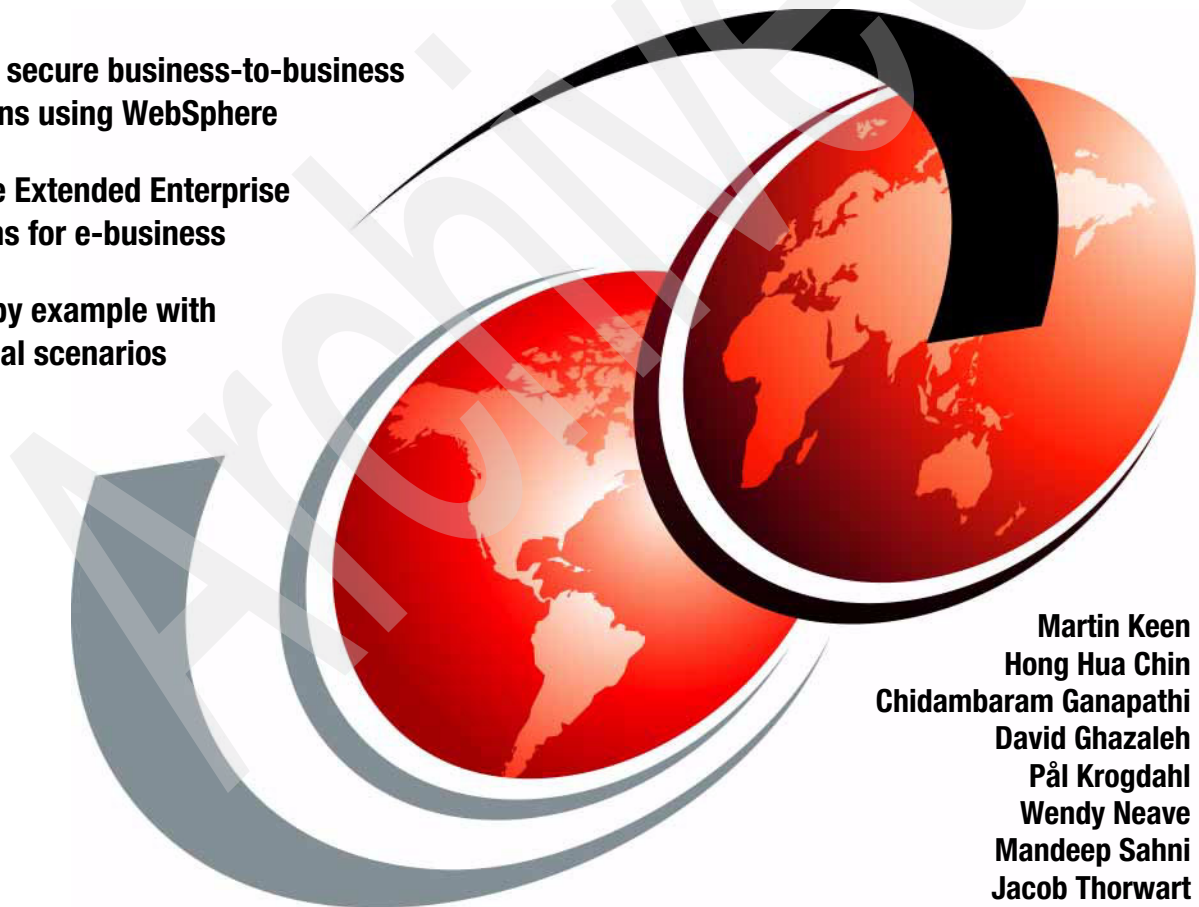


Patterns: Extended Enterprise SOA and Web Services

Design secure business-to-business solutions using WebSphere

Use the Extended Enterprise Patterns for e-business

Learn by example with practical scenarios



Martin Keen
Hong Hua Chin
Chidambaram Ganapathi
David Ghazaleh
Pål Krogdahl
Wendy Neave
Mandeep Sahni
Jacob Thorwart



International Technical Support Organization

Patterns: Extended Enterprise SOA and Web Services

January 2006

Archived

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

First Edition (January 2006)

This edition applies to WebSphere Application Server V6, WebSphere Business Integration Server Foundation V6, and WebSphere Partner Gateway V6.

© Copyright International Business Machines Corporation 2006. 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	xi
Trademarks	xii
Preface	xiii
The team that wrote this redbook	xiv
Become a published author	xvii
Comments welcome	xvii
Part 1. Patterns for e-business and Extended Enterprise	1
Chapter 1. Welcome to this redbook	3
1.1 An introduction to this document	4
1.2 Patterns for e-business SOA series of redbooks	4
1.3 How to read this redbook	6
Chapter 2. Introduction to the Patterns for e-business	11
2.1 The Patterns for e-business layered asset model	12
2.2 How to use the Patterns for e-business	14
2.2.1 Selecting a Business, Integration, Composite pattern, or a Custom design	14
2.2.2 Selecting Application patterns	19
2.2.3 Review Runtime patterns	21
2.2.4 Reviewing Product mappings	23
2.2.5 Reviewing guidelines and related links	24
2.3 Patterns for e-business naming conventions	25
2.4 Summary	25
Chapter 3. Beyond the enterprise	27
3.1 Overview of Extended Enterprise	28
3.2 On Demand Business	28
3.2.1 Key business attributes	29
3.2.2 Key technology attributes	30
3.2.3 Key requirements for integration flexibility	35
3.2.4 The on demand Operating Environment	35
3.3 Approaches for delivering the Extended Enterprise	40
3.3.1 SOA and Web services	41
3.3.2 Traditional approaches	59
3.3.3 Ensuring quality of service	60

Chapter 4. Extended Enterprise pattern	63
4.1 Using the Extended Enterprise business pattern	64
4.2 General guidelines	65
4.2.1 Business and IT drivers	65
4.2.2 Context	65
4.2.3 Solution	67
4.2.4 Employing the pattern	67
4.2.5 What is next?	67
4.3 Extended Enterprise application patterns	68
4.3.1 Exposed Direct Connection application pattern	71
4.3.2 Exposed Direct Connection: Message Connection variation	74
4.3.3 Exposed Direct Connection: Call Connection variation	75
4.3.4 Exposed Broker application pattern	76
4.3.5 Exposed Broker: Router variation	78
4.3.6 Exposed Serial Process application pattern	80
4.3.7 Exposed Serial Process: Workflow variation	83
Chapter 5. Product descriptions	87
5.1 Runtime product descriptions	88
5.1.1 IBM WebSphere Application Server V6	88
5.1.2 IBM DB2 Universal Database Enterprise Server Edition V8.2	91
5.1.3 IBM Cloudscape	92
5.1.4 IBM WebSphere MQ V5.3	93
5.1.5 IBM WebSphere Business Integration Message Broker V5.0	94
5.1.6 IBM WebSphere Business Integration Server Foundation V5.1	94
5.1.7 IBM WebSphere Partner Gateway V6.0	95
5.2 Development product descriptions	96
5.2.1 IBM Rational Application Developer V6	96
5.2.2 IBM WebSphere Studio Application Developer Integration Edition V5.1	97
Chapter 6. Extended Enterprise runtime patterns	99
6.1 Extended Enterprise runtime patterns	100
6.1.1 Generic and SOA profiles	100
6.2 Node types	102
6.2.1 App server/services	102
6.2.2 Network infrastructure	102
6.2.3 Protocol firewall	102
6.2.4 Domain firewall	103
6.2.5 Connector	103
6.2.6 Exposed Connector	104
6.2.7 Exposed ESB Gateway	104
6.2.8 ESB	104

6.2.9 Rules Directory	106
6.2.10 Directory and Security Services	106
6.2.11 Exposed Broker	106
6.2.12 Exposed Router	107
6.2.13 Exposed Process Manager	107
6.2.14 Business Service Choreography	108
6.2.15 Staff Worklist Adapter	108
6.3 Exposed Direct Connection runtime pattern	109
6.3.1 Generic profile	109
6.3.2 SOA profile	111
6.4 Exposed Broker runtime pattern	112
6.4.1 Generic profile	112
6.4.2 SOA profile	113
6.5 Exposed Router variation	115
6.5.1 Generic profile	115
6.5.2 SOA profile	116
6.6 Exposed Serial Process runtime pattern	117
6.6.1 Generic profile	117
6.6.2 SOA profile	119
6.7 Exposed Serial Workflow variation	120
6.7.1 Generic profile	120
6.7.2 SOA profile	121
Chapter 7. Product mappings	125
7.1 Product mappings	126
7.2 Exposed Direct Connection product mapping	126
7.2.1 Generic profile	127
7.2.2 SOA profile	127
7.3 Exposed Broker product mapping	129
7.3.1 Exposed Broker: Generic profile	129
7.3.2 Exposed Router variation: SOA profile	130
7.4 Exposed Serial Process product mapping	132
7.4.1 Generic profile	132
7.4.2 SOA profile	133
Part 2. Business scenario and guidelines	135
Chapter 8. Business scenario used in this book	137
8.1 WS-I sample business scenario	138
8.2 ITSO Good sample business scenario	138
8.2.1 Business context	139
8.2.2 Applications in the supply chain management	139
8.2.3 Example of using the ITSO Good sample application	140

Chapter 9. Technology options	145
9.1 Web services	146
9.1.1 XSD	148
9.1.2 WSDL	148
9.1.3 SOAP	148
9.1.4 UDDI	149
9.1.5 WS-BPEL	150
9.1.6 WS-Security	150
9.2 J2EE	150
9.2.1 JMS	151
9.2.2 Web services for J2EE	151
9.2.3 JAX-RPC	152
9.3 Transport protocols	152
9.3.1 HTTP	152
9.3.2 HTTP/S	153
Part 3. Scenario implementation	155
Chapter 10. Exposed Direct Connection runtime pattern: generic profile	157
10.1 Business scenario	158
10.2 Design guidelines	159
10.2.1 Analyze business requirements	160
10.2.2 Selecting a pattern	160
10.2.3 Analyze design options	162
10.2.4 Products	175
10.3 Development guidelines	177
10.3.1 Exposed Direct Connection interaction: Generic profile	177
10.3.2 Securing applications using WS-Security	179
10.3.3 Generating sample key stores	185
10.3.4 Configuring WS-Security integrity	188
10.3.5 Configuring WS-Security confidentiality	204
10.3.6 Exporting EAR files from Rational Application Developer	214
10.4 Runtime guidelines	215
10.4.1 Solution topology	215
10.4.2 Configuring WebSphere Application Server profiles	217
10.4.3 Hosting the WSDL files	219
10.4.4 Installing the applications	220
10.4.5 Securing the application server using Global Security	223
10.4.6 Configuring an HTTP server for SSL pass-through	224
10.4.7 Changing the Web service client bindings configuration	227
10.4.8 Testing the scenario	229
10.4.9 Viewing SOAP messages using the TCP/IP Monitor	233

Chapter 11. Exposed Direct Connection runtime pattern:	
SOA profile	237
11.1 Business scenario	238
11.2 Design guidelines	239
11.2.1 Analyze IT infrastructure requirements	239
11.2.2 Selecting a pattern	241
11.2.3 Analyze design options	242
11.2.4 Products	249
11.3 Development guidelines	250
11.3.1 Exposed Direct Connection interaction: SOA profile	251
11.4 Runtime guidelines	252
11.4.1 Solution topology	252
11.4.2 Creating the basic infrastructure	254
11.4.3 Create and configure a service integration bus	257
11.4.4 Create and configure the Web service gateway	281
11.4.5 Connecting the ESB and the Exposed ESB Gateway	293
11.4.6 Adding WS-Security to the Web service gateway	304
Chapter 12. Exposed Broker runtime pattern: generic profile	339
12.1 Business scenario	340
12.2 Design guidelines	341
12.2.1 Analyze business requirements	341
12.2.2 Selecting a pattern	341
12.2.3 Analyze design options	343
12.2.4 Products	345
12.3 Development guidelines	346
12.3.1 Scenario implementation: Exposed Broker runtime pattern	347
12.3.2 Mediations	348
12.3.3 Developing a mediation handler class	352
12.3.4 Assigning and exporting the mediation handlers	364
12.4 Runtime guidelines	365
12.4.1 Solution topology	366
12.4.2 Creating the basic infrastructure	367
12.4.3 Configuring the service integration bus	368
12.4.4 Creating the gateway service	371
12.4.5 Installing and defining the mediation application	376
12.4.6 Creating additional destinations	380
12.4.7 Changing the Warehouse endpoint URL	382
12.4.8 Testing the scenario	383
12.4.9 Adding WS-Security to the solution	385
Chapter 13. Exposed Router runtime pattern: SOA profile	387
13.1 Business scenario	388

13.2	Design guidelines	389
13.2.1	Analyze business requirements	389
13.2.2	Selecting a pattern	389
13.2.3	Analyze design options	391
13.2.4	Products	394
13.3	Development guidelines	396
13.3.1	Scenario implementation: Exposed Router SOA profile interaction	396
13.4	Runtime guidelines	397
13.4.1	Solution topology	397
13.4.2	Creating the basic infrastructure	398
13.4.3	Scenario implementation overview	399
13.4.4	Configuring WebSphere Partner Gateway	400
13.4.5	Configuring WebSphere Application Server	414
13.4.6	Testing the WebSphere Partner Gateway configuration	416
Chapter 14. Exposed Serial Process runtime pattern: generic profile		419
14.1	Business scenario	420
14.2	Design guidelines	420
14.2.1	Analyze business requirements	420
14.2.2	Selecting a pattern	421
14.2.3	Analyze design options	422
14.2.4	Products	424
14.3	Development guidelines	425
14.3.1	Scenario implementation: Serial process interaction	425
14.3.2	Creating the basic infrastructure	427
14.3.3	Configuring WebSphere Studio	429
14.3.4	Creating Manufacturer and LoggingFacility Web services clients	429
14.3.5	Create Java proxy classes	435
14.3.6	Create a business process using Process Choreographer	439
14.3.7	Create the Warehouse service	449
14.3.8	Exporting the Enterprise Application files	452
14.4	Runtime guidelines	452
14.4.1	Testing with Web Services Explorer	453
14.4.2	Testing the business process with ITSO Good	455
14.4.3	Deploying the business process	458
Chapter 15. Exposed Serial Process runtime pattern: SOA profile		459
15.1	Business scenario	460
15.2	Design guidelines	460
15.2.1	Analyze the business requirement	460
15.2.2	Selecting a pattern	460

15.2.3 Analyze design options	462
15.2.4 Products	463
15.3 Development guidelines	465
15.3.1 Scenario implementation: Serial process interaction	465
15.3.2 Creating the basic infrastructure	466
15.3.3 Creating a Manufacturer Web service client	468
15.3.4 Modify the Manufacturer proxy class	469
15.3.5 Modify the Warehouse business process	469
15.3.6 Generate deployment code and export the process	471
15.4 Runtime guidelines	471
15.4.1 Configuring the ESB	472
15.4.2 Configuring the Exposed ESB Gateway	473
15.4.3 Testing the business process with ITSO Good	476
Part 4. Appendixes	479
Appendix A. Additional material	481
Locating the Web material	481
Using the Web material	481
System requirements for downloading the Web material	482
How to use the Web material	482
Appendix B. Microsoft .NET Web services	483
B.1 Overview and context of .NET Web services	484
B.1.1 How Microsoft .NET is used in the Redbook scenarios	484
B.1.2 Microsoft .NET Web service development overview	485
B.2 Implementing a Microsoft .NET Web service	486
B.2.1 Create a new Web service project	487
B.2.2 Generating a C# file using a WSDL file and wsdl.exe	487
B.2.3 Modifying the C# file	488
B.2.4 Finalizing and deploying the Web service	491
B.2.5 Testing the .NET Web service	493
Implementing a test J2EE Client	494
B.2.6 Creating a new Rational Application Developer project	494
B.2.7 Importing the necessary WSDLs and XSDs	495
B.2.8 Deploying and testing the J2EEClient	497
B.3 Enabling transport-level security with SSL	500
B.3.1 Configuring the .NET Web service to require SSL	501
B.3.2 Importing the SSL certificate into a key database	507
Appendix C. CICS Transaction Server Web services	513
C.1 CICS Transaction Server V3.1 Web services support	514
C.2 Creating Web services for CICS	514
C.2.1 CICS Web services assistant	515

C.2.2 CICS resources for Web services	515
C.3 Creating and hosting a ManufacturerC Web service	516
Appendix D. WSAdmin Automation Platform	517
D.1 Employing WSAdmin Automation Platform	518
D.1.1 Overview of WSAdmin Automation Platform	518
D.1.2 Downloading WSAdmin Automation Platform	519
D.1.3 Running WSAdmin Automation Platform	520
D.1.4 WSAdmin Automation Platform examples	522
D.1.5 The You Name It option	526
Abbreviations and acronyms	527
Related publications	529
IBM Redbooks	529
Other publications	529
Online resources	530
How to get IBM Redbooks	531
Help from IBM	531
Index	533

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:

CICS®	e-business on demand™	Rational®
Cloudscape™	@server®	Redbooks (logo)  ™
DB2 Connect™	@server®	Redbooks™
DB2 Universal Database™	IBM®	Tivoli®
DB2®	IMS™	Trading Partner®
developerWorks®	iSeries™	WebSphere®
Distributed Relational Database	Lotus®	xSeries®
Architecture™	MQSeries®	z/OS®
Domino®	PartnerLink®	

The following terms are trademarks of other companies:

EJB, Java, Java Naming and Directory Interface, JDBC, JSP, JVM, J2EE, Sun, Sun Certified Programmer for Java, Sun Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Visual C#, Visual Studio, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

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

Preface

Service-oriented architecture (SOA) promotes the ability to communicate with external enterprises. But what are the issues in creating these Extended Enterprise communications? This IBM® Redbook addresses these issues for Web services implementations of SOA, using the Patterns for e-business.

The *Patterns for e-business* are a group of proven, reusable assets that can be used to increase the speed of developing and deploying e-business applications. This IBM Redbook focuses on building Extended Enterprise SOA solutions using WebSphere® Application Server V6, WebSphere Partner Gateway V6, the Web services gateway component of WebSphere Application Server Network Deployment V6, and WebSphere Business Integration Server Foundation V5.1.

Part 1 introduces the Patterns for e-business, and describes the patterns and product mappings within this framework for building Extended Enterprise solutions.

Part 2 describes the business scenario used throughout this book, and describes the technologies for implementing an SOA solution.

Part 3 provides a set of Extended Enterprise scenarios that include simple as well as more complex SOA solutions that use an Enterprise Service Bus.

IBM's SOA announcements:

In September 2005, IBM announced two products intended to be the primary solution for building ESBs:

- ▶ **WebSphere Enterprise Service Bus V6**
Delivers an ESB with Web services connectivity and data transformation.
- ▶ **WebSphere Message Broker V6**
Delivers an advanced ESB with universal connectivity and data transformation.

At the time this redbook was written, WebSphere Enterprise Service Bus was not generally available. In lieu of this product, the service integration bus of WebSphere Application Server V6 is used in the redbook scenario implementations to build ESB solutions.

For more information about IBM's ESB strategy see:

<http://www.ibm.com/software/inf01/websphere/index.jsp?tab=landings/esb>

IBM also announced WebSphere Process Server V6, as a runtime for SOA business processes. This product will ultimately replace WebSphere Business Integration Server Foundation V5.1. WebSphere Process Server was not generally available at the time this redbook was written, so this redbook uses WebSphere Business Integration Server Foundation V5.1 for all scenarios requiring a business process engine.

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.



Figure 0-1 Redbook team (left to right): Martin, Chidu, Pål, Wendy, Mr. David, Mandeep, Hong, and Jake

Martin Keen is a Senior IT Specialist at the ITSO, Raleigh Center. He writes extensively about WebSphere products, SOA, and Patterns for e-business. He also teaches IBM classes worldwide about WebSphere, SOA, and business process management. Before joining the ITSO, Martin worked in the EMEA WebSphere Lab Services team in Hursley, UK. Martin holds a bachelor's degree in Computer Studies from Southampton Institute of Higher Education, in the United Kingdom.

Hong Hua Chin is an Advisory IT Specialist for IBM Software Group, ASEAN/South-Asia. He has five years of experience in J2EE™ and middleware technologies including WebSphere and DB2®. He holds a degree in Computer and Information Sciences from the National University of Singapore. He focuses on technical selling of WebSphere Business Integration products. He has written, presented and taught extensively on enterprise architecture, service-oriented architecture, open computing standards such as Web services, and integration best practices.

Chidambaram Ganapathi is an Associate IT Architect with IBM Global Services India. He has over seven years experience in the IT Industry architecting and delivering e-business solutions for airline, telecommunications and banking industries. He has also worked on e-commerce applications. He specializes in building industry-specific frameworks that helps customers in rapid application development. His current areas of interest include service-oriented architecture, component-based architecture and autonomic computing.

David Ghazaleh is a Software Engineer with IBM in Raleigh, North Carolina, U.S. He has 19 years of experience in the Computer Science industry. His areas

of expertise include System Software Development, System Test Automation, J2EE and Relational Database Management Systems. He is a Sun™ Java™ 2 Certified Programmer. He holds a bachelor's degree in Computer Science from Catholic University of Petropolis, Brazil.

Pål Krogdahl is a Senior IT Architect and Method Exponent with the IBM Nordic Financial Services Sector. He has been working for IBM since 1998 in various areas such as software development, technical pre-sales consulting, and solution architecture. His areas of expertise are in distributed computing, middleware, and Application Services Architecture, with focus on Enterprise Application Integration (EAI) and SOA.

Wendy Neave is a Certified Senior IT Specialist in IBM Global Services in Australia. She has 16 years of experience in application design and development. She holds a Bachelor of Education (Environmental Science) and an Associate Diploma in Computing. She is also a Sun Certified Programmer for Java™ 2. Her areas of expertise include object-oriented analysis, design and development, Java, and WebSphere. Wendy has previously contributed to the IBM Redbooks™ *Patterns: Direct Connections for Intra- and Inter-enterprise* and *BPEL4WS Business Processes with WebSphere Business Integration*.

Mandeep Sahni is an IT Specialist in IBM Global Services, Australia. He has over 5 years of experience in IT. He holds a Masters degree in Information Technology from Swinburne University of Technology, Melbourne. His areas of expertise include J2EE, WebSphere, Web Services, WebSphere Business Integration Server Foundation, WS-BPEL and other Web technologies.

Jacob Thorwart is a Co-op IT Specialist at the IBM ITSO Center in Raleigh, North Carolina. He is currently pursuing a Bachelor of Science degree in Information Sciences and Technology at Pennsylvania State University. His interests include systems integration and Web applications.

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

Jonathan Adams and Paul Verschueren
Patterns for e-business leadership and architecture, IBM UK

Lee Gavin,
Edward Oguejiofor
International Technical Support Organization, Raleigh Center

Bob O'Hanlon
IBM Web Services Test Team, Hursley, UK

Vamsi Namuduri, Mohan Annamalai, Kiran Venkatachala, Ashutosh Arora,
Robert Perry
IBM WebSphere Partner Gateway Development

Stepan Husek
IBM Global Services, Czech Republic

Linda Robinson and Lauren Bedford-Gaines
ITSO Graphics Support

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 email 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 and Extended Enterprise

Archived

Welcome to this redbook

This chapter introduces this redbook to you and provides guidelines for how to read it. It contains the following sections:

- ▶ An introduction to this document
- ▶ Patterns for e-business SOA series of redbooks
- ▶ How to read this redbook

1.1 An introduction to this document

A warm welcome to this redbook, from the IBM Redbook team. We all assembled for five intense weeks in Raleigh, North Carolina, and then again remotely through the review process, to put together this resource. We hope you find it a useful read.

This document focuses on patterns for integrating multiple enterprises together (we call this the Extended Enterprise) using service-oriented architecture (SOA) and Web services. We expand this focus to include a hot topic in SOA architecture called the Enterprise Service Bus (ESB).

This book was designed with IT Architects, System Administrators, and Application Programmers in mind. Throughout this book we discuss a number of architectural patterns for building Extended Enterprise scenarios, highlight the IBM product mappings available to implement these scenarios, then provide step-by-step instructions on how to implement these scenarios in the given products.

The product mappings we use within this book represent IBM products that were generally available at the time of writing. In September 2005, as part of an SOA announcement, IBM announced several new products that can also be applied to these patterns. Specifically, these products are:

- ▶ WebSphere Process Server V6

This product provides Serial Process / Parallel Process / Business Service Choreography capabilities, and succeeds WebSphere Business Integration Server Foundation V5.1.

- ▶ WebSphere Enterprise Service Bus V6

This product provides ESB capabilities, delivering a more powerful alternative to the service integration bus of WebSphere Application Server V6 for building ESB solutions.

- ▶ WebSphere Message Broker V6

This product contains advanced ESB capabilities, primarily for use when integrating a large number of non-Web services based solutions is required.

1.2 Patterns for e-business SOA series of redbooks

The Patterns for e-business describe proven solutions to solve common business problems. A given business problem is mapped as a set of patterns, and the selection of these patterns leads to a number of proven product mappings. In this

redbook we describe these patterns, and provide implementations for most of the product mappings.

The Patterns for e-business contain a number of patterns and product mappings for SOA solutions; from simple point-to-point solutions, to complex Extended Enterprise solutions using an ESB. A series of redbooks have been created to describe these SOA patterns and provide product mapping implementations. This redbook is part of this SOA series.

The following redbooks, shown in order of publication, are part of the Patterns for e-business SOA series:

► *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303

This book introduces SOA concepts and the rudimentary SOA profile of the Patterns for e-business. Scenario chapters are provided, offering design, development, and runtime guidelines for building SOA implementations in WebSphere Application Server V5.

► *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346

This book provides a more in-depth description of SOA and Web services technologies and introduces the SOA concept of the ESB. It expands the Patterns for e-business SOA profile to provide ESB guidelines. This book also provides scenario chapters to show ESB implementations that are created in WebSphere Application Server V5 and WebSphere Business Integration Message Broker V5. An additional scenario chapter describes how WebSphere Business Integration Server Foundation V5.1 can interact with an ESB.

► *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6*, SG24-6494

This book extends the SOA Patterns for e-business introduced in *Patterns: Implementing an SOA Using an Enterprise Service Bus* with product mappings for WebSphere Application Server V6.

► *Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture*, SG24-6773

This book discusses the integration of multiple ESBs within an enterprise. The integration between homogeneous and heterogeneous ESBs is discussed with product mappings using WebSphere Application Server V6 and WebSphere Business Integration Message Broker V5.

1.3 How to read this redbook

As much as the redbook team would love you to read every page of this book cover-to-cover, we anticipate this may not be the case for every reader! To help you locate the information you need, and to provide guidance on which chapters are of most interest to you, this section provides a short description of each chapter.

Part 1. Patterns for e-business and Extended Enterprise

This part introduces the Patterns for e-business and the issues with connecting to an external enterprise. It then describes the products and patterns for implementing Extended Enterprise solutions.

- ▶ Chapter 1. Welcome to this redbook

- ▶ Chapter 2. Introduction to the Patterns for e-business

The Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying e-business applications. This book uses the Patterns for e-business to indicate how to develop and deploy SOA solutions. This chapter provides an introduction to what the Patterns for e-business are at a general level.

- ▶ Chapter 3. Beyond the enterprise

Discusses the issues of connecting to business logic running in external enterprises, from a traditional and Web services viewpoint. This is targeted at readers wishing to gain a fuller understanding of the issues involved in communicating with an Extended Enterprise.

- ▶ Chapter 4. Extended Enterprise pattern

Describes the Extended Enterprise business pattern and Application patterns from the Patterns for e-business. These high level patterns help describe the common interactions between multiple enterprises, and the tiers required to implement these interactions.

- ▶ Chapter 5. Product descriptions

Describes the IBM products discussed and implemented within this redbook to implement Extended Enterprise scenarios.

- ▶ Chapter 6. Extended Enterprise runtime patterns

Explains the Extended Enterprise runtime patterns for the Patterns for e-business. Each Runtime pattern describes the logical architecture that is required to implement an Application pattern. Both generic and SOA Runtime pattern profiles are discussed.

- ▶ Chapter 7. Product mappings

Product mappings are typical and proven implementations of a Runtime pattern, using IBM products. This chapter highlights product mappings for each Extended Enterprise runtime pattern. Most of these product mappings are implemented in Part 3 of this redbook.

Part 2. Business scenario and guidelines

This redbook provides six Extended Enterprise scenario implementations, based on the product mappings from Chapter 7. Each of these scenario implementations uses a common business scenario case study, and a collection of technologies. These are introduced in this section.

- ▶ Chapter 8. Business scenario used in this book
Describes the business scenario used throughout Part 3 of this redbook. The business scenario is based on a sample application provided by the Web Services Interoperability Organization (WS-I).
- ▶ Chapter 9. Technology options
Introduces the technologies required to implement Extended Enterprise interactions using Web services.

Part 3. Scenario implementation

Six Extended Enterprise scenarios are implemented in this section, based on the Runtime patterns identified in Chapter 6 and the product mappings described in Chapter 7.

Each scenario chapter is divided into three distinct parts:

- ▶ Design guidelines
Primarily intended for architects. This section describes the design alternatives that you should consider when designing a particular scenario.
- ▶ Development guidelines
Primarily intended for application developers. This section describes the application development changes that are required when implementing a particular scenario.
- ▶ Runtime guidelines
Primarily intended for system administrators. This section describes how to deploy a particular scenario, and the runtime alternatives that are available.

This redbook contains the following scenario chapters:

- ▶ Chapter 10. Exposed Direct Connection runtime pattern: generic profile
Describes how to design and build a basic Extended Enterprise solution using point-to-point connections between Web services, including the use of

HTTP/S and WS-Security. Rational® Application Developer and WebSphere Application Server are used. As with all the scenarios, this scenario communicates with Web services running in WebSphere Application Server, Microsoft® .NET, and CICS® Transaction Server.

- ▶ Chapter 11. Exposed Direct Connection runtime pattern: SOA profile
Each SOA profile chapter describes how to implement an Extended Enterprise solution in an environment where an ESB is used. This chapter describes how to implement point-to-point Web service connections, using an ESB implemented in WebSphere Application Server to manage WS-Security.
- ▶ Chapter 12. Exposed Broker runtime pattern: generic profile
This chapter describes how to build a WebSphere Application Server mediation to an Extended Enterprise solution.
- ▶ Chapter 13. Exposed Router runtime pattern: SOA profile
Describes how to configure WebSphere Partner Gateway to communicate with the Extended Enterprise.
- ▶ Chapter 14. Exposed Serial Process runtime pattern: generic profile
Examines how to combine Extended Enterprise interactions with Web services based business processes. It uses WebSphere Business Integration Server Foundation as the business process engine.
- ▶ Chapter 15. Exposed Serial Process runtime pattern: SOA profile
Describes how WebSphere Business Integration Server Foundation can be combined with an ESB for Extended Enterprise access.

Appendixes

Each scenario implementation chapter in Part 3 describes how to invoke Web services located in an Extended Enterprise. The Web service implementations in the Extended Enterprise run in WebSphere Application Server, Microsoft .NET, and CICS Transaction Server.

- ▶ Appendix A. Additional material
- ▶ Appendix B. Microsoft .NET Web services
Provides step-by-step instructions for implementing Web services in Microsoft .NET, and how to secure them. These Web services can be used in the scenario implementations in Part 3 of this redbook.
- ▶ Appendix C. CICS Transaction Server Web services
Provides a high-level overview of the steps required to build and deploy a Web service in CICS Transaction Server that can be used by the scenario implementations in Part 3 of this redbook.

- ▶ Appendix D. WSAAdmin Automation Platform

Introduces a useful tool for configuring WebSphere Application Server that we made use of in the development of this redbook.

Archived

Archived

Introduction to the Patterns for e-business

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

To improve the process over time, we need to capture and reuse the experience of the IT architects in such a way that future engagements can be made simpler and faster. We do this by capturing knowledge gained from each engagement and using it to build a repository of assets. IT architects can then build future solutions based on these proven assets. This reuse saves time, money, and effort and helps ensure delivery of a solid, properly architected solution.

The IBM Patterns for e-business help facilitate this reuse of assets. Their purpose is to capture and publish e-business artifacts that have been used, tested, and proven to be successful. 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.

2.1 The Patterns for e-business layered asset model

The Patterns for e-business approach enables architects to implement successful e-business solutions through the reuse 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 and include:

- ▶ 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 that describe how the application components and data within a Business pattern or Integration pattern interact.
- ▶ Runtime patterns that define the logical middleware structure that supports 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.

Figure 2-1 on page 13 shows these assets and their relationships to each other.

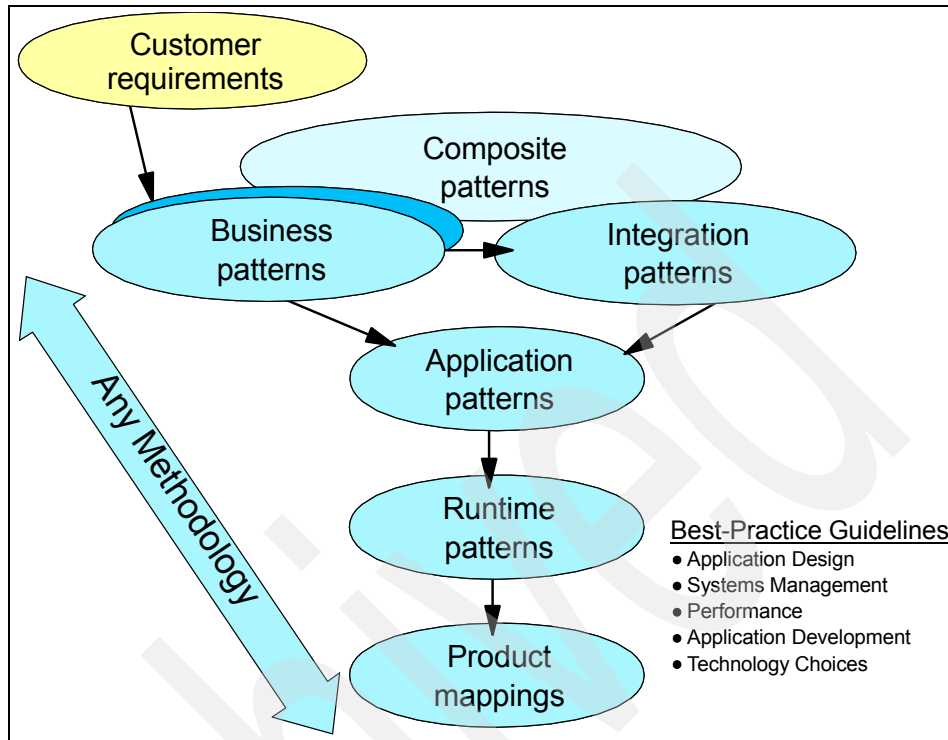


Figure 2-1 The Patterns for e-business layered asset model

Patterns for e-business Web site

The layers of patterns, along with their associated links and guidelines, allow the architect to start with a problem and a vision for the solution, then find a pattern that fits that vision. In order to navigate from top down from one level to another, a decision matrix will be provided to assist the architect in making the right decision.

Then, by drilling down using the patterns process, the architect can further define the additional functional pieces that the application needs to succeed. Finally, the architect can build the application using coding techniques that are outlined in the associated guidelines.

The Patterns Web site provides an easy way of navigating through the layered Patterns assets to determine the most appropriate assets for a particular engagement.

For easy reference, see the Patterns for e-business Web site at:

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

2.2 How to use the Patterns for e-business

As described in the previous section, the Patterns for e-business have 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 2-1 on page 13 above the Business patterns. However, Composite patterns are made up of a number of individual Business patterns and at least one Integration pattern. This section discusses how to use the layered structure of Patterns for e-business assets.

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

When faced with the challenge of designing a solution for a business problem, the first step is to get a high-level view of the goals that you are trying to achieve. You need to describe a proposed business scenario and match each element to an appropriate IBM Pattern for e-business. You might 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. By allowing customers to view their policy information and request changes online, the company can cut back significantly on the resources that are spent handling this type of request by phone. The objective allows policy holders to view policy information that is stored in existing databases.

The Self-Service business pattern fits this scenario perfectly. You can use it in situations where users need direct access to business applications and data. The following sections discuss 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, that are explained in Table 2-1.

Table 2-1 The four primary Business patterns

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

It would be very convenient if all problems fit nicely into these four slots, but reality says that things can often be more complicated. The patterns assume that most problems, when broken down into their basic components, will fit more than one of these patterns. When a problem requires multiple Business patterns, you can use Integration patterns.

Integration patterns

Integration patterns allow you to tie together multiple Business patterns to solve a business problem. Table 2-2 describes the Integration patterns.

Table 2-2 *Integration patterns*

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, data propagators, and data federation engines

The Access Integration pattern maps to User Integration. The Application Integration pattern is divided into two essentially different approaches:

- ▶ Process Integration, which is the integration of the functional flow of processing between the applications.
- ▶ Data Integration, which is the integration of the information that is used by applications.

You can combine the Business and Integration patterns to implement installation-specific business solutions called a Custom design.

Custom design

Figure 2-2 illustrates the use of a Custom design to address a business problem.

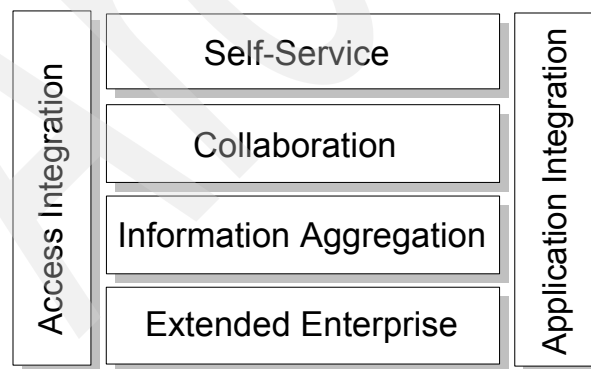


Figure 2-2 *Patterns representing a Custom design*

If you do not use any of the Business or Integration patterns in a Custom design, you can show the unused patterns as lighter blocks than those patterns that you do use. For example, Figure 2-3 shows a Custom design that does not have a Collaboration or an Extended Enterprise business pattern for a business problem.



Figure 2-3 Custom design showing unused patterns

If a Custom design recurs many times across domains that have similar business problems, then it can also be a Composite pattern. For example, the Custom design in Figure 2-3 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. Table 2-3 on page 18 shows the identified Composite patterns.

Table 2-3 Composite patterns

Composite Patterns	Description	Examples
Electronic Commerce	User-to-online-buying	<ul style="list-style-type: none"> • http://www.macys.com • http://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 applications. • Telephone company account manager functions. • Bank, credit card and insurance company online applications.
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.	http://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.	http://www.wwre.org (WorldWide Retail Exchange)

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

2.2.2 Selecting Application patterns

After you identify the Business pattern, 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 usually has multiple possible Application patterns. An Application pattern might have logical components that describe a presentation tier for interacting with users, an application tier, and a back-end application tier.

Application patterns break down the application into the most basic conceptual components that identify 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. Figure 2-4 shows the Self-Service::Directly Integrated Single Channel application pattern, which fulfills this requirement.

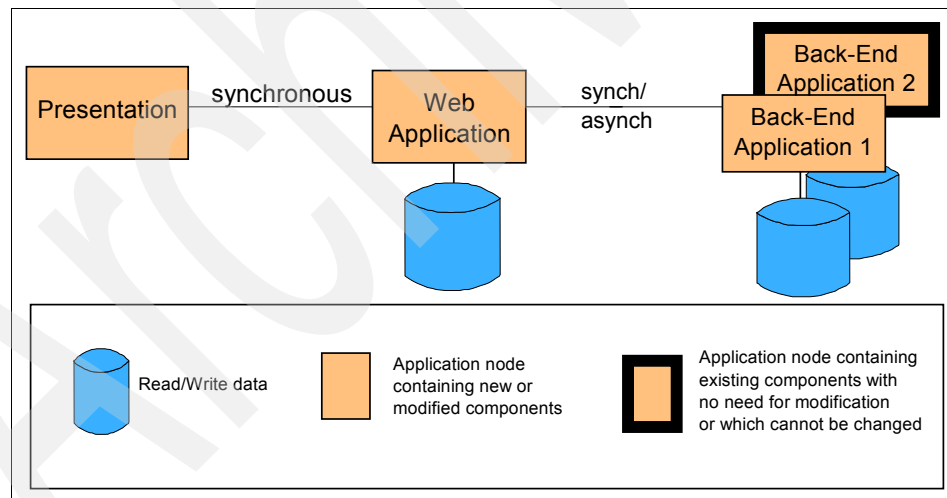


Figure 2-4 Self-Service::Directly Integrated Single Channel pattern

This Application pattern consists of a presentation tier that handles the request and 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. Let's say that the automobile policies and the homeowner policies are kept in two separate and dissimilar databases. The user request actually needs 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 that is sent back from the requests, and put this information into the form of a response (recompose). In this case, the Self-Service::Decomposition application pattern (as shown in Figure 2-5) would be more appropriate.

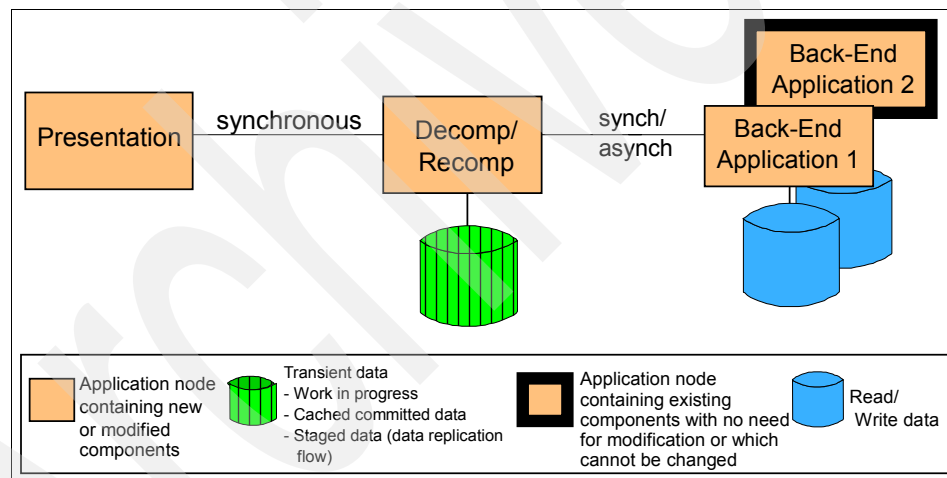


Figure 2-5 Self-Service::Decomposition pattern

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

2.2.3 Review Runtime patterns

You can refine the Application pattern further with more explicit functions. Each function is associated with a runtime node. In reality, these functions, or nodes, can exist on separate physical machines or can coexist on the same machine. In the Runtime pattern the physical location of the function is not relevant. The focus is on the logical nodes that are required and their placement in the overall network structure.

As an example, let's say that our customer has determined that their 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 the situation.

The customer knows that they will have users on the Internet that are accessing their business data; therefore, they require a measure of security. You can implement security 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 the company network.

The customer also needs to determine the functional nodes that are required to implement the application and security measures. Figure 2-6 on page 22 shows the Runtime pattern that is one option.

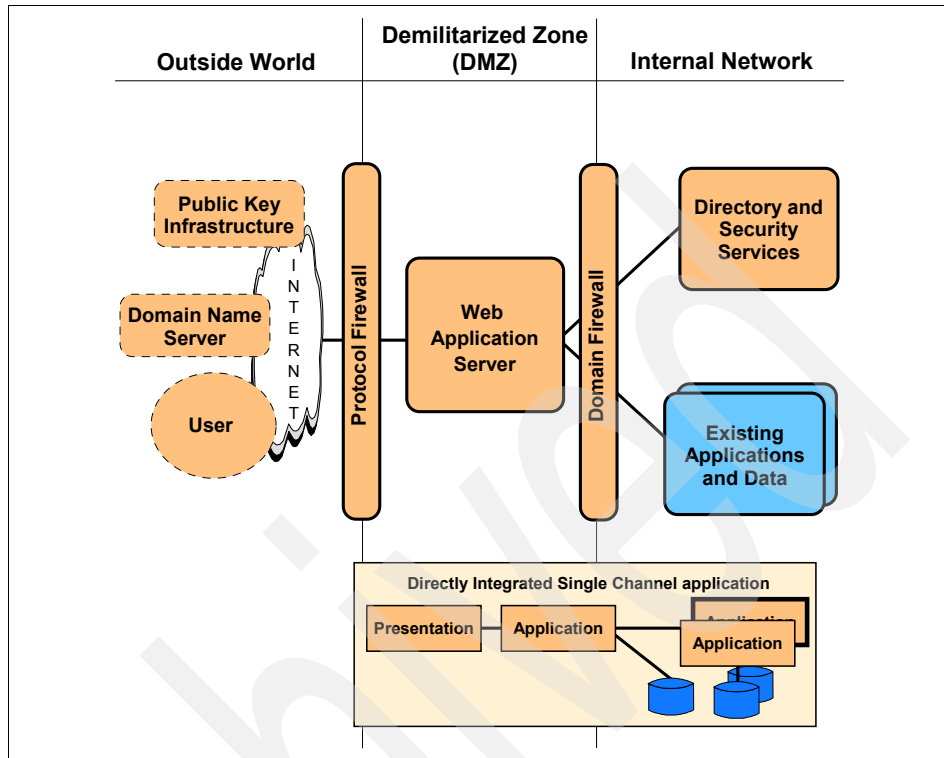


Figure 2-6 Directly Integrated Single Channel application pattern::Runtime pattern

By overlaying the Application pattern on the Runtime pattern, you can see the roles that each functional node fulfills 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. The Application pattern 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. Figure 2-7 on page 23 shows a variation of this pattern. 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) provides static Web pages and redirects other requests to the application server. This pattern moves the application server function behind the second firewall, adding further security.

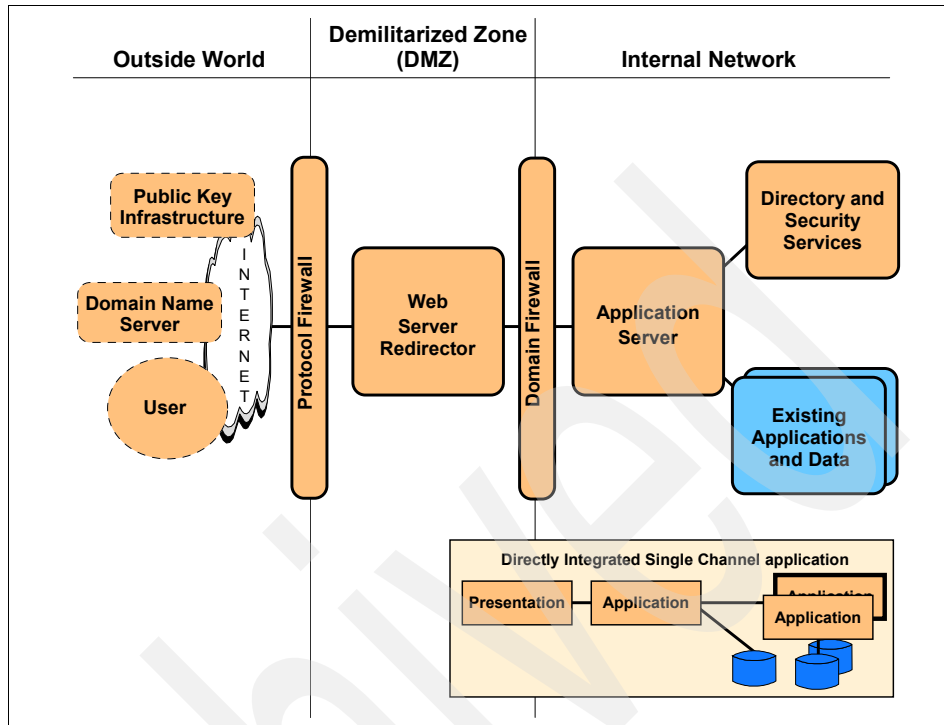


Figure 2-7 Directly Integrated Single Channel application pattern::Runtime pattern

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

2.2.4 Reviewing 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. However, it is more likely that the customer will have a variety of platforms involved in the network. In this case, you can mix and match product mappings (this is dependent on the supported platforms of the IBM products).

For example, you could implement the runtime variation in Figure 2-7 on page 23 using the Product mapping depicted in Figure 2-8.

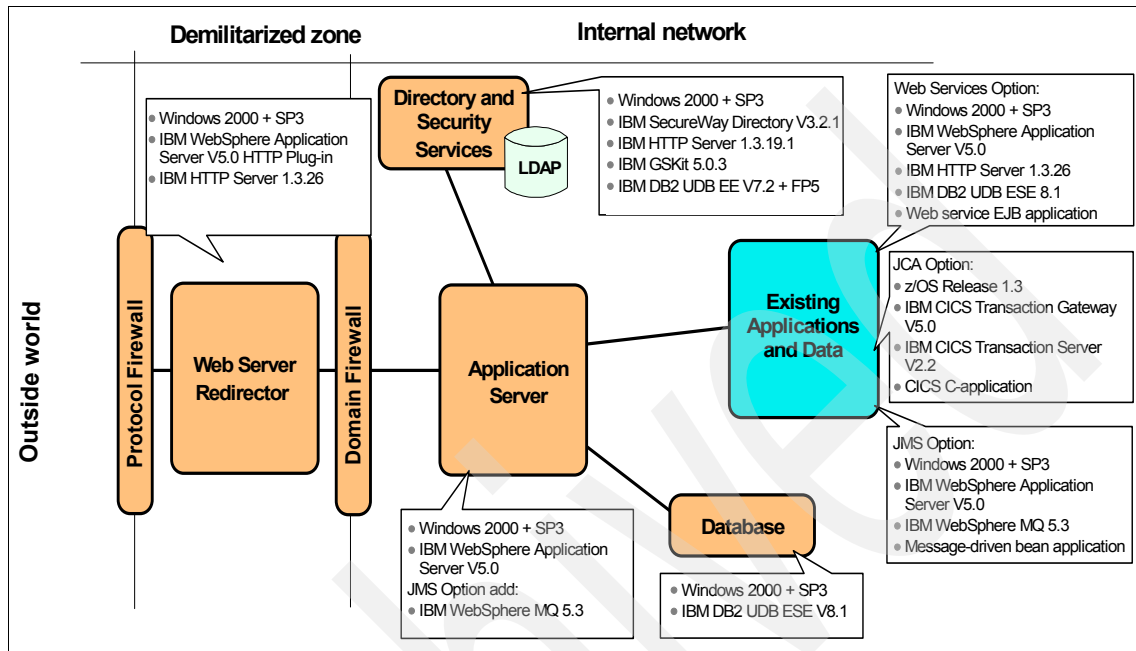


Figure 2-8 Directly Integrated Single Channel application pattern::Product mapping=Windows® 2000

2.2.5 Reviewing guidelines and related links

The Application patterns, Runtime patterns, and Product mappings can 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 guidelines:

- ▶ Design guidelines provide 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, and so forth.
- ▶ Performance guidelines give information about how to improve the application and system performance.

2.3 Patterns for e-business naming conventions

The Patterns for e-business use a standard naming convention with the objective of making it easier for the reader to fully identify the referenced asset.

The capitalization convention is to use lower case for pattern and upper case for the first and most significant qualifier as seen in the following example:

- ▶ Business pattern

When referencing a specific type of pattern, the higher level qualifier (business) is not capitalized as seen in the following example:

- ▶ Self-Service business pattern

The textual notation Business pattern::Application pattern::Runtime pattern::Product mapping is used to represent the position of an asset within the hierarchy. Occasionally an intermediate level or the pattern type will be omitted for brevity as seen in the following examples:

- ▶ Self-Service::Router application pattern
- ▶ Self-Service::Router runtime pattern

In addition, when it is necessary to identify variations or product instances at the same level in the hierarchy an '=' sign will be used as seen in the following examples:

- ▶ Self-Service::Decomposition=Integration Server runtime pattern
- ▶ Application Integration::Direct Connection=Message Connection::Product mapping=Web services

2.4 Summary

The IBM Patterns for e-business are a collected set of proven architectures. You can use this repository of assets to facilitate the development of Web-based applications. Patterns for e-business help you understand and analyze complex business problems and break them down into smaller, more manageable functions that you can then implement.

Archived

Beyond the enterprise

This chapter discusses the fundamental aspects and needs for system and application integration across the enterprise boundaries, by covering the following:

- ▶ An initial overview of the Extended Enterprise, and the business need.
- ▶ The on demand business and operating environment.
- ▶ Approaches for delivering the Extended Enterprise including a discussion about service-oriented architecture and Web services.
- ▶ A quick look at the required Quality of Services.

3.1 Overview of Extended Enterprise

In the past, executives had the luxury of assuming that business models were more or less immortal. Companies always had to work to get better... but they seldom had to get different—not at their core. (From “The Quest for Resilience,” Gary Hamel and Liisa Valikangas, Harvard Business Review, September 2003) But during the last 10 years we have seen an increasing number of organizations forced to get different, and not just better, to survive.

As more organizations are looking to streamline their core business processes, an inherent need to both outsource noncritical business functions, and integrate with customer and supplier IT systems increases. In this fundamental shift from an internally controlled Application Access paradigm, to an Extended Enterprise interacting within a globally distributed ecosystem, as depicted in Figure 3-1, there is a need for a flexible and agile system integration architecture.

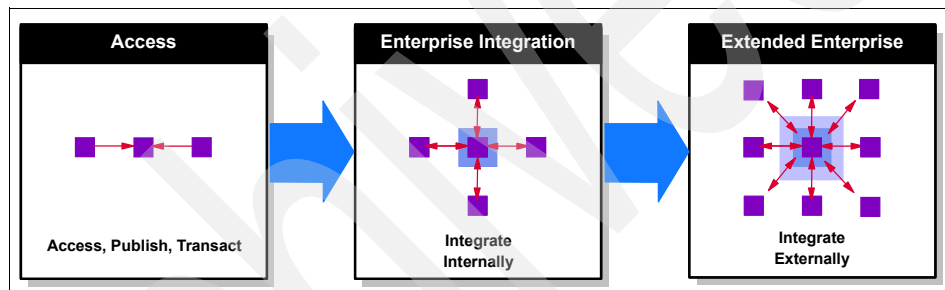


Figure 3-1 The Business Integration Evolution

The fundamental principle of any Extended Enterprise is to support the ability to integrate both business processes and IT systems across organizational boundaries. By ensuring a flexible and manageable integration with both the organizations customers and suppliers as well as value-added business partners, the organization is better positioned to focus on its core competencies.

But this shift in business outsourcing and system integration also creates a number of complications within the underlying IT infrastructure. In an effort to reduce this complexity and streamline the Integration Architecture, IBM introduced the on demand operating environment.

3.2 On Demand Business

The IBM vision of On Demand Business™ is to enable customers to succeed in an environment with an unprecedented rate of change, or within an architecture with a high degree of external integrations.

In today's corporate landscape, businesses want to focus on core competencies, reduce spending, and reuse existing information in new ways without performing a major overhaul of their existing infrastructure. There exists a constant pressure to juggle the often conflicting demands to provide flexibility, cost savings, and efficiency. Figure 3-2 outlines the key components required for any On Demand Business.

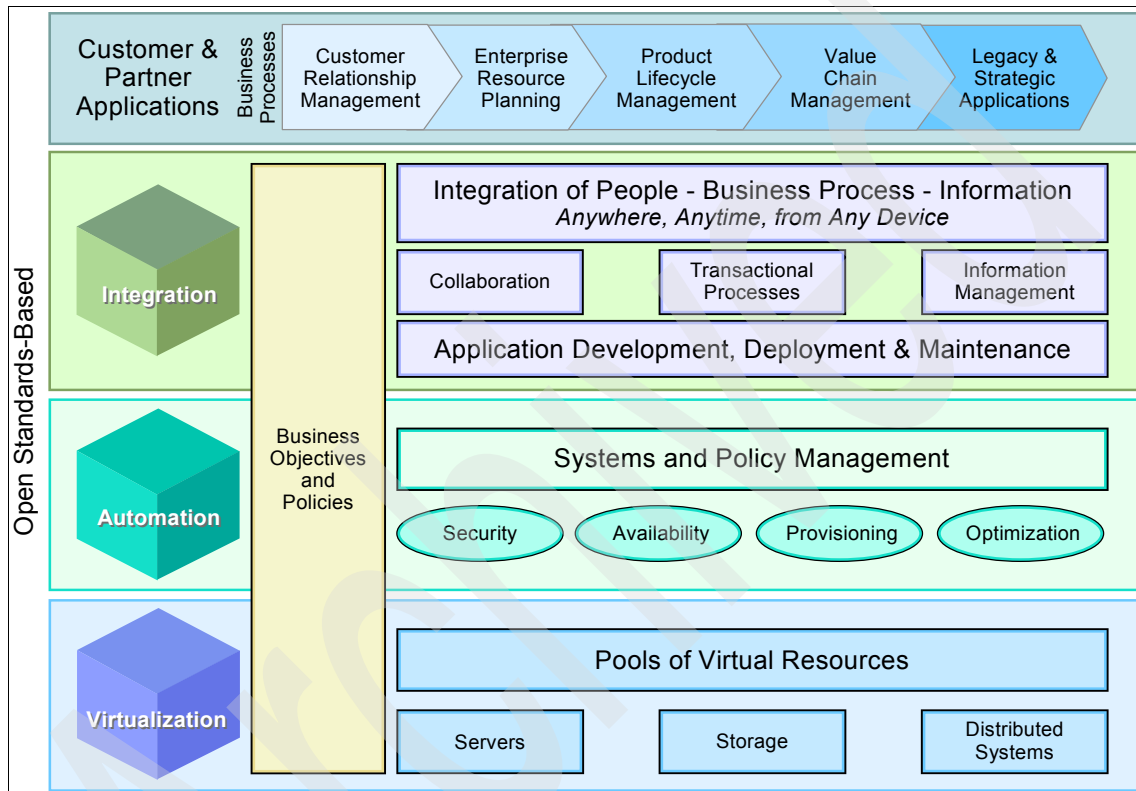


Figure 3-2 On Demand Business overview diagram

3.2.1 Key business attributes

From a business perspective, On Demand Business is about providing a way for companies to realign their business and technology environment to match the request for reusable business functionality.

Business drivers can be summarized as having the following key characteristics:

- Focused

Being *focused* means the enterprise focuses on its core competencies, those activities that make that enterprise successful and unique. Strategic alliances

are formed to provide needs external to these core competencies, and supported by a deployed extended enterprise based IT architecture.

- ▶ Responsive

To say an enterprise is *responsive*, means it has the ability to respond with agility to customer demands, market opportunities, or external threats. These decisions are guided through insight-driven decision-management features, and can be hindered by a badly executed and deployed Extended Enterprise-based architecture.

- ▶ Variable

Successful enterprises achieve operational and business process flexibility. They adapt *variable* cost structures (fixed to variable) to provide a high level of operational efficiency.

- ▶ Resilient

The hallmark of a *resilient* enterprise is the capability and robustness to respond to changes in both business and technical environments. Resilient enterprises manage changes and threats with predictable outcomes.

Companies can achieve these business imperatives by exploiting current technological developments while drawing on experiences that have been learned from past architectural constructs.

3.2.2 Key technology attributes

The business drivers of On Demand Business must be supported by a well-defined technical infrastructure. Again, here we see a strong bias towards a well-defined and executed integration architecture.

These key technological attributes deliver the flexibility, responsiveness, and efficiency that on demand organizations require:

- ▶ Integration
- ▶ Virtualization
- ▶ Automation
- ▶ Open standards

Figure 3-3 on page 31 provides a high-level overview of the range of each On Demand Business attribute.

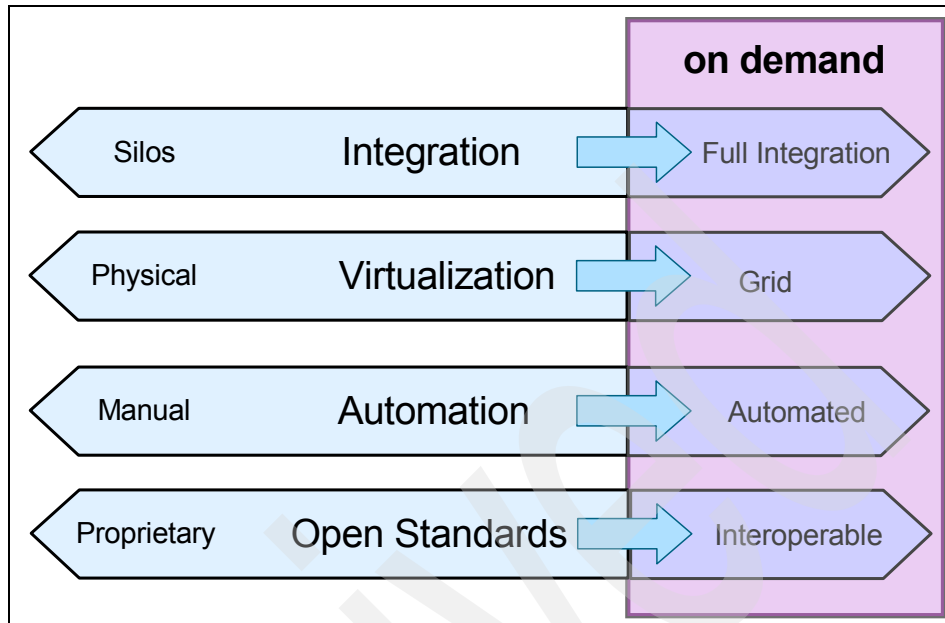


Figure 3-3 Four key technology attributes of On Demand Business

Integration

The fundamental component of on demand infrastructure is integration.

We define *On Demand Business* in the following way: “An On Demand Business is an enterprise whose business processes, integrated end-to-end with key partners, suppliers, and customers, can rapidly respond to any customer demand, market opportunity, or external threat.”

http://www-1.ibm.com/partnerworld/pwhome.nsf/weblook/pub_strategies_ebod.html

Integration can occur at various levels with the following elements:

► People

To function at an on demand operating level, human-to-human and human-to-process interaction requires integration throughout the various levels and is not limited to end users. In on demand processes, trading partners, customers, and employees are all valuable resources in the chain. For example, integration can occur for developers through open tooling paradigms based on open standards, for trading partners by the creation of horizontal processes, and employees through collaboration.

► Process

Recurring elements such as security, service levels, monitoring, and so on can be shared across applications to provide horizontal services to decouple these reusable application components. The use of SOA and Web services, for example, to implement these processes, including the emerging Business Process Execution Language for Web Services (WS-BPEL), will facilitate more rapid changes in these processes, enabling the business to respond with agility to changing market conditions.

► Applications

Organizations have invested enormous resources and capital into custom-designed and off-the shelf applications. Applications sit on disparate systems in an enterprise or across many enterprises. The application integration goal is to leverage, rather than replace, these assets by providing ways of connecting, routing, and transforming the data that is stored or shared among them.

► Systems

Systems manage, process, and deliver data to the people and applications in the solution environment. An on demand operating environment requires the system to be transparent to the elements that interact with it.

► Data

Data is the primary business element of a system. The data is the source of the information and can more easily be shared through the adoption of standards specifications. These specifications allow the transparent sharing of data across applications and organizations.

Virtualization

Various areas of technology in our lives exploit virtualization concepts, including cell phones, PDAs, wireless connectivity, printers, and so forth. Aspects of virtualization draw on widely adopted architectural concepts, including object-oriented design and development, Web services, and XML.

There is a spectrum of virtualization that begins at independent stand-alone systems on one side (a large mainframe system, perhaps) and grid computing on the other. In the middle are varying degrees of client-server implementations.

A grid paradigm, an absolute example of on-demand virtualization, is a collection of distributed computing resources that are available over a local or wide area network and that appear to an end user or application as one large virtual computing system.

The Internet, the most widely recognized example of virtualization, provides a virtual network that supplies access to content and applications.

The vision is to create virtual, dynamic organizations through secure, coordinated resource sharing among individuals, institutions, and resources. *Grid computing* is an approach to distributed computing that spans locations, organizations, machine architectures, and software boundaries.

Figure 3-2 on page 29 depicts virtualization as a set of virtualized resource pools based on:

- ▶ Servers

This category includes partitioning, hypervisors, VM OS, emulators, I/O virtualization, virtual Ethernet, and so forth.

- ▶ Storage

Here, the focus is on the addition of intelligence and value in the network.

- ▶ Distributed systems

This category includes Web services, scheduling, provisioning, workload management, billing and metering, as well as transaction management.

The goal of grid computing, and thus on demand virtualization, is to provide unlimited power, collaboration, and information access to everyone connected to a grid.

Note: Open Grid Services Architecture (OGSA) is an important starting point for grid enablement. For more information about OGSA, refer to the article at:

<http://www-106.ibm.com/developerworks/grid/library/gr-visual/>

Automation

Autonomic computing addresses an organization's need to limit the amount of time and cost that occurs as a result of:

- ▶ Over provisioning
- ▶ High cost of new applications and highly skilled labor
- ▶ IT budget spent on maintenance, not problem resolution
- ▶ Complexities in operating heterogeneous systems
- ▶ Amount of time spent on disparate technology platforms, even within one organization

So how can organizations begin to address these common concerns using an on demand Operating Environment? This is where autonomic computing enters the picture. Autonomic computing can be summarized using the four key components:

- Self-healing

Self-healing is a system's ability to keep functioning, even during component or software malfunction. In order to achieve this, the system must detect, prevent, and recover from disruptions with minimal or no human intervention. This requirement is directly proportional to increased business dependence on technical infrastructures. The need for self-healing is directly proportional to the organization's availability requirement.

- Self-configuring

Self-configuring means the system has the ability to adapt to changing environments dynamically, adding and removing components to and from the systems, and changing the environment to adapt to variable workloads.

- Self-optimization

A *self-optimization* configuration maximizes operational efficiency, including resource tuning and workload management. Self-optimization alleviates the constant drain on resources to perform routine tasks. The goal is to tune systems to respond to workload changes. Systems monitor and self-tune continuously, adapting and learning from the environment around them.

- Self-protecting

Security is one of the inhibitors of the adoption of SOAs as organizations prepare themselves to share data externally. Self-protection requires the system to provide safe alternatives to secure information and data.

Self-protecting automation works by anticipating, detecting, identifying, and protecting systems from external or internal threats.

Open standards

While described as a separate attribute, open standards affect the on demand Operating Environment across the previously defined levels, including automation, integration, and virtualization. Each of these elements leverage open standards specifications in order to achieve their objectives. Open standards are the key element of flexibility and interoperability across heterogeneous systems.

The global adoption of a standard specification enables disparate systems to interact with each other. The underlying platforms can be completely different and independent, but open standards enable processes to be built despite, or because of, these differences.

Open standards provide the On Demand Business Operating Environment with a standard, open mechanism with which to invoke system services.

3.2.3 Key requirements for integration flexibility

To enable the business integration that is required by an on demand business while maintaining the maximum flexibility of implementation, we need to meet the requirements shown in Figure 3-4.

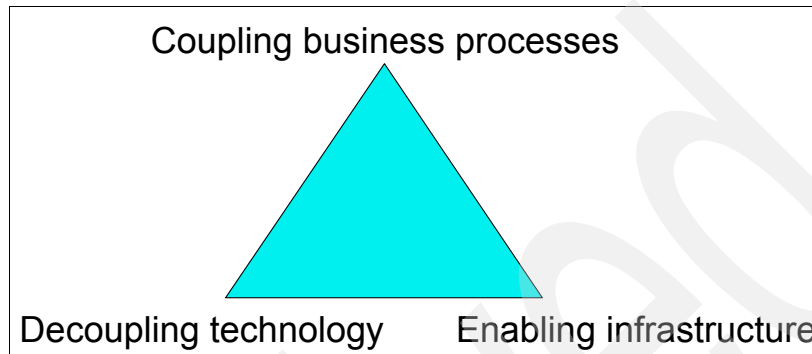


Figure 3-4 On demand key requirements for integration flexibility

Each requirement poses several questions:

- ▶ Coupling business processes
 - How do we model the business?
 - How do we refactor the business into processes, components, and services that can interact dynamically and change agilely?
- ▶ Decoupling technology
 - How do we support business behavior with systems that can interact without joining them too tightly?
 - How can we change and evolve the systems and interactions on the timescales required by the business?
- ▶ Enabling infrastructure
 - How do we build the technical infrastructure to support, execute, manage, and measure these interactions, services, components, and processes?

3.2.4 The on demand Operating Environment

Figure 3-5 on page 36 shows the on demand Operating Environment reference architecture.

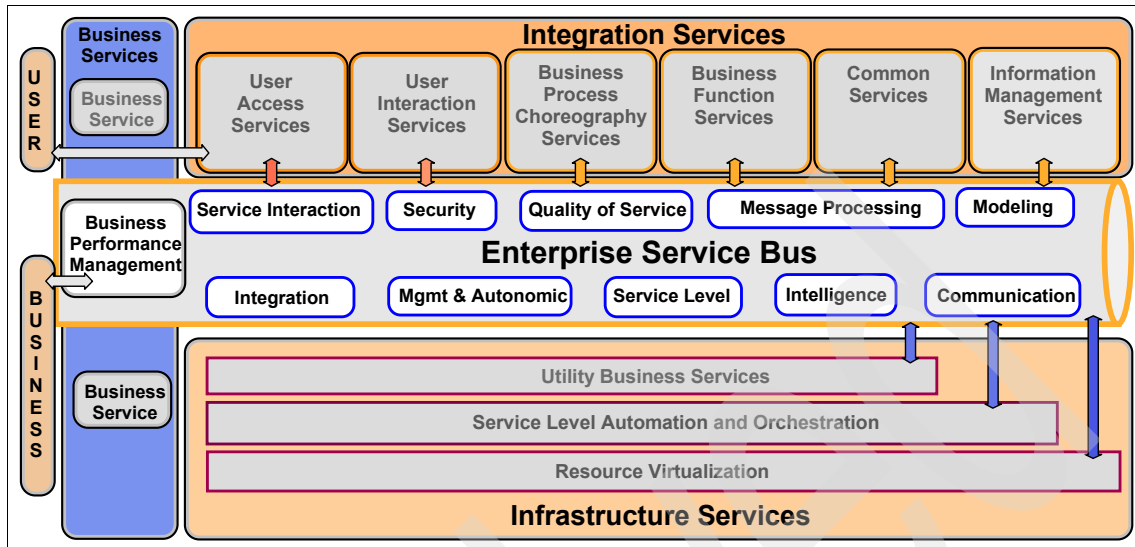


Figure 3-5 On demand Operating Environment architecture

The three core components of the on demand Operating Environment (Integration Services, Enterprise Service Bus, and Infrastructure Services) work together to provide the capability to meet defined business objectives.

Business services leverage the Application and Infrastructure Services, which are mediated by the Enterprise Service Bus, to provide real business processes to end users including customers, employees, and business partners.

Business Service management incorporates the policies and goals of the organization, such as service levels, metrics, and other measurable business guidelines.

Enterprise Service Bus

The Enterprise Service Bus is emerging as a service-oriented infrastructure component that makes large-scale implementation of the SOA principles manageable in a heterogeneous world.

On demand applications are business services built from services that provide a set of capabilities that are worth advertising for use by other services. Typically, a business service relies on many other services in its implementation. Services interact through the Enterprise Service Bus, which facilitates mediated interactions between service endpoints. The Enterprise Service Bus supports event-based interactions as well as message exchange for service request handling. One innovation of the Enterprise Service Bus is a common model for

messages and events. All messages can become events if deploying the service binds the message to a topic in the event space.

For both events and messages, you can use mediations to facilitate interactions for example, to find services that provide capabilities that a consumer wants, or to resolve interface mismatches between consumers and providers with compatible capabilities. In this context, we use the term *service* in a very general sense. Additionally, it is worth noting that from the perspective of the bus, all application components can be specified through WS-* standards because it requires a normalized representation for efficient mediated, capability-based matchmaking. However, this does not imply that they all communicate with SOAP or WS-* protocol standards. The Enterprise Service Bus supports a broad spectrum of ways to get on and off the bus, including on ramps for existing applications or business connections that enable external partners in B2B interaction scenarios to participate in service interaction.

Although they all look the same from the perspective of the Enterprise Service Bus, services implement different facets of an overall on demand application, including:

- ▶ Realize interactions with people involved in the underlying business process.
- ▶ Provide adapters to existing applications that have to be integrated.
- ▶ Choreograph the interaction of several services to achieve a business goal.
- ▶ Manage resources that are needed to perform required business functions.
- ▶ Watch for potential problems in the execution of the process, ready to take action to fix them if they occur.

Therefore, in addition to providing the basic infrastructure for service interactions, the on demand Operating Environment identifies a set of common patterns for construction of on demand applications and provides specific capabilities to support realization of distinct service categories that play particular roles in those patterns. The two distinct service categories are integration services and infrastructure services.

Integration services

The programming model for on demand business services is based on application development using component (service) assembly. The services in the integration category are used by on demand application builders to create new business services; they include services that facilitate integration as well as services that provide business functions to be integrated:

- ▶ *User access services* handle adaptation from three orthogonal perspectives:
 - Endpoint form factor such as display size, memory, and processor limitations ranging from desktop down to pervasive devices

- Modes of interaction including conventional display and keyboard interactions, as well as speech-based interactions and combinations (multimodality)
- Connection types such as peer-to-peer or client/server across a range of connection reliability, including fully disconnected operations
- ▶ *User interaction services* handle the direct interactions of people involved in the business process; for example, processing work items that are spawned by choreography or collaborative process elements.
- ▶ *Business process choreography services* support services that use process flow or rule technology. *Process flows*, for example, describe the interaction of other services, meaning nearly any kind of integration service such as process flow services, to perform the tasks required to realize the functions offered by the new aggregated business service.
- ▶ *Business function services* provide the *atomic* business functions (those that are not composed from other services) that are required by the overall business service. Business function services include adapters for packaged or existing custom applications as well as brand new application components created to meet a functional need that is not already covered by existing applications.
- ▶ *Common services* implement useful features or helper functions that are used by other business services. Examples of common services include implementing personalization of user access and user interaction services, or for reporting status and progress of business services.
- ▶ *Information management services* help to integrate information hosted in a variety of data sources such as databases or existing applications, to access (query, update, and search) that information, to analyze information from those sources in business intelligence scenarios, or taking care of metadata about information and services used and provided by the business services living in the on demand Operating Environment.
- ▶ *Application services* provide containers for integration services, simplifying their participation in interactions with other integration services and on demand Operating Environment infrastructure services. On demand integration service developers focus on realizing the business logic that they care about, assembling integration services that provide required business function and declaring expected quality of service.

Programmers and administrators annotate their applications and services with policy declarations that specify quality of service. The application container and the Enterprise Service Bus automate the interactions with infrastructure services to achieve the expressed policies. An application container also provides generic facilities such as taking care of security or transaction management requirements for the services that it hosts, as well

as kind-specific facilities such as generating events reporting status and progress of business process choreography services.

Infrastructure services

The services in the infrastructure category provide and manage the infrastructure into which business services and their constituents are deployed. These include:

- ▶ *Utility business services* support functions such as billing, metering, rating, peering, and settlement. These services are commonly used, for example, when hosting on demand business services or their components.
- ▶ *Service level automation and orchestration services* implement the quality of service policy declarations for business services. These services implement 'autonomic managers that monitor the execution of services (more precisely, services instrumented to be managed elements) in the on demand Operating Environment, according to the policy declarations they receive. Service level automation and orchestration services then analyze that behavior, and if the analysis indicates a problem, plan a meaningful reaction to that problem then execute that plan. This closed feedback loop is called an *M-A-P-E* (Monitor, Analyze, Plan, Execute) *loop*. Several specializations of such services focus on, for example: managing, availability, configuration or workload for the managed elements, provisioning resources, performing problem management, handling end-to-end security for on demand Operating Environment services, or managing data placement.
- ▶ *Resource virtualization services* provide the instrumentation of server, storage, network, and other resources, including structured (relational) and unstructured information content that is held in a variety of data sources, to enable management and virtualization of those resources under the control of on demand Operating Environment resource managers. Virtualization services include mapping requirements of business services and their components to available resources based on quality of service declarations of the service and knowledge about the current use of available resources.

Besides the fact that they implement very different capabilities that support a variety of on demand Operating Environment patterns, the main difference between integration and infrastructure services is which user roles build and use them. Infrastructure elements are built by middleware providers and ISVs. Integration elements are built by on demand infrastructure and application builders.

One of the most important insights of the on demand Operating Environment is that a common pattern supports both application services and infrastructure services. For example:

- ▶ Adapters enable integration of existing infrastructure components into the Enterprise Service Bus.

- ▶ Service choreography is often used for scripting of M-A-P-E execution plans.
- ▶ The Enterprise Service Bus provides the infrastructure for exchange of events between managed elements and autonomic managers.
- ▶ End users interact with infrastructure services through the portal user interaction services.

3.3 Approaches for delivering the Extended Enterprise

While IT executives have been facing the challenge of cutting costs and maximizing the utilization of existing technology, at the same time they have to continuously strive to serve customers better, be more competitive, and be more responsive to the business's strategic priorities.

There are two underlying themes behind all of these pressures: *heterogeneity* and *change*. Most enterprises today contain a range of different systems, applications, and architectures of different ages and technologies. Integrating products from multiple vendors and across different platforms is almost always challenging. But we also cannot afford to take a single-vendor approach to IT, because application suites and the supporting infrastructure are so inflexible. More importantly, in an extended enterprise environment the control of product and technology selection is dramatically reduced.

Change is the second theme underlying the questions that today's IT executives face. Globalization and e-business are accelerating the pace of change. Globalization leads to fierce competition, which leads to shortening product cycles, as companies look to gain advantage over their competition. Customer needs and requirements change more quickly driven by competitive offerings and the wealth of product information available over the Internet. In response the cycle of competitive improvements in products and services further accelerates.

Improvements in technology continue to accelerate, feeding the increased pace of changing customer requirements. Business must rapidly adapt to survive, not just to succeed, in today's dynamic competitive environment. As a result, the IT infrastructure must enable businesses' ability to adapt.

As a result, business organizations are evolving from the vertical, isolated business divisions of the 1980's and earlier, to the horizontal business-process-focused structures of the 1980's and 1990's, towards the new ecosystem business paradigm. Business services now need to be componentized and distributed. There is a focus on the extended supply chain, enabling customer and partner access to business services. Figure 3-6 on page 41 shows this evolution of business.

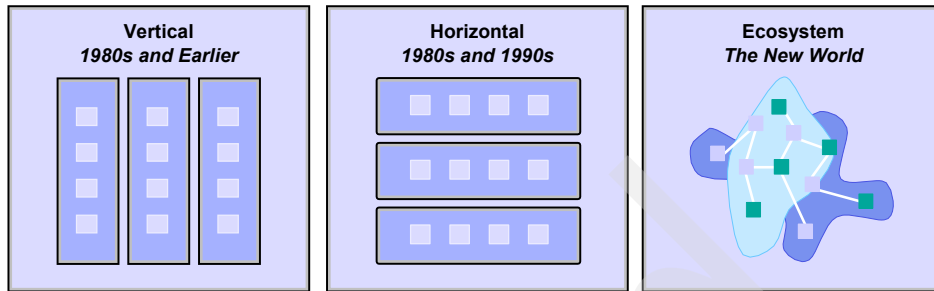


Figure 3-6 The evolution of business

3.3.1 SOA and Web services

The implementation of service-oriented architecture (SOA) using Web services technologies is the current state of the art in systems integration. Both topics are covered extensively in industry literature, but there is some variation in their description, so an introduction is provided here to place the remaining content of this redbook in context.

For some time, the vision of much of the IT industry has been to achieve rapid, flexible integration of IT systems across all elements of the business cycle. The drivers behind this vision include:

- ▶ Increasing the speed with which businesses can implement new products and processes, or change existing ones
- ▶ Reducing implementation and ownership costs
- ▶ Enabling flexible pricing models by outsourcing elements of the business or moving from fixed to variable pricing, based on transaction volumes
- ▶ Simplifying the integration work that is required by mergers and acquisitions
- ▶ Achieving better IT utilization and return on investment
- ▶ Simplifying the enterprise architecture and computing model

Really achieving these goals affects the entire scope of a business's processes and IT systems, as depicted in Figure 3-7 on page 42. Such pictures should be familiar to anyone with an interest in enterprise application integration, business-to-business, or portal technologies. However, perhaps it is fair to say that until recently the industry has lacked a consistent and comprehensive approach to technology and architecture on this scale. Although several systems that cover some elements of this scope have been implemented, there has not been a single, broadly accepted approach.

The combination of SOA, an approach that draws together proven techniques from several preceding architecture and design styles, with new open standards and integration technologies has the potential to provide such a consistent approach.

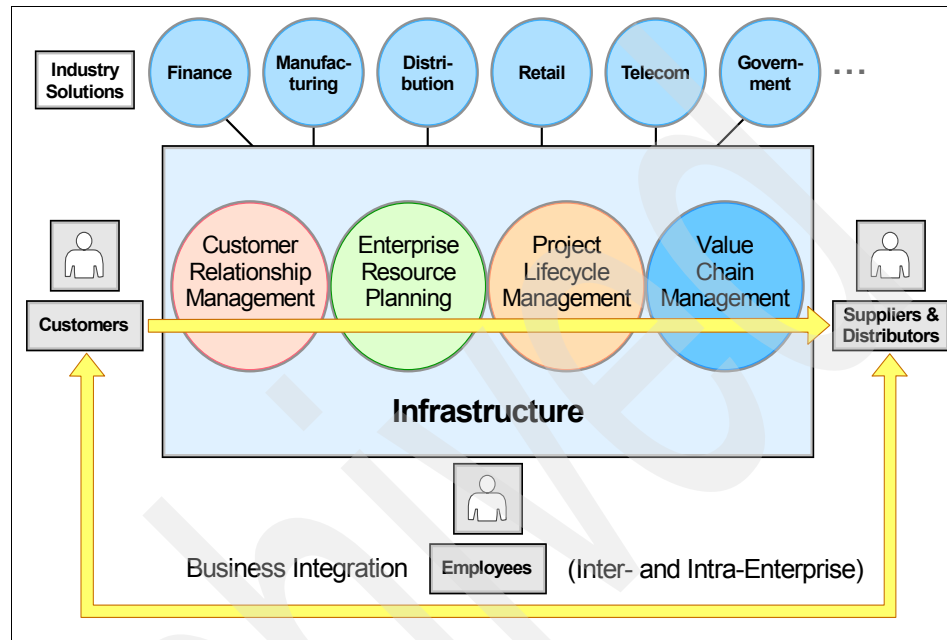


Figure 3-7 Integration across the value chain

To describe why both Web services and SOA are necessary to achieve these goals, it is informative to consider the specific technical issues that arise in any attempt to integrate flexible systems on the scale that we are discussing:

- ▶ Business systems are implemented using a multitude of technologies and platforms.
- ▶ Business processes are implemented by a mixture of people practices, application code, and interactions between people and systems or systems and systems.
- ▶ Changes to one system tend to imply ripples of change at many levels to many other systems.
- ▶ No single, fully functional integration solution will talk to all of the systems in the enterprise.
- ▶ Deployment of any single, proprietary integration solution across the enterprise is complex, costly, and time-consuming.

- ▶ All issues that are involved in internal integration are encountered again when integrating with partners and their systems.
- ▶ There is no single data, business, or process model across or beyond the enterprise.
- ▶ Not all integration technologies work as well across a wide area network or the Internet as they do across a local area network, perhaps due to:
 - The use of exotic protocols
 - Constraints imposed by security technologies, including firewalls
 - Constraints imposed by network bandwidth

As we discuss Web services and SOA in this section, we see how those issues are addressed. More to the point, it is only the appropriate combination of both Web services technology and the SOA approach that enables us to address them all on the broadest scales. In that vein, we should take stock briefly of what both Web services and SOA have achieved separately to date:

- ▶ Most significant SOAs are proprietary or customized implementations based on reliable messaging and Enterprise Application Integration middleware (for example WebSphere Business Integration) and do not use Web services technologies. They have, however, demonstrated the benefits of SOA, usually within a single enterprise.
- ▶ Most existing Web services implementations consist of point-to-point integrations that address a limited set of business functions between a defined set of cooperating partners, and they use HTTP, an unreliable transport, as the communication mechanism. They have, however, demonstrated the efficacy of the Web services technologies in integrating heterogeneous systems both within and among organizations.

There are several more ambitious efforts underway using both Web services and SOA. However, many of these efforts involve building significantly customized infrastructure functions in addition to using off-the-shelf products and technologies.

It is also worth noting that because we are dealing with enterprise integration and implementation in this redbook, we have to be aware of all of the usual requirements for enterprise class systems to, for example:

- ▶ Leverage existing assets.
- ▶ Support customized systems and commercial off-the-shelf (COTS) packages.
- ▶ Support incremental adoption and implementation.
- ▶ Provide for loose coupling between systems.
- ▶ Incorporate synchronous and asynchronous communication and transactions.

- ▶ Maintain security.
- ▶ Support multiple programming languages and platforms.
- ▶ Handle high volumes and transaction rates that are the hallmark of peak behavior.
- ▶ Support global deployment, including multiple languages, currency independence, and 24/7 operations.

Finally, we should be clear that there is no magic for achieving every aspect of the Extended Enterprise. We contend that all of this is possible with Web services and SOA, but you cannot just deploy a Web services SOA and start it. Instead, we describe an incremental approach to designing and deploying what can become an enterprise-class SOA using Web services over an appropriate timescale.

What is SOA?

SOA is an integration architecture approach based on the concept of a service. The business and infrastructure functions that are required to build distributed systems are provided as services that, collectively or individually, deliver application functionality to either end-user applications or other services.

SOA specifies that, within any given architecture, there should be a consistent mechanism for services to communicate. That mechanism should be loosely coupled and support the use of explicit interfaces.

SOA brings the benefits of loose coupling and encapsulation to integration at an enterprise level. It applies successful concepts proved by Object Oriented development, Component Based Design, and Enterprise Application Integration technology to an architectural approach for IT system integration.

Services are the building blocks of SOA, providing function out of which distributed systems can be built. Services can be invoked independently by either external or internal service consumers to process simple functions, or can be chained together to form more complex functionality and, therefore to devise new functionality quickly.

By adopting an SOA approach and implementing it using supporting technologies, companies can build flexible systems that implement changing business processes quickly, and can make extensive use of reusable components. This concept is shown in Figure 3-8 on page 45.

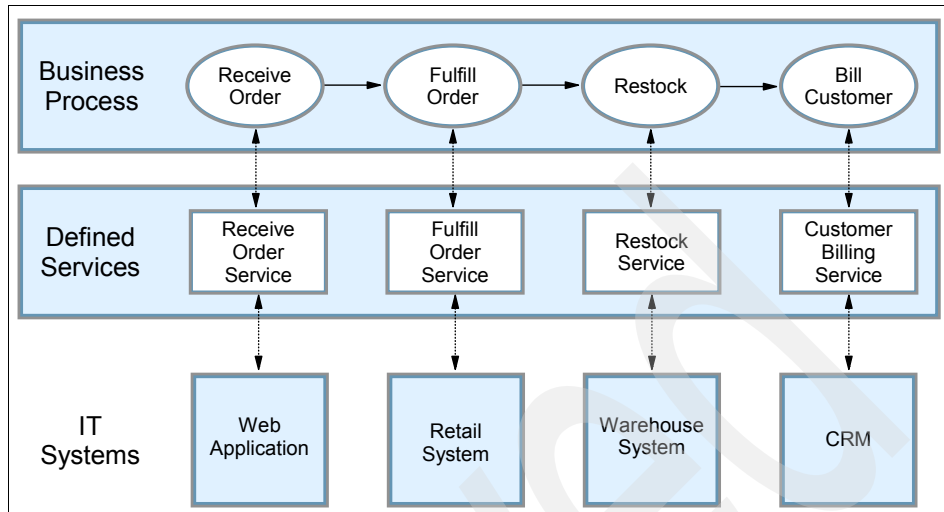


Figure 3-8 A service oriented approach to building systems

Figure 3-8 illustrates a company that wants to implement a new business process to support customers placing orders from a Web site. The company already has existing retail, warehouse, and billing systems. It would like to build the new process by reusing the functionality provided by those systems, rather than having to write new applications or new interfaces to the existing systems.

If the company has already adopted an SOA approach, it has defined the interfaces to its existing systems in terms of the functions, or services, that they can offer to support building business processes. This makes building the new Web front end to the system very simple. To complete the new business process, all they have to do is to develop an application that makes calls to the services.

The SOA approach means companies are able to build horizontal business processes that integrate systems, people, and processes from across the enterprise, quickly and easily responding to changing business needs.

As shown in Figure 3-8, existing systems can be used to implement new business processes that extend the use of the system beyond the processes they were originally written to support. This means the company is able to maximize previous IT investment by reusing existing IT systems without having to invest extensively to build new interfaces to the systems.

What is a service?

Having outlined SOA as being an architectural approach to defining integration architectures based on services, it is important to define what is meant by a

service in this context to fully describe SOA and understand what can be achieved by using it.

A *service* can be defined as any discrete function that can be offered to an external consumer. This can be an individual business function, or a collection of functions that together form a process.

There are many additional aspects to a service that must also be considered in the definition of a service within an SOA. The most commonly agreed upon aspects are:

- ▶ Services encapsulate reusable business functions.
- ▶ Services are defined by explicit, implementation-independent interfaces.
- ▶ Services are invoked through communication protocols that stress location transparency and interoperability.

In this redbook we define an SOA as being defined by these common aspects.

Reusable function

Any business function can be a service. SOA often focusses on business functions. However as you read further in this redbook, you can see that many technical functions can also be exposed as services. When defining *function* there are several levels of granularity that can be considered. Many descriptions of SOA refer to the use of large-grained services, however, some powerful counter-examples of successful, reusable, fine-grained services exist. For example, `getBalance` is a very useful service, but hardly large-grained.

More realistically, there are many useful levels of service granularity in most SOAs. For example, all of the following list items are services, however they have different granularity. Some degree of choreography or aggregation is required between each granularity level for them to be integrated in an SOA:

- ▶ Technical Function Services, for example `auditEvent`, `checkUserPassword`, `checkUserAuthorization`
- ▶ Business Function Services, for example `calculateDollarValueFromYen`, `getStockPrice`
- ▶ Business Transaction Services, for example `checkOrderAvailability`, `createBillingRecord`
- ▶ Business Process Services, for example `openAccount`, `createStockOrder`, `reconcileAccount`, `renewPolicy`

A service can be any business function. In an SOA however, it is preferable that the function is genuinely reusable. The goal of a service in SOA is that it can be used by one or more systems that participate in the architecture. For example,

while the reuse of a Java logging API could be described as *design time* (when a decision is made to reuse an available package and bind it into application code), the intention of SOA is to achieve the reuse of services at:

- ▶ Runtime

Each service is deployed in only one place, and remotely invoked by anything that must use it. The advantage of this approach is that changes to the service (for example, to the calculation algorithm or the reference data it depends on) need only be applied in a single place.

- ▶ Deployment time

Each service is built once but redeployed locally to each system or set of systems that must use it. The advantage of this approach is increased flexibility to achieve performance targets or to customize the service (perhaps according to geography).

The service definition should encapsulate the function well enough to make reuse possible. The encapsulation of functions as services and their definition using interfaces enables the substitution of one service implementation for another. For example, the same service might be provided by multiple providers (such as a car insurance quote service, which might be provided by multiple insurance companies), and individual service consumers might be routed to individual service providers through some intermediary agent.

Granularity in SOA

The concept of *granularity* is used to mean several things in SOA, each of which is actually quite separate. We will not be greatly concerned with these issues but it is worth identifying them:

- ▶ Level of abstraction of services

Is the service a high-level business process, a lower-level business subprocess or activity, or a very low-level technical function?

- ▶ Granularity of service operations

How many operations are in the service: one, a few, or many? What determines which operations are collected together in a service?

- ▶ Granularity of service parameters

How are the input and output data of service operations expressed? SOA prefers a small number of large, structured parameters rather than a small number of primitive types.

Explicit implementation independent interfaces

Using explicit interfaces to define and encapsulate service function is particularly important to making services genuinely reusable. The interface should

encapsulate only those aspects of process and behavior that are used in the interaction between the service consumer and the service provider. An explicit interface definition, or contract, is used to bind a service consumer and a service provider. It should specify only the mutual behavior required for the interaction, and nothing about the implementation by the consumer or the provider.

By explicitly defining the interaction in this way, those aspects of either system (for example the platform they are based on) that are not part of the interaction are free to change without affecting the other system. This allows either system to change implementation or identity freely.

The use of explicit interfaces to define and encapsulate services function is illustrated in Figure 3-9

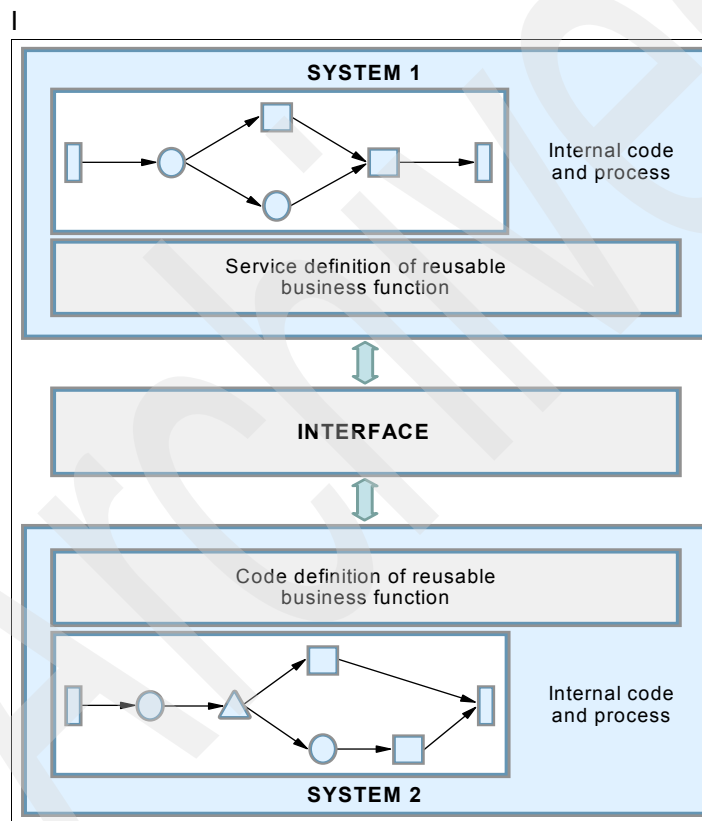


Figure 3-9 Service implementation in SOA

Communication protocols that stress location transparency

Companies have a variety of choices when deciding how to connect applications. HTTP, HTTP/S, JMS, CORBA and SMTP are all examples of protocols that can

be used to connect applications. There are also many middleware products, for example WebSphere MQ, that are used to provide application-to-application connectivity. Even within a single company, a variety of techniques, products, and protocols can be used to address different integration requirements. This could create problems when trying to extend the integration to connect to applications that do not use the same protocols.

SOA does not specify that any one protocol should be used to provide access to a service. A key principle of SOA is that a service is not defined by its communication protocol, but instead the service is defined as independent of protocol. The benefit of being protocol-independent is that it allows different protocols to access the same service.

Ideally, a service should only be defined once through a service interface and have many implementations with different access protocols. This helps to increase the reusability of any service definition.

Also, services should be invoked, published, and discovered in a way that is abstracted away from the actual implementation using a single, standards-based interface.

All of this is to say that there is a complimentary nature between SOA and Web services.

On Demand Business and SOA

SOA plays a crucial role for companies trying to implement the IBM vision of On Demand Business. The vision of IBM On Demand Business (formerly called On Demand Business) is to enable customers to succeed in an environment with an unprecedented rate of change.

In an on demand world companies need to be able to respond to any customer requirement, opportunity, or threat quickly and easily. To succeed in this environment a company must be able to implement new processes quickly while leveraging existing investment. From a business perspective, On Demand Business is about providing a way for companies to realign their business and technology environment to match the need for reusable business functionality. For a fuller discussion on IBM On Demand Business and how it relates to SOA refer to Chapter 10 of the IBM Redbook *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303.

SOA can be seen as an architectural enabler for On Demand Business. The basic relationship between SOA and On Demand Business is that SOA touches on the four key elements of On Demand Business:

- ▶ Open standards
 - SOA provides a standard method of invoking services (business logic and functionality) for disparate organizations to share across network boundaries.
- ▶ Integration
 - Interfaces are provided to wrap service endpoints to provide a system-independent architecture to promote cross-industry communication and so integrate end-to-end both inside and outside of the enterprise.
 - SOAs can provide dynamic service discovery and binding, which means that service integration can occur on demand.
 - SOA provides an approach to integrate heterogeneous technologies inside an enterprise.
- ▶ Virtualization
 - A key principle of SOA is that services should be invoked by service consumers that are oblivious to service implementation details, including location, platform, and even the identity of the service provider, if appropriate to the business scenario.
- ▶ Automation
 - Technologies such as grid technologies that apply SOA principles to implementing infrastructure services that provide an evolutionary approach to increased automation.

What are Web services?

Web services are a recent set of technology specifications that leverage existing proven open standards such as XML, URL, and HTTP to provide a new system-to-system communication standard. Based on this communication model, additional higher-level Web services standards have also been defined to address transactions, security, business processes, and so forth: the higher-order functions that are required to get systems, applications, and processes (rather than objects and components) talking to each other.

Web services learn from the way the Web revolutionized how people talk to systems: new customers, new business models, extensions of opportunity, new transparency and improved collaboration between employees and employers, and, in some cases, reductions in infrastructure costs and complexity. The key to these successes is a universal server-to-client model consistent with a highly distributed environment, based on simple open standards and industry support.

Web services promises to integrate one business directly with another so that the process does not have to wait for people to provide the glue, get your own

business talking to itself, or your partners to provide integrated IT systems, and again the potential for dramatic reductions in infrastructure costs and complexity. Once again, the key is a universal program-to-program communication model based on simple open standards and industry support.

Figure 3-10 shows the basic interaction model supported by Web services.

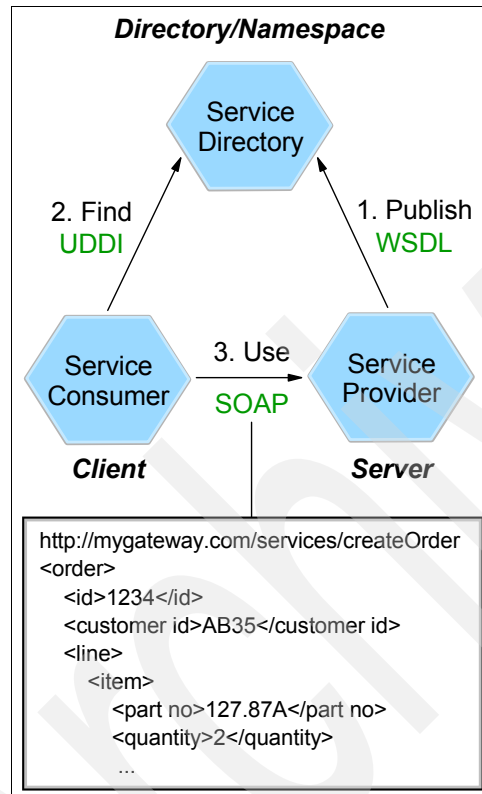


Figure 3-10 Basic Web services

Basic Web services define interactions among Service Consumers, Service Providers, and Service Directories as follows:

Service Consumers find Web services in a UDDI Service Directory. They retrieve WSDL descriptions of Web services offered by Service Providers, who previously published those descriptions to the Service Directory. After the WSDL has been retrieved, the Service Consumer binds to the Service Provider by invoking the service through SOAP.

The basic Web services are often described in terms of SOAP, WSDL, and UDDI, each of which we define and discuss. However, it should be noted that

each of these standards can be used in isolation, and there are many successful implementations of SOAP alone, or SOAP and WSDL, in particular.

SOAP

SOAP is an XML messaging protocol that is independent of any specific transport protocol. SOAP defines a framework within which messages contain headers, which are used to control the behavior of SOAP-enabled middleware, and a message body. As SOAP is an XML format, and as XML is text-based, SOAP is supportable in the vast majority of existing and new technical environments and can be transported over a vast variety of protocols.

In practice, SOAP is most often communicated over HTTP, although this is likely to evolve rapidly because HTTP is an unreliable protocol. (For instance, it is already possible to send SOAP messages through JMS implementations such as WebSphere MQ.) Basic SOAP also makes no reference to characteristics of interactions such as security and transactionality. However, as SOAP headers provide an extensible model, these aspects are being added gradually to the Web services specifications as extensibility elements, as we describe further in the next section. The use of SOAP over specific protocols, such as HTTP, is usually written as SOAP/HTTP, SOAP/JMS, and so forth.

The SOAP V1.2 specification is available from the World Wide Web Consortium, and deliberately does not define a meaning for SOAP as an acronym. (SOAP is sometimes referred to as Service Oriented Architecture Protocol, or by its definition in the more widely supported SOAP V1.1 specification, Simple Object Access Protocol.)

WSDL: Web Services Description Language

WSDL is an XML-based interface definition language that separates function from implementation, and enables design by contract as recommended by SOA. WSDL descriptions contain a PortType (the functional and data description of the operations that are available in a Web service), a Binding (providing instructions for interacting with the Web service through specific protocols, such as SOAP/HTTP), and a Port (providing a specific address through which a Web service can be invoked using a specific protocol binding).

The value of WSDL is that it enables development tooling and middleware for any platform and language to understand service operations and invocation mechanisms. For example, given the WSDL interface to a service that is implemented in Java, running in a WebSphere environment, and offering invocation through HTTP, a developer working in the Microsoft .NET platform can import the WSDL and easily generate application code to invoke the service.

As with SOAP headers, the WSDL specification is extensible and provides for additional aspects of service interactions to be specified, such as security and transactionality.

UDDI: Universal Description, Discovery, Integration

UDDI servers act as a directory of available services and service providers. UDDI can be searched a couple of ways:

- ▶ You can use SOAP to query UDDI to find the locations of WSDL service definitions.
- ▶ You can perform a search through a user interface during design or development.

The original UDDI classification was based on a U.S. government taxonomy of businesses, and recent versions of the UDDI specification have added support for custom taxonomies.

A public UDDI directory is provided by IBM, Microsoft, and SAP, each of whom runs a mirror of the same directory of public services. However, there are many patterns of use that involve private registries; see Steve Graham's articles:

- ▶ *The role of private UDDI nodes in Web services, Part 1: Six species of UDDI*
<http://www.ibm.com/developerworks/webservices/library/ws-rpu1.html>
- ▶ *The role of private UDDI nodes, Part 2: Private nodes and operator nodes*
<http://www.ibm.com/developerworks/webservices/library/ws-rpu2.html>

SOAP/HTTP uses existing namespaces and infrastructure

One of the important potential benefits of Web services is to reduce the reliance of integration on specific integration technologies that require heavyweight deployment where such deployment would be problematic or impossible. A particular example of this situation is business-to-business interactions, particularly as they become more widespread and dynamic.

The specific use of Web services with HTTP as a communication protocol has some extraordinary benefits in this area. Because SOAP/HTTP uses HTTP as a communication protocol and URL as the addressing format, the entire global network of distributed, resilient routing and communications infrastructure that the Internet provides can be used. Allowances must be made for the unreliable nature of HTTP, but the advantages of a service communication protocol that is already deployed and globally pervasive should not be underestimated.

Web services standards

There are many successful implementations of the basic Web Services standards, particularly SOAP and WSDL, but as we previously described, many

aspects of service interaction and integration are not directly supported by those basic standards. These integration aspects include security, transactionality, delivery assurance, and process modeling.

Web services standards are evolving and maturing to address these aspects of interaction and integration, increasing their value to SOA. In this section we cover some of the recent and emerging Web services standards that support more sophisticated aspects of service interactions and SOA.

Production-level product support for some of these standards is not yet available, but early implementations exist. The IBM Emerging Technologies Toolkit (ETTK), for example, provides an implementation of WS-ReliableMessaging. The toolkit can be downloaded from:

<http://www.alphaworks.ibm.com/tech/ettk>

Web services security

In theory, Web services can leverage any security model that is appropriate to the underlying communication technologies. (SOAP/HTTP can utilize basic HTTP authentication or SSL authentication and encryption.) However, such simple point-to-point models are insufficient for the widespread integration needs of SOA. For example:

- ▶ Communication security does not recognize the difference between SOAP message headers and the SOAP message body.
- ▶ Credentials can be technology-specific for the communication mechanism, but inappropriate for communication mechanisms that are used farther down the interaction chain.
- ▶ Combining many interactions in a secure overall chain involves trust models between the participants in the chain. Such models are often customized or proprietary and are not consistent. Flexibly changing participants in the chain imply a technology barrier to participation.

In 2002, IBM and Microsoft proposed an architecture and roadmap for Web services security (WS-Security). This set out a framework consisting of several Web services specifications, including WS-Security, WS-Trust, WS-Privacy, and WS-Policy. It also accommodated existing security technologies such as Kerberos, XML Digital Signatures, and XML Encryption.

Support for the basic WS-Security standards is available in existing products and can be used to implement secure Web Services solutions. Understanding the security requirements of specific SOA situations and selecting appropriate technologies, include those compliant with the WS-Security standards, is a key decision in SOA implementation.

Further information about security

- ▶ Security in a Web Services World: a Proposed Architecture and Roadmap
<http://www.ibm.com/developerworks/library/ws-secmap/>
- ▶ Web Services Security: Moving up the stack
<http://www.ibm.com/developerworks/webservices/library/ws-secroad/>

WS-ReliableMessaging and SOAP/JMS

The HTTP protocol that is used widely in SOAP interactions and specified in the WS-I Basic Profile offers relatively poor reliability in contrast to communication protocols that are often associated with valuable business transactions, such as WebSphere MQ. Many SOA scenarios involve interactions that require a level of delivery assurance beyond that provided by HTTP.

The WS-ReliableMessaging specification defines a protocol for reliable communication (including SOAP messages) that use a variety of communication technologies, which might themselves be less reliable. An updated specification was published in March 2004, but production support is not yet available in middleware products.

Until WS-ReliableMessaging is widely available, alternative approaches are possible using implementations of SOAP over more reliable communication infrastructures. For example, SOAP messaging is supported through the JMS API to WebSphere MQ by WebSphere MQ, WebSphere Application Server, and WebSphere Business Integration Server Foundation. However, such approaches tend to be implementations by specific technology vendors so, although they are useful in particular SOA implementations, they do not have all of the potential benefits of a fully open-standard implementation.

Further information about messaging

- ▶ Updated: Web Services Reliable Messaging: A new protocol for reliable delivery between distributed applications
<http://www.ibm.com/developerworks/webservices/library/ws-rm/>
- ▶ Implementation Strategies for WS-ReliableMessaging: How WS-ReliableMessaging can interact with other middleware communication systems
<http://www.ibm.com/developerworks/webservices/library/ws-rmimp/>

Business Process Execution Language for Web Services

Because the encapsulation and exposure of business functions as services in an SOA enables the definition of processes containing those services, Business Process Execution Language for Web Services (WS-BPEL) provides a standard, XML language for expressing business processes consisting of functions that are

defined through WSDL interfaces. WS-BPEL supports both short-lived and long-lived processes. These are processes that must wait at certain points until some event occurs, such as the receipt of an event.

As with WSDL, WS-BPEL has both design time and runtime uses. At design time, development or modeling tools can use, import, or export WS-BPEL to enable business analysts to specify processes. Developers can refine them and bind process steps to specific service implementations. However, runtime choreography and workflow engines can use WS-BPEL to control the execution of process and invoke the services that are required to implement them.

Although WS-BPEL is a relatively new standard, product support such as WebSphere Business Integration Server Foundation V5.1 and WebSphere Process Server V6 is available. This provides additional facilities to compensate failed processes (a proprietary equivalent to the WS-BusinessActivity standard described in the next section, “Web services transactions”) and provide a user workflow interface to enable human actions to fulfill WSDL-defined steps in a WS-BPEL process.

Further information about WS-BPEL

- ▶ WS-BPEL specification
<http://www.ibm.com/developerworks/library/ws-bpel/>
- ▶ Business Process with WS-BPEL, a series of introductory articles and references
<http://www.ibm.com/developerworks/webservices/library/ws-bpelcoll/>
- ▶ WS-BPEL support in WebSphere Business Integration Server Foundation
<http://www.ibm.com/software/integration/wbisf/features/>
- ▶ WS-BPEL support in WebSphere Studio Application Developer Integration Edition
<http://www.ibm.com/software/integration/wsadie/features/>

Web services transactions

Although WS-ReliableMessaging provides a means to assure the delivery of individual communications in a Web services interaction, a means is also required to control the integrity of business transactions in an SOA that consist of one or more Web services invocations or interactions.

Within the framework of the Web services coordination (WS-Coordination) specification, both synchronous (WS-AtomicTransaction) and long-lived (WS-BusinessActivity) transaction models have been defined. These replace the previous WS-Transaction specification.

The WS-AtomicTransaction specifies a model for synchronous, two-phase committal of distributed transactions using Web services protocols. WS-BusinessActivity defines an asynchronous model for compensating failed processes using undo actions to reverse the affects of individual steps of the process. Neither specification has mature product support to date.

Further information transactions

- ▶ WS-AtomicTransaction specification
<http://www.ibm.com/developerworks/library/ws-atomtran/>
- ▶ WS-BusinessActivity specification
<http://www.ibm.com/developerworks/webservices/library/ws-busact/>
- ▶ Transactions in the world of Web Services, part 1 and part 2
<http://www.ibm.com/developerworks/webservices/library/ws-wstx1/>
<http://www.ibm.com/developerworks/webservices/library/ws-wstx2/>
- ▶ WS-Coordination specification
<http://www.ibm.com/developerworks/library/ws-coor/>

Web Services Policy Framework (WS-Policy)

The Web Services Policy Framework is intended to provide a set of languages by which service consumers and providers can express their requirements and capabilities for qualities of service of service interactions, such as security, transactionality, and communication reliability. Eventually a framework of such languages, supported by Enterprise Service Bus middleware, will enable open-standard implementations of negotiated coupling between various aspects of service interactions.

A WS-Policy specification is available, although specific policy languages for quality of service aspects such as security are still required, and product support has yet to emerge.

Further information about WS-Policy

- ▶ WS-Policy framework specification
<http://www.ibm.com/developerworks/library/ws-polfram/>
- ▶ Web Services Policy Framework: New specifications improve WS-Security
<http://www.ibm.com/developerworks/webservices/library/ws-polfram/summary.html>

Web Services Resource Framework (WS-ResourceFramework)

As we write this redbook, WS-ResourceFramework is an architectural proposal rather than a standard. However, WS-ResourceFramework relates to some

aspects of SOA that we have only touched on in discussing Web services, principally the design, rather than the technical implementation of services.

To enable middleware to provide increasingly sophisticated support for such stateful interactions, the Web Services Resource Framework provides a model for associating Web services with stateful resources (for example data such as rows in a database), as opposed to stateful processes (as can be accomplished with WS-BPEL). In essence, WS-ResourceFramework is essentially a model for making Web services middleware and infrastructure aware of stateful identifiers such as transaction IDs.

Further information about WS-ResourceFramework

- ▶ WS-ResourceFramework overview

<http://www.ibm.com/developerworks/webservices/library/ws-resource/ws-wsrfpaper.html>

What are the advantages of SOA and Web services?

Using SOA has the following advantages in achieving loosely coupled, flexible integration of IT systems:

- ▶ Using implementation-independent interfaces to describe services means that heterogeneous systems can be integrated.
- ▶ Describing service interfaces in terms of a common business process and data model minimizes any interdependencies to only what matters to the business.
- ▶ Encapsulating services with standard interfaces enables reuse and flexibility. Each service is defined and implemented in only one place, so changing it is straightforward.

There are benefits for development and maintenance costs too, but flexibility is the primary goal of SOA.

With clearly defined interfaces between all business systems, it is possible to model and change the business process captured by them at a level above individual systems. This means SOA is an enabler for process modelling and automation at the enterprise scale.

Currently, and for some time to come, many of the technologies used to implement SOAs are evolving rather than mature and stable. Therefore, individual SOA solutions must make carefully balanced decisions among customized, proprietary, and open-standard technologies, which SOA characteristics and components to implement, and which areas of business function and process to which to apply them. Of course, these decisions will be

balanced between business benefits, technology maturity, and implementation or maintenance efforts.

3.3.2 Traditional approaches

Traditionally, integration approaches to any extended enterprise scenarios are technology-centric. Organizations that want to integrate their systems select a product- or platform-related solution based on the installed infrastructure and communications protocols to be used between the partner systems.

While this approach can be seen as inflexible and inappropriate for any integrations crossing departmental or organizational boundaries, there are a number of scenarios which still favor a more traditional integration over the use of SOA and Web services. The following list represents examples where a more traditional approach would be favored over the use of SOA:

- ▶ A low number of static integration points

There are a number of situations where the integration between two systems can be seen as contained and static, and will not require any future additional integrations or constant changes. In this scenario, there are limited drivers for implementing the integration using SOA principles.

A good example of this is the integration between two systems on a production line. While the overriding production process might change (alterations in design or change of production model, for example), there is limited scope for the communication protocol or implemented technology changing frequently. In this situation, the performance and development gains obtained by the tight-coupling is preferable to the benefits obtained by SOA principles.

- ▶ High dependency on real-time communication

While it might not be a big issue in day-to-day transactional enterprise integrations, a number of systems rely on real-time communication. A good example is an early warning missile system that has to detect an attack, and facilitate a counter-attack with minimal or no delay. In this scenario, a five-minute delay in a queue is not acceptable. Another example is any delay in having to translate a transaction request from any proprietary to open standards.

Most business systems however, can be viewed in a very different light. If we look at most business scenarios for any extended enterprise, the need for flexibility and open standards support is more critical.

- ▶ Within a homogeneous environment

Admittedly, there are few enterprises these days that are running a true homogeneous environment. But in scenarios where you are connecting

together two systems which are built and deployed on identical platforms , the need for open standards is reduced.

While this scenario is probably more relevant to smaller, specialized organizations, there are instances in any integration where this could be true. One example which is commonly repeated is the integration of ERP based systems, such as SAP, between business partners where the application and protocols are the same. In this scenario, there are benefits in both the SOA and traditional approach to system integration. The selection then, must be based on both future change cases and frequency of change.

- Technology centric integrations

In some integration scenarios, the integration is focused on the technology requirements rather than the business operations or interactions. In general this would be related to micro level integrations, such as within a service.

An example of this could be the interactions between a number of Java or .NET components within an application, or the interaction between a number of CICS Transaction Server transactions. We use a technology-centric integration approach to facilitate the interactions between these components at the micro level, but externalize the operations in a open standards-based interface at the macro level.

3.3.3 Ensuring quality of service

The following quality of service concerns are of particular importance when working in the Extended Enterprise domain.

Important: This profile is intended as a rough preliminary guide to quality of service concerns that differentiate this domain, high-level architectural design. However, it is not a substitute for thorough analysis at a later design stage.

Availability

High availability can be a particularly significant issue in the interenterprise integration domain. It is important that you use careful availability management to provide acceptable levels of customer service or, in some cases, to meet contractual obligations regarding the availability of the application service being provided.

Federation

To avoid overlap and inconsistencies in the implementation and management of an interenterprise application integration scenario, it is crucial to clearly define and agree to the responsibilities of each partner. In particular, agreed upon

mechanisms are needed to pass resource and user authentication and authorization information between domains.

Performance

With interenterprise application integration, components of the end-to-end solution are outside the enterprise boundaries and cannot be directly influenced. A client to an external application will find it difficult to control such variables as response time, workload, and availability.

To minimize such dependencies, consider loosely coupled and reliable communications.

Security

This topic includes a range of complex issues. For the purpose of this discussion, the communication channel is secured by using firewalls and proper authentication, authorization, and so forth.

In addition, we secure the exchange of the data itself. In the case of an intraenterprise scenario, it can be sufficient to use a trustworthy network. For interenterprise communication, this is most likely not possible. There is a need to protect (encrypt) the data and be certain of the sender's identity (signature).

Standards compliance

To enable interoperability between enterprises, standards compliance is a key capability in the interenterprise integration domain. Widely accepted public standards normally are required to have agreement between partners. There is also usually a need for compatibility with standard firewalls when communicating between trusted private networks and untrusted networks, such as the Internet.

Extended Enterprise pattern

The Extended Enterprise business pattern from the Patterns for e-business, also known as the Business-to-Business (B2B) pattern, addresses the interactions and collaborations between business processes in separate enterprises. This pattern can be observed in solutions that implement programmatic interfaces to connect interenterprise applications. It does not cover applications that are directly invoked through a user interface by business partners across organizational boundaries.

Note: The Application patterns for Extended Enterprise are basically the implementation of Application Integration across organizational boundaries. The difference is mainly how quality of service aspects affect the Runtime patterns.

4.1 Using the Extended Enterprise business pattern

Table 4-1 shows some cross-industry examples of the Extended Enterprise pattern.

Table 4-1 Cross-industry examples

Service	Examples
Buy Side	<ul style="list-style-type: none">▶ Direct Procurement▶ Indirect Procurement▶ Supply chain execution
Sell Side	<ul style="list-style-type: none">▶ B2B e-commerce (Distributors)
Trading Partner® Modernization	<ul style="list-style-type: none">▶ EDI modernization
Exchange Participation	<ul style="list-style-type: none">▶ Private e-exchanges▶ Public e-exchanges

Table 4-2 lists some industry-specific example applications that can be implemented though the Extended Enterprise pattern.

Table 4-2 Industry-specific examples

Industry	Example applications
Manufacturing	<ul style="list-style-type: none">▶ Supply chain planning▶ Supply chain execution▶ Vendor-managed inventory
Travel	<ul style="list-style-type: none">▶ Checking flight or room availability▶ Making or modifying reservations
Retail	<ul style="list-style-type: none">▶ Checking supplier inventory▶ Placing replenishment orders▶ Paying suppliers automatically
Financial	<ul style="list-style-type: none">▶ Transferring payments▶ Checking account balances▶ Obtaining credit information▶ Loan origination▶ Processing securities
Telecommunication	<ul style="list-style-type: none">▶ OSS integration▶ Cross-organization order management▶ Managed service provider interconnect

If you are not yet sure that your business problem can be solved by the functionality enabled through an Extended Enterprise solution design, the

guidelines in the next section provide additional information about choosing this business pattern. Business and IT drivers, the e-business context appropriate for this solution type, and additional solution details are discussed here.

If you determine that the Extended Enterprise business pattern can provide an appropriate solution design for your business needs, the next step is to select an Application pattern. The Extended Enterprise business pattern can be implemented using any one of three Application patterns discussed here. They provide solution flexibility so that the selected Business pattern can address the specific needs of the business process being automated.

4.2 General guidelines

This section details a business and IT scenario for an Extended Enterprise solution and helps you to determine whether the Extended Enterprise business pattern is appropriate for your interenterprise application integration.

4.2.1 Business and IT drivers

Use the Extended Enterprise business pattern if your business is developing a solution with the following characteristics:

- ▶ The business processes need to be integrated with existing business systems and information.
- ▶ The business processes need to integrate with processes and information that exist at partner organizations.

4.2.2 Context

Figure 4-1 on page 66 illustrates the general problem addressed by this pattern. Interactions between partners form a public process, or potentially multiple public processes. Each of these processes must be integrated into the private business process flows of each partner. Such integration can be as simple as passing data to a particular application, or as sophisticated as initiating or resuming a multistep workflow involving several applications and user interactions.

For example, Partner A, the source application, and Partner B, the target application, agree upon sharing specific business processes and a process flow. Partner A invokes a public process flow that, in turn, invokes a private, internal process flow within Partner B's organization. Partner A is not concerned with the details of Partner B's private process flow. Instead, Partner A cares only about the results that it expects in response to the invoked process in Partner B.

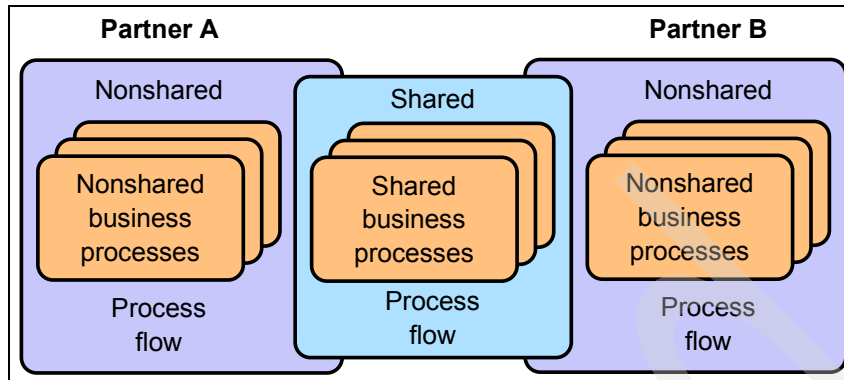


Figure 4-1 Extended Enterprise context

The golden rule of business-to-business integration is that it is better to know less about the trading partner's private processes and the implementation details of their applications. This loose coupling enables organizations to evolve their applications without affecting trading partner's applications.

Obviously, specific functionality supported by these applications depends on the particular details of the trading partner agreements and service level agreements between the organizations involved. Yet a survey of such applications in multiple industries reveals certain common approaches that have been successful. These commonalities of success are harvested as the various Application patterns that can be used to implement this Business pattern.

For example, consider the case of a company that wants to integrate their retail organization with a range of external manufacturers. The Extended Enterprise pattern improves organizational efficiency and reduces the latency of business events by integrating the external manufacturers with the inventory replenishment system and reducing the likelihood of unfilled orders. The Extended Enterprise pattern also applies a structured exchange with trading partners and supports real-time access to and from applications. This allows the resellers to receive the benefits of an updated inventory and receive real-time notice of any out of stock items.

The Extended Enterprise pattern also allows this company to minimize application complexity and to integrate their applications with resellers that have unique infrastructure designs. They can leverage their current skills and existing investments, while eliminating the need for extensive retraining and infrastructure investments.

4.2.3 Solution

The Extended Enterprise pattern may consist of all or some of the following elements:

- ▶ Business entities consisting of programs, applications, or databases that exist within an organization and need access and connection to other business entities across the network.
- ▶ A network that:
 - Is based on TCP/IP and other Internet technologies
 - Can be a dedicated wide area network (WAN) connection
- ▶ Business rules that:
 - Manage the integration between the business entities
 - Describe trading partner agreements
 - Use workflow rules to determine the sequence of steps and the data flow that needs to be used to facilitate the integration. These rules:
 - Describe the sequence of steps that a message needs to go through before transferring to the other business entity
 - Specify how and where the message should be delivered
 - Use transformation rules to specify format and protocol transformations that need to be applied to messages that flow between the business entities
- ▶ A set of interactions that includes the execution of a jointly-agreed business process
- ▶ Patterns based on those described by the business interaction patterns framework

4.2.4 Employing the pattern

This pattern can be observed in such solutions as:

- ▶ A manufacturer or wholesaler enabling external resellers to place orders to their inventory management system
- ▶ Extended value chain functions within e-Marketplaces that support cross-enterprise processes such as demand planning and collaborative design

4.2.5 What is next?

If you determine that the Extended Enterprise business pattern can provide an appropriate solution design for the application you are developing, then selecting

an appropriate Application pattern is your next step. If the Extended Enterprise business pattern is not appropriate for your development efforts, review the Business patterns again to determine which pattern best addresses your e-business needs.

4.3 Extended Enterprise application patterns

We present the Extended Enterprise Application patterns in order of increasing flexibility and sophistication. As the Application patterns build on each other, their capabilities and reliance on middleware increase, and they require less application development. Use the graph show in Figure 4-2 to select the Application pattern that best fits your requirements.

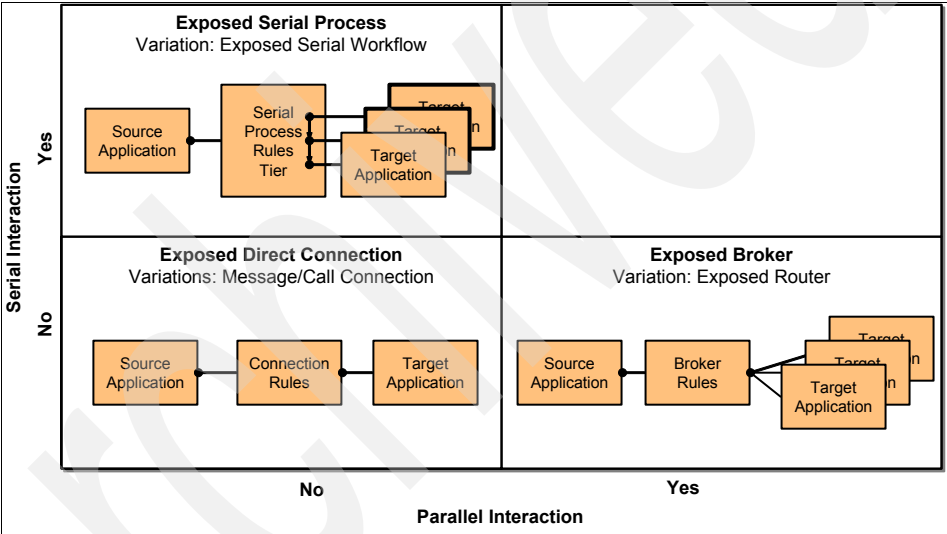


Figure 4-2 The Extended Enterprise application patterns

Note: While the Exposed Parallel Process Application pattern is another possibility, it is not currently being observed in the Extended Enterprise domain. We expect the Exposed Parallel Process Application pattern to appear at a later stage.

Business and IT drivers

Table 4-3 and Table 4-4 on page 70 summarize the business and IT drivers for the Extended Enterprise application patterns and their variations.

Table 4-3 Business drivers

Business drivers	Exposed Direct Connection Message connection	Exposed Direct Connection Call connection	Exposed Router variation	Exposed Broker	Exposed Serial Process	Exposed Serial Workflow variation
Improve organizational efficiency	✓	✓	✓	✓	✓	✓
Reduce the latency of business events	✓	✓	✓	✓	✓	✓
Support a structured exchange with business partners	✓	✓	✓	✓	✓	✓
Support real-time one-way message flows to partner processes	✓		✓	✓	✓	✓
Support real-time request/reply message flows to partner processes		✓	✓	✓	✓	✓
Support dynamic routing of message between partners to one of many target applications			✓	✓	✓	✓
Support dynamic distribution of message between partners to multiple target applications				✓	✓	✓
Support automated coordination of business process flow between partners					✓	✓
Support human interaction and intervention within the process flow between partners						✓

Table 4-4 IT drivers

IT drivers	Exposed Direct Connection Message connection	Exposed Direct Connection Call connection	Exposed Router variation	Exposed Broker	Exposed Serial Process	Exposed Serial Workflow variation
Minimize total cost of ownership (TCO)			✓	✓	✓	✓
Leverage existing skills	✓	✓	✓	✓	✓	✓
Leverage the existing investment	✓	✓	✓	✓	✓	✓
Enable back-end application integration	✓	✓	✓	✓	✓	✓
Minimize application complexity	✓	✓	✓	✓	✓	✓
Minimize enterprise complexity			✓	✓	✓	✓
Improve maintainability			✓	✓	✓	✓
Improve flexibility by externalizing process logic from application logic					✓	✓
Support long running transactions						✓

Legend for Extended Enterprise application patterns

The conventions shown in Figure 4-3 are used to describe the Extended Enterprise application patterns in illustrations that follow.

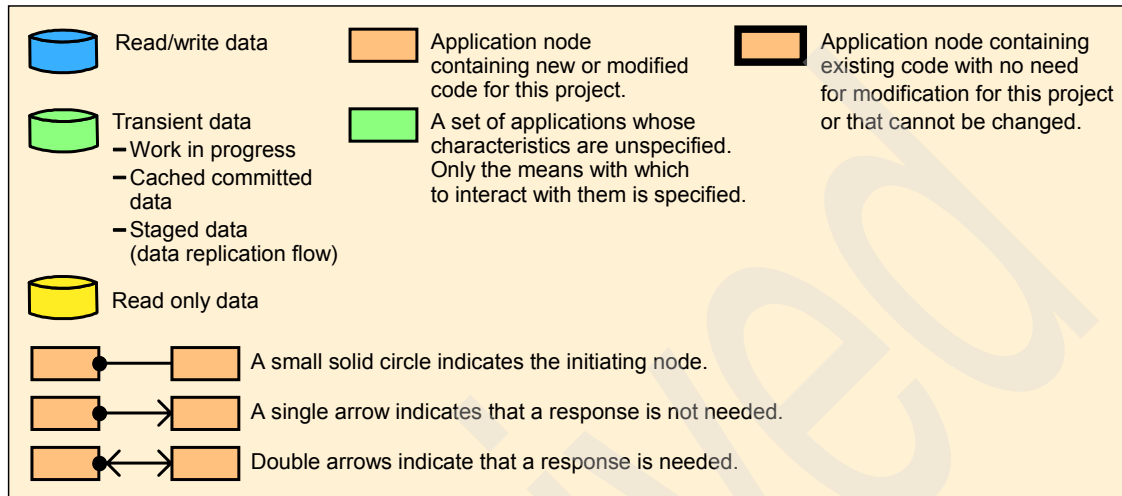


Figure 4-3 Application pattern diagram conventions

4.3.1 Exposed Direct Connection application pattern

The Exposed Direct Connection application pattern represents the simplest interaction type based on a one-to-one topology. It allows a pair of applications to directly communicate with each other across organization boundaries. Interactions between a source and a target application can be arbitrarily complex. Generally, complexity can be addressed by breaking down interactions into more elementary interactions.

More complex point-to-point connections have modeled connection rules, such as business rules, associated with them, as shown in Figure 4-4 on page 72. Connection Rules generally control the mode of operation of a connector, depending on external factors. Examples of Connection Rules are:

- ▶ Business data mapping rules (for adapter connectors)
- ▶ Autonomic rules (such as priority in a shared environment)
- ▶ Security rules
- ▶ Capacity and availability rules

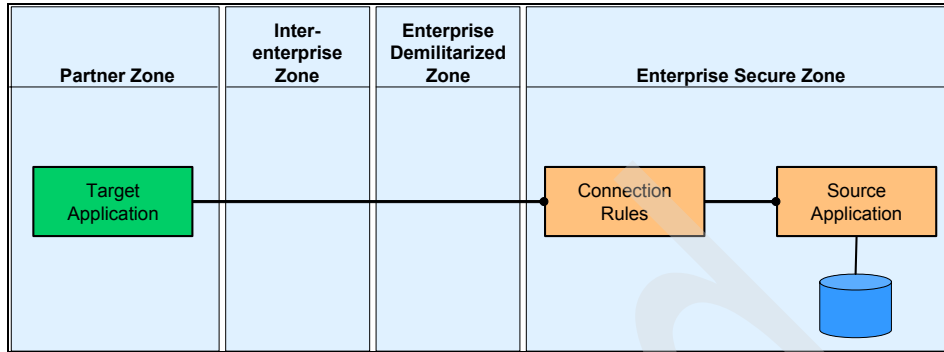


Figure 4-4 Exposed Direct Connection application pattern

Note: The Connection Rules component is not needed when no modeled rules are associated with the connection.

The Exposed Direct Connection application pattern has two variations:

- ▶ Message Connection variation
- ▶ Call Connection variation

All applications of the Direct Connection application pattern are one variation or the other. The variation required depends on whether the initiating source application needs an immediate response from the target application to proceed with execution.

Both variations can be used either with synchronous or asynchronous communication protocols. However, there are preferences for a specific protocol type, depending on the variation. For example, the Call Connection variation has a more natural fit with synchronous protocols. The Message Connection variation favors asynchronous protocols.

We examine these two variations in more detail later in this section.

Business and IT drivers

The business and IT drivers for choosing the Exposed Direct Connection application pattern are to:

- ▶ Improve the organizational efficiency
- ▶ Reduce the latency of business events
- ▶ Support a structured exchange with business partners
- ▶ Support real-time one-way message flows to partner processes
- ▶ Support real-time request/reply message flows to partner processes
- ▶ Leverage existing skills

- ▶ Leverage the existing investment
- ▶ Enable back-end application integration
- ▶ Minimize application complexity

The primary goal of the Exposed Direct Connection application pattern is to permit an application to gain direct and real-time access to another application that is outside the organization to reduce the latency of business events.

Solution

The Exposed Direct Connection application pattern, as shown in Figure 4-4 on page 72, is divided into a number of logical components:

- ▶ The *Source Application tier* represents one or more applications that are interested in initiating an interaction with the target application in another organization.
- ▶ The *Connection* is the line between the source application and the target application representing a point-to-point connection between the two applications.
- ▶ The *Connection Rules tier* represents any business rules associated with the connection, such as data mapping rules and security rules.
- ▶ The *Target Application tier* represents a new application, a modified existing application, or an unmodified existing application. This application is responsible for implementing the necessary business services.

Because this application is directly exposed across organizational boundaries, it must implement or exploit the necessary security features such as authentication, authorization, confidentiality, integrity, and logging for non-repudiation purposes.

Guidelines for use

Direct integration between applications can be inflexible. That is any changes to one application can have *knock-on*, unexpected, effects on other applications. This is especially dangerous when integrating across organizational boundaries. Any changes to the exposed target application might require changes to many partner applications. Such changes can be both expensive and time-consuming.

You can minimize such knock-on effects by using document-based adapters that wrap the applications in the exposed connection. Document-based adapters are small programs that convert the mutually agreed upon messages into application programming interface (API) calls to existing or new back-end applications. This layering technique isolates the exposed applications from partner applications and increases flexibility. Any changes to these exposed applications only impact the adapter, provided there is no need to change the mutually agreed upon messages.

The message definition should be generalized to further promote flexibility. Do not tightly couple messages with back-end application APIs. Rather the message should capture all the necessary information required for that logical interaction across business boundaries. Such generalization helps to cope with changes to the back-end application API without changing the agreed upon message format.

Benefits

The use of this pattern allows the complete integration of applications belonging to different companies, assuring a real-time and service-oriented access to external data and processes. Source and target applications are clearly decoupled, as are business logic and communication details. Therefore, it is possible to develop different parts of the whole system independently.

Limitations

This pattern implements a direct connection between the source and target application. Therefore, this pattern cannot be used for the intelligent routing of requests, decomposition and recomposition of requests, or for invoking complex business process workflow as a result of a request from a partner application. Under such circumstances, consider a more advanced Application pattern, such as Exposed Broker or Exposed Serial Process.

4.3.2 Exposed Direct Connection: Message Connection variation

The Message Connection variation, shown in Figure 4-5, applies to solutions where the business process does not require a response from the exposed target application within the scope of the interaction.

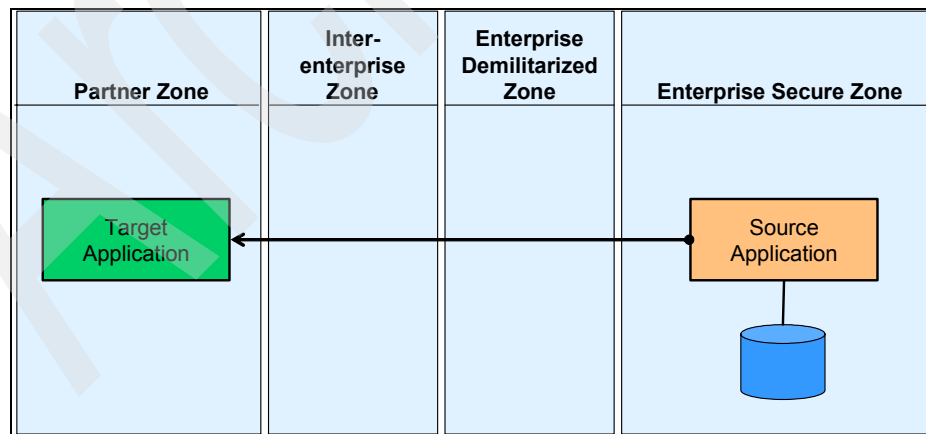


Figure 4-5 Message Connection variation

Note: We do not show the Connection Rules box in Figure 4-5 because we want to focus on the connection itself.

Business and IT drivers

The business and IT driver for choosing the Message Connection variation of the Direct Connection application pattern is to support real-time, one-way message flows. The main driver for selecting this variation is when the business process has no interest in the result of the operation. This variation also has the most natural fit when message-oriented middleware is used, such as IBM WebSphere MQ.

4.3.3 Exposed Direct Connection: Call Connection variation

The Call Connection variation, shown in Figure 4-6, applies to solutions where the business process depends on the exposed target application to process a request and return a response within the scope of the interaction.

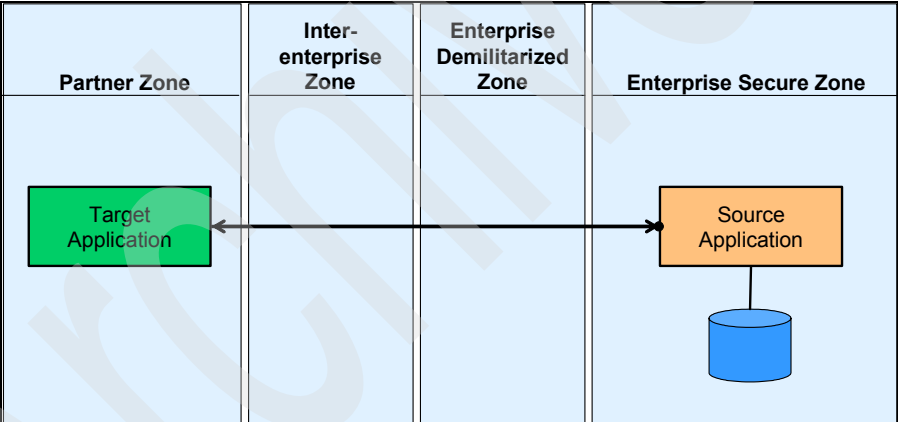


Figure 4-6 Call Connection variation

Note: We do not show the Connection Rules box in Figure 4-6 because we want to focus on the connection itself.

Business and IT drivers

The business and IT driver for choosing the Call Connection variation of the Direct Connection application pattern is to support real-time request/reply message flows. The main driver for selecting this variation is when the business process requires a result message in the interaction.

4.3.4 Exposed Broker application pattern

The Exposed Broker application pattern, shown in Figure 4-7, is based on a one-to- n topology that separates distribution rules from the applications. It allows a single interaction from a partner's source application to be distributed to multiple target partner applications concurrently.

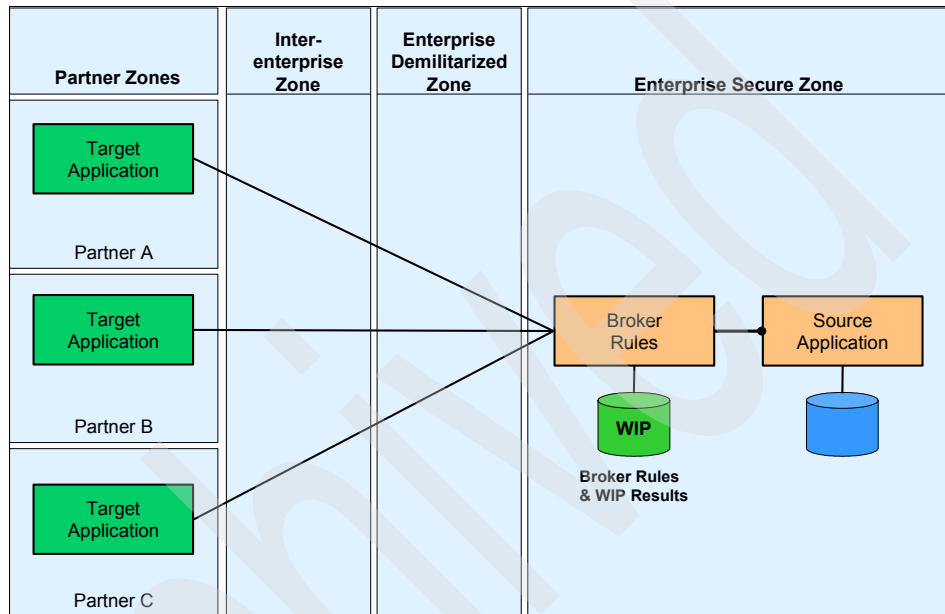


Figure 4-7 Exposed Broker application pattern

The Exposed Broker application pattern applies to solutions where the source application starts an interaction that is distributed to multiple target applications across organization boundaries. It separates the application logic from the distribution logic based on broker rules. The decomposition and recomposition of the interaction is managed by the Broker Rules tier.

The Exposed Broker pattern reuses the Exposed Direct Connection pattern to provide connectivity between the tiers. The Broker Rules tier can support the Message Connection variation or Call Connection variation (or both variations) of the Exposed Direct Connection pattern.

Business and IT drivers

The primary business driver for selecting this Application pattern is to permit one application to interact with one or more of multiple partner applications across organization boundaries. Using a hub-and-spoke architecture instead of a

point-to-point architecture allows for the seamless integration of applications while minimizing the complexity.

A request for information can be routed to one of many targets or simultaneously to multiple targets. The resulting request message can be decomposed into multiple request messages. The reply messages are then recomposed into a single reply message using the appropriate recombination rules. This externalization of routing, decomposition, and recombination rules from individual source and target applications increases the maintainability and flexibility and reduces the enterprise-wide integration complexity.

The primary IT driver for selecting this Application pattern is to permit loose coupling of clients and services with minimum modification to each. The solution should allow for multiple transmission protocols to be used and for transformation of protocols between client and service.

Solution

This Application pattern, as shown in Figure 4-7 on page 76, is divided into several logical components:

- ▶ The *Source Application tier* represents one or more applications that are interested in interacting with the target applications in another organization.
- ▶ The *Broker Rules tier* reduces the proliferation of direct connections. In addition, it supports message routing, decomposition and recombination, and message enhancement and transformation. These rules are often captured as business rules that govern the behavior of the broker tier. This tier also uses a work-in-progress data store to retain the intermediate results from the responses coming back from target applications until all the necessary responses are received.
- ▶ The *Target Application tier* represents new, modified existing, or unmodified existing applications in a partner organization. These applications are responsible for implementing the necessary business services.

Guidelines for use

To increase the flexibility of the solution and responsiveness to changing business requirements, we recommend that you pay particular attention to the definition of reusable messages and services that pass through the Broker Rules tier.

Use robust transaction processing systems to implement the back-end applications to ensure availability, scalability, and performance.

A decomposition implementation (one source message to multiple target messages) requires state persistence and re-composition of the response messages.

Use standards where possible to minimize future changes required to the source and target applications. This is particularly important in an interenterprise solution.

Security is a primary concern when opening up business processes to external organizations. The solution should include robust security mechanisms to protect resources.

Benefits

The benefits of this Application pattern are:

- ▶ It allows the integration of multiple, diverse applications between partner organizations.
- ▶ It minimizes the impact to existing applications.
- ▶ The broker provides routing services, relieving the source application from being aware of the target application.
- ▶ The broker provides transformation services that allow the source and target to use different communication protocols.
- ▶ The broker can provide decomposition and recomposition of messages, allowing one request from the source to be satisfied using multiple target applications. The fact that the response is a composite of multiple requests and responses is hidden from the source application.
- ▶ The use of a central broker minimizes the impact of changes in the location of the target application.

Limitations

You must implement logic at the broker for routing, decomposition, and recomposition tasks.

4.3.5 Exposed Broker: Router variation

The Router variation of the Exposed Broker application pattern, shown in Figure 4-8 on page 79, applies to solutions where the partner's source application initiates an interaction that is forwarded to, at most, one of multiple target applications. The selection of the target application is controlled by the distribution rules that govern functioning of the connector component.

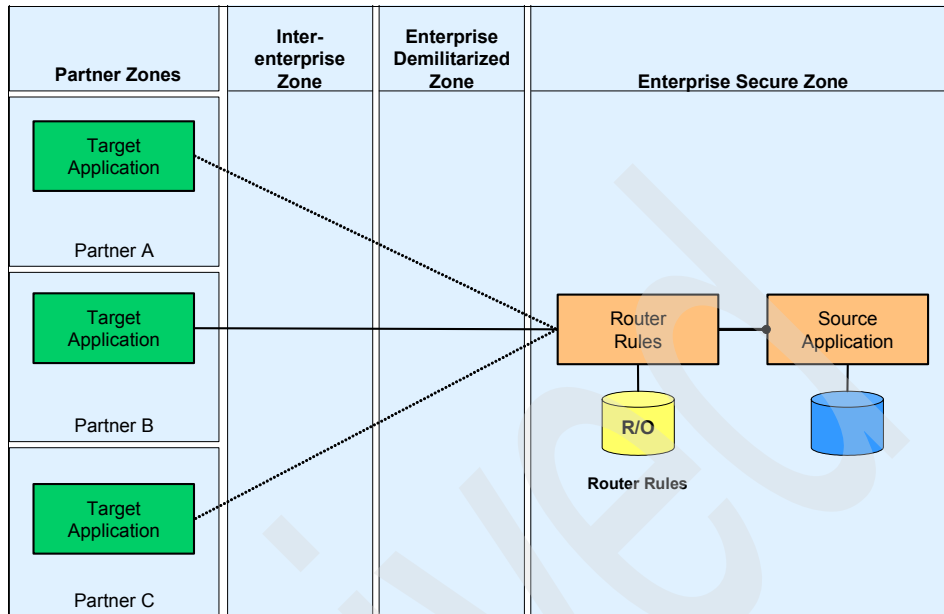


Figure 4-8 Router variation

Business and IT drivers

The requirements and their solution are less complex than those defined by the Exposed Broker application pattern in that the decomposition of messages and transmission to multiple targets simultaneously are not required.

This pattern is also applicable to the one-to-one application integration scenario. In this case, the source and target do not adhere to the same message and interface formats. Therefore, they require a transformation service in the Enterprise Application Integration (EAI) infrastructure.

Solution

This Application pattern, as shown in Figure 4-8 on page 79, is divided into several logical components:

- The *Source Application tier* represents one or more applications that are interested in interacting with the target partner applications, one target at a time.
- The *Router Rules tier* represents any business rules associated with the message handling, such as routing and transformation. It receives requests from multiple source applications and routes them intelligently to the appropriate partner applications. This tier implements minimal business logic.

- ▶ The *Target Application tier* represents new, modified, existing, or unmodified existing applications. These applications are responsible for implementing the necessary business services.

Guidelines for use

The guidelines for this Application pattern are the same as those for the Exposed Broker application pattern.

Benefits

The benefits of this Application pattern are:

- ▶ It allows the integration of multiple, diverse partner applications.
- ▶ It minimizes the impact to existing applications.
- ▶ It provides routing services, relieving the source application from being aware of the target application.
- ▶ It provides transformation services that allow the source and target to use different communication protocols.
- ▶ The use of a central router minimizes the impact of changes in the location of the target application.

Limitations

With the Router variation, there is limited ability in the router to manipulate the requests. It performs intelligent routing and protocol transformation. However, it does not have the ability to send simultaneous requests to the target applications based on one incoming request, nor does it have the ability for decomposition or recomposition.

4.3.6 Exposed Serial Process application pattern

The Exposed Serial Process application pattern, shown in Figure 4-9 on page 81, extends the one-to-*n* topology provided by the Exposed Broker application pattern by facilitating the sequential execution of business services hosted by a number of target applications. It enables the orchestration of a serial business process across enterprise boundaries, in response to an interaction initiated by the source application.

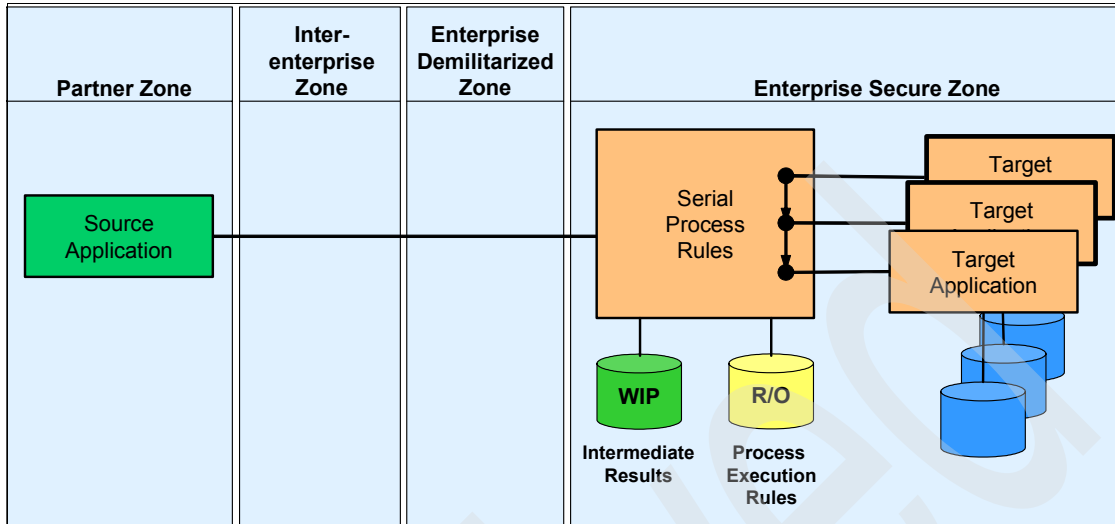


Figure 4-9 Exposed Serial Process application pattern

The Exposed Serial Process application pattern separates the process logic from application logic that is distributed across organization boundaries. The process logic is governed by Serial Process Rules that define execution rules for each target application, together with control flow and data flow rules. It can also include any necessary adapter rules.

Business and IT drivers

The primary business driver for selecting this Application pattern is to support the automated coordination of business process flow between partners. From an IT perspective, the key driver for selecting this Application pattern is to improve the flexibility and responsiveness of IT by externalizing the process flow logic from the application logic.

Solution

The Exposed Serial Process application pattern is broken down into three logical tiers:

- ▶ The *Source Application tier* represents an application in another organization that is interested in interacting with the Exposed Serial Process.
- ▶ The *Serial Process Rules tier* supports most of the services provided by the broker tier in the Exposed Broker application pattern, including the routing of requests, protocol conversion, message broadcasting, and message decomposition and recomposition. In addition, it supports the separation of business process flow logic from individual application logic.

The process logic is governed by serial process rules that define execution rules for each target application, together with control flow and data flow rules. It can also include any necessary adapter rules. The combination of these process execution rules is stored in read-only databases. This externalization of process flow logic is essential for the implementation of a flexible and responsive IT environment that can respond quickly to changing business needs. It also makes it possible to compose new end-to-end processes by combining different business services provided by different applications. Finally, this tier uses a work-in-progress (WIP) database to store the intermediate results from the execution of different process steps.

- The *Target Application tier* represents new, modified, existing, or unmodified existing applications that are responsible for implementing the necessary business services.

Guidelines for use

The flexibility and responsiveness provided by this Application pattern heavily depend on the externalization of process execution logic from individual partner applications. Applications that are based on a service-oriented architecture (SOA) approach, which have well-defined and coarse-grained business services that represent a unit of work, are better suited for participation in this Application pattern. You must be able to compose these business services into an end-to-end process flow. A given service may need to participate in more than one end-to-end process.

Standards should be used where possible to minimize future changes required to the source and target applications. This is particularly important in an inter-enterprise solution.

Security is a primary concern when opening business processes to external organizations. The solution should include robust security mechanisms to protect resources.

Benefits

The Exposed Serial Process application pattern improves the flexibility and responsiveness of an organization. It does this by implementing end-to-end process flows across organization boundaries and by externalizing process logic from individual applications. In addition, it provides a foundation for automated support for Business Process Management that enables the monitoring and measurement of the effectiveness of business processes.

Limitations

This Application pattern is ideally suited for straight-through processing where human interactions are not necessary to complete an end-to-end process. If

support for human interactions is needed to complete certain process steps, consider the Workflow variation of this Application pattern.

Similarly it does not support the parallel execution of multiple tasks. As the process composition technologies mature, we expect to see more widespread use of the Exposed Parallel Process application pattern in Extended Enterprise scenarios.

4.3.7 Exposed Serial Process: Workflow variation

The Workflow variation of the Exposed Serial Process application pattern, shown in Figure 4-10, extends the basic serial process orchestration capability by supporting human interaction for completing certain process steps.

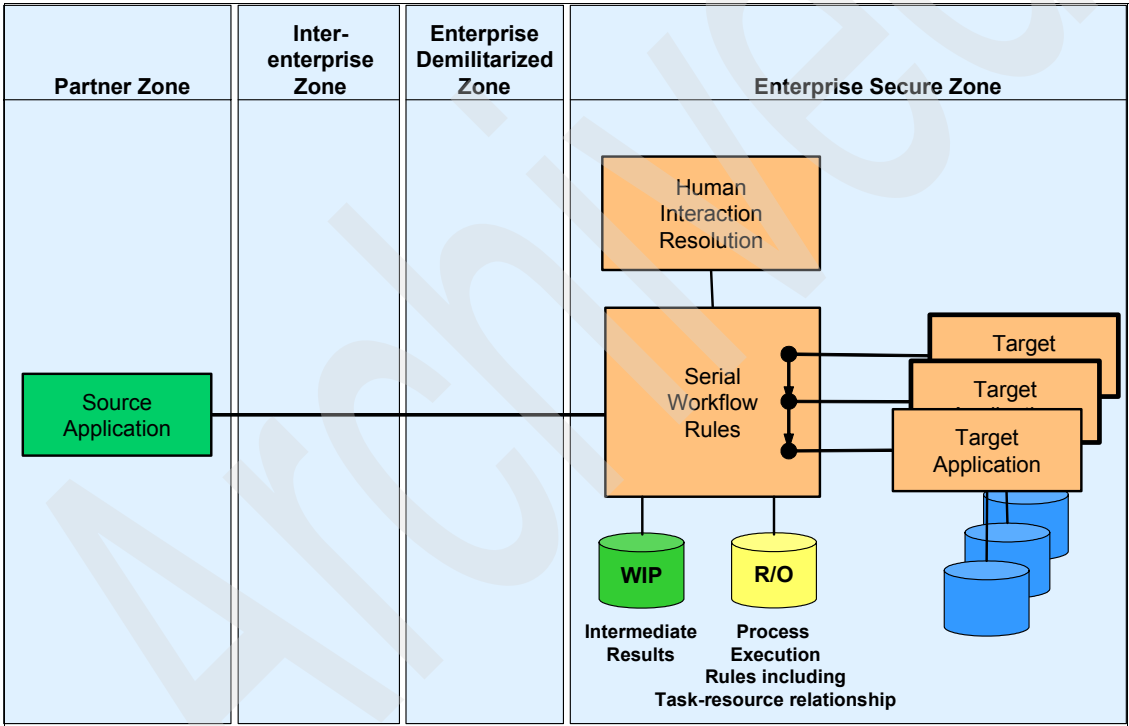


Figure 4-10 Exposed Serial Workflow variation

Business and IT drivers

All the business and IT drivers listed under the Exposed Serial Process application pattern also apply to this variation. The additional business driver for selecting this variation is the need to support human interaction and intervention within the process flow between partners. Support for long-running transactions

is another IT driver. This is often a prerequisite for the automation of complex process flows involving human interaction.

Solution

The Serial Workflow variation is broken down into three logical tiers:

- ▶ The *Source Application tier* is the same as for the Exposed Serial Process application pattern.
- ▶ The *Serial Workflow Rules tier* supports all the services provided by the serial process rules tier within the Exposed Serial Process application pattern. In addition, it supports certain tasks within the process to be routed to a person or people for completion. To accomplish this, the process execution rules are augmented with task-resource relationships that define which resources are capable of performing which tasks.

In this context, consider the following points:

- A task is a portion of the end-to-end process.
- Resources are capable of executing these tasks.
- People, departments, and target applications can all be resources capable of executing a particular task.

This tier resolves the task-resource relationship during the execution of a process. If the need for human interaction is identified, the task is added to a work list associated with an individual or a department as a work item to be completed by a person. The process is typically suspended until the completion of the task.

Finally, this tier provides support for long-running transactions. It uses a WIP database to store the intermediate results from the execution of different process steps until the complete execution of the end-to-end process.

- ▶ The *Target Application tier* is the same as for the Exposed Serial Process application pattern.

Guidelines for use

The following guidelines apply to this variation in addition to the guidelines that are documented in the Exposed Serial Process application pattern. We recommend that people-based exception handling be implemented for the majority of the automated tasks within the process. If an automated task reaches certain error conditions, people must be able to intervene and handle the exceptions.

Benefits

The Exposed Serial Workflow application pattern improves the flexibility and responsiveness of an organization. It implements end-to-end process flows across organization boundaries that externalize process logic from the individual application. Further flexibility is introduced by the externalization of task-resource resolution rules.

In addition, it provides a foundation for automated support for Business Process Management. This enables monitoring and measurements of the effectiveness of business processes.

Limitations

Exposed Serial Process: Workflow variation does not support the parallel execution of multiple tasks. As the process composition technologies mature, we expect to see more widespread use of the Exposed Parallel Workflow variation in Extended Enterprise scenarios.

Product descriptions

This chapter describes products that are discussed and used throughout this book for both development and runtime activities. The products described are:

- ▶ IBM WebSphere Application Server V6
- ▶ IBM DB2 Universal Database Enterprise Server Edition V8.2
- ▶ IBM Cloudscape
- ▶ IBM WebSphere MQ V5.3
- ▶ IBM WebSphere Business Integration Message Broker V5.0
- ▶ IBM WebSphere Business Integration Server Foundation V5.1
- ▶ IBM WebSphere Partner Gateway V6.0
- ▶ IBM Rational Application Developer V6
- ▶ IBM WebSphere Studio Application Developer Integration Edition V5.1

5.1 Runtime product descriptions

This section describes the IBM products that are discussed and used in runtime scenarios throughout this book.

5.1.1 IBM WebSphere Application Server V6

WebSphere Application Servers are a suite of servers that implement the J2EE specification. This simply means that any enterprise applications that are written to the J2EE specification can be installed and deployed on any of the servers in the WebSphere Application Server family.

The foundation of the WebSphere brand is the application server. The application server provides the runtime environment and management tools for J2EE and Web services-based applications. Clients access these applications through standard interfaces and APIs. The applications, in turn, have access to a wide variety of external sources such as existing systems, databases, and Web services, that can be used to process the client requests (see Figure 5-1).

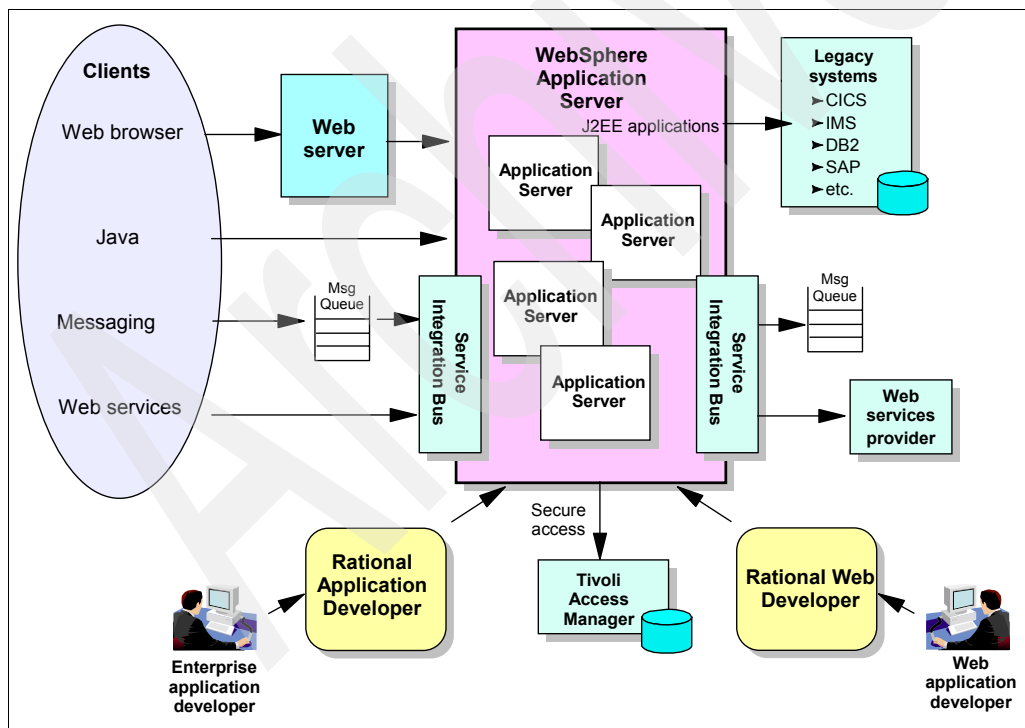


Figure 5-1 WebSphere Application Server product overview

WebSphere Application Servers are available in multiple packages to meet specific business needs. They are also available on a wide range of platforms, including UNIX®-based platforms, Microsoft operating systems, IBM z/OS®, and iSeries™. Although branded for iSeries, the WebSphere Application Server products for iSeries are functionally equivalent to those for the UNIX and Microsoft platforms.

Highlights and benefits

WebSphere Application Server provides the environment to run your Web-enabled e-business applications. You can think of an application server as *Web middleware*, or a middle tier, in a three-tier e-business environment. The first tier is the HTTP server that handles requests from the browser client. The third tier is the business database (for example, DB2 UDB for iSeries) and the business logic (for example, traditional business applications such as order processing). The middle tier is IBM WebSphere Application Server, which provides a framework for consistent, architected linkage between the HTTP requests and the business data and logic.

IBM WebSphere Application Server is intended for organizations that want to take advantage of the productivity, performance advantages, and portability that Java provides for dynamic Web sites. IBM WebSphere Application Server includes the following benefits:

- ▶ J2EE V1.4 support
- ▶ High performance connectors to many common back-end systems to reduce the coding effort required to link dynamic Web pages to real line-of-business data
- ▶ Application services for session and state management
- ▶ Web services that enable businesses to connect applications to other business applications, to deliver business functions to a broader set of customers and partners, to interact with marketplaces more efficiently, and to create new business models dynamically
- ▶ The service integration bus infrastructure to complement and extend WebSphere MQ and the application server

It is suitable for those that are currently using the WebSphere Application Server V5 embedded messaging and for those that need to provide messaging capability between WebSphere Application Server and an existing WebSphere MQ backbone.

The service integration bus features include:

- Multiple messaging patterns (APIs) and protocols for message-oriented and service-oriented applications.
- J2EE V1.4 compliant JMS default messaging provider

- Web services standards for supporting JAX-RPC APIs
- Reliable message transport capability
- Tightly and loosely coupled communications options
- Intermediary logic (mediations) to intelligently adapt message flow in the network
- Clustering support to provide scalability and high availability
- Quality of service options
- Support for the WebSphere Business Integration programming model, converging functions from workflow, message brokering, collaborations, adapters, and the application server.
- Fully integrated within WebSphere Application Server, including security, installation, administration console, performance monitoring, trace, and problem determination.
- Support for connectivity into a WebSphere MQ network

Packaging for distributed platforms

Because different levels of application server capabilities are required at different times for various e-business application scenarios, WebSphere Application Server is available in multiple packaging options. Although they share a common foundation, each product provides unique benefits to meet the needs of applications and the infrastructure that supports them. So, at least one WebSphere Application Server product package will fulfill the requirements of any particular project and the prerequisites of the infrastructure that supports it. As your business grows, the WebSphere Application Server family provides a migration path to higher configurations.

WebSphere Application Server - Express V6

The Express package is geared to those who need to get started quickly with e-business. It is specifically targeted at medium-sized businesses or departments of a large corporation, and is focused on providing ease of use and application development. It contains full J2EE V1.4 support, but is limited to a single server environment.

The WebSphere Application Server - Express offering is unique from the other packages in that it is bundled with an application development tool. Although there are WebSphere Studio and Rational Developer products designed to support each WebSphere Application Server package, they are normally ordered independently of the server. WebSphere Application Server - Express includes the Rational Web Developer application development tool. It provides a development environment geared toward Web developers and includes support

for most J2EE V1.4 features, with the exception of EJB™ and J2EE Connector Architecture development environments.

However, keep in mind that WebSphere Application Server - Express does contain full support for EJB and the J2EE Connector Architecture, so you can deploy applications with them.

WebSphere Application Server V6

The WebSphere Application Server package is the next level of server infrastructure in the WebSphere Application Server family. Though the WebSphere Application Server is functionally equivalent to that shipped with Express, this package differs slightly in packaging and licensing. The development tool included is a trial version of Rational Application Developer, the full J2EE V1.4 compliant development tool.

WebSphere Application Server Network Deployment V6

WebSphere Application Server Network Deployment is an even higher level of server infrastructure in the WebSphere Application Server family. It extends the WebSphere Application Server base package to include clustering capabilities, edge components, and high availability for distributed configurations. These features become more important at larger enterprises, where applications tend to service a larger customer base, and more elaborate performance and availability requirements are in place.

Application servers in a cluster can reside on the same or multiple machines. A Web server plug-in installed in the Web server can distribute work among clustered application servers. In turn, Web containers running servlets and JSPs can distribute requests for EJBs among EJB containers in a cluster.

More information can be found at the IBM WebSphere Application Server Web site:

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

More information about using IBM WebSphere Application Server V6 can be found in the redbook *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

5.1.2 IBM DB2 Universal Database Enterprise Server Edition V8.2

IBM DB2 Universal Database™ Enterprise Server Edition is a multi-user version of DB2 Universal Database that allows you to create and manage single partitioned or partitioned database environments. Partitioned database systems can manage high volumes of data and provide benefits such as high availability and increased performance. Other features include:

- ▶ A data warehouse server and related components
- ▶ DB2 Connect™ functionality for accessing data stored on midrange and mainframe database systems
- ▶ Satellite administration capabilities

DB2 Universal Database V8.2 delivers new features to address the ever increasing demands and requirements on important data, which include:

- ▶ Broadened autonomic computing that automate and simplify potentially time consuming and complex database tasks
- ▶ A significant amount of new capabilities as well as further integration of DB2 tooling into the Microsoft .NET and WebSphere Java environment

These new capabilities simplify the development and deployment of DB2 applications and allow application developers to take advantage of the openness, performance, and scalability of DB2, without regard to the back-end database or the chosen application architecture

- ▶ Integration of industry proven high availability disaster recovery technology allow line-of-business managers and the enterprise itself to benefit because applications face less risk of downtime

You can find more information about the IBM DB2 Universal Database Enterprise Server Edition at:

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

5.1.3 IBM Cloudscape

IBM Cloudscape™ is an open source Java relational database management system that can be embedded in Java programs and used for online transaction processing. IBM Cloudscape features include:

- ▶ Rapid application development through the Java-based relational database management system (RDBMS) built from the ground up for the embedded environment. This platform-independent, small footprint database integrates tightly with any Java based solution, allowing shortened development cycles.
- ▶ Supports Java technology standards. Single application versions can be created that run on any standard Java Virtual Machine (JVM™).
- ▶ Does not require database administration or resource management and is invisible to non-technical users, thus eliminating the need for database administration at each client installation site. IBM Cloudscape can also be deployed anywhere, from notebook or desktop applications to robust server solutions.

- ▶ Tuned for high performance as well as efficient use of resources, with a straightforward migration path to various IBM DB2 versions.
- ▶ Supports international characters and formats as well as a rich set of RDBMS features that are based on SQL-92E, including row locking, triggers, and stored procedures.
- ▶ Available access to IBM Cloudscape from inside Java programs using JDBC™ and the ability to embed the IBM Cloudscape database inside Java applications on the server.

Cloudscape Network server comes as part of the IBM Cloudscape package. This provides multi-user connectivity to IBM Cloudscape databases within a single system or over a network using Standard Distributed Relational Database Architecture™ protocol.

You can find more information about IBM Cloudscape at:

<http://www.ibm.com/software/data/cloudscape>

5.1.4 IBM WebSphere MQ V5.3

IBM WebSphere MQ provides assured, once-only delivery of messages across more than 35 industry platforms using a variety of communication protocols. The transportation of message data through a network is made possible through the use of a network of WebSphere MQ queue managers. Each queue manager hosts local queues that are containers used to store messages. Through remote queue destinations and message channels, data can be transported to its destination queue manager.

To use the services of a WebSphere MQ transport layer, an application must make a connection to a WebSphere MQ queue manager, the services of which enable it to receive (*get*) messages from local queues or send (*put*) messages to any queue on any queue manager. The application's connection can be made directly (where the queue manager runs locally to the application) or as a client to a queue manager that is accessible over a network.

Dynamic workload distribution is another important feature of WebSphere MQ. This feature shares the workload among a group of queue managers that are part of the same cluster. This enables WebSphere MQ to balance the workload across available resources automatically and provide hot standby capabilities if a system component fails. This is a critical feature for companies that must maintain around-the-clock availability.

WebSphere MQ supports a variety of application programming interfaces (APIs), including MQI, AMI, and JMS that provide support for several programming languages as well as point-to-point and publish/subscribe communication

models. In addition to support for application programming, WebSphere MQ provides several connectors and gateways to a variety of other products, such as Microsoft Exchange, Lotus® Domino®, SAP/R3, CICS Transaction Server, and IMS™, to name just a few.

To learn more, visit the IBM WebSphere MQ Web site:

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

5.1.5 IBM WebSphere Business Integration Message Broker V5.0

IBM WebSphere Business Integration Message Broker V5.0 extends the messaging capabilities of WebSphere MQ by adding message routing, transformation, and publish/subscribe features. WebSphere BI Message Broker provides a runtime environment that executes message flows, which consist of a graph of nodes that represent the processing that is needed for integrating applications. They can be designed to perform a wide variety of functions, including:

- ▶ Routing of messages to zero or more destinations based on the contents of the message or message header. Both one-to-many and many-to-one messaging topologies are supported.
- ▶ Transformation of messages into different formats so that diverse applications can exchange messages that each of them can understand.
- ▶ Processing message content in several message domains, including the XML domain that handles self-defining (or generic) XML messages, the Message Repository Manager (MRM), which handles predefined message sets, and unstructured data (BLOB domain).

For more information, visit the IBM WebSphere Business Integration Message Broker Web site:

<http://www.ibm.com/software/integration/wbimessagebroker/>

5.1.6 IBM WebSphere Business Integration Server Foundation V5.1

IBM WebSphere Business Integration Server Foundation V5.1 builds on WebSphere Application Server to provide a premier Java 2 Enterprise Edition (J2EE) and Web services technology-based application platform for deploying enterprise Web services solutions for dynamic On Demand Business.

It includes Business Process Choreographer, which provides IBM WebSphere Application Server with the ability to choreograph intraenterprise and interenterprise services into business processes described with open-standard Business Process Execution Language for Web Services (WS-BPEL). Each

activity in the business process is defined as a service using WSDL. The business process itself is also exposed as a WSDL-defined Web service.

The business processes that are implemented in an enterprise typically require a mixture of human and IT resources, and these processes are supported by Business Process Choreographer. A *process* is a directed graph that starts with an Input node and ends with an Output node. A process itself is described in WSDL. Its input and output are described as WSDL messages.

A process can contain many activities. An *activity* can be the invocation of an EJB, a Java class, a service, or another process. A process can also be event-driven. For example, it can be paused to wait for an event and then resumed when a message arrives.

Business Process Choreographer supports processes that can be:

- ▶ Long-running and interruptible (with human intervention)
- ▶ Short-running and part of a single transaction

For more information about IBM WebSphere Business Integration Server Foundation V5.1, visit the Web site:

<http://www.ibm.com/software/integration/wbisf/>

5.1.7 IBM WebSphere Partner Gateway V6.0

WebSphere Partner Gateway enables business-to-business process integration and data sharing between partners of all types and sizes. It also provides support for Web services. It is implemented on top of J2EE, and designed for multitier and single-server implementations.

WebSphere Partner Gateway is provided in three editions:

- ▶ WebSphere Partner Gateway Express

This edition is specifically designed for the small-and-medium business (SMB) market. It has a small footprint and is easy to use, but it has limited features and deployment options.

- ▶ WebSphere Partner Gateway Advanced

This edition has all the features of WebSphere Partner Gateway. It has a rich set of features and can handle any complexity in your business-to-business environment. Many more protocols and standards are supported for both inbound and outbound communication. Also, WebSphere Partner Gateway Advanced has a document engine that can handle the transformation and validation of documents. The document engine also checks for duplicate documents.

- ▶ WebSphere Partner Gateway Enterprise

This edition has the same features of WebSphere Partner Gateway Advanced. Its main differentiator is that a user of WebSphere Partner Gateway Enterprise is licensed to implement more connections than WebSphere Partner Gateway Advanced.

WebSphere Partner Gateway offers the following features:

- ▶ Applies security management using SSL, public-key cryptography, certificate validation, authentication, AS1, AS2.
- ▶ Document Manager component of WebSphere Partner Gateway provides the mediation capabilities like message routing, transformation and validation. It can also handle digital signature verification.
- ▶ Business-to-business capabilities allow us to specify the type of documents and also specific document attributes that each partner is allowed to send and receive.
- ▶ Non-repudiation repository
- ▶ Partner profile management
- ▶ Customization of document-handling and workflow through *user exits*. User exits can be used to extend or modify an existing document-handling function or to create entirely new ones for transport & business protocol options not offered by WebSphere Partner Gateway.

More information about IBM WebSphere Partner Gateway V6 can be found at:

<http://www.ibm.com/software/integration/wspartnergateway/>

5.2 Development product descriptions

This section describes products discussed and used in development scenarios throughout this book.

5.2.1 IBM Rational Application Developer V6

Rational Application Developer is an integrated development environment with full support for the J2EE programming model including EJB development, Web services, Web applications and Java. In previous releases, this product was known as WebSphere Studio Application Developer. This tool includes integrated portal development, UML editing, code analysis, automated test and deployment tools, built-in version control, and team tools. Everything needed to allow a developer to be productive, and make sure written code is well designed, scalable, and ready for production is included in Rational Application Developer.

Additionally everything is provided for version control and protection when developers work in large teams or on complex projects. Rational Application Developer is optimized for IBM WebSphere software.

Rational Application Developer V6.0 is part of the Rational Software Development Platform used to develop applications to be deployed to IBM WebSphere Application Server V6.0, V5.0.x, and IBM WebSphere Portal V5.0.2.2 and V5.1. The Rational Software Development Platform provides an integrated development environment (IDE) and tooling used to design, develop, test, debug, and deploy applications in support of the application development life cycle.

The IBM Rational Software Development Platform is built upon the IBM Eclipse SDK 3.0, which is an IBM supported version of the Eclipse V3.0 Workbench containing many new features, and a new look and feel. When used with the Rational Software Development Platform, developers can access a broad range of requirements directly from Rational Application Developer for WebSphere software:

- ▶ Rational Web Developer tools for accelerated use of portal, SOA and J2EE
- ▶ Shorten the Java learning curve with use of drag-and-drop components and point-and-click database connectivity
- ▶ Automated tools for applying coding standard reviews, component, and Web service unit testing and multi-tier runtime analysis to help improve code quality
- ▶ Business applications integrated with Web services and SOA

More information can be found at the IBM Rational Application Developer Web site:

<http://www.ibm.com/software/awdtools/developer/application>

5.2.2 IBM WebSphere Studio Application Developer Integration Edition V5.1

IBM WebSphere Studio Application Developer Integration Edition builds on the complete set of functionality offered by WebSphere Studio Application Developer to deliver a development environment designed to deliver on-demand e-business applications by simplifying build-to-integrate tasks, accelerating large-scale application development, and enabling real-time application flexibility for applications that deploy to WebSphere Business Integration Server Foundation V5.1. WebSphere Studio Application Developer Integration Edition V5 is used to develop processes for Business Process Choreographer.

The visual process editor provides intuitive drag-and-drop tools to easily compose and choreograph application interactions and dynamic workflows among J2EE components, Web services, existing applications, and human activities. Developers can quickly and easily build, debug, and deploy complex applications using powerful workflow tools and advanced messaging capabilities to streamline and automate business processes. New services can be added, or existing services modified, without affecting the other components in the business process.

More information can be found at the IBM Web site:

<http://www.ibm.com/software/awdtools/studiointegration/about/>

Extended Enterprise runtime patterns

This section describes the Runtime patterns that are relevant to the Extended Enterprise business pattern including both generic and SOA profiles.

This chapter describes the following Extended Enterprise runtime patterns:

- ▶ Exposed Direct Connection runtime pattern including Message and Call variations
- ▶ Exposed Broker runtime pattern
- ▶ Exposed Router variation
- ▶ Exposed Serial Process runtime pattern
- ▶ Exposed Serial Process Workflow variation

It also provides a description of each of the node types used in the runtime patterns.

6.1 Extended Enterprise runtime patterns

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.

Runtime patterns describe the logical architecture that is required to implement an Application pattern. These patterns depict the major nodes, their roles and the interactions between these nodes.

We can overlay the Application pattern onto the Runtime pattern to identify where business logic is deployed on nodes. The Runtime patterns illustrated give some typical examples of possible solutions, but should not be considered exhaustive.

The Extended Enterprise Runtime patterns are:

- ▶ Exposed Direct Connection: Call variation
- ▶ Exposed Direct Connection: Message variation
- ▶ Exposed Broker
- ▶ Exposed Broker: Exposed Router variation
- ▶ Exposed Serial Process
- ▶ Exposed Serial Process: Workflow variation

To understand the Runtime pattern, review the node definitions provided in 6.2, “Node types” on page 102.

6.1.1 Generic and SOA profiles

Each Extended Enterprise runtime pattern has a generic and an SOA profile.

The generic profile describes the basic implementation of each Runtime pattern in the context of the Extended Enterprise business pattern. The generic profile specifies an infrastructure that can be used by all applications, including services in an SOA.

Note: The Generic profile can be applied when designing solutions for use with an SOA. The distinction between SOA solutions designed using the generic profile and the SOA profile is that the SOA profile introduces and exploits more specific SOA concepts such as an ESB.

The SOA profile describes an infrastructure tailored specifically for services in an SOA. When moving to an SOA, existing applications must first be exposed as services. This allows each service to communicate to other services using SOA techniques as shown in Figure 6-1 on page 101:

1. Service provider publishes service interface to registry.
2. Service consumer retrieves service interface from registry.
3. Service consumer uses the retrieved interface to make calls to the service hosted by the service provider.

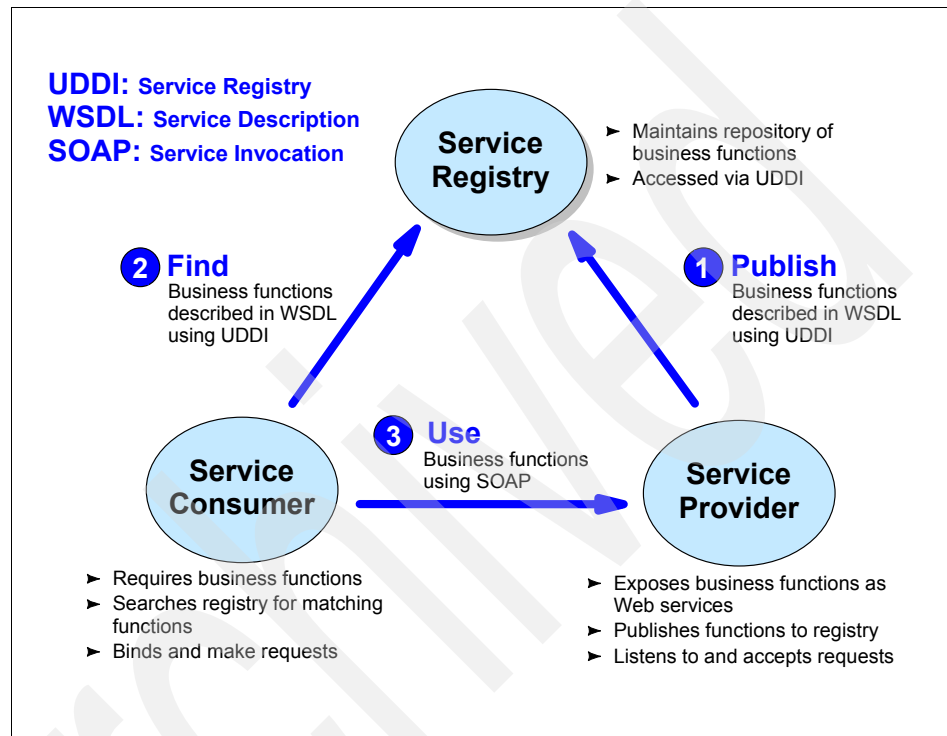


Figure 6-1 Basic Web service interaction model

The generic profile, which covers a wide range of implementations, includes this level of SOA.

The second step in creating an SOA is to build an SOA infrastructure for these services. This includes things like an ESB, an Exposed ESB Gateway for Extended Enterprise services, and Business Service Choreography for choreographing the services. The SOA profile describes how to build this SOA infrastructure. This profile provides a more extensive set of quality of service capabilities over the generic profile, by including support for:

- Multiple messaging styles
- Protocol transformation
- Decoupling of service consumers from service providers
- Namespace transformation

These capabilities provide a more robust and secure implementation of a service-oriented architecture. To achieve these capabilities, the SOA profile for the Extended Enterprise pattern always includes Exposed ESB Gateway and ESB nodes.

6.2 Node types

A Runtime pattern consists of several nodes representing specific functions. Most Runtime patterns consist of a core set of common nodes, with the addition of one or more nodes unique to that pattern. To understand the Runtime pattern, you need to review the following node definitions.

6.2.1 App server/services

The *application server / services* (App Server / Services) 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 it generally does not serve HTTP requests. In other situations, it can be used for business logic only. The application server node supports hosting of Web services applications.

Services can be implemented in a variety of technologies and can be custom-developed enterprise applications, such as those typically implemented in WebSphere Application Server, CICS Transaction Server, IMS Transaction Manager, and software packages.

6.2.2 Network infrastructure

Inter-enterprise network infrastructure includes the network infrastructure allowing connectivity between enterprises. Network infrastructure has unspecified internal characteristics; only the means with which to interact with it is specified.

6.2.3 Protocol firewall

A *firewall* is a hardware/software system that manages the flow of information between the Internet and an organization's private network. Firewalls can prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets, and can block some virus attacks (as long as those viruses are coming from the Internet). A firewall can separate two or more parts of a local network to control data exchange between departments. Components of firewalls include filters or screens, each of which controls transmission of

certain classes of traffic. Firewalls provide the first line of defense for protecting private information, but comprehensive security systems combine firewalls with encryption and other complementary services, such as content filtering and intrusion detection.

Firewalls control access from a less trusted network to a more trusted network. Traditional implementations of firewall services include:

- ▶ Screening routers (the protocol firewall)
- ▶ Application gateways (the domain firewall)

A pair of firewall nodes provides increasing levels of protection at the expense of increasing computing resource requirements. The protocol firewall is typically implemented as an IP router.

6.2.4 Domain firewall

The domain firewall is implemented typically as a dedicated server node.

A *domain* firewall is usually used to separate a secure zone, such as the internal network, from a demilitarized zone. This provides added security protection from the unsecure zone, such as the Internet.

6.2.5 Connector

This node, which is deployed in the demilitarized zone (DMZ) between two firewalls, provides a communication link over the internet for incoming requests from external applications as well as outgoing requests to external services.

It provides connectivity from the Enterprise Secure Zone to the Interenterprise Zone. It might be a low-level component (for example a TCP/IP infrastructure) which is omitted from the runtime pattern diagrams, or it might have more advanced capabilities such as caching of reusable content (for example a Web server).

Depending on the required level of detail, a connector can be:

- ▶ A primitive (or unmodeled) connector, represented by a simple line between components.
- ▶ A component (or modeled) connector, represented by a rectangle on a line between components.

A *connector* can be an adapter connector, a path connector, or both.

Adapter connector

Adapter connectors are concerned with enabling logical connectivity by bridging the gap between the context schema and protocols used by the source and target components. An adapter connector is one that supports the transformation of data and protocols.

Path connector

Path connectors are concerned with providing physical connectivity between components. A path connector can be very complex (for example, the Internet), or very simple (an area of shared storage).

6.2.6 Exposed Connector

The Exposed Connector provides the connectivity between two partner enterprises. A connector is always present to facilitate interaction between two components within these enterprises.

6.2.7 Exposed ESB Gateway

An Exposed ESB Gateway makes the services of one organization available to others, and vice versa, in a controlled and secure manner. Although this might require capabilities such as partner provisioning and management, which are distinct from ESB capabilities, the intent of this component is different from the intent of the ESB, which is to provide a service infrastructure *within* an organization. For both of these reasons, the Exposed ESB Gateway is likely to be integrated with, but not be a part of, the ESB.

6.2.8 ESB

The Enterprise Service Bus (ESB) is a key enabler for an SOA because it provides the capability to route and transport service requests from the service consumer to the correct service provider. The ESB controls routing within the scope of a service namespace.

The true value of the ESB, however, is to enable the infrastructure for SOA in a way that reflects the needs of today's enterprise: to provide suitable service levels and manageability that enable you to operate and integrate in a heterogeneous environment. Furthermore, the ESB needs to be centrally managed and administered and have the ability to be physically distributed.

The following are the minimum set of functions that this node should support:

- Routing

This function removes the need for applications to know anything about the bus topology or its traversal. The interaction that a consumer initiates is sent to one provider.

- Addressing

Addressing complements routing to provide location transparency and support service substitution. Service addresses are transparent to the service consumer and can be transformed by the ESB. The ESB obtains the service address from the namespace directory.

- Messaging styles

The ESB should support at least one or more messaging styles. The most common are request/response, fire and forget, events, publish/subscribe, and synchronous and asynchronous messaging.

- Transport protocols

The ESB should support at least one transport that is or can be made widely available, such as HTTP/S. The ESB can provide protocol transformation. If a protocol transformation is required that is not supported by the ESB, then a specific connector can be used to perform the transformation (see 6.2.6, “Exposed Connector” on page 104).

- Service interface definition

Services should have a formal definition, ideally in an industry-standard format, such as WSDL.

- Service messaging model

The ESB should support at least one model such as SOAP, XML, or a proprietary EAI model.

In addition to these capabilities, the ESB can support more advanced capabilities, such as:

- Integration

Additional integration services that can be provided include service mapping and data enrichment.

- Quality of service

These services can include transaction management (for example, ACID properties, compensation, or WS-Transaction), various assured delivery paradigms (such as WS-ReliableMessaging), or support for Enterprise Application Integration middleware.

- Message processing

The ESB can support more advanced message processing capabilities such as encoded logic, content-based logic, message and data transformations,

message/service aggregation and correlation, validation, intermediaries, object identity mapping, service/message aggregation, and store and forward.

► **Modelling**

The ESB can support more advanced modelling capabilities such as object modelling, common business object models, data format libraries, public versus private models for business-to-business integration, and development and deployment tooling.

► **Service level**

Service level indicators might need to be measured, particularly in an enterprise mission critical environment. The key indicators are availability and performance, which includes response time, throughput, and capacity.

► **Infrastructure intelligence**

More advanced infrastructure capabilities can be provided. These include:

- Business rules
- Policy-driven behavior, particularly for service levels
- Security and quality of service capabilities (WS-Policy)

6.2.9 Rules Directory

This node holds the read-only process execution rules that support the process flow execution. These rules control the sequencing of activities and, therefore, support flow control within the context of an end-to-end process flow. The implementation of this node involves persistent data technologies, such as a flat file or a DBMS.

6.2.10 Directory and Security Services

This node supplies authentication and authorization services. It also holds the user ID and password and related privileges. This node typically leverages LDAP-based directories.

This node contains configuration information needed to support secure and controlled access to services.

6.2.11 Exposed Broker

The Exposed Broker node allows distribution, decomposition, and recomposition of messages so a single interaction from a source component can be switched, split, and joined to multiple target components concurrently. The Exposed Broker separates the application logic from the distribution logic based on broker rules.

The broker also manages the decomposition and recomposition of the interaction using these rules.

The Exposed Broker node exposes external processes to the broker functions within the node. A variation of this would be to use the Exposed Broker to expose internal processes to external partners.

6.2.12 Exposed Router

The Exposed Router node is a variation of the Exposed Broker node. It allows a single interaction from a source component to be switched and adapted to only one of multiple target components. It separates the application logic from the distribution logic based on router rules.

It exposes external processes to the Router functions in the node. A variation of this would be to use the Exposed Router node to expose internal processes to external partners.

6.2.13 Exposed Process Manager

This node contains the process flow execution engine. It provides the capability for model-driven business process automation. It also enables tracking by leveraging the process execution rules stored in the associated database.

These processes can span multiple applications and organizational boundaries. The node maintains state and tracks sequencing through the process flow. In doing so, it often leverages the associated repository to store intermediate results.

This node is also responsible for invoking target applications as necessary through their associated connectors.

The Process Manager node can support serial processes in which there is a sequential execution of process steps and parallel processes where process steps or sub-processes can execute concurrently.

The Process Manager should support the following key capabilities:

- ▶ Process definition standards, such as WS-BPEL, and the ability to execute process definitions that have been defined and exported from a modelling tool
- ▶ Monitoring and analysis of processes by capturing information about process execution for historical analysis

It should also support integration with system management and administration tools.

- ▶ Ability to meet non-functional requirements such as performance, availability and scalability will be important for mission critical enterprise applications
Other key non-functional requirements are security and transaction management particularly supporting the integrity and recovery of long running business processes.
- ▶ Multiple levels of process abstractions.
- ▶ Correlation of events or incoming messages with existing process instances.
- ▶ Support for branching, parallel branch execution and recomposing if the process manager supports parallel process execution.

The Exposed Process Manager exposes external processes to the process flow execution engine in the node. A variation of this would be to use the Exposed Process Manager node to expose internal serial processes to external partners.

6.2.14 Business Service Choreography

The Business Service Choreography node executes the business process flow logic that governs the sequence and control of service invocations. The business process is controlled centrally and is not part of the program logic in individual applications. As a result, the business process can be modelled and implemented centrally. The Business Service Choreography node facilitates the implementation of changes to the business process and monitoring and analysis of business process execution.

The Business Service Choreography node provides the same support for model-driven business process automation as the Exposed Process Manager node, but in this instance the responsibility for routing and transporting service requests is the responsibility of the ESB. The Exposed ESB Gateway provides support for secure Extended Enterprise interactions. An external partner has no direct interaction with the business processes available on the node.

6.2.15 Staff Worklist Adapter

A specialized Staff Worklist Adapter is responsible for presenting the work items to be executed by a particular person or a department. This adapter enables human interaction within automated business processes. It is the primary interface through which the humans interact within the end-to-end workflow.

6.3 Exposed Direct Connection runtime pattern

When using the Exposed Direct Connection runtime pattern, the source application uses a connector to access the target application. This allows a single interaction from the source application to be adapted and transported to one partner target application.

6.3.1 Generic profile

This Runtime pattern allows two different organizations to talk to each other with a mutually agreed upon message format and protocol. Each partner can use their own internal messaging format, then use a connector adapter to convert from the internal format to the external format.

The connector itself can be explicitly or implicitly modeled. If the connector is explicitly modeled, the modeler can use decomposition and abstraction techniques to expand the connector to the appropriate level of detail.

The term *Connector* can be qualified by both the connector variation and by the interaction variation. Some examples are:

- ▶ Adapter Connector
- ▶ Path Connector
- ▶ Message Connector
- ▶ Call Connector
- ▶ Call Adapter Connector

The target application relies on services provided by its hosting server. These are modeled using the Application Server/Services component.

The Rules Directory might or might not exist. If it does exist, it is a modeling decision as to whether the rules need to be shown in the Runtime pattern. For example, analysis might determine that connection rules are not an important part of the solution, so the Rules Directory might be left off the Runtime pattern.

The Directory and Security Services node supplies authentication and authorization services. It also holds the user ID as well as password and related privileges. This node typically leverages LDAP-based directories. It also contains configuration information needed to support secure access between the enterprise and partner services.

The generic profile of the Exposed Direct Connection runtime pattern is shown in Figure 6-2 on page 110.

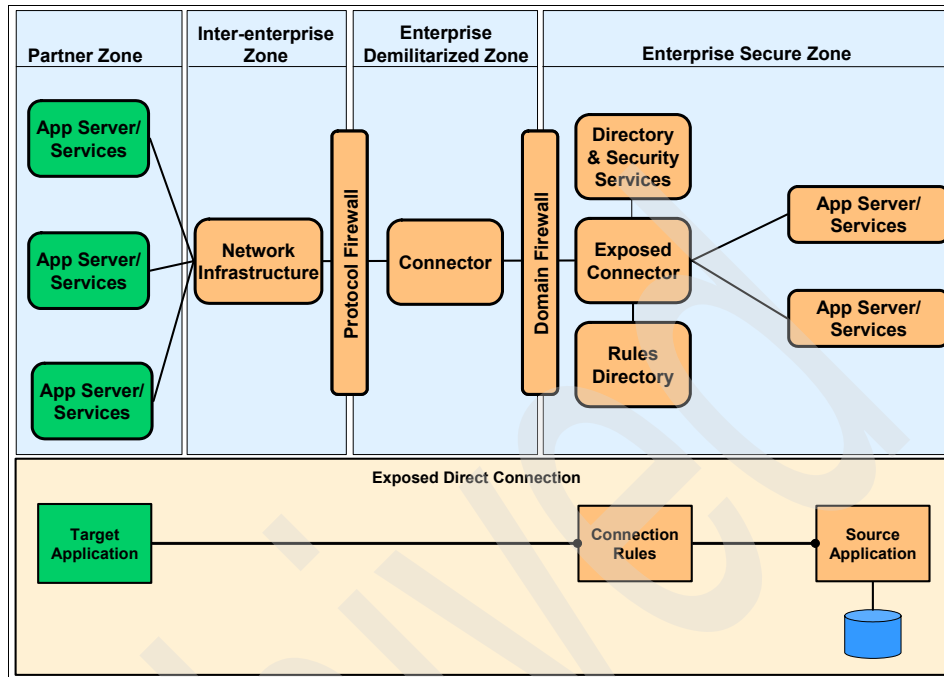


Figure 6-2 Exposed Direct Connection runtime pattern: generic profile

Figure 6-2 shows a standard pattern of Path Connectors (firewalls and network infrastructure), but other variations do exist with fewer or more firewalls.

The secure zone Connector is primarily concerned with logical connection of the Path Connector to the Application Services, and will therefore often be modeled as an Adapter Connector.

Less secure applications and connectors can be placed within the Demilitarized Zone, depending on local security policies. The less secure applications are usually placed as shown in Figure 6-2.

We do not have separate Runtime patterns for the message and call variations of the Exposed Direct Connection application pattern. It is still important to identify that your business scenario requires a message or call application pattern because you can use this knowledge as a consideration when selecting a Product mapping.

6.3.2 SOA profile

In the SOA profile for the Exposed Direct Connection runtime pattern, the Exposed Connector, Rules Directory, and Partner Infrastructure are specialized as shown in Figure 6-3 to instead become:

- ▶ An Exposed ESB Gateway
- ▶ An ESB
- ▶ Service Consumers and Providers

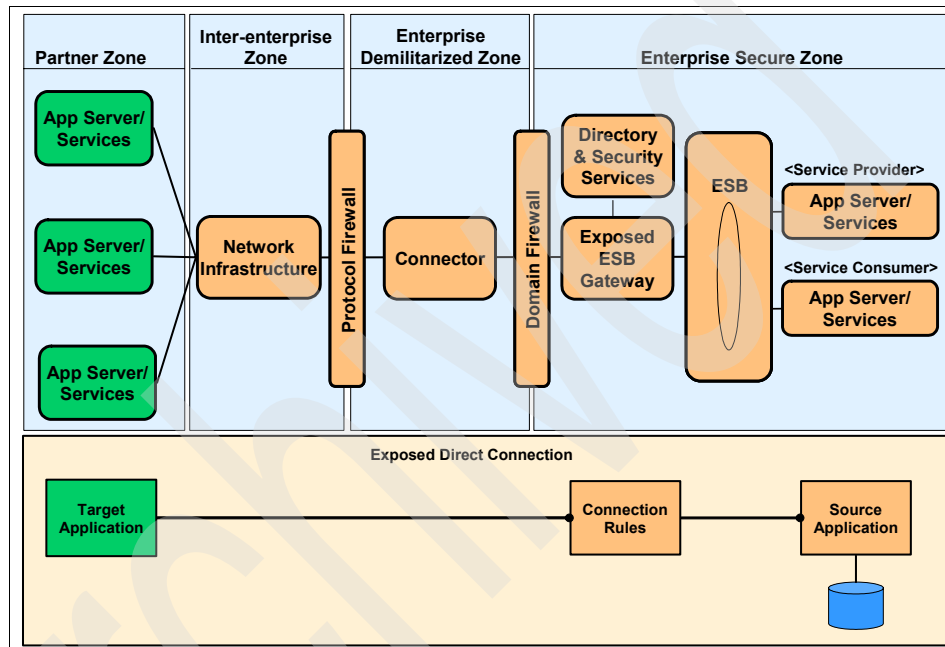


Figure 6-3 Exposed Direct Connection runtime pattern: SOA profile

The Exposed ESB Gateway provides support for the *exposed* requirement of the Extended Enterprise pattern. It provides a single point of access between:

- ▶ External Service Consumers and Service Providers in the Enterprise Secure Zone
- ▶ Service Consumers in the Enterprise Secure Zone and external Service Providers

The Exposed ESB Gateway secures the connection between enterprises and provides namespace mapping.

The ESB meets the Connector and Rules Directory requirement of the Direct Connection runtime pattern generic profile, but also gives support for the SOA infrastructure by providing for service location transparency and interoperability,

encapsulated reusable business function and explicit implementation- independent interfaces within the enterprise.

Service Consumers and Service Providers replace the Application Server and Services from the generic profile.

6.4 Exposed Broker runtime pattern

The Exposed Broker runtime pattern applies to solutions where the source application starts an interaction that is distributed to multiple target applications across organizational boundaries. It allows a single interaction from a partner's source application to be distributed to multiple target partner applications concurrently. It separates the application logic from the distribution logic based on broker rules from the Rules Directory.

The Exposed Broker pattern reuses the Exposed Direct Connection pattern to provide connectivity between the tiers. The Broker Rules tier can support Message variation or Call variation (or both variations) of the Exposed Direct Connection pattern.

6.4.1 Generic profile

The Broker Rules tier of the application pattern is implemented in this Runtime pattern with an Exposed Broker node as shown in Figure 6-4 on page 113. The Exposed Broker node is responsible for the routing and distribution of incoming or outgoing messages to the target applications. It has the ability to decompose and recompose messages. It also includes functionality to include external partners by exposing their processes to internal processes.

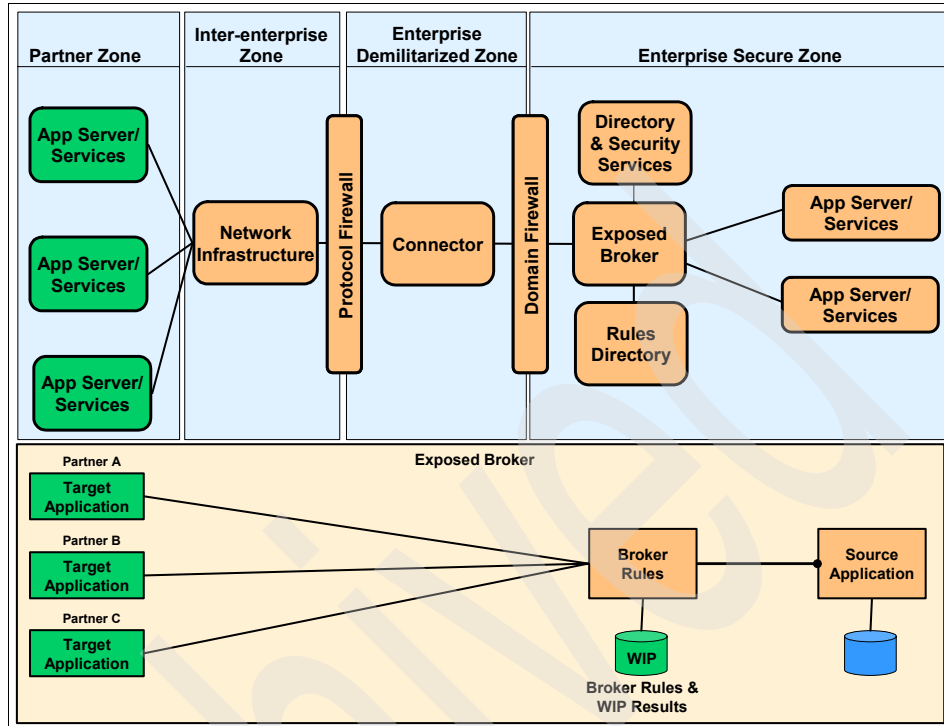


Figure 6-4 Exposed Broker runtime pattern: generic profile

The Directory and Security Services supplies authentication and authorization services. It also holds the user ID and password and related privileges. This node typically leverages LDAP-based directories. It also contains configuration information needed to support secure access between the enterprise and partner services.

The Application Server and Services nodes execute the logic of the target and source applications.

6.4.2 SOA profile

In the SOA profile for the Exposed Broker runtime pattern, the Exposed Broker, Rules Directory and Partner Infrastructure are specialized as shown in Figure 6-5 on page 114 to instead become:

- ▶ an Exposed ESB Gateway
- ▶ an ESB
- ▶ Service Consumers and Providers

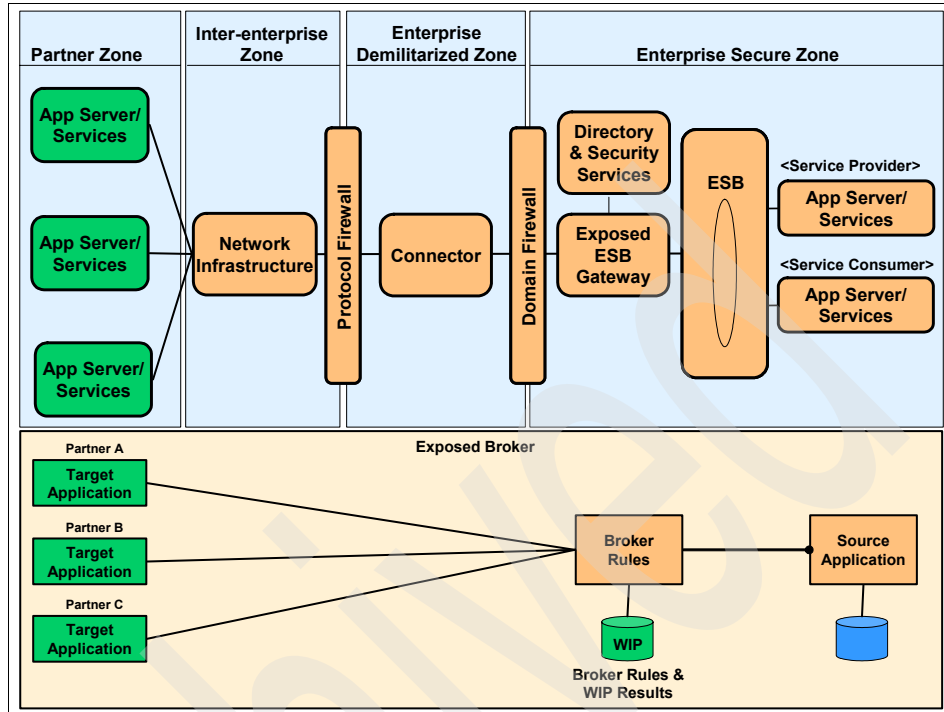


Figure 6-5 Exposed Broker runtime pattern: SOA profile

The Exposed ESB Gateway provides support for the *exposed* requirement of the Extended Enterprise pattern. It provides a single point of access between:

- ▶ External Service Consumers and Service Providers in the Enterprise Secure Zone
- ▶ Service Consumers in the Enterprise Secure Zone and external Service Providers

The Exposed ESB Gateway secures the connection between enterprises and provides namespace mapping.

The ESB meets the broker and rules directory requirement of the Runtime pattern generic profile, but also gives support for the SOA infrastructure by providing for service location transparency and interoperability, encapsulated reusable business function and explicit implementation-independent interfaces within the enterprise.

Service Consumers and Service Providers replace the Application Server/Services in the generic profile.

6.5 Exposed Router variation

The Exposed Router variation of the Exposed Broker runtime pattern applies to solutions where the partner's source application initiates an interaction that is forwarded to, at most, one of multiple target applications. The selection of the target application is controlled by the distribution rules that govern functioning of the connector component.

6.5.1 Generic profile

In the Exposed Router variation shown in Figure 6-6, the Exposed Router node provides the logic to perform intelligent routing of messages to one target application at a time. It does not include the simultaneous distribution or decomposition capabilities that the Exposed Broker node provides.

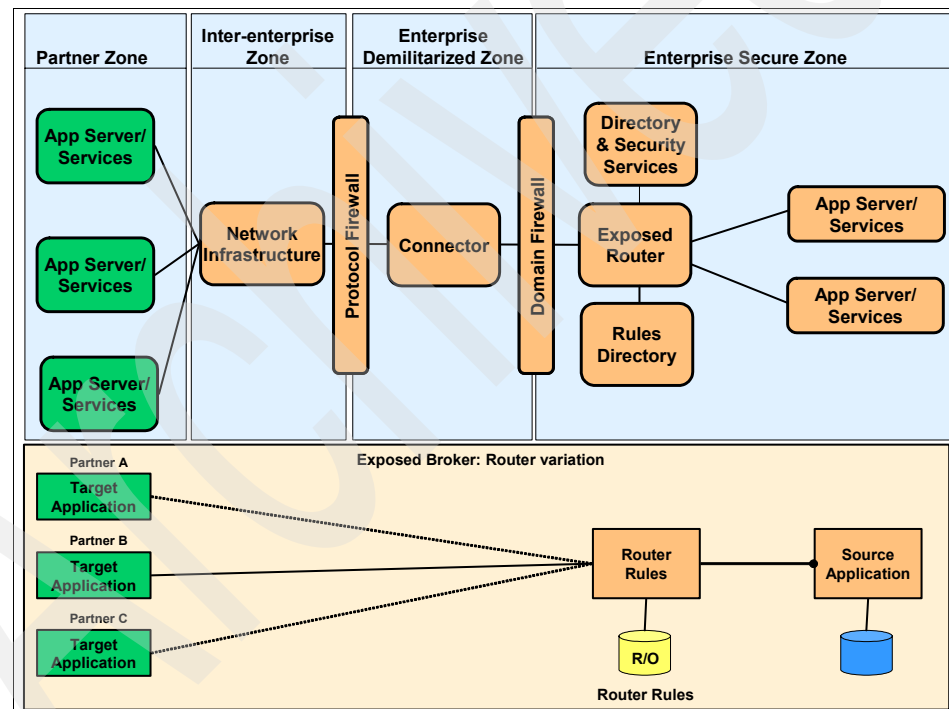


Figure 6-6 Exposed Router runtime pattern: generic profile

The Exposed Router variation of the Exposed Broker application pattern can be thought of as an extension to the Router variation of the Broker application pattern. It also includes functionality to include external partners in a solution by exposing their processes to internal processes.

The Directory and Security Services supplies authentication and authorization services. It also holds the user ID and password and related privileges. This node typically leverages LDAP-based directories. It also contains configuration information needed to support secure access between the enterprise and partner services.

The Application Server/Services nodes execute the logic of the target and source applications.

6.5.2 SOA profile

In the SOA profile for the Exposed Router runtime pattern, the Exposed Router, Rules Directory, and Partner Infrastructure are specialized as shown in Figure 6-7 to instead become:

- An Exposed ESB Gateway
- An ESB
- Service Consumers and Providers

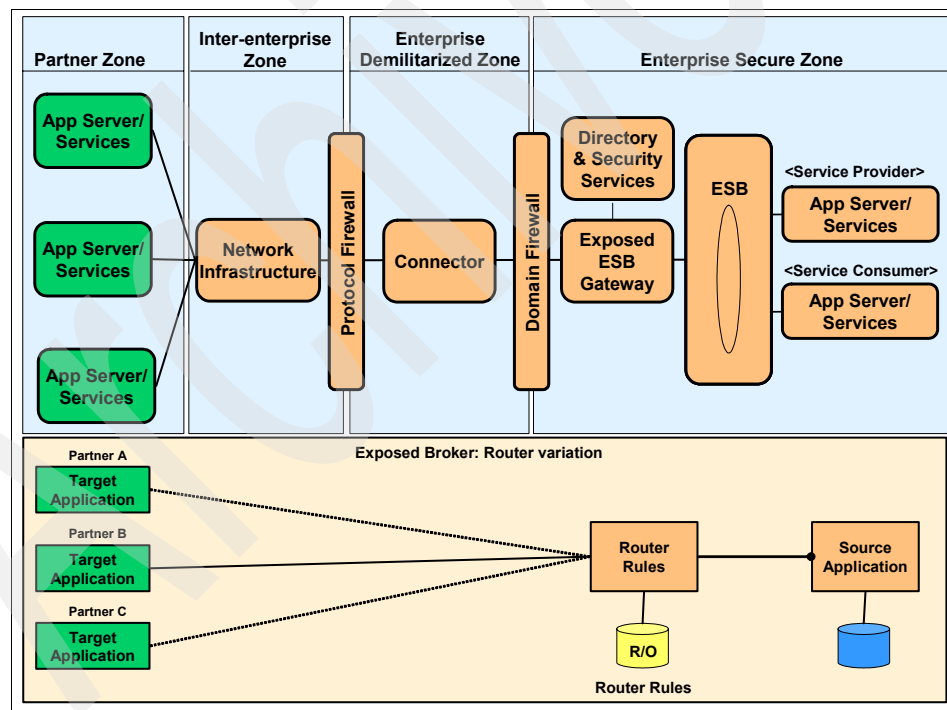


Figure 6-7 Exposed Router runtime pattern: SOA profile

The Exposed ESB Gateway provides support for the *exposed* requirement of the Extended Enterprise pattern. It provides a single point of access between:

- ▶ External Service Consumers and Service Providers in the Enterprise Secure Zone
- ▶ Service Consumers in the Enterprise Secure Zone and external Service Providers

The Exposed ESB Gateway secures the connection between enterprises and provides namespace mapping.

The ESB meets router and rules directory requirement of the Runtime pattern generic profile, but also gives support for the SOA infrastructure by providing for service location transparency and interoperability, encapsulated reusable business function and explicit implementation-independent interfaces within the enterprise.

Service Consumers and Service Providers replace the Application Server and Services in the generic profile.

6.6 Exposed Serial Process runtime pattern

The Exposed Serial Process runtime pattern applies to solutions where a single interaction from the partner's source application executes a sequence of target applications based on process logic. This process logic is separated from the application logic. The process logic is governed by serial process rules that define execution rules for each target application, together with control flow and data flow rules. It can also include any necessary adapter rules.

6.6.1 Generic profile

In the Exposed Serial Process runtime pattern, the Exposed Process Manager node shown in Figure 6-8 on page 118 contains the process flow execution engine which provides the capability for model-driven business process automation. It also enables tracking by leveraging the process execution rules stored in the associated database.

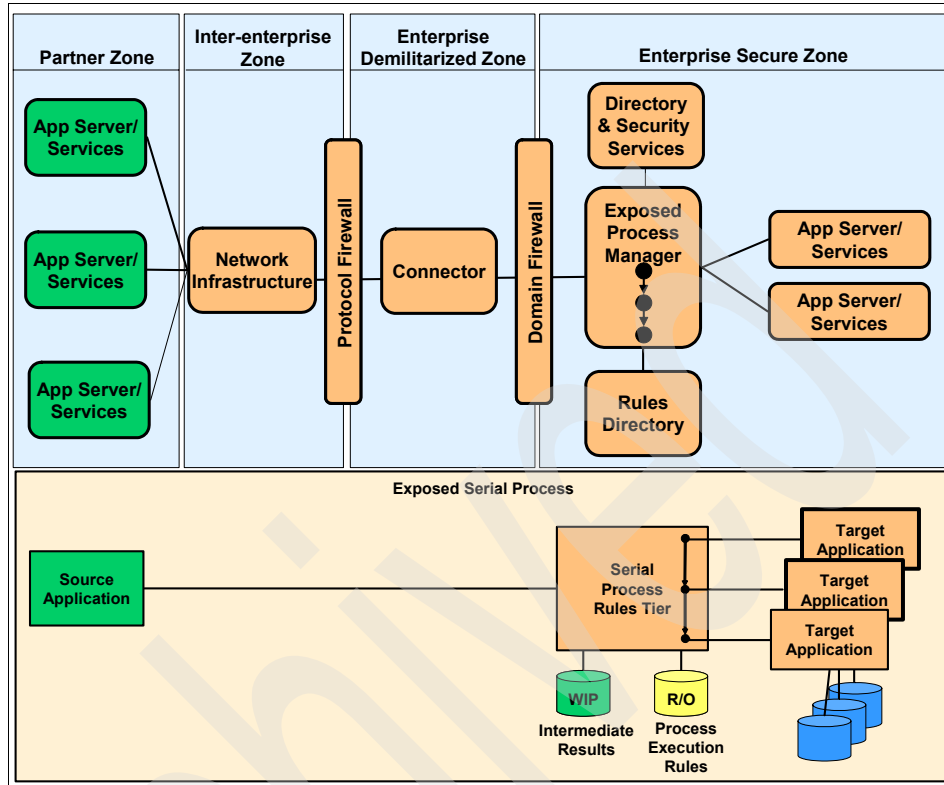


Figure 6-8 Exposed Serial Process runtime pattern: generic profile

These processes can span multiple applications and organizational boundaries within an enterprise. The node maintains state and tracks sequencing through the process flow. In doing so, it often leverages the associated repository to store intermediate results. This node is also responsible for invoking target applications as necessary through their associated connectors.

The Directory and Security Services supplies authentication and authorization services. It also holds the user ID and password and related privileges. This node typically leverages LDAP-based directories. It also contains configuration information needed to support secure access between the enterprise and partner services.

The Application Server and Services nodes execute the logic of the target and source applications.

6.6.2 SOA profile

In the SOA profile for the Exposed Serial Process runtime pattern, the Exposed Process Manager, Rules Directory, and Partner Infrastructure are specialized as shown in Figure 6-9 to instead become:

- ▶ An Exposed ESB Gateway
- ▶ An ESB
- ▶ A Business Service Choreography node
- ▶ Service Consumers and Providers

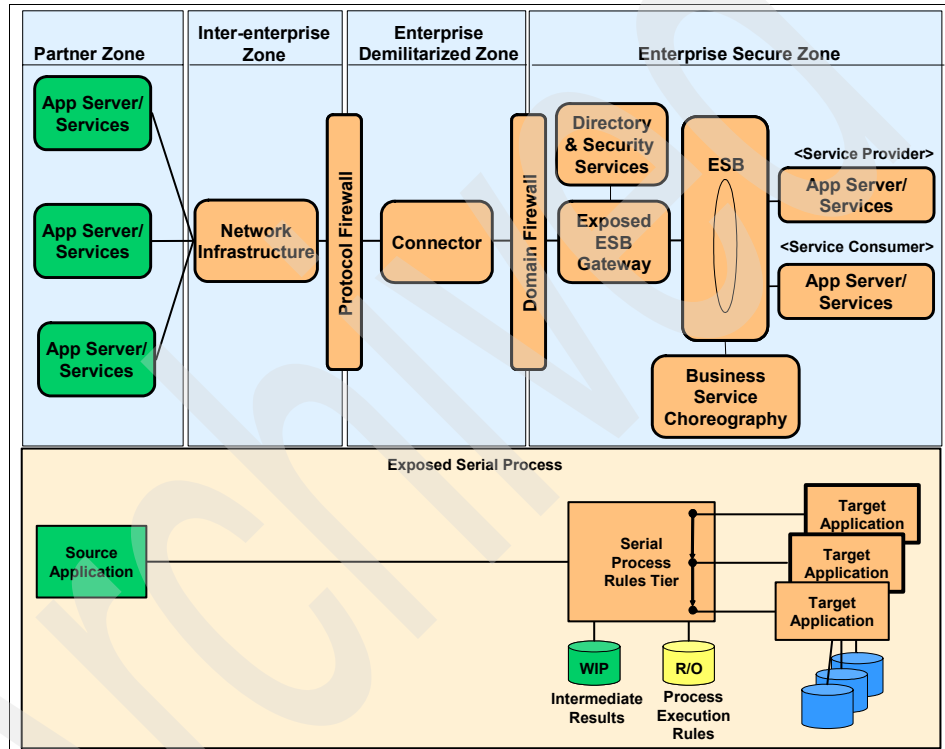


Figure 6-9 Exposed Serial Process runtime pattern: SOA profile

The Exposed ESB Gateway provides support for the *exposed* requirement of the Extended Enterprise pattern. It provides a single point of access between:

- ▶ External Service Consumers and Service Providers in the Enterprise Secure Zone
- ▶ Service Consumers in the Enterprise Secure Zone and external Service Providers

The Exposed ESB Gateway secures the connection between enterprises and provides namespace mapping.

The ESB gives support for the SOA infrastructure by providing for service location transparency and interoperability, encapsulated reusable business function and explicit implementation-independent interfaces within the enterprise.

The Business Service Choreography node provides support for the Process Manager requirement of the Serial Process pattern.

Service Consumers and Service Providers replace the Application Server/Services in the generic profile.

6.7 Exposed Serial Workflow variation

The Serial Workflow variation of the Serial Process runtime pattern extends the basic serial process orchestration capability by supporting human interaction for completing certain process steps.

6.7.1 Generic profile

The Serial Workflow variation of the Runtime pattern shown in Figure 6-10 on page 121 provides specialized support for tasks requiring human interaction within the process logic. When a particular task requires human interaction, the Exposed Process Manager node creates a work item, identifies a particular person or a department responsible for executing that task, and adds the work item to its worklist.

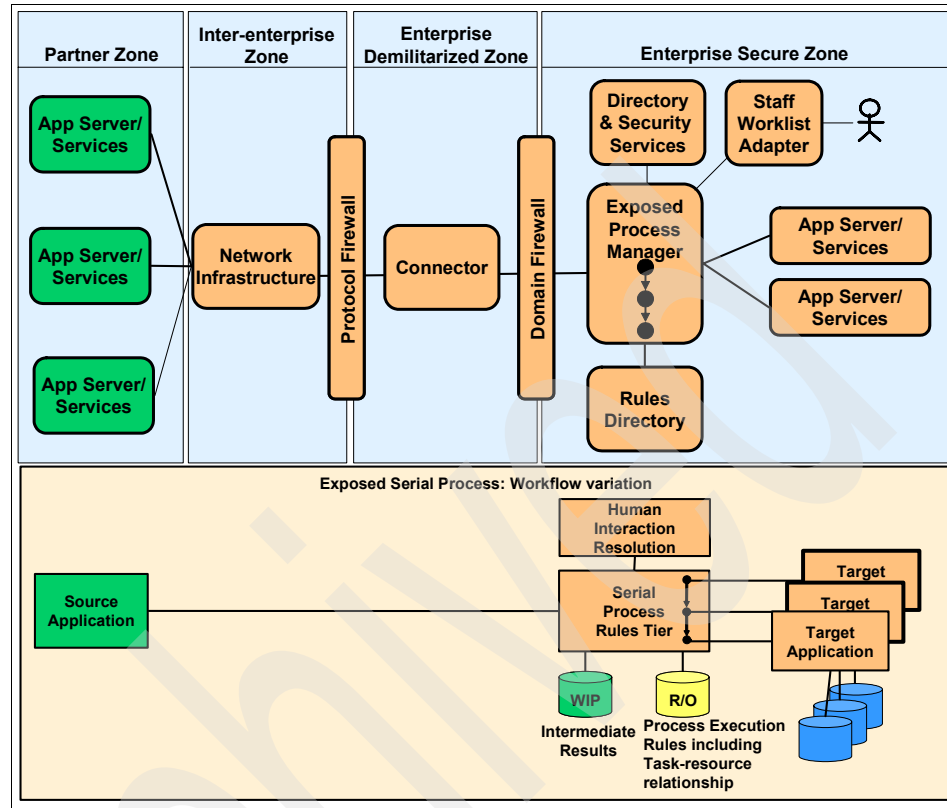


Figure 6-10 Exposed Serial Workflow runtime pattern: generic profile

The Directory and Security Services supplies authentication and authorization services. It also holds the user ID and password and related privileges. This node typically leverages LDAP-based directories. It also contains configuration information needed to support secure access between the enterprise and partner services.

The Staff Worklist Adapter then presents the work items to be executed by a particular person or a department.

6.7.2 SOA profile

In the SOA profile for the Exposed Serial Workflow runtime pattern, the Exposed Process Manager, Rules Directory and Partner Infrastructure are specialized as shown in Figure 6-11 on page 122 to instead become:

- An Exposed ESB Gateway
- An ESB
- A Business Service Choreography node
- Service Consumers and Providers

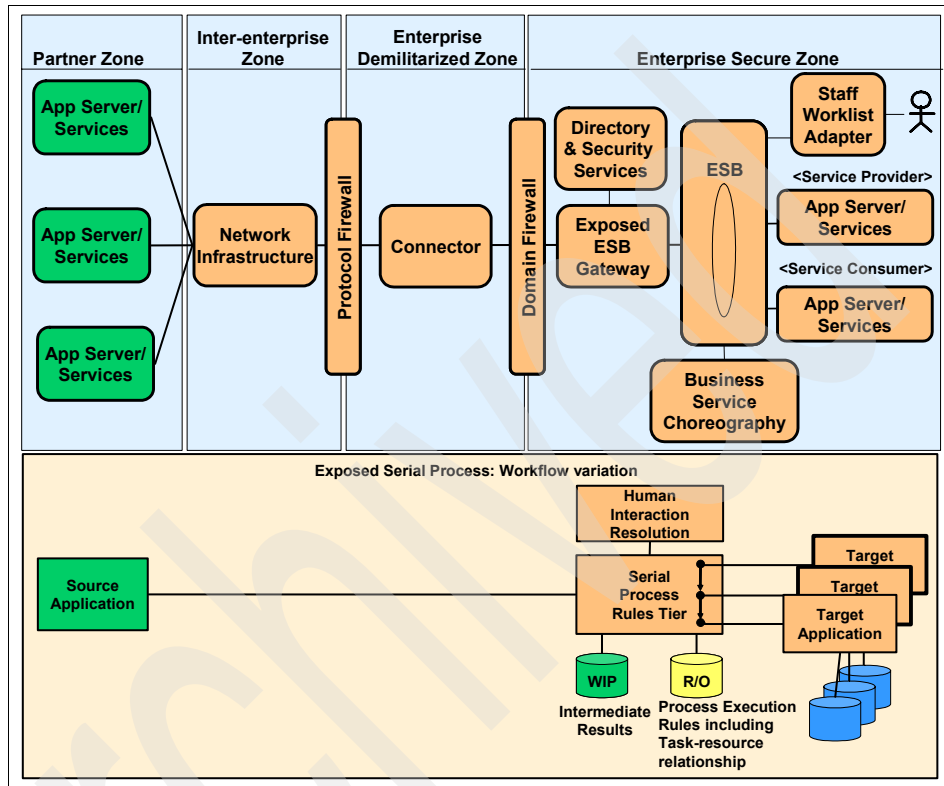


Figure 6-11 Exposed Serial Process Workflow runtime pattern: SOA profile

The Exposed ESB Gateway provides support for the *exposed* requirement of the Extended Enterprise pattern. It provides a single point of access between:

- External Service Consumers and Service Providers in the Enterprise Secure Zone
- Service Consumers in the Enterprise Secure Zone and external Service Providers

The Exposed ESB Gateway secures the connection between enterprises and provides namespace mapping.

The ESB gives support for the SOA infrastructure by providing for service location transparency and interoperability, encapsulated reusable business function and explicit implementation-independent interfaces within the enterprise.

The Business Service Choreography node provides support for the Process Manager requirement of the Serial Workflow pattern.

Service Consumers and Service Providers replace the Application Server/Services in the generic profile.

Archived

Product mappings

This chapter provides Product mappings for the following Runtime patterns:

- ▶ Exposed Direct Connection runtime pattern
 - Generic profile
 - SOA profile
- ▶ Exposed Broker runtime pattern
 - Generic profile
- ▶ Exposed Router variation
 - SOA profile
- ▶ Exposed Serial Process runtime pattern
 - Generic profile
 - SOA profile

You can find an overview of the products used in these Product mappings in Chapter 5, “Product descriptions” on page 87.

7.1 Product mappings

After choosing a Runtime pattern, you need to determine the products and platforms that you will use. The Product mappings in this section are suggested mappings and address the scenario implementations that Part 3, “Scenario implementation” starting on page 155 of this book discusses. These Product mappings are also typical Product mappings used for production systems.

We suggest that you make the final product selection decisions based on your particular non-functional requirements, such as volumetric data, performance, availability, scalability, security, manageability, and supportability. You typically define these non-functional requirements during the solution analysis process.

Other considerations that influence the product selection include:

- ▶ Specific technology and product standards
- ▶ Existing systems and platform investments
- ▶ Existing development skills

Each scenario highlights the interoperability benefits of Web services technology with each of the external partner applications being developed and hosted using a different technical environment:

- ▶ WebSphere Application Server V6.0.2
- ▶ Microsoft .NET
- ▶ CICS Transaction Server V3.1

Note: The Product mappings in this section do not include hardware nodes and operating systems. The sample scenarios in Part 3, “Scenario implementation” starting on page 155 of this book were implemented on xSeries® servers running the Windows 2000 operating system.

7.2 Exposed Direct Connection product mapping

Our sample scenario illustrates use of both the Call and Message variations of the Exposed Direction Connection pattern. One-way and two-way intra enterprise requests are used between components within the Enterprise Secure Zone. Component interactions are performed with Web services using the SOAP/HTTP protocol.

The Extended Enterprise interaction uses the Message variation of the Exposed Direct Connection pattern, with the external partner responding to acknowledge that the request has been processed. These Web service interactions also use the SOAP/HTTP protocol.

7.2.1 Generic profile

This Product mapping uses WebSphere Application Server V6.0.2 to host the Application Services in the Enterprise Secure Zone as shown in Figure 7-1.

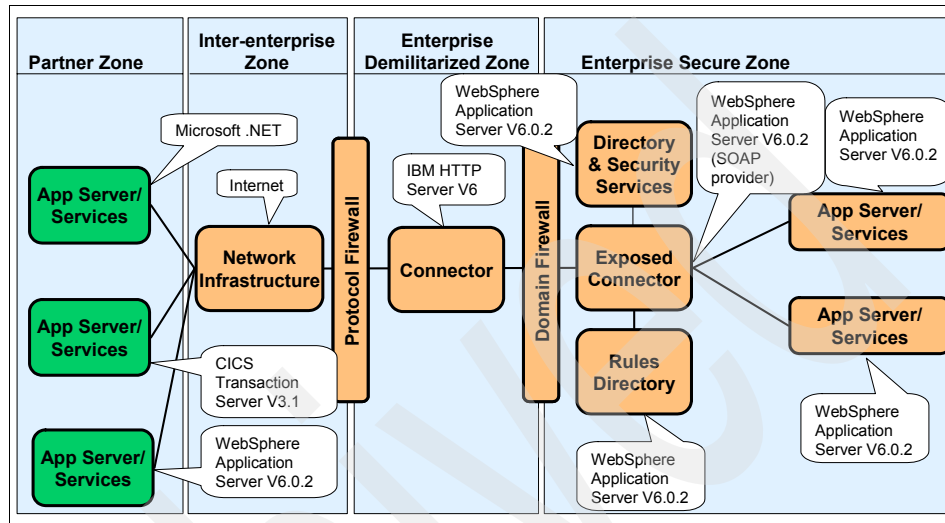


Figure 7-1 Exposed Direct Connection: generic profile product mapping

WebSphere Application Server V6.0.2 acts as an Exposed Connector by performing as a SOAP provider.

The Rules Directory node is implemented by WebSphere Application Server V6.0.2. WebSphere Application Server V6.0.2 allows you to override the Endpoint URL in the Web services client binding for installed Web and EJB modules providing the ability to update service locations at runtime.

The Directory and Security services node provides support for WS-Security as the application code for the Application Services in the Enterprise Secure Zone has been developed, so that all interactions include support for WS-Security integrity and confidentiality.

The IBM HTTP Server V6 acts as an Adapter Connector by providing protocol transformation from SOAP/HTTP to SOAP/HTTPS thus delivering transport level security between the enterprise and the partner organizations.

7.2.2 SOA profile

The Direct Connection SOA profile shown in Figure 7-3 introduces the ESB and Exposed ESB Gateway components which are central to all the SOA profiles.

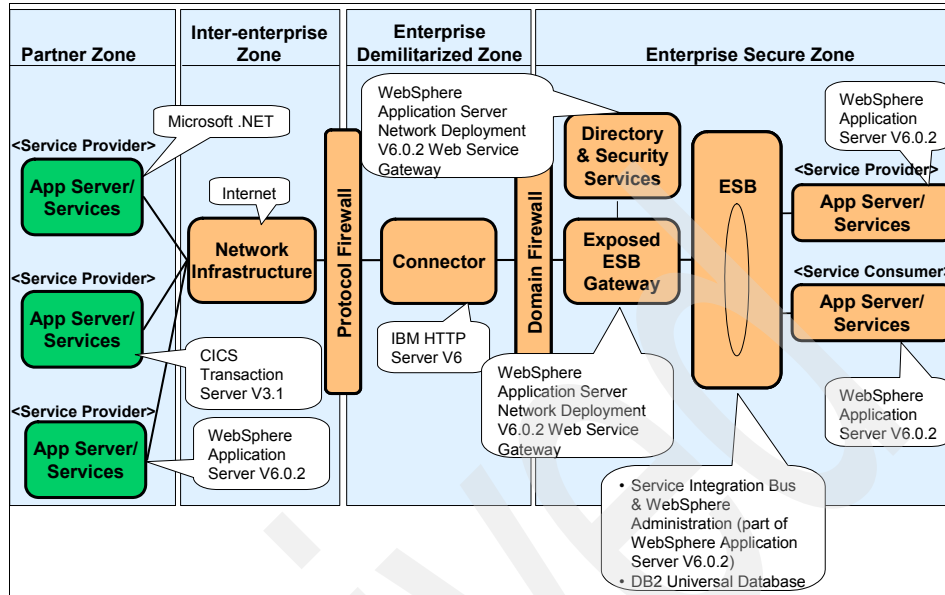


Figure 7-2 Exposed Direct Connection: SOA profile product mapping

This Product mapping uses WebSphere Application Server V6.0.2 to host the Application Services in the Enterprise Secure Zone.

The ESB is run in a service integration bus within WebSphere Application Server Network Deployment V6.0.2, providing service location transparency between Service Consumers and Service Providers within the enterprise. With the Network Deployment offering, you can implement a scalable clustering of multiple WebSphere Application Servers.

The Web services gateway provided with WebSphere Application Server Network Deployment V6.0.2 is the Exposed ESB Gateway in our Product mapping. It is used to provide a standard, consistent interface for the internal processes that access external processes. Using an Exposed ESB Gateway minimizes the disruption caused by changes in the external partner infrastructure.

The Directory and Security services node is configured to secure all transactions to the external Partner Zone to use WS-Security integrity and confidentiality. In this scenario, the Application Services in the Enterprise Secure Zone do not include support for WS-Security. WebSphere Application Server Network Deployment V6.0.2 allows you to configure a service integration bus to use WS-Security to secure the SOAP messages that pass between the Service

Consumer and the target partner Service Provider. Interactions within the enterprise will not be secured.

A local DB2 Universal Database database is used to store the SDO (Service Data Object) repository.

The IBM HTTP Server V6 acts as an Adapter Connector by providing protocol transformation to SOAP/HTTPS thus delivering transport level security between the enterprise and the partner organizations.

7.3 Exposed Broker product mapping

For our Exposed Broker scenarios, the responsibility for implementing the business rules to identify which external partner organization to communicate with has been implemented with an Exposed Broker or Exposed Router node.

7.3.1 Exposed Broker: Generic profile

Figure 7-3 illustrates the Product mapping for the Exposed Broker. The Exposed Broker includes the means to expose partner processes to internal processes.

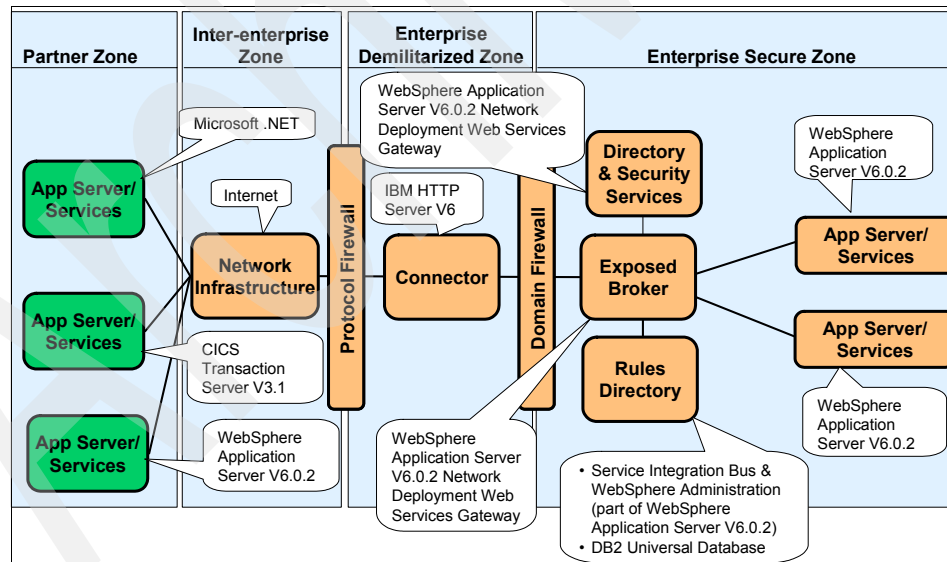


Figure 7-3 Exposed Broker: generic profile product mapping

This Product mapping uses WebSphere Application Server V6.0.2 to host the Application Services in the Enterprise Secure Zone.

The Exposed Broker node is implemented with the Web services gateway that is part of WebSphere Application Server Network Deployment V6.0.2, which supports the exposed nature of this node by providing a standard, consistent interface for the internal processes to access external processes.

The Rule Directory node is implemented using the service integration bus mediation support within WebSphere Application Server Network Deployment V6.0.2.

In the Directory and Security services node, the Web services gateway security will be configured for all transactions to the external Partner Zone to use WS-Security integrity and confidentiality. In this scenario, the Application Services in the Enterprise Secure Zone do not include support for WS-Security. WebSphere Application Server Network Deployment V6.0.2 allows you to configure a service integration bus to use WS-Security to secure the SOAP messages that pass between the Service Consumer and the target partner Service Provider. Interactions within the enterprise will not be secured.

A local DB2 Universal Database database is used to store the SDO repository.

The IBM HTTP Server V6 acts as an Adapter Connector by providing protocol transformation to SOAP/HTTPS, thus delivering transport level security between the enterprise and the partner organizations.

7.3.2 Exposed Router variation: SOA profile

Figure 7-4 on page 131 illustrates the Product mapping for the Exposed Router variation. The Router logic is provided by the ESB node performing intelligent routing of messages to one target application at a time. It does not include the simultaneous distribution or decomposition capabilities that the Broker node provides.



The ESB is run in a service integration bus within WebSphere Application Server Network Deployment V6.0.2, providing service location transparency between Service Consumers and Service Providers within the enterprise. With the Network Deployment offering, you can implement a scalable clustering of multiple WebSphere Application Server servers.

The WebSphere Partner Gateway acts as the Exposed ESB Gateway node providing a standard, consistent interface for the internal processes to access external processes. An Exposed ESB Gateway minimizes the disruption caused by changes in the external partner infrastructure.

The IBM HTTP Server V6 acts as an Adapter Connector by providing protocol transformation to SOAP/HTTPS thus delivering transport level security between the enterprise and the partner organizations.

7.4 Exposed Serial Process product mapping

The Product mappings documented in this section for Serial Process patterns and their workflow variations primarily focus on the key nodes (the Exposed Process Manager and Business Service Choreography nodes) in the appropriate Runtime patterns.

7.4.1 Generic profile

In this Product mapping for the Serial Process runtime pattern, shown in Figure 7-5, the Exposed Process Manager node is implemented using WebSphere Business Integration Server Foundation V5.1.

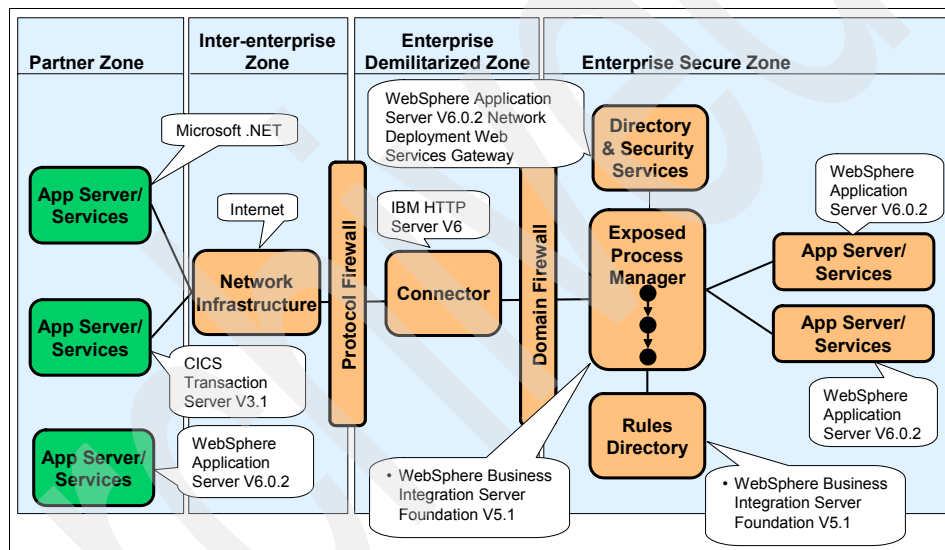


Figure 7-5 Exposed Serial Process: generic profile product mapping

This Product mapping uses WebSphere Application Server V6.0.2 to host the Application Services in the Enterprise Secure Zone.

An application service uses the Web Services Invocation Framework (WSIF) to invoke the automated process instance implemented by the Process Manager node. The Exposed Process Manager implemented using WebSphere Business Integration Server Foundation V5.1 invokes the external partner application services.

The Rules Directory node implemented using WebSphere Business Integration Server Foundation V5.1 identifies which external partner organization application service to invoke.

In the Directory and Security services node, the service integration bus within WebSphere Application Server Network Deployment V6.0.2 is configured secure all transactions to the external Partner Zone to use WS-Security integrity and confidentiality.

The IBM HTTP Server V6 acts as an Adapter Connector by providing protocol transformation to SOAP/HTTPS thus delivering transport level security between the enterprise and the partner organizations.

7.4.2 SOA profile

The SOA mapping shown in Figure 7-6 shares many of the same characteristics as the Generic profile. The Process Manager functionality is replaced with the Business Service Choreography node, although this node is still implemented by WebSphere Business Integration Server Foundation V5.1.

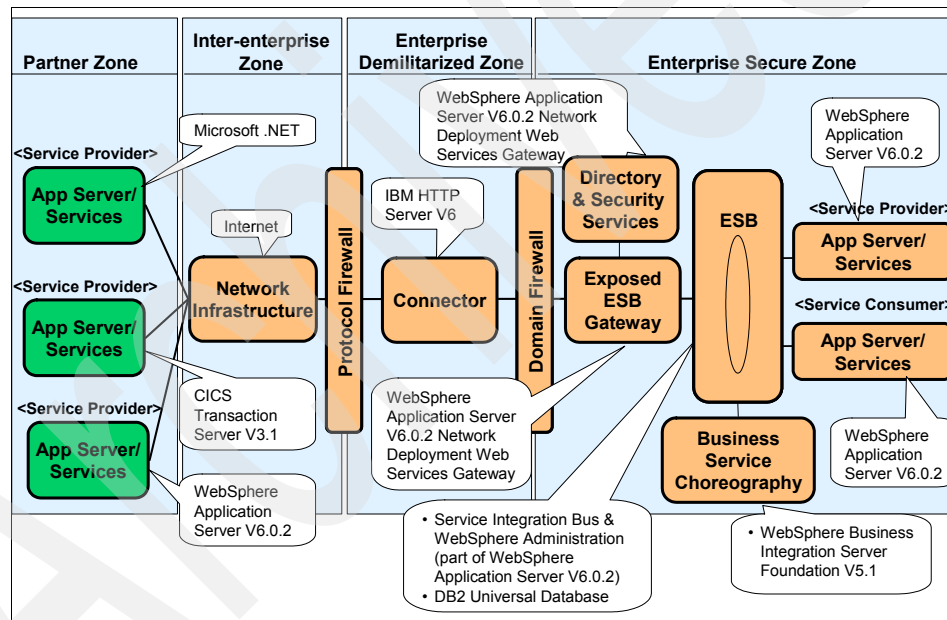


Figure 7-6 Exposed Serial Process: SOA profile product mapping

This Product mapping uses WebSphere Application Server V6.0.2 to host the Application Services in the Enterprise Secure Zone.

The ESB is run as a service integration bus within WebSphere Application Server Network Deployment V6.0.2, providing service location transparency between Service Consumers and Service Providers within the enterprise. With

the Network Deployment offering, you can implement a scalable clustering of multiple WebSphere Application Server servers.

An application service uses the ESB to invoke the Business Service Choreography node's automated process instance by the using the Web Services Invocation Framework (WSIF). The Business Service Choreography node is implemented using WebSphere Business Integration Server Foundation V5.1.

A local DB2 Universal Database database is used to store the SDO repository.

The Web services gateway provided with WebSphere Application Server Network Deployment V6.0.2 is the Exposed ESB Gateway in our Product mapping. It is used to provide a standard, consistent interface for the internal processes to access external processes. Using an Exposed ESB Gateway minimizes the disruption caused by changes in the external partner infrastructure.

In the Directory and Security services node, the service integration bus within WebSphere Application Server Network Deployment V6.0.2 is configured secure all transactions to the external Partner Zone to use WS-Security integrity and confidentiality.

The IBM HTTP Server V6 acts as an Adapter Connector by providing protocol transformation to SOAP/HTTPS, thus delivering transport level security between the enterprise and the partner organizations.



Part 2

Business scenario and guidelines

Archived

Business scenario used in this book

Part 3, “Scenario implementation” on page 155 uses a common business scenario for all of the scenario implementation chapters. This business scenario is called *ITSO Good*. The ITSO Good business scenario is based on the WS-I sample business scenario.

This chapter contains the following sections:

- ▶ A description of the WS-I sample business scenario
- ▶ The business context of the ITSO Good sample business scenario
- ▶ A definition of each application in the ITSO Good sample business scenario
- ▶ An example of using the ITSO Good sample business scenario

8.1 WS-I sample business scenario

The Web Services Interoperability Organization (WS-I) has developed a supply chain management business scenario that demonstrates the features, and tests for compliance of the WS-I Basic Profile V1.0. The following documents describe the WS-I sample business scenario and the technical solution overview:

- ▶ WS-I Supply Chain Management Use Cases V1.0
- ▶ WS-I Usage Scenarios V1.0
- ▶ WS-I Supply Chain Management Technical Architecture V1.0

For full details, see the Web Services Interoperability Organization Web site:

<http://www.ws-i.org>

This WS-I sample business scenario is a simplified supply chain for a consumer electronics retailer. The scenario consists of several Web service interfaces (WSDL files) and a Web-based GUI. Each Web services engine provider wishing to demonstrate the WS-I sample business scenario must provide an implementation for the Web services interfaces.

IBM provides an implementation of the WS-I sample business scenario using Web services implementations written in J2EE and which run in WebSphere Application Server. Other Web service engine providers have created other implementations, using other programming languages and application servers.

In this redbook we use a modified version of the IBM implementation of the WS-I sample business scenario, the ITSO Good sample business scenario, to demonstrate extended enterprise capabilities using SOA and Web services.

8.2 ITSO Good sample business scenario

In this redbook we use a fictional company called ITSO Good to demonstrate design, development, and runtime considerations for extended enterprise solutions. The ITSO Good sample is a modified version of the IBM implementation of the WS-I sample business scenario. This section describes:

- ▶ The high-level business context of the sample business scenario
- ▶ How the supply chain works
- ▶ Screenshots from a run through of the ITSO Good sample application implementation.

8.2.1 Business context

The ITSO Good enterprise provides a Web site which allows users to place orders for electronic goods such as televisions, DVD players, and video cameras. The ITSO Good supply chain management application handles the order of these goods through a Retailer, the delivery of these goods through a Warehouse, and the restocking of the Warehouse through a number of Manufacturers. The Manufacturers are all external organizations to ITSO Good. The users of the ITSO Good Web site are all internal ITSO Good employees.

The applications that make up the ITSO Good supply chain management scenario are shown in Figure 8-1.

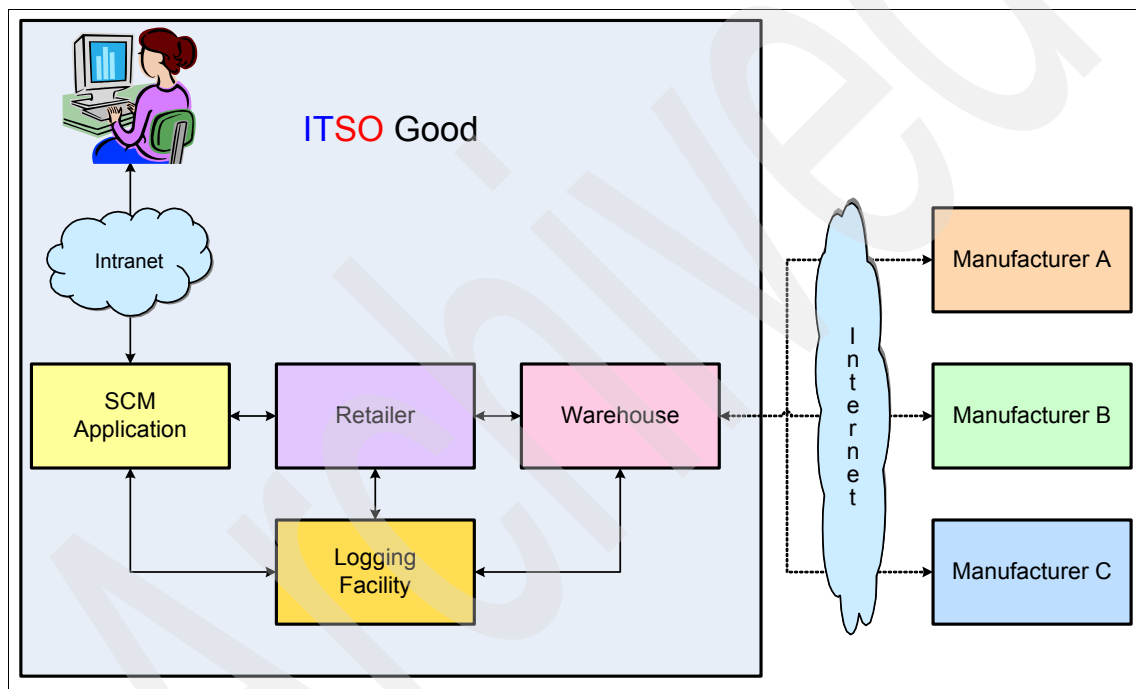


Figure 8-1 High level business context of ITSO Good

8.2.2 Applications in the supply chain management

The supply chain management scenario works as follows:

- ▶ The *SCMSampleUI* application provides a Web front-end for users to access the ITSO Good supply chain management process.
- ▶ The *Retailer* application can be used to retrieve a list of products sold by the Retailer, and to place orders.

- ▶ The *Warehouse* application ships product orders if the Warehouse has sufficient stock.
- ▶ The *Manufacturer* applications, three of those, replenish the Warehouse of products if the Warehouse stock levels fall below a certain threshold. Each Manufacturer produces different products, and all are external to the ITSO Good organization.
- ▶ The *Logging Facility* application records the status of orders as they pass through the supply chain management scenario. This can be used to track the progress of a given order.

8.2.3 Example of using the ITSO Good sample application

This section shows an implementation of the ITSO Good sample application running in WebSphere Application Server. It shows the Web-based GUI and how the supply chain management process works.

The first screen the user sees is shown in Figure 8-2.

Parameter	Value
Customer Name	A Shopper
Customer Number	A12345-9876543-xyz
Customer Address	123 Customer Lane

Configurator Options: ☒ Use all local endpoints - no configuration options

Place New Order

Figure 8-2 SCM Sample application

To use the application, a user clicks Place New Order. This retrieves a list of all the products that the Retailer application sells. This list of products is shown in the Shopping Cart screen, as shown in Figure 8-3.



Supply Chain Management Sample Application

Shopping Cart

Enter quantities for products to order. Then Submit Order.

Quantity	Product Number	Name	Description
<input type="text"/>	605001	TV, Brand1	24in, Color, Advanced Velocity Scan Modulation, Stereo
<input type="text"/>	605002	TV, Brand2	32in, Super Slim Flat Panel Plasma
<input type="text"/>	605003	TV, Brand3	50in, Plasma Display
<input type="text"/>	605004	Video, Brand1	S-VHS
<input type="text"/>	605005	Video, Brand2	HiFi, S-VHS
<input type="text"/>	605006	Video, Brand3	s-vhs, mindv
<input type="text"/>	605007	DVD, Brand1	DVD-Player W/Built-In Dolby Digital Decoder
<input type="text"/>	605008	DVD, Brand2	Plays DVD-Video discs, CDs, stereo and multi-channel SACDs, and CD-RWs, 27MHz/10-bit video DAC,
<input type="text"/>	605009	DVD, Brand3	DVD Player with SmoothSlow forward/reverse; Digital Video Enhancer; Text; Custom Parental Control (20-disc); Digital Cinema Sound mode
<input type="text"/>	605010	TV, Brand4	Designated invalid product code that is allowed to appear in the catalog but is unable to be ordered

Submit Order

Configure

Figure 8-3 SCM Sample product listing

You can order multiple quantities of each product. If the Warehouse has sufficient stock for the product, an order is placed.

If the placement of the order causes the Warehouse's stock level of that product to drop below a certain threshold, then a reorder request is sent to the external Manufacturer of the product.

The Order Status screen, Figure 8-4 on page 142, shows which orders were placed and which orders were not placed due to insufficient stock.

Order Status

Review order status. Then Track Order or pick a different configuration.

Product Number	Quantity	Price	Comment
605001	3	899.85	in stock from Warehouse
605002	6	8999.94	in stock from Warehouse

 Track Order

 Configure

Figure 8-4 SCM Sample order status page

The progress of orders can be tracked by clicking the Track Order button. This retrieves information stored in the Logging Facility application. Figure 8-5 on page 143 shows the results of an order in which products 605001 and 605002 were shipped and a reorder for 19 units of product 605002 was placed with Manufacturer B.

Supply Chain Management Sample Application

Track Order

Review details of the order. Then Configure another order or review Order Status.

Service ID	Event ID	Description	Vendor
Retailer.submitOrder	UC1-5	Order placed by A12345-9876543-xyz for 605001, 605002	
WarehouseA.ShipGoods	UC2-2-1	WarehouseA will determine its ability to ship product(s) 605001, 605002.	
WarehouseA.ShipGoods	UC2-2-2	WarehouseA is able to ship 605001, 605002.	
Retailer.submitOrder	UC1-9	Processing of the order from A12345-9876543-xyz has finished normally	
ManufacturerB.submitPO	UC3-3	ManufacturerB is replenishing stock for 605002.	
ManufacturerB.submitPO	UC5-5	ManufacturerB has produced additional units of 605002 and is shipping 19 units.	
WarehouseA.submitSN	UC3-7-1	WarehouseA has received notice that product 605002 has been shipped by ManufacturerB	
WarehouseA.submitSN	UC3-7-2	WarehouseA has replenished stock for product 605002	

 Order Status

 Configure

Figure 8-5 SCM Sample track order page

The ITSO Good sample application does not retain state. Therefore all Warehouse stock levels return to their default values the next time an order is placed.

Archived

Technology options

This chapter discusses the essential technologies that this book uses to implement the Extended Enterprise SOA patterns.

It describes the following technologies:

- ▶ **Web services**

Web services are one of the most popular set of open standards used to implement SOA. This section describes the basic Web services specifications including SOAP, WSDL, and UDDI. This section then goes on to describe some of the advanced Web services specifications including WS-BPEL and WS-Security.

- ▶ **J2EE**

Java 2 Enterprise Edition is an important technology in implementing SOA and Extended Enterprise capabilities as it is the technology behind WebSphere Application Server. This section introduces some of the main J2EE technologies for Extended Enterprise scenarios including JMS and Web services for J2EE.

- ▶ **Transport protocols**

Open, widely available transport protocols are vital in an Extended Enterprise SOA environment. This section describes the HTTP and HTTP/S transport protocols.

9.1 Web services

Web services are a recent reinvention of concepts that have been around for some time. They introduce many new advantages and capabilities. In a sense, none of the function that Web services provide is new. CORBA has provided much of this function for many years. Web services, however, builds upon existing open Web technologies, such as XML, URL, and HTTP. Web services are defined in several different standards, such as SOAP and WSDL which build upon general Web and other Web services standards. These standards are defined by the World Wide Web Consortium, the Organization for the Advancement of Structured Information Standards (OASIS), and Web Services Interoperability Organization (WS-I). The basic Web services support provides for three simple usage models:

- ▶ One-way usage scenario

A Web services message is sent from a consumer to a provider and no response message is expected.

- ▶ Synchronous request/response usage scenario

A Web services message is sent from a consumer to a provider and a response message is expected.

- ▶ Basic callback usage scenario

A Web service message is sent from a consumer to a provider using the two-way invocation model, but the response is just treated as an acknowledgement that the request has been received. The provider then responds by calling, making use of a Web service callback to the consumer.

Other Web service standards are built upon these basic standards and invocation models to provide higher level functions and qualities of service.

Examples of these standards are WS-Transaction, WS-Security, and WS-ResourceFramework. One of the main aims of Web services is to provide a loose coupling between service consumers and service providers. While this is limited to a certain extent by a requirement for the consumers and providers to agree on a WSDL interface definition, Web services have been created with significant flexibility with regard to the location of these Web services. Figure 9-1 on page 147 shows how the Web services interaction model has been designed with this form of loose coupling.

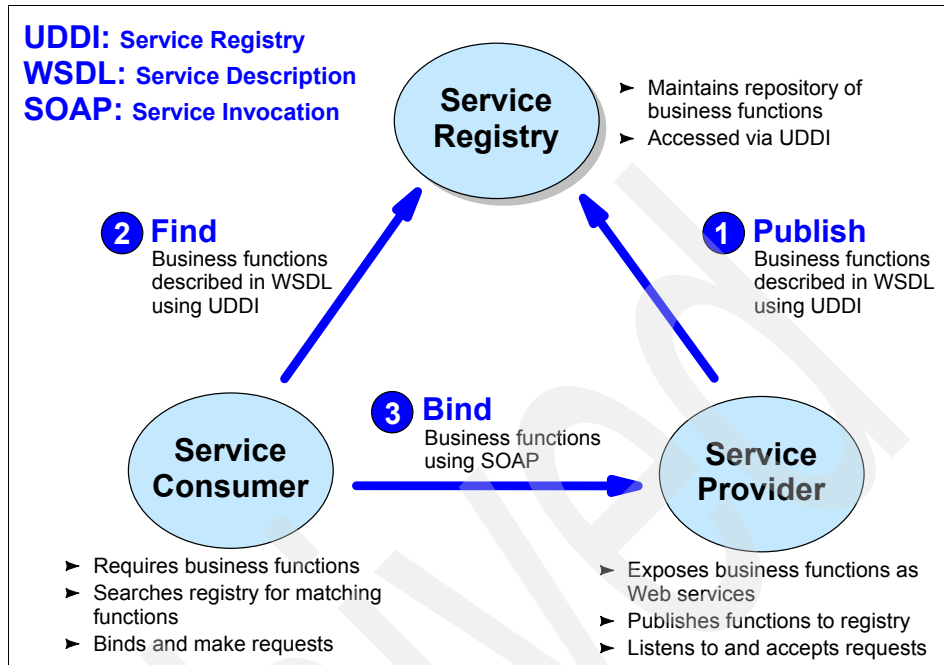


Figure 9-1 Web services interaction model

The interactions work as follows:

1. The service provider publishes WSDL, defining its interface and location to a service registry.
2. The service consumer contacts the service registry to obtain a reference to a service provider.
3. The service consumer, having obtained the location of the service provider, makes calls on the service provider.

Note: This section lists some of the Web services standards. This list is in no way exhaustive because new standards are emerging and maturing over time. For further information about Web services, visit the Web services section of the IBM developerWorks® Web site:

<http://www.ibm.com/developerworks/webservices>

9.1.1 XSD

XML Schema Definition (XSD), is a recommendation of the World Wide Web Consortium (W3C). It specifies the formal description of elements within an Extensible Markup Language (XML) document. This description is used to verify that each item of content in a document adheres to the description of the element in which the content is to be placed.

In general, a *schema* is an abstract representation of an object's characteristics and relationship to other objects. An XML schema represents the interrelationship between the attributes and elements of an XML object (for example, a document or a portion of a document). To create a schema for a document, you analyze its structure, defining each structural element as you encounter it. For example, within a schema for a document describing a Web site, you would define a Web site element, a Web page element, and other elements that describe possible content divisions within any page on that site. Just as in XML and HTML, elements are defined within a set of tags.

XSD has several advantages over earlier XML schema languages, such as document type definition (DTD) or Simple Object XML (SOX). For example, it is more direct: XSD, in contrast to the earlier languages, is written in XML, which means that it does not require intermediary processing by a parser. Other benefits include self-documentation, automatic schema creation, and the ability to be queried through XML Transformations (XSLT). Despite the advantages of XSD, it has some detractors who claim, for example, that the language is unnecessarily complex.

9.1.2 WSDL

Web Services Description Language (WSDL) is an XML-based interface definition language that separates function from implementation and enables design by contract as recommended by SOA.

WSDL descriptions contain a port type (the functional and data description of the operations that are available in a Web service), a binding (providing instructions for interacting with the Web service through specific protocols, such as SOAP over HTTP), and a port (providing a specific address through which a Web service can be invoked using a specific protocol binding).

9.1.3 SOAP

SOAP is an XML-based format for constructing messages in a transport independent way and a standard on how the message should be handled. SOAP messages consist of an envelope that contains a header and a body. It also

defines a mechanism for indicating and communicating problems that occurred while processing the message, which are known as SOAP *faults*.

The headers section of a SOAP message is extensible and can contain many different headers that are defined by different schemas. The extra headers can be used to modify the behavior of the middleware infrastructure. For example, the headers can include information about transactions that can be used to ensure that actions performed by the service consumer and service provider are coordinated.

The body section contains the content of the SOAP message. When used by Web services, the SOAP body contains XML-formatted data. This data is specified in the WSDL that describes the Web service. When talking about SOAP, it is common to talk about SOAP in combination with the transport protocol that is used to communicate the SOAP message. For example, SOAP that is transported using HTTP is referred to as *SOAP over HTTP* or *SOAP/HTTP*.

The most common transport that is used to communicate SOAP messages is HTTP. This is expected because Web services are designed to make use of Web technologies. However, SOAP can also be communicated using JMS as a transport. When using JMS, the address of the Web service is expressed in terms of a JMS connection factory and a JMS destination. Although using JMS provides a more reliable transport mechanism, it is not an open standard, requires extra and potential expensive investment, and does not interoperate as easily as SOAP over HTTP.

The SOAP version 1.1 and 1.2 specifications are available from the World Wide Web Consortium at:

<http://www.w3.org/TR/soap/>

9.1.4 UDDI

Universal Description, Discovery, Integration (UDDI) servers act as a directory of available services and service providers. SOAP can be used to query UDDI to find the locations of WSDL definitions of services, or the search can be performed through a user interface at design or development time. The original UDDI classification was based on a U.S. government taxonomy of businesses and recent versions of the UDDI specification have added support for custom taxonomies.

A public UDDI directory is provided by IBM, Microsoft, and SAP, each of whom runs a mirror of the same directory of public services. However, there are many patterns of use that involve private registries. For more information, see the following articles:

- ▶ *The role of private UDDI nodes in Web services, Part 1: Six species of UDDI*
<http://www.ibm.com/developerworks/webservices/library/ws-rpu1.html>
- ▶ *The role of private UDDI nodes, Part 2: Private nodes and operator nodes*
<http://www.ibm.com/developerworks/webservices/library/ws-rpu2.html>

9.1.5 WS-BPEL

Business Process Execution Language for Web Services (WS-BPEL) provides a means to formally specify business processes and interaction protocols.

WS-BPEL (formerly BPEL4WS) provides a language for the formal specification of business processes and business interaction protocols. By doing so, it extends the Web services interaction model and enables it to support business transactions. WS-BPEL defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

9.1.6 WS-Security

The security protocols for Web services begin with the WS-Security specification that defines a token-based architecture for secure communications. There are six major component specifications built on this base:

- ▶ WS-Policy and its related specifications define the policy rules on how services interact.
- ▶ WS-Trust defines the trust model for secure exchange.
- ▶ WS-Privacy defines how privacy of information is maintained.
- ▶ WS-SecureConversation defines how to establish a secured session between services for exchanging data using the rules defined in WS-Policy, WS-Trust, and WS-Privacy.
- ▶ WS-Federation defines the rules of distributed identity and how it is managed.
- ▶ WS-Authorization handles the processing for authorization to access and exchange data.

9.2 J2EE

Java technology is both an object-oriented programming language and a platform originally developed by Sun Microsystems. The Java platform consists of the Java Application Programming Interface (API) and the Java Virtual Machine (JVM), an interpreter between the programming language and the underlying

software and hardware architectures. The Java API is a large collection of ready-made software components to ease the development and deployment of applets and applications, including robust, secure and interoperable enterprise applications.

J2EE (Java 2 Enterprise Edition) is the enterprise version of Java that simplifies the construction and deployment of multitier enterprise applications by basing them on standardized modular components. It provides a complete set of services to those components, and by handling many details of application behavior automatically without complex programming.

Note: A large community of developers, testers and technology experts contribute to the Java APIs through a community process known as the Java Community Process (JCP). IBM has contributed significantly to the JCP since the birth of J2EE and continues to do so. You can track the JCP at:

<http://www.jcp.org/en/home/index/>

Java technology is critical to the IBM On Demand Business initiative. Java was one of the first technologies to support open standards in the enterprise, enabling customers to adopt XML and Web services in seamless information and application integration. Additionally, Java serves as the cornerstone of many IBM products and technology consulting services.

9.2.1 JMS

Java Message Service (JMS) is an API that adds a provider framework that enables the development of portable, message-based applications for the Java platform by defining a common set of messaging concepts and programming strategies that will be supported by all JMS technology-compliant messaging systems.

9.2.2 Web services for J2EE

Web services for J2EE (WSEE) leverages J2EE technologies, defining the needed mechanism to standardize a deployment model for Web services. This standardization aims to achieve interoperability across different, compliant J2EE platforms, transforming the migration among them into a routine process and ensuring that vendors interoperate.

WSEE defines the concepts, interfaces, file formats, and responsibilities to support the development and runtime models for Web services.

WSEE-compliant Web service providers certify that their services can be redeployed in other compliant servers. WSEE enables developers, assemblers, and deployers to configure Web services through XML-based deployment descriptors.

9.2.3 JAX-RPC

Java API for XML-based RPC (JAX-RPC) facilitates distributed computing in a Web services environment. JAX-RPC-based Java applications can easily communicate with non-Java-based technologies in the RPC style fashion.

A JAX-RPC server application's entry point is also known as an *endpoint*. A Web service endpoint is described using a WSDL document. JAX-RPC is about Web services interoperability across heterogeneous platforms and languages. This makes JAX-RPC a key technology for Web services-based integration.

9.3 Transport protocols

The fundamental transport mechanisms between Web components are HTTP and HTTP/S.

9.3.1 HTTP

HyperText Transfer Protocol (HTTP) is a request/response protocol between clients and servers. It is also the default communication protocol of the World Wide Web.

An HTTP server listening on that port waits for an HTTP client to send a request string, such as GET/HTTP/1.1 followed by an optional body of arbitrary data.

Upon receiving the request string (and message, if any), the server sends back a response string, such as 200 OK, and a message of its own, the body of which is perhaps the requested file, an error message, or some other requested information.

An HTTP client, such as a Web browser, typically initiates a request by establishing a TCP/IP connection to a particular port on a remote host (typically port 80).

Both responses and requests have headers which contain useful information. Some headers are optional, while others, such as Host, are required by the HTTP/1.1 protocol.

9.3.2 HTTP/S

HTTP/S is the secure version of HTTP. Instead of using plain text socket communication, HTTP/S encrypts the session data using either a version of the Secure Socket Layer (SSL) protocol or the Transport Layer Security (TLS) protocol, thus ensuring reasonable protection from eavesdroppers, and man in the middle attacks. The default TCP/IP port of HTTP/S is 443.

The level of protection depends on the correctness of the implementation by the Web browser and the server software and the actual cryptographic algorithms supported.

In Web pages that use HTTP/S, the URL begins with `https://` rather than `http://`.



Part 3

Scenario implementation

Archived

Exposed Direct Connection runtime pattern: generic profile

The simplest form of integrating with an Extended Enterprise is by using the Exposed Direct Connection runtime pattern, where service consumers and providers communicate through direct point-to-point interactions.

This chapter describes the design, development and runtime guidelines to implement the Exposed Direct Connection runtime pattern using a generic profile.

This includes step-by-step instructions for configuring WebSphere Application Server V6.0.2 to make secure Web service calls to a variety of service providers including Microsoft .NET and CICS Transaction Server. Step-by-step instructions also describe how to use Rational Application Developer to secure Web service interactions using WS-Security.

10.1 Business scenario

The business scenario implemented in this chapter represents the supply chain management on demand scenario defined in Chapter 8, “Business scenario used in this book” on page 137.

The supply chain management application of ITSO Good makes requests to the Retailer system to help customers buy electronic goods online. The Retailer receives stock from the Warehouse, and the Warehouse replenishes stock from the external Manufacturers that reside outside the the enterprise, on a one-to-one basis, as shown in Figure 10-1.

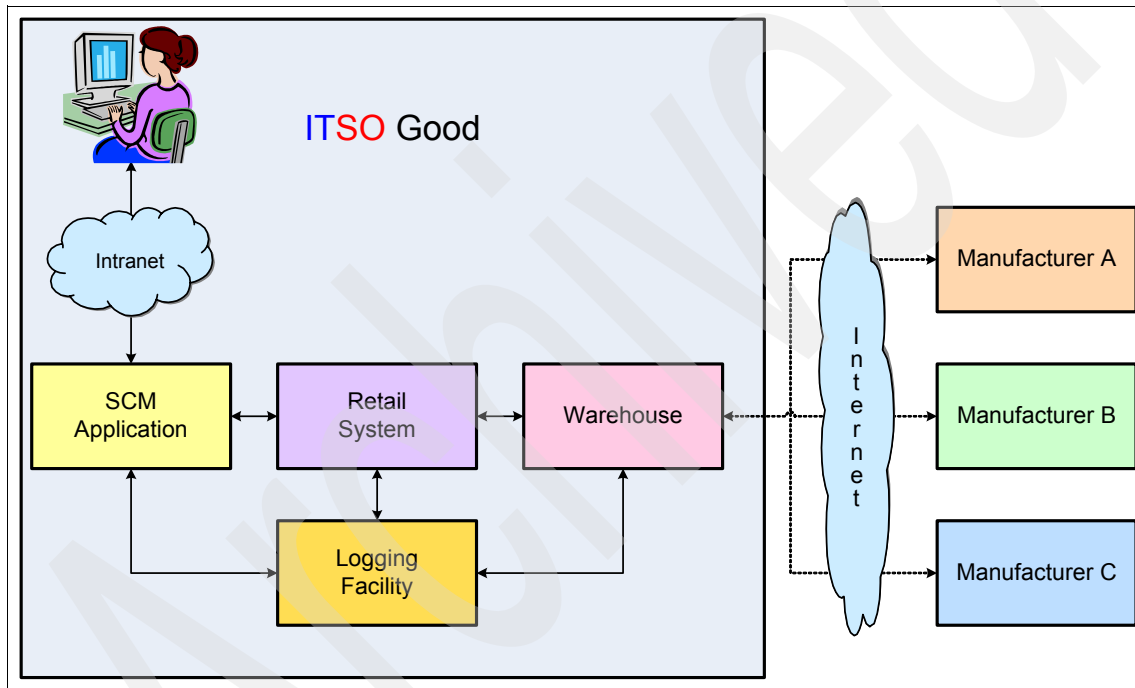


Figure 10-1 High-level business context of the scenario

Having successfully integrated their internal Retail and Warehouse systems, ITSO Good have now decided to integrate with their external Manufacturing partners. As a result, ITSO Good have come up with the following business requirements related to the integration of their Warehouse system with the external Manufacturing partner systems:

- ▶ ITSO Good requires the Warehouse system to have real-time access to the Manufacturer systems that reside outside the enterprise to reduce the latency

of business events and also requires the Manufacturer to respond in real-time to the replenishment order placed by the Warehouse.

- ▶ The Warehouse system also needs to have the flexibility to support the heterogeneous and existing environments of their Manufacturing partner systems. ITSO Good expects its IT systems to use interoperable standards for its integration with external systems.
- ▶ This scenario represents the organization's first attempt to implement a service-oriented architecture (SOA) solution. The systems that make up the scenario are fixed and established. Their location and naming conventions are not expected to change.

10.2 Design guidelines

Besides the usual guidelines that govern application integration design, integration in an extended enterprise has some additional guidelines related to the qualities of service, policies, and so forth.

In an intraenterprise application integration scenario, interactions are made within an enterprise's trusted networks. But in an extended enterprise, integration with the partner's business processes might involve interactions that are exposed to less secured zones such as the internet or shared Wide Area Networks (WANs). As a result, qualities such as security, reliability, and interoperability need special attention. These service qualities manifest themselves with differing degrees of importance and specificity in different integration scenarios.

The design guidelines detailed here and in the following chapters are also influenced by the trading partner agreements and service level agreements between the organizations involved.

The design guidelines section is split into the following categories:

- ▶ **Analyze business requirements**
Describes the system context of the problem this chapter is trying to solve.
- ▶ **Selecting a pattern**
Describes which Extended Enterprise runtime pattern from the Patterns for e-business is suitable to solve this business problem.
- ▶ **Analyze design options**
Describes design decisions in building the solution.

► Products

Lists the appropriate IBM products that can be used to implement the solution, and provides a product mapping of the products used in the actual implementation.

10.2.1 Analyze business requirements

Figure 10-2 shows the system context diagram depicting the interactions of ITSO Good with external systems.

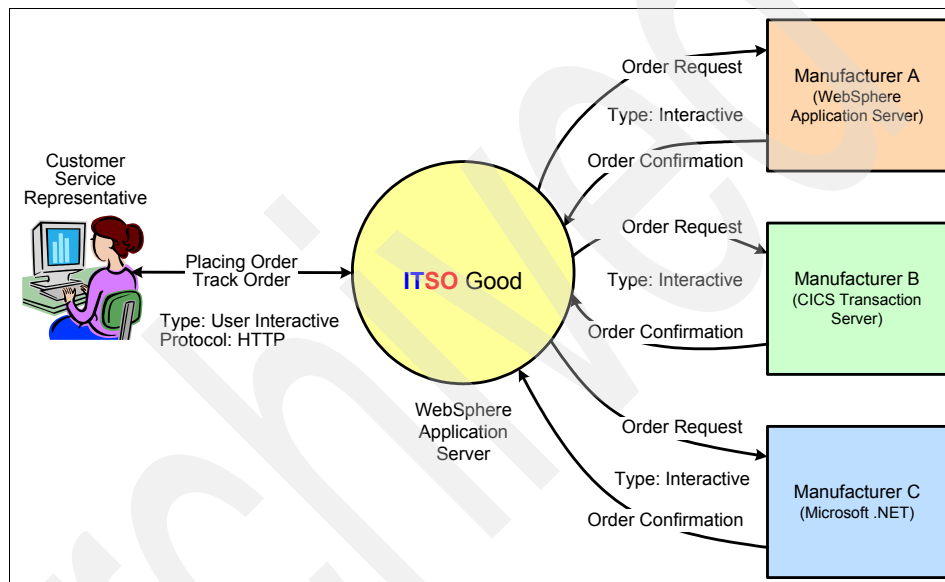


Figure 10-2 System context for the Exposed Direct Connection generic profile scenario

The Warehouse system is expected to integrate with three different types of Manufacturer systems, implemented in Microsoft .NET, CICS Transaction Server, and a WebSphere Application Server J2EE-based environment respectively. The interactions are two-way (request/response) with the Manufacturer's systems responding to the submitted purchase order with an order confirmation.

10.2.2 Selecting a pattern

We use the Patterns for e-business to determine the appropriate Runtime pattern to apply to this scenario. The Patterns approach is based on a set of layered assets that can be exploited by any existing development methodology.

Figure 10-3 depicts these assets and their relationships to each other.

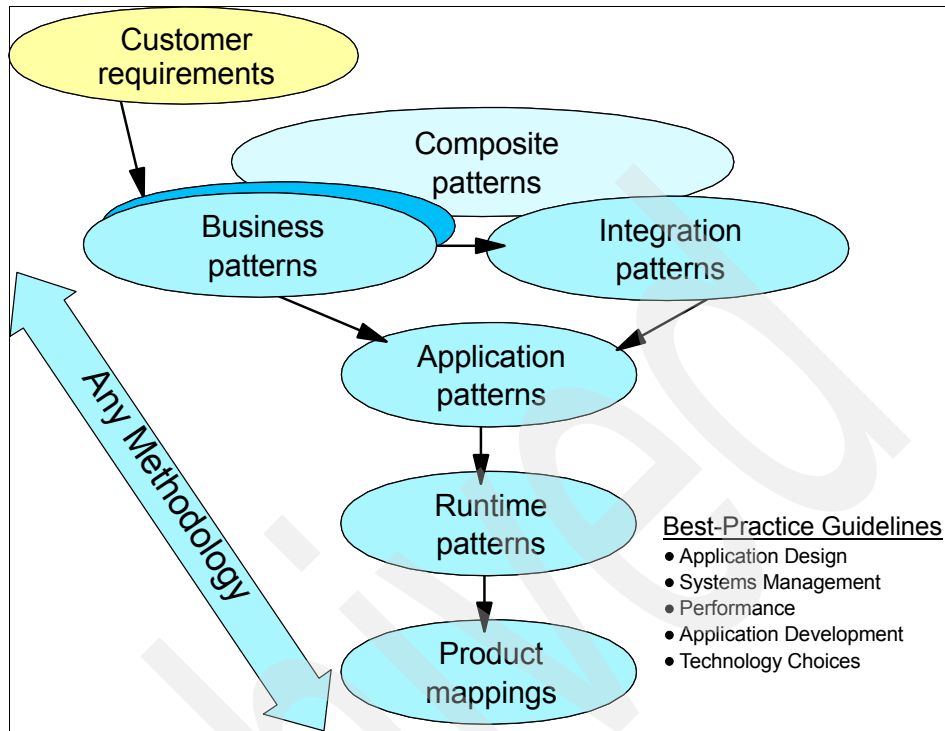


Figure 10-3 The Patterns for e-business layered asset model

Here, we describe a step-by-step approach used to navigate the Patterns for e-business asset catalog:

1. Business pattern

We select the Extended Enterprise business pattern since the given scenario requires interactions between the business processes in the Warehouse and Manufacturer systems that reside in separate enterprises.

2. Application pattern

The Warehouse and Manufacturer systems are required to interact on a one-to-one basis representing point-to-point connections. Therefore we select the Exposed Direct Connection application pattern. This pattern has two variations:

- Message Connection variation
- Call Connection variation

Because the business scenario requires the proposed solution to support real-time request/reply message flows to partner processes, we select the Call Connection variation.

3. Runtime pattern

The Application pattern provides us with the Direct Connection runtime pattern for the proposed solution. Since this solution represents the first-step in SOA transformation for ITSO Good, we select the generic profile of the Direct Connection runtime pattern to keep the proposed solution simple.

Figure 10-4 shows the level 0 decomposition of the generic profile of the Exposed Direct Connection runtime pattern, mapped onto the Exposed Direct Connection application pattern.

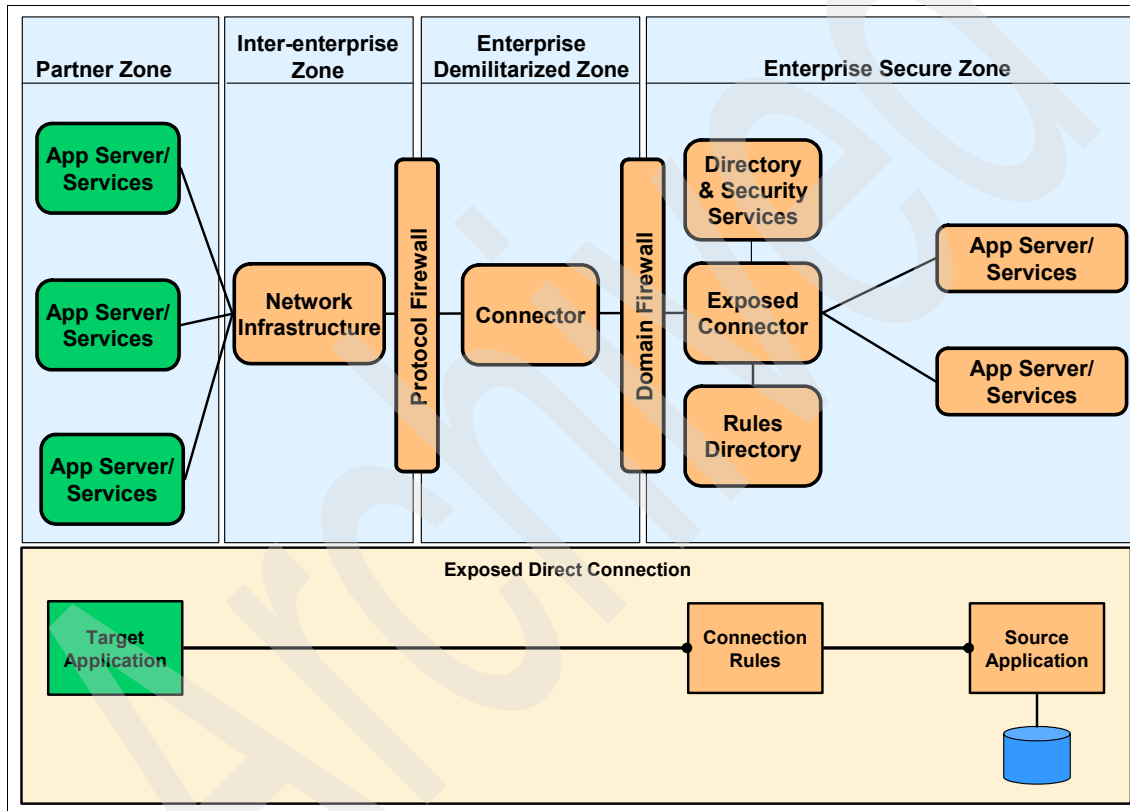


Figure 10-4 Exposed Direct Connection::Runtime pattern = Generic profile

10.2.3 Analyze design options

Direct integration between applications can provide fast response time but at the same time can be inflexible, in that any changes to one application can have knock-on effects on other applications. This is especially dangerous when integrating across organizational boundaries. Any changes to the exposed target

application might require changes to many partner applications. Such changes can be both expensive and time consuming.

Such knock-on effects can be minimized using message adapters that wrapper the applications in the exposed connection. This is represented by the Exposed Connector component in the Exposed Direct Connection application pattern in Figure 10-4 on page 162. These Connectors can be explicitly or implicitly modeled. They convert the mutually agreed upon messages into API calls to existing or new backend applications. This layering technique isolates the exposed applications from partner applications and increases flexibility. Any changes to these exposed applications would only impact the Connector, provided there is no need to change the mutually agreed upon messages.

Message definition should also be generalized to further promote flexibility. In other words, messages should not be tightly coupled with backend application APIs. Rather the message should capture all the necessary information required for that logical interaction across business boundaries. Such generalization will help cope with changes to the backend application API without having to change the agreed upon message format.

For example, the use of WSDL (Web Services Definition Language) to describe the services offered by the Manufacturers would provide for a standard way to define the interaction semantics.

Also since the interaction is directly exposed across organizational boundaries, it must implement or exploit the necessary security features such as authentication, authorization, confidentiality, integrity, and logging for nonrepudiation purposes.

Using these guidelines, the following architectural decisions were made for the given scenario.

Note: The following decisions are based on the sample business scenario for this Redbook. In reality, there might be other factors such as existing assets, organization skills, business strategies, and policies that need to be considered while making any architectural decisions.

Architectural decision: integration options

The requirement to integrate ITSO Good with external service providers presents a number of integration options. These are discussed in Table 10-1 on page 164.

Table 10-1 Integration options

Subject area	Integration options between Warehouse and Manufacturer
Issue or problem statement	To select the technology solution used to integrate the replenishment business process of the Warehouse with the Manufacturing process in the external partner systems across organizational boundaries.
Assumptions	The three partner Manufacturer systems are built using J2EE, Microsoft .NET and CICS Transaction Server platforms respectively.
Motivation	ITSO Good requires the Warehouse system to have the flexibility to support the heterogeneous and existing environments of their Manufacturing partner systems.
Alternatives	<ol style="list-style-type: none"> 1. Web-service standard based integration using SOAP over HTTP(s). 2. Use custom technology specific adapters to integrate with each of the CICS Transaction Server, Microsoft .NET and WebSphere Application Server J2EE based Manufacturer systems. 3. Use of COTS (Commerical-Off-The-Shelf) packages like WebSphere Business Integration Adapters to integrate with each of the CICS Transaction Server, Microsoft .NET and WebSphere Application Server J2EE based Manufacturer systems.
Decision	Web-services using SOAP over HTTP/S will be used to integrate the Warehouse system with each of the Manufacturer system.

Subject area	Integration options between Warehouse and Manufacturer
Justification	<p>Alternative#1: The use of Web services to integrate the Warehouse and Manufacturer services allows loose coupling between the two and is also independent of the service implementation of either systems. Use of Web services also fosters inter-operability since it is not technology dependent.</p> <p>Alternative#2: The use of custom technology-specific adapters to connect to the different heterogeneous platform environments of the Manufacturer. This might improve performance in some cases but it comes at the cost of flexibility and maintainability. Flexibility/adaptability is a key factor in an extended enterprise scenario. Also firewall restrictions could pose problems for certain native communication/transport protocols.</p> <p>Alternative#3: The use of WebSphere Business Integration Adapters, which are off-the-shelf software based on the WebSphere Adapter Framework, is a useful asset. It could help in rapid application development. But even this option would not be a flexible one. Firewall restrictions could also pose concerns.</p> <p>In conclusion, Web services are an open standard that is supported by WebSphere Application Server, Microsoft .NET and CICS Transaction Server along with many other application platforms. Web services provides an inter-operable way to integrate heterogeneous systems with minimal cost.</p>

Securing Web services

Having decided to use Web services as the means to integrate the Warehouse and Manufacturer systems that reside in separate enterprises, we need to ensure that these Web services interactions are secured.

In this section, we take a brief look into the various security options available and design a security solution based on the business requirements.

Today, securing Web services involves:

- ▶ Message level security (WS-Security)
- ▶ Transport level security (SSL/TLS)

These elements are shown in Figure 10-5.

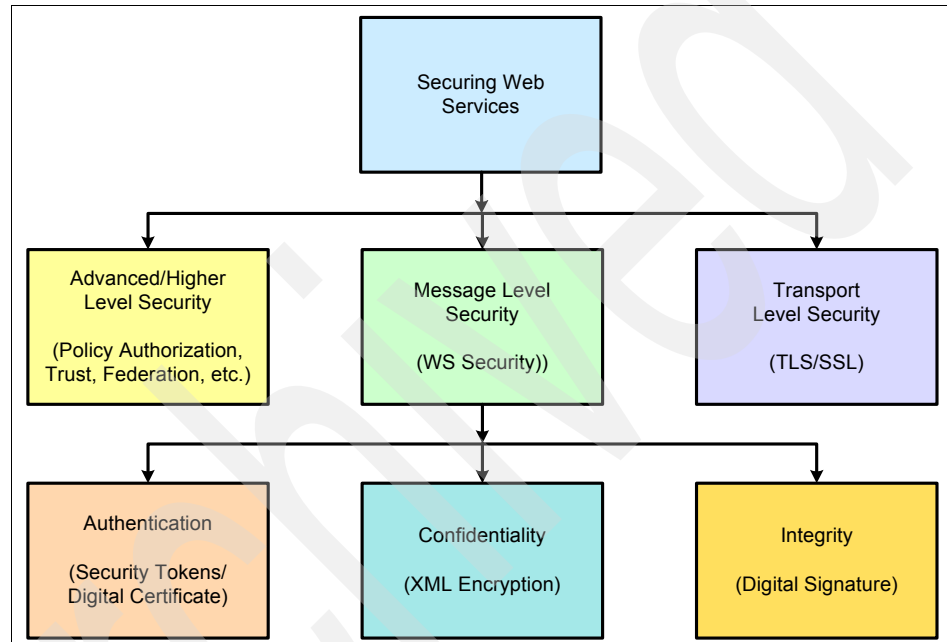


Figure 10-5 Securing Web services

Message level security: WS-Security

WS-Security is a standard set of SOAP extensions that handles three aspects of Web services security (commonly known as the *security triad* of WS-Security) namely: authentication, integrity, and confidentiality.

These three mechanisms can be used independently (for example, to pass a security token) or in a tightly integrated manner (for example, signing and encrypting a message and providing a security token hierarchy associated with the keys used for signing and encryption).

Authentication

Authentication ensures that the security credentials (business identity, user name, digital certificates, and so forth) of the service consumer passed in the

Web service request are really who they claim. *Security tokens* are used in WS-Security to pass these credentials. They are broadly of two types:

► UsernameToken

An XML-based security token specified in the <Security> header of a SOAP message. It contains the username and an optional password information that can be passed as plaintext or as a digest format.

The syntax of this element is:

```
<UsernameToken Id="...">
  <Username>...</Username>
  <Password Type="...">...</Password>
</UsernameToken>
```

In this form of authentication, also known as *basic authentication*, the user credentials (user name and password) are validated and authenticated by the service provider against a local security repository such as Tivoli® Access Manager.

► BinarySecurityToken

The BinarySecurityToken element defines a security token that is binary encoded. The encoding is specified using the ValueType attribute that indicates what the security token is (for example, *X.509 Certificates*, *Kerberos tickets*) and the EncodingType attribute indicates how the security token is encoded (for example, Base64 encoding, hex encoding).

The syntax of this element is:

```
<BinarySecurityToken Id=...
  EncodingType=...
  ValueType=.../>
```

Digital Certificates such as X.509 Certificates contain:

- Identity credentials that can be mapped to a user by using public key infrastructure
- A pair of public and private keys associated with it

Using the certificate on its own would make for relatively easy replay attacks. As a result, a common practice is to use the private key of the certificate to digitally sign a piece of information, elements in the SOAP header or body. By validating the signed information using the public key associated with the sender's certificate, the receiver can authenticate the sender as being the owner of the certificate, thereby validating the sender's identity. When both the service consumer and provider authenticate each other using their respective certificates it is called *mutual authentication*. This also requires the consumer and provider to maintain the public keys of each other's certificate in a private key store.

Integrity

Message integrity is provided by leveraging an XML signature in conjunction with security tokens to ensure that messages are transmitted without modifications. In order to validate the integrity of the information contained in the SOAP message, the data can be digitally signed using the security keys.

When building trust into an application based on a digital signature, there are other technologies, such as certificate evaluation, that should be incorporated.

Confidentiality

Message confidentiality leverages XML encryption in conjunction with security tokens to keep portions of a SOAP message confidential. Encryption of data can be achieved using symmetric or asymmetric algorithms:

- ▶ Symmetric algorithms such as Triple DES use a shared key for both encryption and decryption.
- ▶ Asymmetric algorithms such as RSA-V1.5 use a pair of keys, public and private keys, for encryption and decryption. One of the keys is used for encrypting the information while the other is used for decrypting it.

Symmetric algorithms are more efficient than asymmetric algorithms; however, they require management of shared keys between the parties and have inherent security risks of being exposed to others outside of your organization or business partners. On the other hand asymmetric algorithms are highly performance intensive and hence care should be taken to ensure that only highly confidential elements of a SOAP message are encrypted using this method.

A common practice is to use a symmetric algorithm to encrypt the SOAP message, then include the shared key itself in the SOAP message with the key encrypted and decrypted using an asymmetric algorithm. This provides an efficient solution and one that is easy to manage.

WS-Security design guidelines

This section discusses a set of guidelines for designing Web services security using WS-Security.

- ▶ Message level security for Web services using WS-Security should be used when application to application security is required and not just security over the internet between nodes at the enterprise boundary.
- ▶ Digital signature capability provided by WS-Security is an ideal choice for providing message integrity when mission-critical or confidential data needs to be exchanged in a SOAP message between applications.
- ▶ Both XML digital signature and XML encryption will have an impact on the performance of Web services. If performance is a primary concern over all other requirements, then encryption at the transport level using SSL is

probably a better choice. On the other hand, XML digital signature can also be used judiciously to sign only specific elements of a SOAP message like the security tokens in the SOAP header to provide authentication.

- ▶ As noted in the W3C specification of XML encryption, cryptographic vulnerabilities could be introduced when combining digital signatures and encryption over a common XML element. Encrypting digitally signed data, while leaving the digital signature in the clear, might allow plaintext guessing attacks. This vulnerability can be mitigated by using secure hashes and nonces in the text being processed.
- ▶ When messages are exchanged over an untrusted zone, it is recommended that messages include digitally signed elements to allow message receivers to detect replays of the message. Digital signatures must be combined with an appropriate means to ensure the uniqueness of the message, such as timestamps or sequence numbers. This could otherwise lead to *denial of service* attacks on the service provider.
- ▶ Interoperability should be a primary goal. The Web Services-Interoperability Organization (WS-I) is working on a draft called *WS-I Basic Security Profile 1.0* that is based on a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications to promote interoperability. As an extension of the Basic Profile, the Basic Security Profile is designed to support the addition of security functionality to SOAP messaging, in an interoperable manner.

The latest version of this specification is available at:

<http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html>

WS-Security by itself does not provide a complete security solution. Instead, WS-Security is a building block that can be used in conjunction with other Web service extensions and higher-level, application-specific protocols to accommodate a wide variety of security models and encryption technologies.

Important: When designing the security aspects of a system, it is important to understand that absolute security is unobtainable. The security measures that we put in place only reduce the chances of a system being attacked by malicious users and cannot prevent it totally. A resource can be considered safe if the cost of an attack is made so high as to make it unattractive. This principle should be borne in mind when designing security. *Security policies* also play an important role in monitoring and managing the security measures put in place.

Transport level security for Web services

HTTP, the most widely used Internet communication protocol, is currently also the most popular protocol for Web services. HTTP is an inherently insecure

protocol, because all information is sent in clear text between unauthenticated peers over an insecure network.

To secure HTTP, you can apply transport-level security. Secure Sockets Layer (SSL) and Transport Layer Security (TLS), its successor, are cryptographic protocols which provide secure communications on the Internet.

Trivia: You can use TLS / SSL connections to encrypt data transferred over other application level protocols, such as File Transfer Protocol (FTP), Lightweight Directory Access Protocol (LDAP), and Simple Mail Transfer Protocol (SMTP).

In typical use, only the server is authenticated (its identity is ensured) while the client remains unauthenticated; mutual authentication requires PKI deployment to clients.

Unlike message-level security, HTTPS encrypts the entire HTTP data packet. There is no option to apply security selectively on certain parts of the message. SSL and TLS provide security features including authentication, data protection, and cryptographic token support for secure HTTP connections. Therefore transport level security is recommended when there are no intermediaries involved between the service provider and requester.

Architectural decision: Securing the Web service interaction

The business requirements as described in 10.1, “Business scenario” on page 158 require us to integrate the Warehouse service with the Manufacturing service in the external partner systems, providing a real-time and service-oriented access between them. The security of these interactions is discussed in Table 10-2 on page 171.

Table 10-2 Securing the Web services

Subject Area	Securing the interaction between the ITSO Good enterprise and external Manufacturer
Issue Or Problem Statement	<p>The Web service interaction between the Warehouse system and the partner Manufacturing system that reside in separate enterprises needs to be secured.</p> <p>Some of the additional security requirements that need to be addressed by the solution are:</p> <ul style="list-style-type: none"> ▶ The proposed solution needs to be secure with no dependency on the network infrastructure used to connect the Warehouse and Manufacturing systems. They are initially expected to be connected over the ubiquitous but untrusted internet. ▶ The corporate security policy requires that all confidential information exchanged between the boundary nodes of the enterprise and partner systems be protected using SSL.
Assumptions	Web-services using SOAP over HTTP/S is used by the Warehouse system to invoke the Manufacturing service exposed by the partner systems.
Motivation	Because the partner systems are implemented in different technology environments, we need to use standard and inter-operable security features.
Alternatives	<ol style="list-style-type: none"> 1. Providing message-level security using WS-Security 2. Providing transport-level security using SSL/TLS. 3. A combination of WS-Security and TLS/SSL.
Decision	Option#3: Message level security using XML digital signature & XML encryption and transport layer security using SSL would be used to secure the Web service interaction. Username token would be used for identification.

Subject Area	Securing the interaction between the ITSO Good enterprise and external Manufacturer
Justification	<ul style="list-style-type: none"> ▶ Option 1: Message-level security alone doesn't satisfy the requirement to have the messages exchanged across the boundary nodes of the enterprises to be secured at the transport layer as well. ▶ Option 2: Given that there are no intermediaries involved (Direct Connection) between the Warehouse and Manufacturer systems, transport level security could have been an ideal option. But it does not satisfy the requirement for the solution to be independent of the network infrastructure. ▶ Option 3: This provides an ideal choice as this provides security at two levels. In case the use of XML encryption induces performance overhead, we could limit the encryption of data at the transport level alone using SSL along with XML digital signature at the message-level for authentication and data integrity.
Related Decision	Non-repudiation would be handled by logging the events and transactions into the Logging Service utility provided by ITSO Good.

The proposed security model shown in Figure 10-6 on page 173, uses multiple security tokens in the SOAP header. The UsernameToken sends the user credentials and the keys associated with the X.509 Certificate (BinarySecurityToken) digitally sign the UsernameToken using XML Digital signature. By validating the signature, the Manufacturing system authenticates the sender and also validates the integrity of the message. The SOAP body was not digitally signed because, in the given business scenario, it did not contain any confidential data.

Both the Warehouse and Manufacturing systems import each other's Digital certificate (X.509 Certificate) into their keystores.

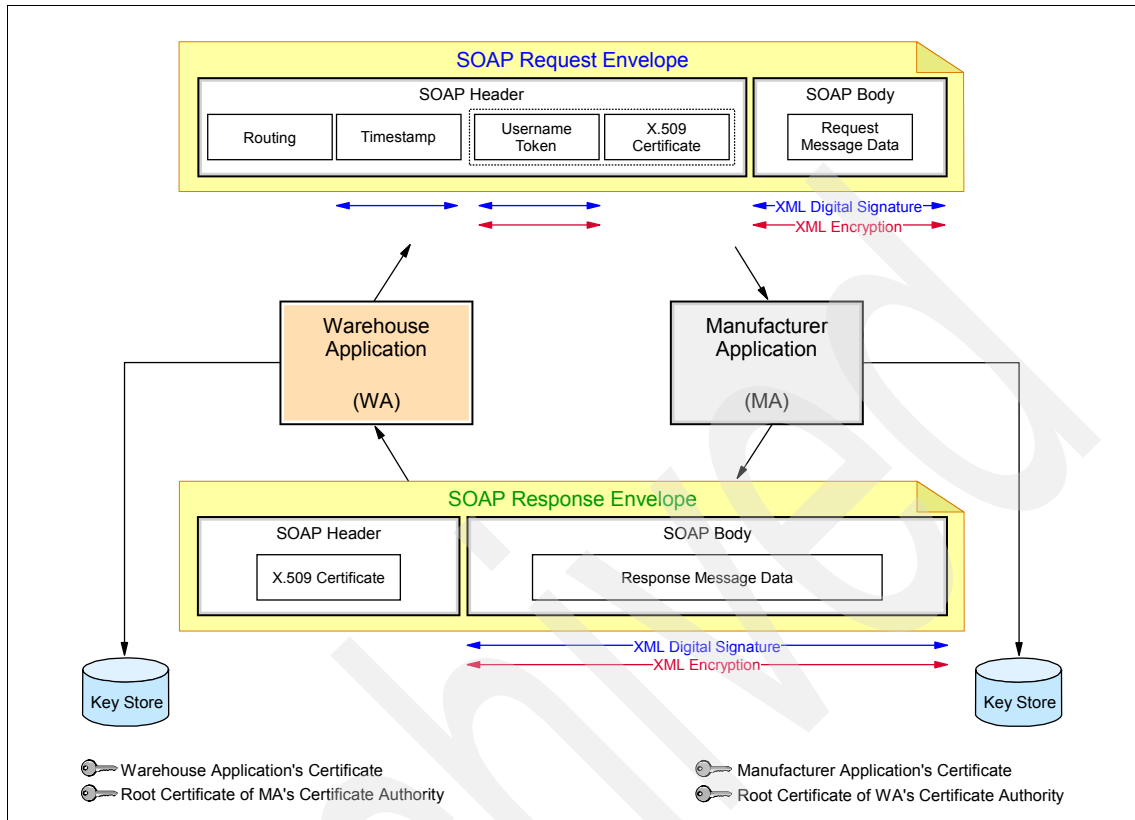


Figure 10-6 Web services security model between the Warehouse and Manufacturer.

In Figure 10-6, the Timestamp detail in the SOAP header is also digitally signed to protect against replay attacks. This also ensures that no duplicate replenishment orders are processed by the Manufacturing service as a result of the replays.

XML encryption at the message layer provides confidentiality. If the overhead associated with XML encryption results in unacceptable performance, transport level security using SSL could be used for data confidentiality to ensure better performance.

A peek into the advanced security features

The WS-Security specification discussed previously only includes a subset of security services. A more general security model is needed to cover other security aspects, such as logging and nonrepudiation in a real-world scenario. The definition of those requirements is given in a common Web services security model framework, a security white paper of Web Services Security Roadmap

proposed by IBM and Microsoft. A brief description of this Roadmap is provided here. For more detailed information about this roadmap, refer to these resources:

<http://www-128.ibm.com/developerworks/library/specification/ws-secmap/>

<http://msdn.microsoft.com/library/en-us/dnwssecur/html/securitywhitepaper.asp>

Web services security model framework

The Web services security model introduces a set of individual interrelated specifications to form a layering approach to security. It includes several aspects of security: identification, authentication, authorization, integrity, confidentiality, auditing, and nonrepudiation. It is based on the WS-Security specification, codeveloped by IBM, Microsoft, and VeriSign.

The Web services security model schema is shown in Figure 10-7.

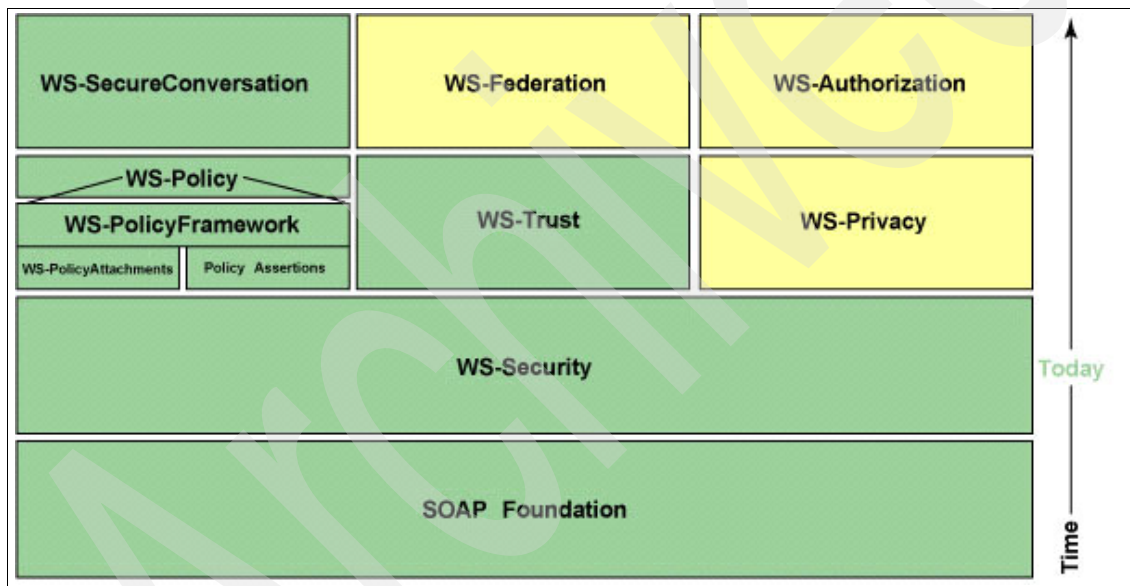


Figure 10-7 Web Services Security Roadmap

These specifications include different aspects of Web services security including:

- **WS-Policy**

This policy describes the capabilities and constraints of the security and other business policies on intermediaries and endpoints, required security tokens, supported encryption algorithms, and privacy rules, for example. The WS-SecurityPolicy specification defines a set of security policy assertions which apply to Web Services Security: SOAP Message Security, WS-Trust, and WS-SecureConversation.

Attention: An updated version of the Web Services Security Policy Language (WS-SecurityPolicy) specification has been released by IBM, Microsoft, RSA Security, and VeriSign. IBM, Microsoft and 12 other coauthors also announced that this WS-SecurityPolicy specification, together with Web Services Trust Language (WS-Trust) and Web Services Secure Conversation Language (WS-SecureConversation), was submitted to OASIS for standardization in September 2005.

- ▶ **WS-Trust**
This policy describes a framework for trust models that enables Web services to securely interoperate. This specification is responsible for managing trusts and establishing trust relationships.
- ▶ **WS-Privacy**
This policy describes a model for how Web services and consumers state privacy preferences and organizational privacy practice statements.
- ▶ **WS-Federation**
This policy describes how to manage and broker the trust relationships in a heterogeneous federated environment, including support for federated identities.
- ▶ **WS-Authorization**
WS-Authorization describes how to manage authorization data and authorization policies.
- ▶ **WS-SecureConversation**
This specification is built on top of the WS-Security and WS-Policy models to provide secure communication between services. This specification defines mechanisms for establishing and sharing security contexts, and deriving keys from security contexts, to enable a secure conversation.

10.2.4 Products

We discussed several design decisions which influence product choice in 10.2.3, “Analyze design options” on page 162. This section looks at the products that you can use to implement these design decisions and the product choices that were made for this particular implementation.

Product implementation options

Product choices for this scenario are based on:

- Design decisions made in 10.2.3, “Analyze design options” on page 162
- Extended Enterprise capabilities of the products
- Products that are currently available

For this scenario, we can use the following currently available products to implement the generic profile of the Exposed Direct Connection runtime pattern:

- WebSphere Application Server V6.0.2
- WebSphere Application Server Network Deployment V6.0.2
- WebSphere Application Server Network Deployment V6.0.2 Web Services Gateway

You can find a comparison between available products and their Extended Enterprise capabilities in Chapter 5, “Product descriptions” on page 87.

Product mapping selected

For this scenario, we will use WebSphere Application Server V6.0.2 because it meets all of the requirements. The Product mapping is shown in Figure 10-8.

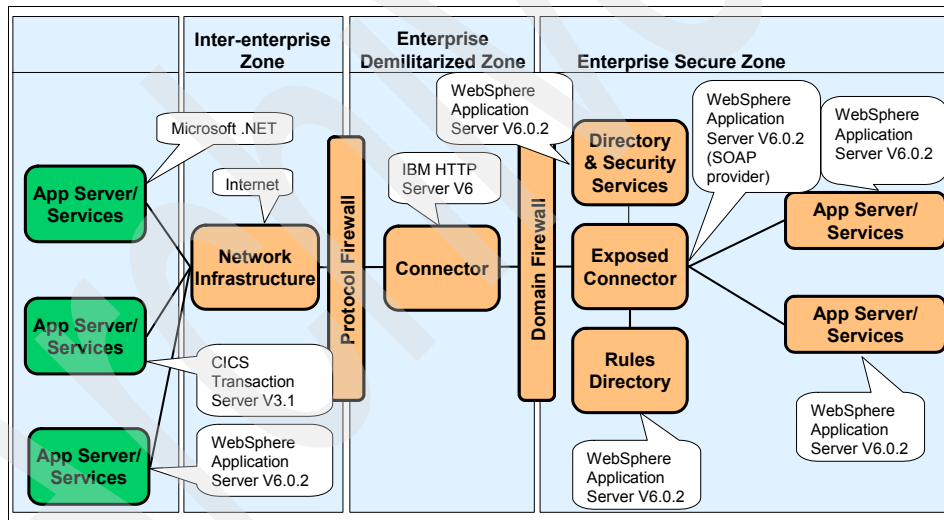


Figure 10-8 Exposed Direct Connection:: Product mappings

In this Product mapping, WebSphere Application Server V6.0.2 was used for all services within the ITSO Good enterprise. The Warehouse service invokes the Manufacturing service that are implemented in the three heterogeneous technical environments namely, CICS Transaction Server V3.1, WebSphere Application Server V6.0.2 and Microsoft .NET respectively. The Warehouse service connects to the Manufacturing service through a SOAP/HTTP Web

service call using the SOAP Provider in the WebSphere Application Server Network Deployment V6.0.2.

The Directory and Security services are being provided by the WebSphere Application Server Network Deployment in this scenario to keep the solution simple. For more advanced configuration, consult the IBM redbook *Enterprise Security Architecture Using IBM Tivoli Security Solutions*, SG24-6014.

10.3 Development guidelines

The scenario in this book describes the development guidelines for implementing the product mapping shown in Figure 10-8 on page 176 for the Exposed Direct Connection runtime pattern. This product mapping describes how to add WS-Security to Web service interactions by configuring enterprise applications in Rational Application Developer.

Note: The development guidelines in this section use Rational Application Developer V6.0.1.

10.3.1 Exposed Direct Connection interaction: Generic profile

Figure 10-9 on page 178 shows the interactions that are made by each component in the sample application.

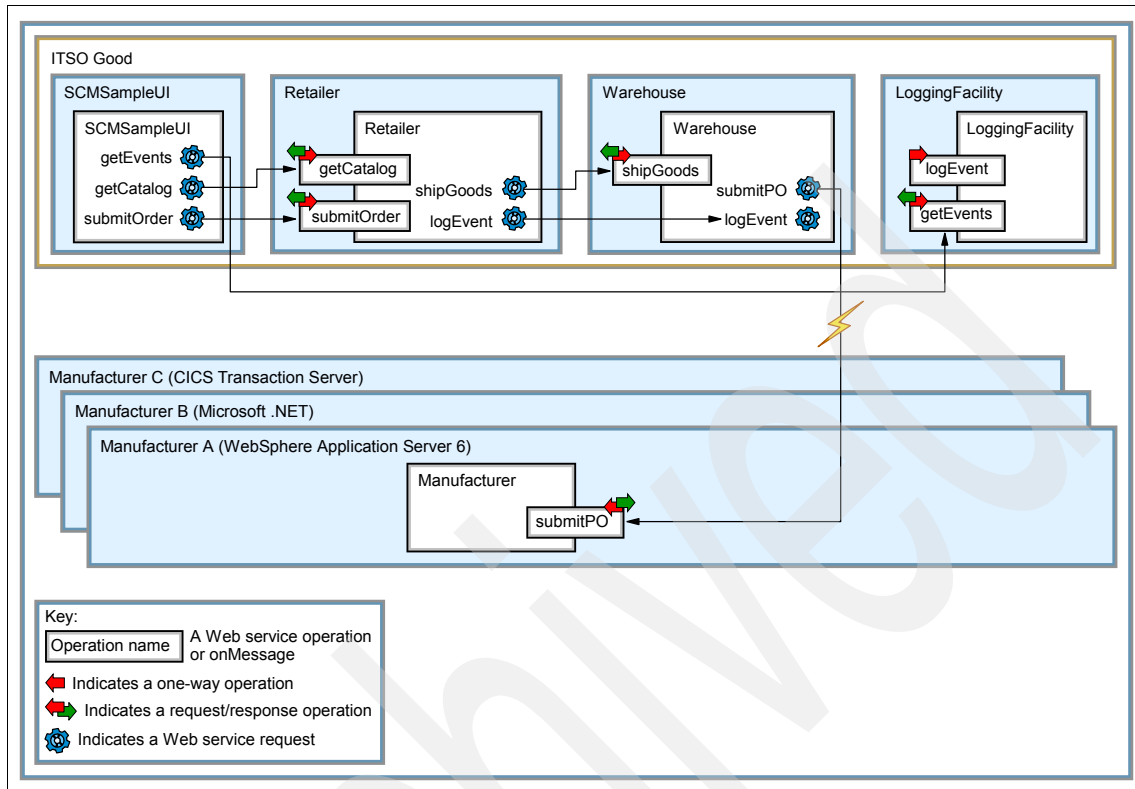


Figure 10-9 Scenario implementation using the Exposed Direct Connection pattern::Generic profile

Figure 10-9 shows how the application has been written and how it interacts with, other components. Shown are the enterprise applications (blue boxes), Web services (white boxes), and the Web service operations (smaller white boxes such as getCatalog). The connectivity between the application's components is synchronous. Arrows connecting the operations indicate Web service invocations.

The application interacts as follows:

1. The SCMSampleUI application:
 - a. Provides a Web user interface.
 - b. Invokes the Retailer Web service to get a list of all the items for purchase.
 - c. Invokes the Retailer Web service to order an item.
 - d. Invokes the LoggingFacility to track an order.
2. When an order is submitted, the Retailer Web service:
 - a. Invokes the LoggingFacility to log events that occur in the order.

- b. Invokes the Warehouse to obtain whether the order can be shipped and, if so, has it shipped.
3. When a request to ship goods is made, the Warehouse Web service:
 - a. Determines if there is enough of an item in stock to ship the order.
 - If there is not enough quantity in stock, it refuses to ship the order.
 - If there is enough quantity in stock, it ships the order.
 - b. Determines if more of the goods need to be ordered:
 - If more goods need to be ordered, it submits a purchase order to the relevant Manufacturer.
 - If there is enough of a particular item in stock, it does nothing.
4. When a purchase order is submitted, the Manufacturer Web service receives and process it, then returns to the Warehouse.

The calls between the Warehouse Web service and the Manufacturer Web services are interenterprise across an untrusted network, therefore these calls need to be secured.

10.3.2 Securing applications using WS-Security

The WS-Security specification provides message-level security, which is used to implement message content integrity and confidentiality.

The advantage of using WS-Security rather than SSL is that WS-Security can provide end-to-end message-level security. This means that message security can be protected even if the message goes through multiple services called *intermediaries*.

Additionally, WS-Security is independent of the transport layer protocol; it can be used for any Web service binding (for example, HTTP, JMS, RMI). Using WS-Security, end-to-end security can be achieved (Figure 10-10).

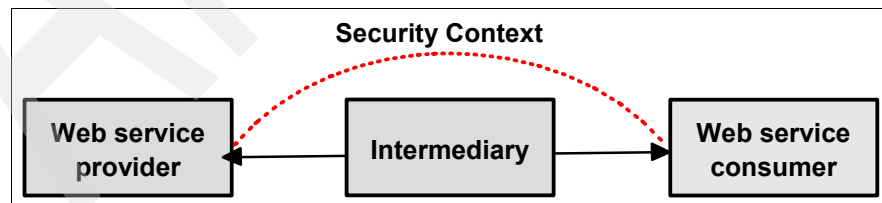


Figure 10-10 End-to-end security with message-level security

Configuring WS-Security

To secure a Web service application with WS-Security, it is necessary to define WS-Security configurations using Rational Application Developer V6.0 or the Application Server Toolkit. If the appropriate WS-Security configurations exist in the application EAR file, the WS-Security runtime is invoked when the SOAP message is outgoing or incoming and the SOAP messages are secured. This chapter provides information about how to configure WS-Security using Rational Application Developer V6.0.1.

There are a vast array of possible configurations using WS-Security. The particular configuration needed for a service is dictated by the security requirements of the project. To limit the scope of this chapter, we describe how to configure two mechanisms, Integrity and Confidentiality, to secure the request SOAP message from Warehouse to Manufacturer (Figure 10-11):

- Integrity is provided by applying a digital signature to a SOAP message.
- Confidentiality is applied by SOAP message encryption.

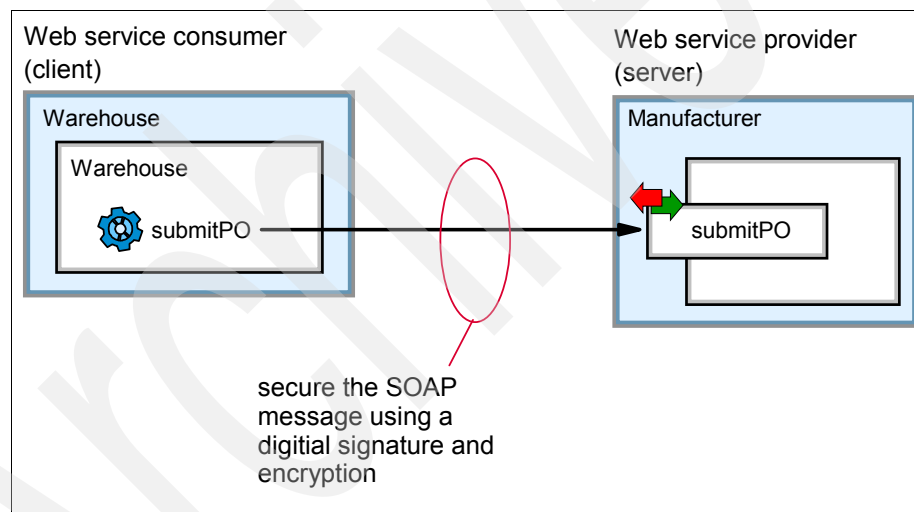


Figure 10-11 Securing the connection between the Warehouse and Manufacturer

The responses from the Manufacturer back to the Warehouse will not be signed or encrypted. A real WS-Security configuration will almost certainly be more complex than shown in this section and might involve both signing and encrypting requests and responses.

Example 10-1 on page 181 shows a partial SOAP message without WS-Security.

Example 10-1 SOAP message without applying security

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <p67:Configuration

xmlns:p67="http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08
/Configuration.xsd">
    <p67:UserId>1b63ce7e:10581adfdda:-7ff9</p67:UserId>
  </p67:Configuration>
  <p900:StartHeader>
```

Example 10-2 shows a partial SOAP message with integrity and confidentiality, appropriate when the message includes personal information, such as a credit card or bank account number. The whole SOAP body is signed and encrypted.

Example 10-2 SOAP message with WS-Security

```
<soapenv:Envelope
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secect-1.0.xsd">
    <wsse:BinarySecurityToken
EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-mess
age-security-1.0#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-p
rofile-1.0#X509"
    wsu:Id="x509bst_5963606011209195163"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-u
tility-1.0.xsd">

MIIBzjCCATegAwIBAgIEQueSaTANBgkqhkiG9w0BAQQFADAsMQswCQYDVQQGEwJVUzEMMAoGA1UECxM
DSUJNMQ8wDQYDVQQDEwZDbG11bnQwHhcNMDUwNzI3MTM1NTUzWWhcNMDkwNzI2MTM1NTUzWjAsMQswCQ
YDVQQGEwJVUzEMMAoGA1UECxMDSUJNMQ8wDQYDVQQDEwZDbG11bnQwZ4wDQYJKoZIhvcNAQEBBQADg
YwAMIGIAoGAYg6ZmQFokKxQmupTPGOGNeJAFxx8EWhSmhwQzm4Hww7WVv3BUvjKtv4Do02aeTXo/e2f
07oq5cPn7ZazKSkV48p30t32UHZisJ4ZyRG13iu3W9HFSGDiskLNIwYxN8NDE7+Vxr1gf/Q0Fnn7J9M
YgtKV7oDH4wTds5gUQg0qB6UCAwEAATANBgkqhkiG9w0BAQQFAA0BgQA789EMk9oaw3cpYbKs1VjZbz
```

LCIQ2XSscYeyj0Vh7H930/7wJwG+7rgeL1n3lwn47gpfnJyeIOaYIGDBKFbzJ4mKKK+bWjpNjMiJ0r
DqH5RH+xL3DMYhqLCDLH+6Uh3JmUbBhzhfjI4uZxfCHpzR5AbIoeWWKcLiTnSFLZIIeTQ==
</wsse:BinarySecurityToken>

How to define a WS-Security configuration

While many steps are necessary to configure WS-Security, we describe how to configure two security mechanisms: integrity, and confidentiality. Figure 10-12 shows an overview of the resources that need to be defined for these mechanisms.

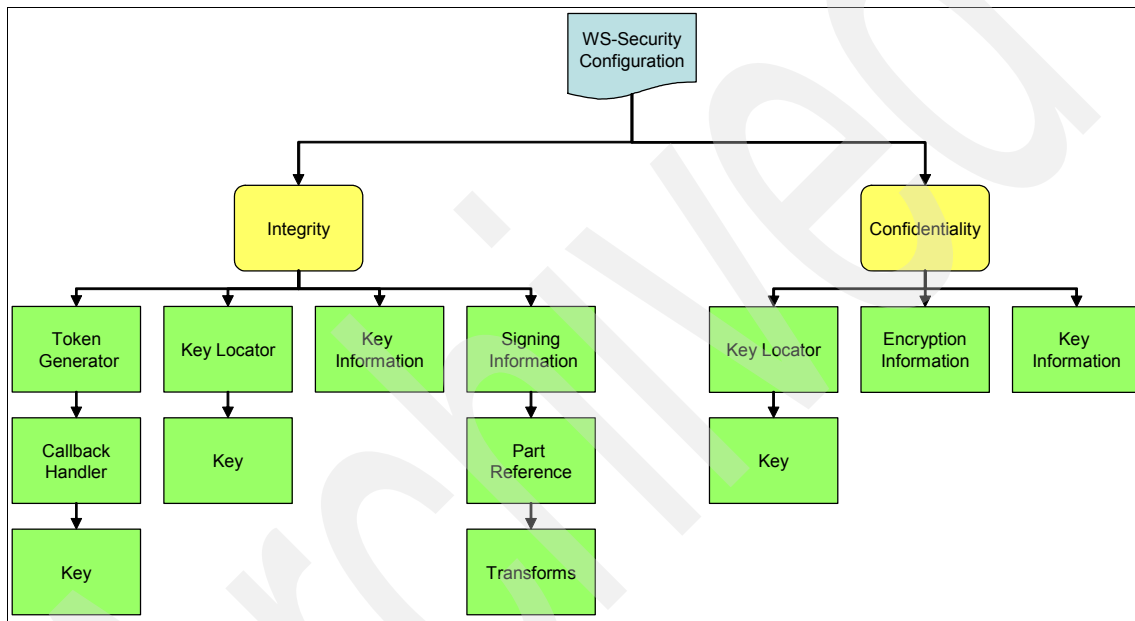


Figure 10-12 WS-Security binding elements

You can use GUI editors for specifying WS-Security configurations for both the client and server. Rational Application Developer V6.0 provides two GUI editors for WS-Security configuration: one for the client and one for the server.

Here is the outline of the steps used to configure WS-Security in our sample application:

1. Configure WS-Security for sending the request message (WarehouseEJB - client)

Edit the `ejb-jar.xml` file for the request generator with the Deployment Descriptor Editor.

2. Configure WS-Security for receiving the request message (Manufacturer - server)

Edit the `webservices.xml` file for the request consumer with the Web Services Editor.

Note: The client-side security information is accessed through the Deployment Descriptor Editor of the client module:

- ▶ For a Web client (servlet, JSP™, JavaBean), edit the `web.xml` file.
- ▶ For an EJB client (a session bean accessing a Web service), edit the `ejb-jar.xml` file.
- ▶ For a J2EE application client, edit the `application-client.xml` file.

In each editor, there are two tabs for WS-Security configuration:

- ▶ **WS Extension** — *What* security measures to apply
- ▶ **WS Binding** — *How* to apply the security measures

Examining the client configuration

Before you configure WS-Security, import the projects where security needs to be added into a Rational Application Developer workspace. The projects for this scenario are located in the `DirectConnection\projects` directory of the additional material that is supplied with this book (see Appendix A, “Additional material” on page 481).

To import the projects:

1. In Rational Application Developer, click **File** → **Import**. Select **Project Interchange** and click **Next**.
2. Locate **DirectConnectionProjects.zip** in the `DirectConnection\projects` directory of the additional material that is supplied with this book.
3. Select all the projects in this file then click **Finish**. Ignore all errors that appear in the Problems view.

Note: The projects you import contain WSDL files. These WSDL files import resources from an HTTP server with the address `http://appsrv1a.itso.ral.ibm.com`. In order to resolve these errors in the WSDL file you need to configure this HTTP server. See 10.4.3, “Hosting the WSDL files” on page 219 for details.

For the client configuration, you access the GUI for the WS-Security configuration from the Deployment Descriptor Editor:

1. To open the GUI in Rational Application Developer, expand the client project (**EJB Projects** → **WarehouseEJB**) in the Project Explorer and double-click the deployment descriptor. You can alternatively expand **ejbModule** → **META-INF** and open the **ejb-jar.xml** file directly.
2. When the Deployment Descriptor Editor opens, select the **WS Extension** or **WS Binding** tab. The WS Extension page is for editing the client's deployment descriptor extension file, so you can specify what security is required. The WS Binding page is for editing the client's binding file, so you can specify how to apply the required security. Figure 10-13 shows the WS-Binding page in the Deployment Descriptor Editor.

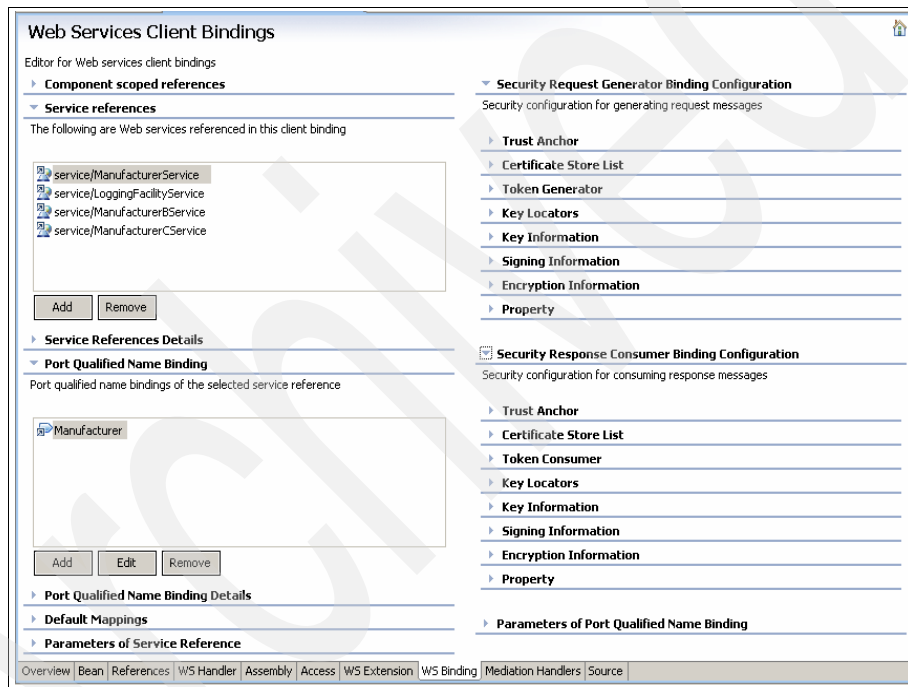


Figure 10-13 WS-Binding page in the deployment descriptor

Examining the server configuration

For the server configuration, you access the GUI for the WS-Security configuration using the Web Services Editor.

1. To open the GUI, expand the server project (**Dynamic Web Projects** → **ManufacturerWeb**) in the Project Explorer and double-click the **webservices.xml** file under **WebContent** → **WEB-INF**.

- When the Deployment Descriptor Editor opens, select the **Extensions** or **Binding Configurations** tab. The Extensions page is for editing the server's extension file, so you can specify what security is required. The Binding Configurations page is for editing the server's binding file, so you can specify how to apply the required security. Figure 10-14 shows the Extensions page in the Web Services Editor.

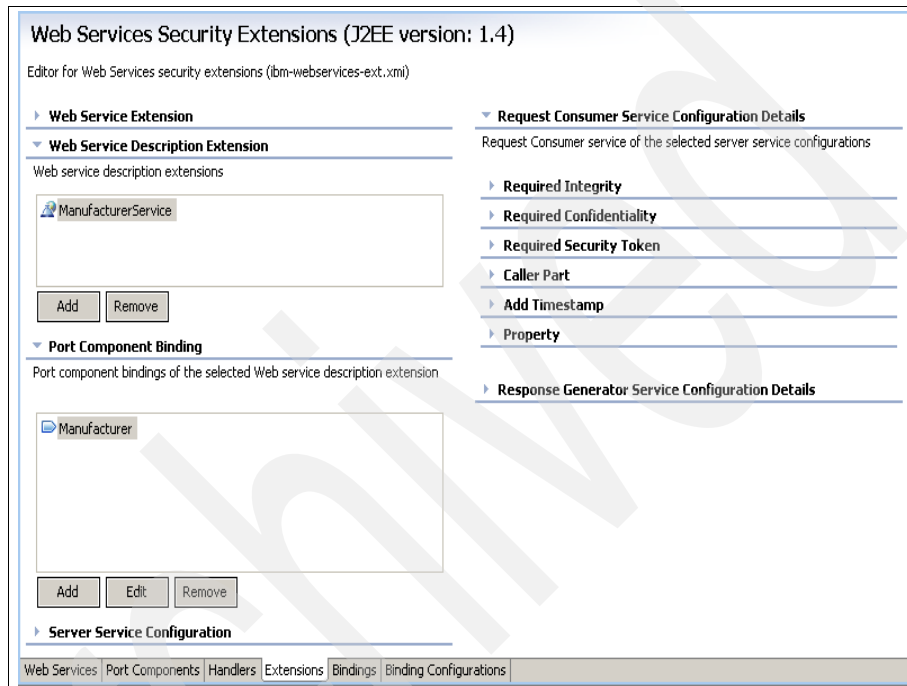


Figure 10-14 Extensions page in the Web Services Editor

10.3.3 Generating sample key stores

Before starting the WS-Security configuration, you have to prepare the client and server key stores to sign or encrypt the message. This section shows how to create a sample key store for testing WS-Security. If you want to apply WS-Security to a real application, you should prepare the appropriate certificate and keys. Do not use this sample key store for a real application.

We create two key stores that contain the following keys. The keys are created by using the Java keytool:

Client key

Algorithm	RSA
-----------	-----

Distinguished Name	CN=Client, OU=IBM, C=US
Key size	1024
Storepass	client
Storetype	JKS
Keypass	client_rsa
Alias	client_rsa
Key store file	client.jks
Certificate file	client_rsa.cer

Server key

Algorithm	RSA
Distinguished Name	CN=Server, OU=IBM, C=US
Key size	1024
Storepass	server
Storetype	JKS
Keypass	server_rsa
Alias	server_rsa
Key store file	server.jks
Certificate file	server_rsa.cer

Using the Java keytool

To generate sample key stores, follow these steps:

1. Go to the directory where you want to generate the key stores. Here, we generate key stores in <WAS_HOME>\etc\ws-security\samples\ITSGood.
2. Run the keytool in a command prompt as follows. The keytool is located in <RAD_HOME>\runtimes\base_v6\java\jre\bin if you are using the Integrated Test Server, or <WAS_HOME>\java\jre\bin if you are using a standalone WebSphere Application Server.

- a. Set the PATH environment variable:

```
set PATH=<RAD60_HOME>\runtimes\base_v6\java\jre\bin;%PATH%
```

- b. Generate a client RSA key:

```
keytool -genkey -keyalg RSA -keystore client.jks -storetype jks
-storepass client -alias client_rsa -keypass client_rsa -dname
"CN=Client, OU=IBM, C=US" -keysize 1024 -validity 1460
```

- c. Generate a server RSA key:

```
keytool -genkey -keyalg RSA -keystore server.jks -storetype jks
-storepass server -alias server_rsa -keypass server_rsa -dname
"CN=Server, OU=IBM, C=US" -keysize 1024 -validity 1460
```

- d. Export a client RSA certificate:

```
keytool -export -keystore client.jks -storetype jks -storepass client
-alias client_rsa -file client_rsa.cer
```

e. Import a client RSA certificate to a server key store:

```
keytool -import -noprompt -keystore server.jks -storetype jks -storepass  
server -alias client_rsa -file client_rsa.cer
```

f. Export a server RSA certificate:

```
keytool -export -keystore server.jks -storetype jks -storepass server  
-alias server_rsa -file server_rsa.cer
```

g. Import a server RSA certificate to a client key store:

```
keytool -import -noprompt -keystore client.jks -storetype jks -storepass  
client -alias server_rsa -file server_rsa.cer
```

3. You should now have two key store files and two certificate files:

Client key store file	client.jks
Server key store file	server.jks
Client certificate file	client_rsa.cer
Server certificate file	server_rsa.cer

4. To verify the key information in these key stores, run the following commands:

```
keytool -list -keystore client.jks -storepass client -v  
keytool -list -keystore server.jks -storepass server -v
```

The expected output is shown in Figure 10-3.

Example 10-3 Expected output from the keytool command

```
Keystore type: jks  
Keystore provider: IBMJCE
```

Your keystore contains 2 entries

```
Alias name: client_rsa  
Creation date: Nov 2, 2005  
Entry type: keyEntry  
Certificate chain length: 1  
Certificate[1]:  
Owner: CN=Client, OU=IBM, C=US  
Issuer: CN=Client, OU=IBM, C=US  
Serial number: 43684e07  
Valid from: 11/2/05 1:26 PM until: 11/1/09 1:26 PM  
Certificate fingerprints:  
MD5: DD:EA:E6:60:6F:C1:2A:C5:09:0B:17:F3:D2:CA:2E:0D  
SHA1: A0:16:70:58:91:F4:0B:3D:DA:0A:F8:B1:BD:CF:83:5E:43:43:98:BB
```

```
*****  
*****
```

Alias name: server_rsa
Creation date: Nov 2, 2005
Entry type: trustedCertEntry

Owner: CN=Server, OU=IBM, C=US
Issuer: CN=Server, OU=IBM, C=US
Serial number: 43685105
Valid from: 11/2/05 1:39 PM until: 11/1/09 1:39 PM
Certificate fingerprints:
MD5: 5B:D9:3A:C4:1D:5B:C6:C0:6E:77:04:5E:48:E2:F2:70
SHA1: 26:AD:2A:54:8F:F8:20:4D:B8:38:57:2A:C1:83:F9:29:68:35:CB:EE

Important: When using the keytool, only a self-signed certificate can be created. If you want to create a certificate path, you have to use another tool. More detailed information about creating certificates and keys is provided in *WebSphere Application Server V6: Security Handbook*, SG24-6316.

10.3.4 Configuring WS-Security integrity

Integrity is provided by applying a digital signature to a SOAP message. This section describes how to configure integrity in Rational Application Developer. It contains the following steps:

- ▶ Configuring the client integrity
- ▶ Configuring the server integrity
- ▶ Integrity on the response message

WS-Security integrity requires a number of components to be defined, as highlighted in Figure 10-15 on page 189.

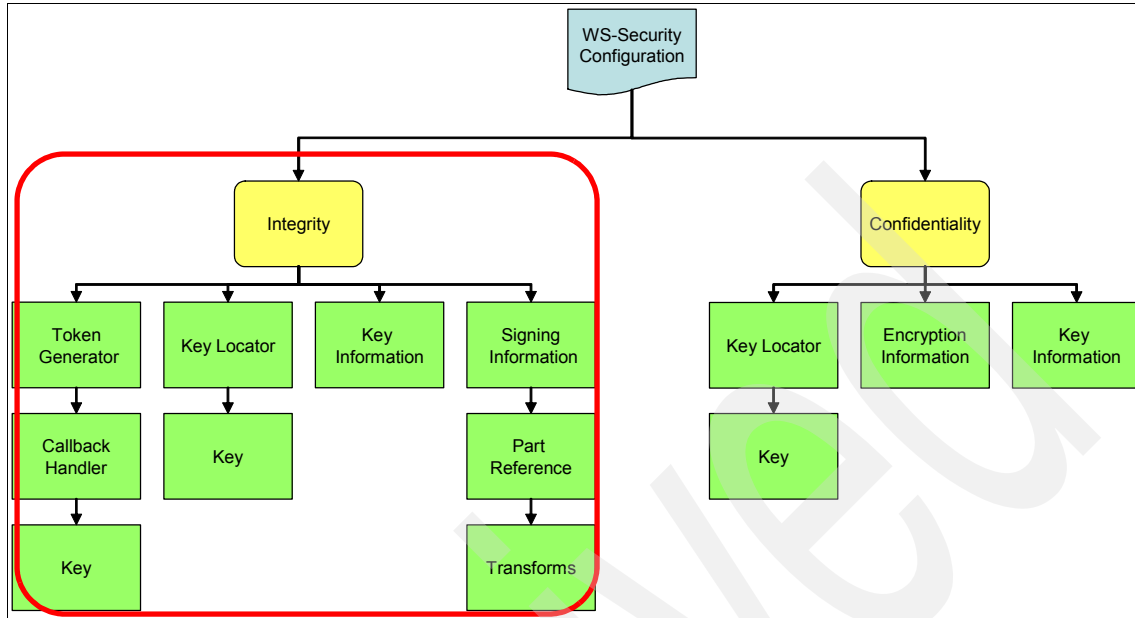


Figure 10-15 WS-Security binding elements - integrity

Configuring the client integrity

To configure integrity for a request message sent by a client, open the WarehouseEJB deployment descriptor and go to the **WS Extension** tab.

1. Under Service References select **service/ManufacturerService**.
2. Expand **Request Generator Configuration**, then expand **Integrity**.
3. Click **Add** and enter the following information in the Integrity dialog window (Figure 10-16 on page 190):
 - a. Enter a name identifying the part in the Integrity Name field of the value `int_body`.
 - b. Select the order in which the signature is generated. In our case, we select **1**.
 - c. Click **Add** under Message Parts. This creates one integrity part. The default created part is for signing the SOAP body. If you want to sign the SOAP body only, you have nothing more to do.
 - d. Click **OK**.

The Integrity Dialog window is shown with the following configuration:

- Integrity Name:** `int_body`
- Order:** `1`
- Message Parts:**

Message parts dialect	Message parts keyword
<code>http://www.ibm.com/websphere/webservices...</code>	<code>body</code>
- Nonce:**

Nonce dialect	Nonce keyword
- Timestamp:**

Timestamp dialect	Timestamp keyword	Timestamp exp

Buttons: Add, Remove, OK, Cancel.

Figure 10-16 Integrity dialog

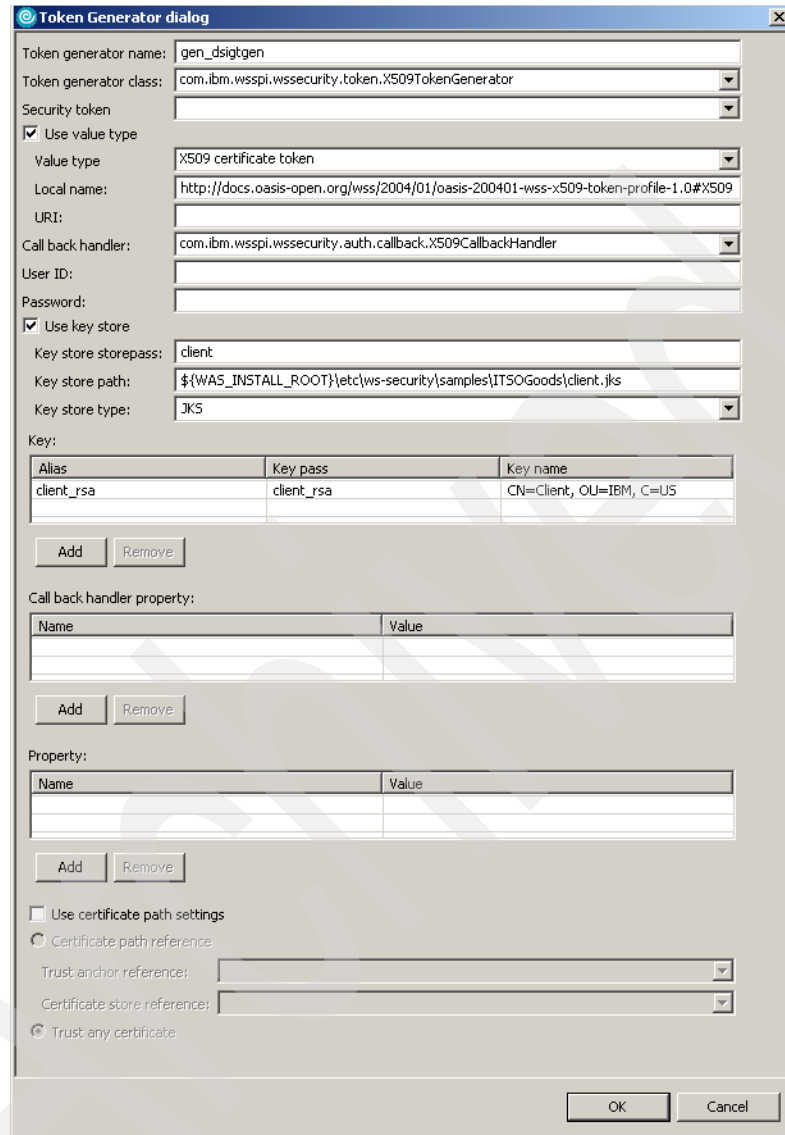
4. Save the configuration.

The corresponding integrity part information must be specified on the WS Binding page:

1. Click the **WS Binding** tab and expand **Security Request Generator Binding Configuration**, then expand **Token Generator**.
2. To insert a security token into the message for signing click **Add** and perform the following:
 - a. Enter a Token generator name of `gen_dsigtgen`.
 - b. For the Token generator class, select the field ending in **X509TokenGenerator**.
 - c. Do not select a Security token.
 - d. Select **Use value type**, and then select **X509 certificate token** in the Value type field and the **X509CallbackHandler** in the Call back Handler field.

- e. Select **Use key store**. In our case, we enter `client` as the storepass, `${WAS_INSTALL_ROOT}\etc\ws-security\samples\ITSGood\client.jks` as the key store path, and select **JKS** as key store type.
- f. Click **Add** under Key and enter `client_rsa` as the alias, `client_rsa` as the key pass, and `CN=Client, OU=IBM, C=US` as the key name.
- g. Click **OK**, and a token generator is created. Save the changes.

Figure 10-17 on page 192 shows the Token Generator dialog for specifying a signature by an X.509 certificate.



Token Generator dialog

Token generator name:

Token generator class:

Security token:

☒ Use value type

Value type:

Local name:

URI:

Call back handler:

User ID:

Password:

☒ Use key store

Key store storepass:

Key store path:

Key store type:

Key:

Alias	Key pass	Key name
client_rsa	client_rsa	CN=Client, OU=IBM, C=US

Call back handler property:

Name	Value

Property:

Name	Value

☐ Use certificate path settings

☐ Certificate path reference

Trust anchor reference:

Certificate store reference:

☒ Trust any certificate

Figure 10-17 Token Generator dialog for signing by an X.509 certificate

3. Expand **Key Locators** and click **Add**. Specify how to retrieve a key for signing:
 - a. Enter a Key locator name of gen_dsiglocator.

- b. Select or enter a **Key locator class name**. The class to retrieve a key implements the `com.ibm.wsspi.wssecurity.keyinfo.KeyLocator` interface. In our case, we select the **KeyStoreKeyLocator**.
- c. Specify the key store and key by selecting **Use key store**. Enter the same information as in the previous dialog, including the key store details and the addition of a key called `client_rsa`.
- d. Click **OK**, and a key locator is created.

Figure 10-18 shows the Key Locator dialog for specifying a signature by an X.509 certificate.

Key Locator dialog

Key locator name:

Key locator class:

☒ Use key store

Key store storepass:

Key store path:

Key store type:

Key:

Alias	Key pass	Key name
client_rsa	client_rsa	CN=Client, OU=IBM, C=US

Property:

Name	Value
------	-------

Figure 10-18 Key Locator dialog for signing by an X.509 certificate

4. Expand **Key Information** and click **Add**. Specify which type of security token reference is used:
 - a. Enter a name of `gen_dsigkeyinfo`.
 - b. Select **STRREF** in the Key information type. The key information class name is filled in when a type is selected.
 - c. Select **Use key locator** and select a Key locator from the list. Key locators that have been defined are listed. Select **gen_dsiglocator**. Also select the predefined key **CN=Client, OU=IBM, C=US** as the key name.
 - d. If a security token is inserted into the message, and a token generator is specified already, select which token generator is used. Select **Use token**

and select **gen_dsigtgen** from the list. The selected token generator is invoked to generate the token that is referenced from this key information. Click **OK**, and the key information is created.

Figure 10-19 shows a Key Information dialog for specifying a signature by an X.509 certificate.

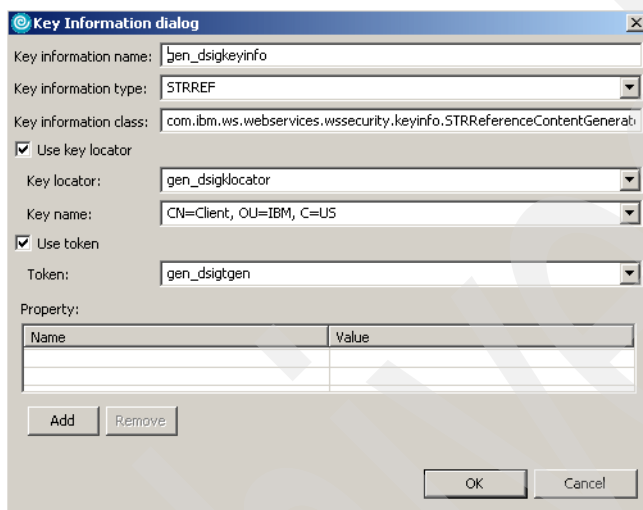
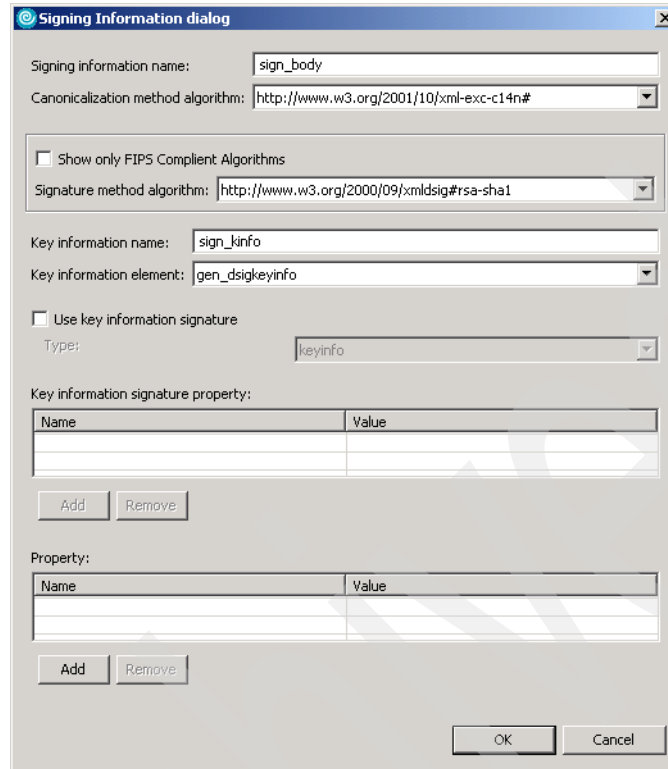


Figure 10-19 Key Information dialog for signing by an X.509 certificate

5. Expand **Signing Information** and click **Add**. You have to specify how to sign:
 - a. Enter a name of sign_body.
 - b. Select the Canonicalization method algorithm **<http://www.w3.org/2001/10/xml-exc-c14n#>**.
 - c. Select the Signature method algorithm **<http://www.w3.org/2000/09/xmldsig#rsa-sha1>**
 - d. Enter a Key information name of sign_kinfo.
 - e. Select a Key information element from the list. Predefined key information is listed. Select **gen_dsigkeyinfo**.
 - f. Click **OK**, and the signing information is created.

Figure 10-20 on page 195 shows a Signing Information dialog for specifying a signature by an X.509 certificate.



The image shows a 'Signing Information dialog' window. It contains several input fields and checkboxes. The 'Signing information name' is 'sign_body'. The 'Canonicalization method algorithm' is 'http://www.w3.org/2001/10/xml-exc-c14n#'. There is a checkbox for 'Show only FIPS Compliant Algorithms' which is unchecked. The 'Signature method algorithm' is 'http://www.w3.org/2000/09/xmldsig#rsa-sha1'. The 'Key information name' is 'sign_kinfo'. The 'Key information element' is 'gen_dsigkeyinfo'. There is a checkbox for 'Use key information signature' which is unchecked. The 'Type' is 'keyinfo'. Below this is a section for 'Key information signature property' with a table with two columns: 'Name' and 'Value'. There are 'Add' and 'Remove' buttons below the table. At the bottom, there is a 'Property' section with another table with 'Name' and 'Value' columns, and 'Add' and 'Remove' buttons. At the very bottom are 'OK' and 'Cancel' buttons.

Signing information name:

Canonicalization method algorithm:

☐ Show only FIPS Compliant Algorithms

Signature method algorithm:

Key information name:

Key information element:

☐ Use key information signature

Type:

Key information signature property:

Name	Value

Property:

Name	Value

Figure 10-20 Signing Information dialog for signing by an X.509 certificate token

6. Expand **Part References** and click **Add** to specify an integrity part that applies to this signature information:
 - a. Enter a Part reference name of sign_part.
 - b. Select an Integrity part from the list of parts defined on the WS Extensions page. In our case, we select int_body.
 - c. Select a Digest method algorithm from the list. The supported digest method algorithm is **http://www.w3.org/2000/09/xmldsig#sha1**.
 - d. Click **OK**, and a part reference is created.

Figure 10-21 on page 196 shows a Part Reference dialog for specifying a signature by an X.509 certificate.

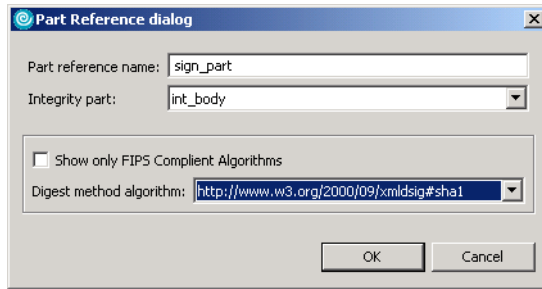


Figure 10-21 Part Reference dialog for signing by an X.509 certificate

7. After adding part references, adding a transform becomes enabled. Expand **Transforms** and click **Add** and complete the dialog:
 - a. Enter a name of sign_trans.
 - b. Select the <http://www.w3.org/2001/10/xml-exc-c14n#> transform algorithm from the list.
 - c. Click **OK**, and a transform is created.

Figure 10-22 shows a Transform dialog for specifying a signature by an X.509 certificate.

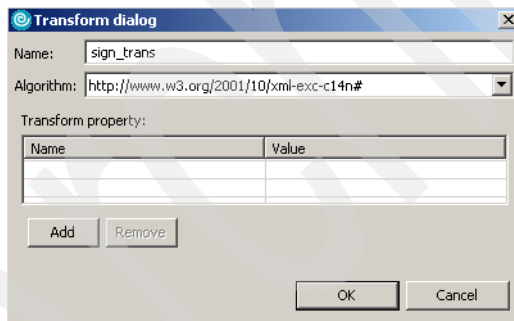


Figure 10-22 Transform dialog for signing by an X.509 certificate

8. Save your changes to the deployment descriptor.

Configuring the server integrity

To configure how to verify integrity, open the **webservices.xml** file in the **ManufacturerWeb** project with the Web Services Editor. To receive the digitally signed request message from a client, a server should configure signature verification on the Extensions page as follows.

1. Under Port Component Binding, click **Manufacturer** to highlight it.

2. Expand **Request Consumer Service Configuration Details**, then **Required Integrity** and click **Add**:
 - a. Enter a name denoting the part of **reqint_body**.
 - b. Select the Usage type, **Required**.
 - c. Click **Add** under Message Parts to specify an integrity required part. We select **http://....../dialect-was** and **body**.
 - d. Click **OK**, and a required integrity part is created. Save the changes.

Figure 10-23 shows a Required Integrity dialog for specifying that integrity is required on the SOAP body.

The image shows a 'Required Integrity Dialog' window. It has a title bar with a close button. Inside, there's a 'Required Integrity Name' field with 'reqint_body' entered. Below it is a 'Usage type' dropdown menu set to 'Required'. There are three main sections: 'Message Parts', 'Nonce', and 'Timestamp'. Each section has a table with columns for dialect, keyword, and (for Timestamp) expiration. The 'Message Parts' table has one row with 'http://www.ibm.com/websphere/webservices...' and 'body'. Below each table are 'Add' and 'Remove' buttons. At the bottom right are 'OK' and 'Cancel' buttons.

Message parts dialect	Message parts keyword
http://www.ibm.com/websphere/webservices...	body

Nonce dialect	Nonce keyword

Timestamp dialect	Timestamp keyword	Timestamp exp

Figure 10-23 Required Integrity dialog

3. We have to provide corresponding information about the binding configuration.
 - a. Select the **Binding Configurations** tab and expand **Request Consumer Binding Configuration Details**.
 - b. Add a Token Consumer; expand **Token Consumer** and click **Add**:
 - c. Enter a name of **con_dsgtcon**.

- d. Select the Token consumer class **X509TokenConsumer**.
- e. Do not select a security token.
- f. Select **Use value type** and select **X509 certificate token**.
- g. Select **Use jaas.config** and enter `system.wssecurity.X509BST` as the name.
- h. Select **Use certificate path settings** and select **Trust any certificate**.
- i. Click **OK**.

Figure 10-24 on page 199 shows a Token Consumer dialog for signature verification by X.509 certificates.

The dialog box is titled "Token Consumer dialog". It contains the following fields and options:

- Token consumer name:
- Token consumer class:
- Security token:
- ☒ Use value type
 - Value type:
- Local name:
- URI:
- ☒ Use jaas.config
 - jaas.config name:
 - jaas.config property:

Name	Value
- ☐ Use trusted ID evaluator
 - Trusted ID evaluator class:
 - Trusted ID evaluator property:

Name	Value
- ☐ Use trusted ID evaluator reference
 - Trusted ID evaluator reference:
 - Property:

Name	Value
- ☒ Use certificate path settings
 - ☐ Certificate path reference
 - Trust anchor reference:
 - Certificate store reference:
 - ☒ Trust any certificate

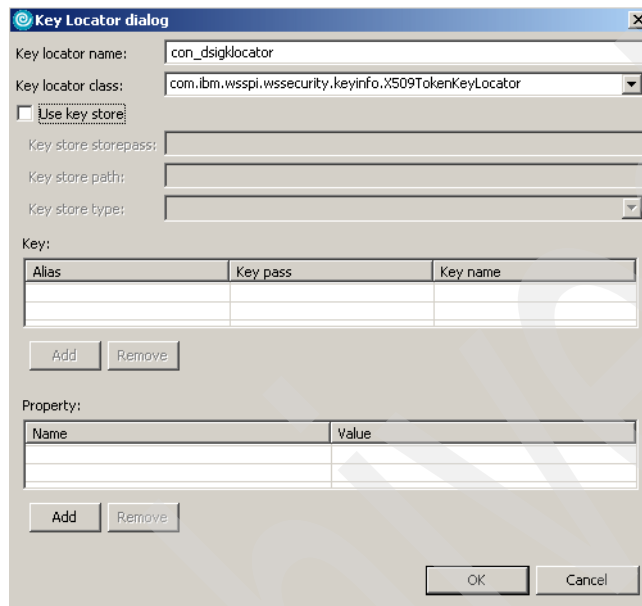
At the bottom right are and .

Figure 10-24 Token Consumer dialog for signature verification (X.509 certificate)

4. Expand **Key Locators** and click **Add** to specify how to retrieve a key for a signature verification:
 - a. Enter a name of `con_dsigtlocator`, and select the **X509TokenKeyLocator** class.

- b. Click **OK**.

Figure 10-25 shows a Key Locator dialog for signature verification by X.509 certificates.



The Key Locator dialog box contains the following fields and controls:

- Key locator name:** con_dsiglocator
- Key locator class:** com.ibm.wsspi.wsssecurity.keyinfo.X509TokenKeyLocator
- ☐ **Use key store:**
- Key store storepass:** (empty text field)
- Key store path:** (empty text field)
- Key store type:** (empty dropdown menu)
- Key:** (table with 3 columns: Alias, Key pass, Key name)
- Property:** (table with 2 columns: Name, Value)
- Buttons:** Add, Remove, OK, Cancel

Alias	Key pass	Key name

Name	Value

Figure 10-25 Key Locator dialog for signature verification (X.509 certificate)

5. Expand **Key Information** and click **Add** to specify which type of security token reference is required. The type should match the client configuration:
- Enter a name of con_dsigkeyinfo, select **STRREF** as the type.
 - Select **Use key locator** and set the Key locator to **con_dsiglocator**.
 - Select **Use token** and select the token **con_dsigtcon**. On the consumer side, we do not have to specify the key name.
 - Click **OK**.

Figure 10-26 on page 201 shows a Key Information dialog for signature verification by X.509 certificates.

The image shows a 'Key Information dialog' window. It contains the following fields and controls:

- Key information name:** con_dsigkeyinfo
- Key information type:** STRREF
- Key information class:** com.ibm.ws.webservices.wssecurity.keyinfo.STRReferenceContentConsume
- ☒ **Use key locator**
 - Key locator:** con_dsiglocator
 - Key name:**
- ☒ **Use token**
 - Token:** con_dsigtcon
- Property:**

Name	Value

At the bottom, there are 'Add' and 'Remove' buttons, and 'OK' and 'Cancel' buttons.

Figure 10-26 Key Information dialog for signature verification (X.509 certificate)

6. Expand **Signing Information** and click **Add** to define how to verify a required integrity part:
 - a. Enter a name of sign_body.
 - b. Select **http://www.w3.org/2001/10/xml-exc-c14n#** as the Canonicalization method algorithm and **http://www.w3.org/2000/09/xmldsig#rsa-sha1** as the Signature method algorithm.
 - c. For the Signing key information, click **Add**, enter sign_kinfo as the Key information name, and select **con_dsigkeyinfo** as the Key information element.
 - d. Click **OK**.

Figure 10-27 on page 202 shows a Signing Information dialog for signature verification by X.509 certificates.

Signing Information dialog

Signing information name:

Canonicalization method algorithm:

☐ Show only FIPS Compliant Algorithms

Signature method algorithm:

Signing key information:

Key information name:	Key information element:
sign_kinfo	con_dskeyinfo

☐ Use key information signature

Type:

Key information signature property:

Name	Value
------	-------

Property:

Name	Value
------	-------

Figure 10-27 Signing Information dialog for signature verification (X.509 certificate)

7. Expand **Part References** and click **Add** and complete the dialog:
 - a. Enter a name of sign_part.
 - b. Select **reqint_body** from the list and **http://www.w3.org/2000/09/xmldsig#sha1** as the algorithm.
 - c. Click **OK**.

Figure 10-28 on page 203 shows a Part Reference dialog for signature verification by X.509 certificates.

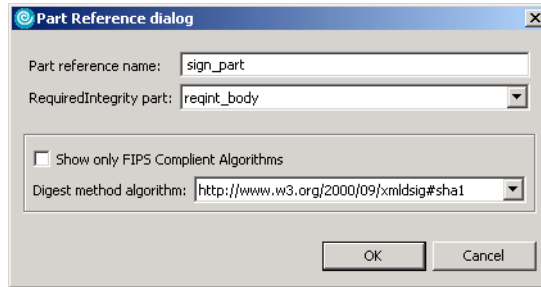


Figure 10-28 Part Reference dialog for signature verification (X.509 certificate)

8. Expand **Transforms** and click **Add** and complete the dialog:
 - a. Enter a name of sign_trans.
 - b. Select **http://www.w3.org/2001/10/xml-exc-c14n#** as the algorithm.
 - c. Click **OK**.

Figure 10-29 shows a Transform dialog for signature verification by X.509 certificates.

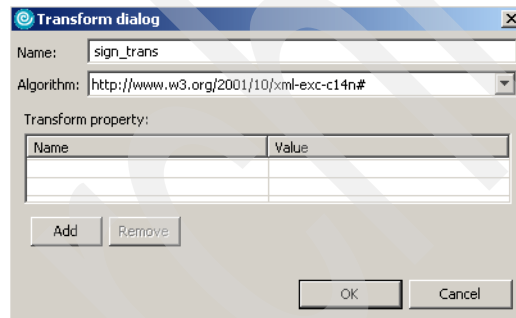


Figure 10-29 Transform dialog for signature verification (X.509 certificate)

9. Save the changes to the deployment descriptor.

Integrity on the response message

Optionally, you can add integrity on the response message from the server. To do this you need to configure the Response Generator in the server configuration and the Response Consumer in the client configuration:

- For the server configuration, you can configure the Response Generator Service Configuration Details in the Extensions page and the Response Generator Binding Configuration Details in the Binding Configurations page by referring to “Configuring the client integrity” on page 189.

- For the client configuration, you can configure the Response Consumer Configuration in the WS Extension page and the Security Response Consumer Binding Configuration in the WS Binding page by referring to “Configuring the server integrity” on page 196.

Securing the response message in a real scenario is important, but for the simple scenario in this chapter we do not secure the response message.

10.3.5 Configuring WS-Security confidentiality

Confidentiality is applied by SOAP message encryption. This section describes how to configure confidentiality in Rational Application Developer. It contains the following steps:

1. Configuring the client confidentiality
2. Configuring the server confidentiality
3. Confidentiality on the response message

WS-Security confidentiality requires a number of components to be defined, as highlighted in Figure 10-30.

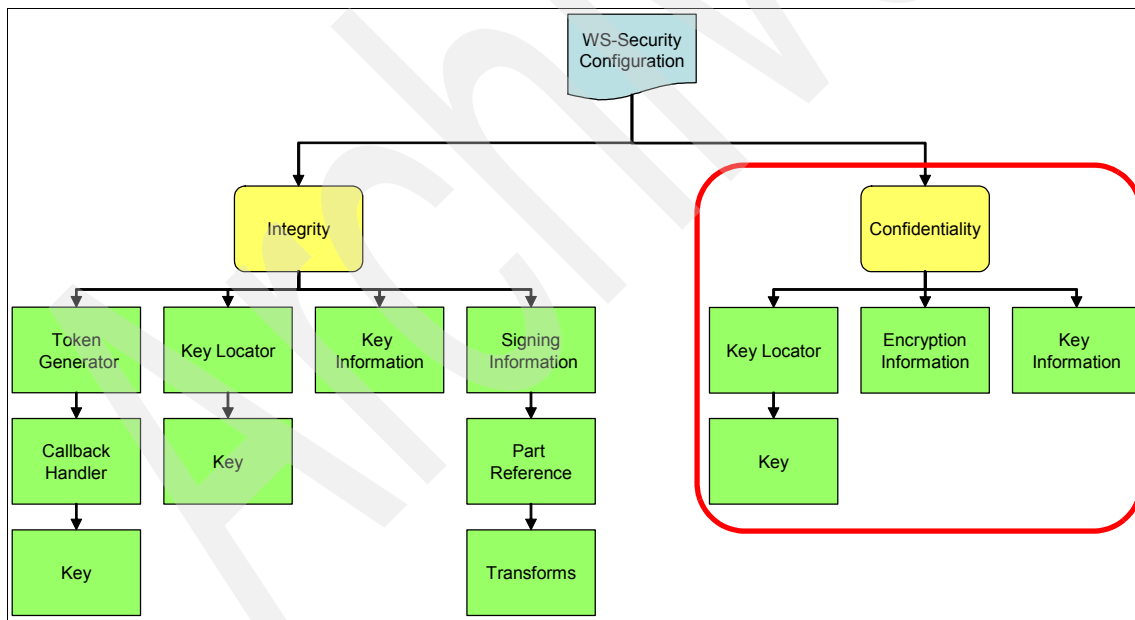


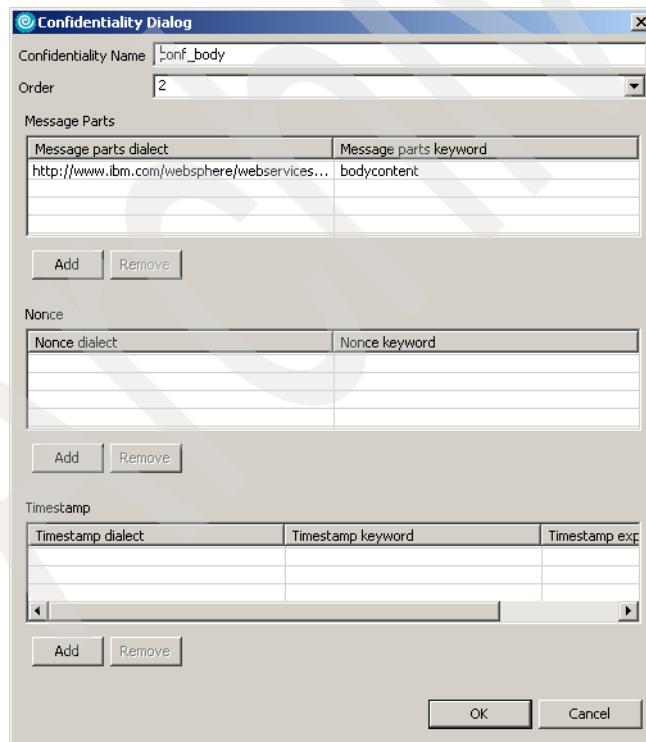
Figure 10-30 WS-Security binding elements - confidentiality

Configuring client confidentiality

To configure confidentiality for a request message sent by a client, open the WarehouseEJB deployment descriptor and go to the **WS Extension** page.

1. Expand **Request Generator Configuration**, expand **Confidentiality**, then click **Add**:
 - a. Enter a name of `conf_body`.
 - b. Select the order in which the encryption is generated. In our case, we select **2**.
 - c. Click **Add** for Message Parts, and select **http://.../dialect-was** and **bodycontent** as the keyword.
 - d. Click **OK**, and a confidentiality part is created. If you need multiple confidentiality parts, you can add more.
2. Save the configuration.

Figure 10-31 shows a Confidentiality dialog for specifying the encryption of the SOAP body content.



The image shows a 'Confidentiality Dialog' window. It has a title bar with a close button. The dialog is divided into several sections. The first section is 'Confidentiality Name' with a text box containing 'conf_body'. Below it is an 'Order' dropdown menu set to '2'. The next section is 'Message Parts', which contains a table with two columns: 'Message parts dialect' and 'Message parts keyword'. The first row has 'http://www.ibm.com/websphere/webservices...' and 'bodycontent'. Below the table are 'Add' and 'Remove' buttons. The next section is 'Nonce', which contains a table with two columns: 'Nonce dialect' and 'Nonce keyword'. Below the table are 'Add' and 'Remove' buttons. The final section is 'Timestamp', which contains a table with three columns: 'Timestamp dialect', 'Timestamp keyword', and 'Timestamp exp'. Below the table are 'Add' and 'Remove' buttons. At the bottom right of the dialog are 'OK' and 'Cancel' buttons.

Message parts dialect	Message parts keyword
http://www.ibm.com/websphere/webservices...	bodycontent

Nonce dialect	Nonce keyword
---------------	---------------

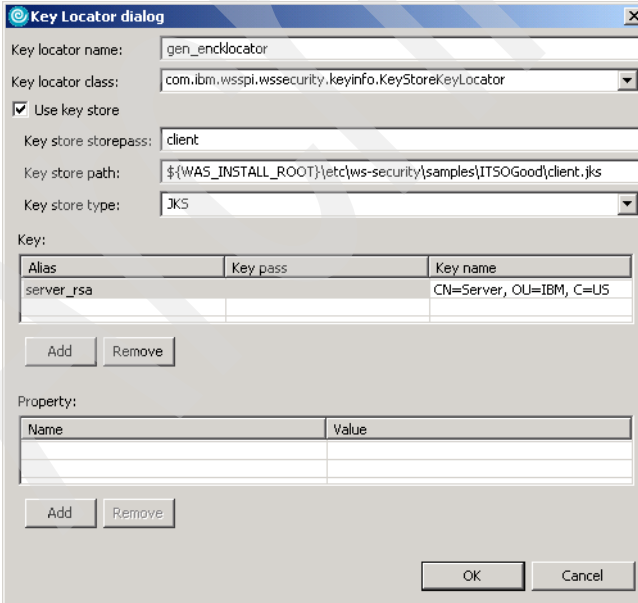
Timestamp dialect	Timestamp keyword	Timestamp exp
-------------------	-------------------	---------------

Figure 10-31 Confidentiality dialog for body content encryption

After specifying a confidentiality part, the corresponding information must be specified. Go to the **WS Binding** page and complete the following:

1. Expand **Security Request Generator Binding Configuration**. Expand **Key Locators** and click **Add** to specify how to retrieve a key for encryption:
 - a. Enter a name of `gen_encklocator`.
 - b. Select **KeyStoreKeyLocator** as the class name. The class to retrieve a key implements the `com.ibm.wsspi.wssecurity.keyinfo.KeyLocator` interface.
 - c. Select **Use key store** and specify a client key store and server public key. We specify `client`, `${WAS_INSTALL_ROOT}\etc\ws-security\samples\ITSOGood\client.jks` and **JKS**.
 - d. Click **Add** under Key to define the key. Enter `server_rsa` as the alias and `CN=Server, OU=IBM, C=US` as the key name. Key pass should be empty, because a client does not know the key password of a server key in the client key store.
 - e. Click **OK**.

Figure 10-32 shows a Key Locator dialog for specifying the encryption of the SOAP body content.



The Key Locator dialog box is shown with the following configuration:

- Key locator name:** `gen_encklocator`
- Key locator class:** `com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator`
- ☒ **Use key store**
- Key store storepass:** `client`
- Key store path:** `${WAS_INSTALL_ROOT}\etc\ws-security\samples\ITSOGood\client.jks`
- Key store type:** `JKS`
- Key:**

Alias	Key pass	Key name
<code>server_rsa</code>		<code>CN=Server, OU=IBM, C=US</code>

Buttons: Add, Remove
- Property:**

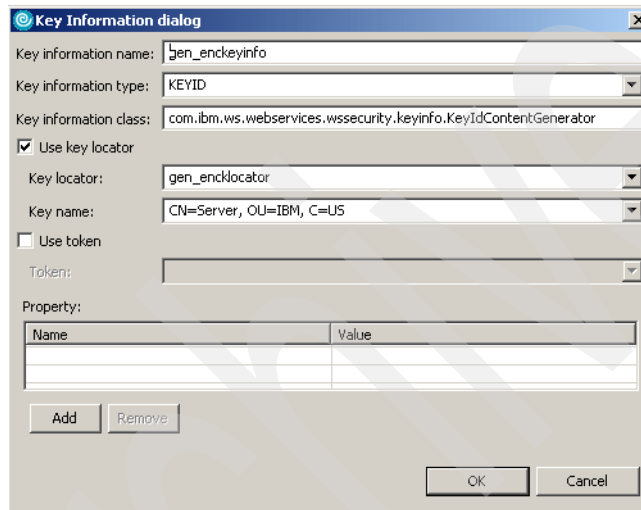
Name	Value
------	-------

Buttons: Add, Remove
- Buttons:** OK, Cancel

Figure 10-32 Key Locator dialog for body content encryption

2. Expand **Key Information** and click **Add** to specify which type of key reference is used:
 - a. Enter a name of `gen_enckeyinfo`, and select a type of **KEYID** from Key information type list.
 - b. Select **Use key locator**, and then select `gen_encklocator` and **CN=Server, OU=IBM, C=US**.
 - c. Click **OK**.

Figure 10-33 shows a Key Information dialog for specifying the encryption of the SOAP body content.



The image shows a 'Key Information dialog' window. It contains several input fields and checkboxes. The 'Key information name' field is set to 'gen_enckeyinfo'. The 'Key information type' dropdown is set to 'KEYID'. The 'Key information class' field is set to 'com.ibm.ws.webservices.wssecurity.keyinfo.KeyIdContentGenerator'. The 'Use key locator' checkbox is checked. The 'Key locator' dropdown is set to 'gen_encklocator'. The 'Key name' dropdown is set to 'CN=Server, OU=IBM, C=US'. The 'Use token' checkbox is unchecked. The 'Token' dropdown is empty. Below these fields is a 'Property:' section with a table with two columns: 'Name' and 'Value'. The table is currently empty. At the bottom of the dialog are 'Add' and 'Remove' buttons, and at the very bottom are 'OK' and 'Cancel' buttons.

Name	Value
------	-------

Figure 10-33 Key Information dialog for body content encryption

3. Expand **Encryption Information** and click **Add** and specify how to encrypt:
 - a. Enter a name of `enc_body`.
 - b. Select `http://www.w3.org/2001/04/xmlenc#tripledes-cbc` as Data encryption method algorithm.
 - c. Select `http://www.w3.org/2001/04/xmlenc#rsa-1_5` as Key encryption method algorithm from the list.
 - d. Enter Key information name to specify the key information reference. We specify `enc_keyinfo`.
 - e. Select a Key information element from the list of the key information that was defined. The selected key information is used for encryption. In our case, we select `gen_enckeyinfo` from the list.

- f. Select a Confidentiality part from the list of confidentiality parts that were defined in the extensions. In our case, we select **conf_body** from the list.
- g. Click **OK**, and the encryption information is created.

Figure 10-34 shows an Encryption Information dialog for specifying the encryption of the SOAP body content.

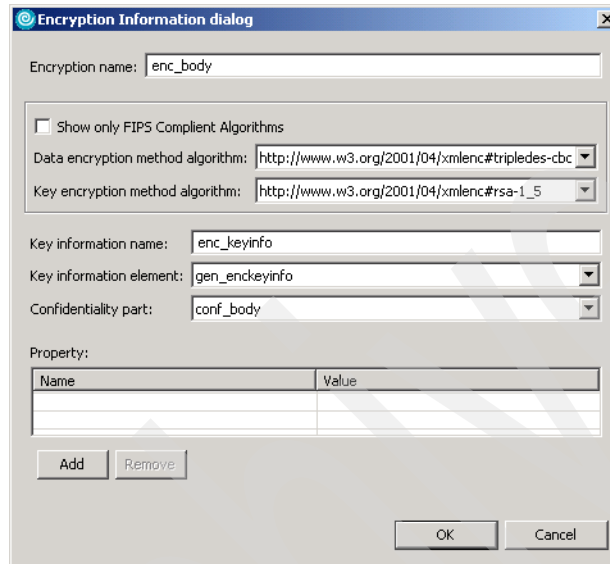


Figure 10-34 Encryption Information dialog for body content encryption

4. Save the changes to the deployment descriptor.

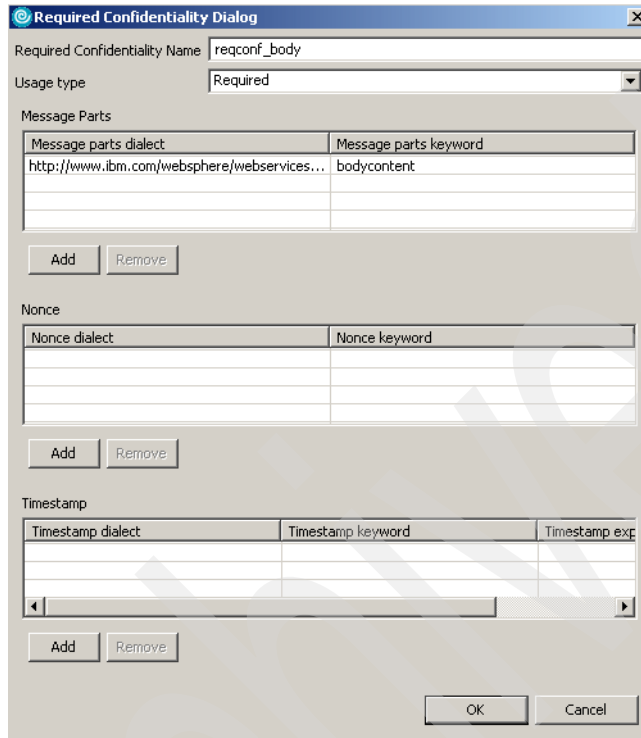
Configuring server confidentiality

To configure how to decrypt the message, open the **webservices.xml** file in the **ManufacturerWeb** project with the Web Services Editor.

To receive an encrypted message from a client, the server should configure how to decrypt the message on the **Extension** page under **Request Consumer Service Configuration Details** as follows:

1. Expand **Required Confidentiality** and click **Add**:
 - a. Enter a name of reqconf_body.
 - b. Select the Usage type **Required**.
 - c. Click **Add** for Message Parts and type `http://.../dialect-was` and `bodycontent` as the keyword.
 - d. Click **OK**, and a required confidentiality part is created.

Figure 10-35 shows a Required Confidentiality dialog for decryption of the SOAP body content.



The dialog box is titled "Required Confidentiality Dialog". It contains the following sections:

- Required Confidentiality Name:** reqconf_body
- Usage type:** Required (dropdown menu)
- Message Parts:**

Message parts dialect	Message parts keyword
http://www.ibm.com/websphere/webservices...	bodycontent
- Nonce:**

Nonce dialect	Nonce keyword
- Timestamp:**

Timestamp dialect	Timestamp keyword	Timestamp exp

Each section has "Add" and "Remove" buttons. At the bottom are "OK" and "Cancel" buttons.

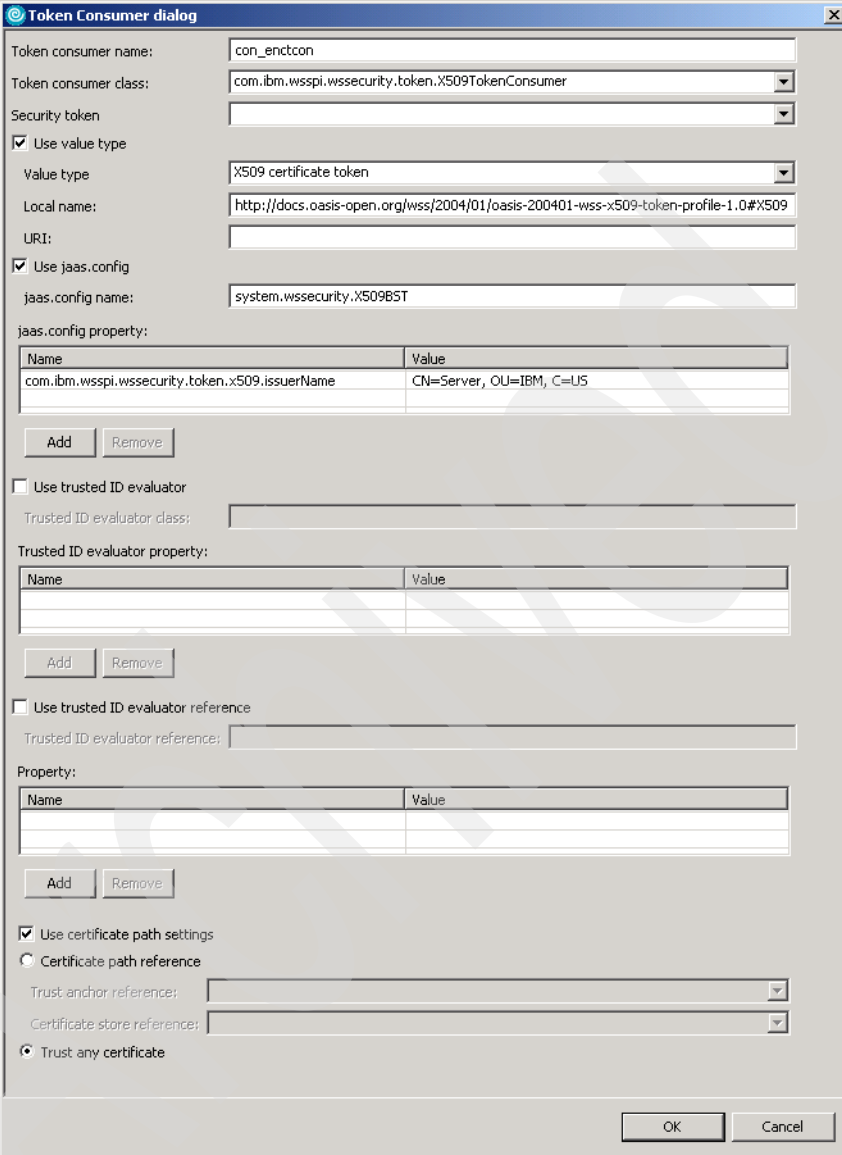
Figure 10-35 Required Confidentiality dialog for body content decryption

After specifying the required confidentiality part, the corresponding information must be specified on the **Binding Configurations** page. Switch to this tab and perform the following:

1. Expand **Request Consumer Binding Configuration Details**. These settings should match the client configuration. Expand **Token Consumer**, click **Add** and complete the dialog:
 - a. Enter a name, for example, con_enctcon.
 - b. Select **com.ibm.wsspi.wssecurity.token.X509TokenConsumer** as the class.
 - c. Select **Use value type** and select **X509 certificate token**.
 - d. Select **Use jaas.config** and enter system.wssecurity.X509BST as the name. In addition, specify a jaas.config properties by clicking **Add**.

- e. Specify a `jaas.config` property name of `com.ibm.wsspi.wssecurity.token.x509.issuerName`. Set the value of this property to `CN=Server, OU=IBM, C=US`.
- f. Click **Use certificate path settings** and then select **Trust any certificate**.
- g. Click **OK**.

Figure 10-36 on page 211 shows a Token Consumer dialog for decryption of the SOAP body content.



Token Consumer dialog

Token consumer name:

Token consumer class:

Security token:

☒ Use value type

Value type:

Local name:

URI:

☒ Use jaas.config

jaas.config name:

jaas.config property:

Name	Value
com.ibm.wsspi.wsssecurity.token.x509.issuerName	CN=Server, OU=IBM, C=US

☐ Use trusted ID evaluator

Trusted ID evaluator class:

Trusted ID evaluator property:

Name	Value
------	-------

☐ Use trusted ID evaluator reference

Trusted ID evaluator reference:

Property:

Name	Value
------	-------

☒ Use certificate path settings

☐ Certificate path reference

Trust anchor reference:

Certificate store reference:

☒ Trust any certificate

Figure 10-36 Token Consumer dialog for body content decryption

2. Expand **Key Locators** and click **Add** to specify how to retrieve a key for decryption:
 - a. Enter a name of con_encklocator.

- b. Select **com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator** as the class.
- c. Select **Use key store** and specify the server key store and server private key. In our case, we specify server, `${WAS_INSTALL_ROOT}\etc\ws-security\samples\ITSOGood\server.jks` and JKS.
- d. Click **Add** under Key and enter `server_rsa` as the alias, `server_rsa` as the key pass, and `CN=Server, OU=IBM, C=US` as the key name.
- e. Click **OK**.

Figure 10-37 shows a Key Locator dialog for the decryption of the SOAP body content.

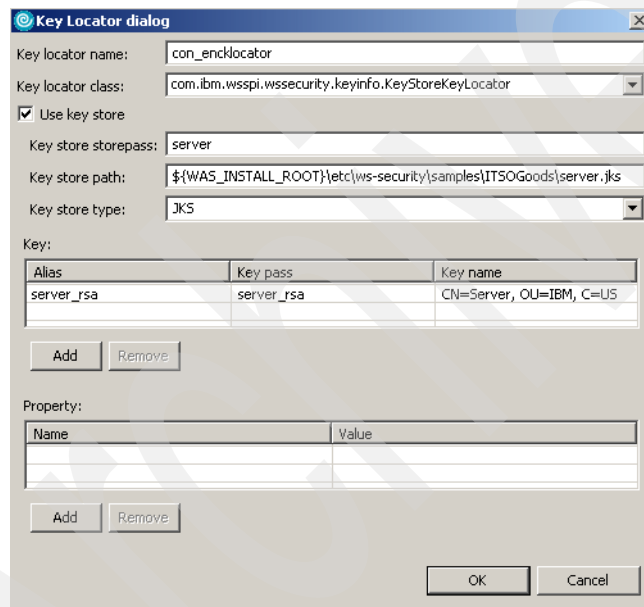
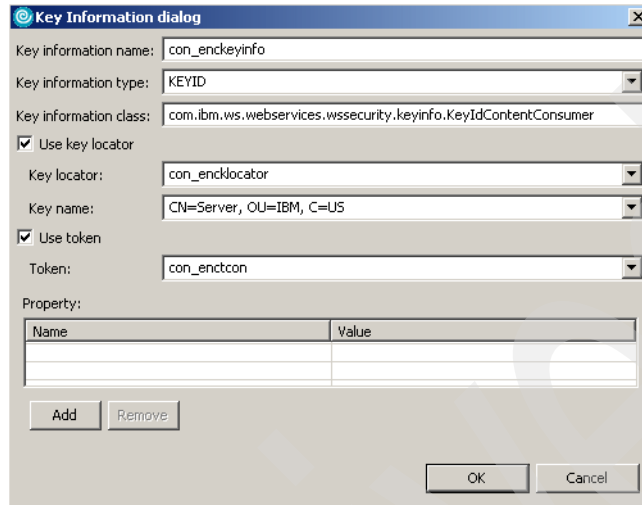


Figure 10-37 Key Locator dialog for body content decryption

3. Expand **Key Information** and click **Add** to specify which type of security token reference is required:
 - a. Enter a name of `con_enckeyinfo`. The type should match the client configuration, select **KEYID**.
 - b. Select **Use key locator**, and then select `con_encklocator` and **CN=Server, OU=IBM, C=US** from the pull-down menus.
 - c. Select **Use token** and select `con_enctcon`.
 - d. Click **OK**.

Figure 10-38 shows a Key Information dialog for the decryption of SOAP body content.



The image shows a 'Key Information dialog' window. It contains the following fields and controls:

- Key information name:** con_enckeyinfo
- Key information type:** KEYID (dropdown)
- Key information class:** com.ibm.ws.webservices.wssecurity.keyinfo.KeyIdContentConsumer
- ☒ **Use key locator**
 - Key locator:** con_enclocator (dropdown)
 - Key name:** CN=Server, OU=IBM, C=US (dropdown)
- ☒ **Use token**
 - Token:** con_enctcon (dropdown)
- Property:**

Name	Value

Below the table are 'Add' and 'Remove' buttons.
- At the bottom right are 'OK' and 'Cancel' buttons.

Figure 10-38 Key Information dialog for body content decryption

4. Expand **Encryption Information** and click **Add** to define how to decrypt a required confidentiality part:
 - Enter a name of enc_body.
 - Select **http://www.w3.org/2001/04/xmlenc#tripledes-cbc** for the Data encryption method algorithm and **http://www.w3.org/2001/04/xmlenc#rsa-1_5** for the Key encryption method algorithm.
 - Click **Add** under Encryption key information. Enter enc_kinfo as the Key information name and select **con_enckeyinfo** as the Key information element.
 - Select **reqconf_body** as the RequiredConfidentiality part.

Figure 10-39 on page 214 shows an Encryption Information dialog for the required confidentiality part.

Encryption name:

Data encryption method algorithm:

Key encryption method algorithm:

Encryption key information

Key information name	Key information element
enc_kinfo	con_enckeyinfo

RequiredConfidentiality part:

Property:

Name	Value
------	-------

Figure 10-39 Encryption Information dialog for body content decryption

5. Save your changes to the configuration.

Confidentiality on the response message

If you want to add confidentiality on the response message from the server, you have to configure the Response Generator in the server configuration and the Response Consumer in the client configuration:

- ▶ For the server configuration, you can configure the Response Generator Service Configuration Details on the Extensions page and the Response Generator Binding Configuration Details on the Binding Configurations page.
- ▶ For the client configuration, you can configure the Response Consumer Configuration on the WS Extension page and the Security Response Consumer Binding Configuration on the WS Binding page.

10.3.6 Exporting EAR files from Rational Application Developer

Now that you have configured WS-Security, you need to export ITSOGood and Manufacturer EAR files in order to install them on the applications servers.

Perform the following steps:

1. In the Project Explorer view of the J2EE perspective, expand **Enterprise Applications**.

2. Right-click **ITSOGood** and select **Export** → **EAR file**. Enter a destination to save the EAR file, making sure to name it `ITSOGood.ear`. Click **Finish**.
3. Repeat this process for the Manufacturer project, saving the file as `Manufacturer.ear`.

10.4 Runtime guidelines

This section takes you through the steps that are involved for configuring the sample application using the Extended Enterprise Direct Connection pattern. The following activities are needed to successfully setup and test the sample application:

- ▶ Configuring WebSphere Application Server server profiles
- ▶ Hosting the WSDL files
- ▶ Installing and configuring the applications
- ▶ Securing the Application Servers using Global Security
- ▶ Configuring the Web server for SSL routing
- ▶ Running and using the Supply Chain application

Important: Any described port number in this chapter and the following ones should be translated to the local equivalent in your scenario implementation. For instance, port 9080 used in some of our URLs might not necessarily be the HTTP Transport port in your scenario. Rather, your port number might be 9081 instead. The same concept of localized translation applies to hostnames and IP addresses as well.

10.4.1 Solution topology

In order to represent the complete business scenario, the sample application is divided into four subapplications:

- ▶ *ITSOGood* contains the `SCMSampleUI`, `Retailer`, `Warehouse`, and `LoggingFacility` services.
- ▶ *Manufacturer*, *ManufacturerB*, *ManufacturerC* are three individual services, each packaged separately, and deployed to three different enterprises.

As described in the Product mapping in “Product mapping selected” on page 176, we use WebSphere Application Server V6.0.2 to host the *ITSOGood* and *Manufacturer* applications, Microsoft .NET to host the *ManufacturerB* application, and CICS Transaction Server to host the *ManufacturerC* application. This arrangement is shown in Figure 10-40 on page 216.

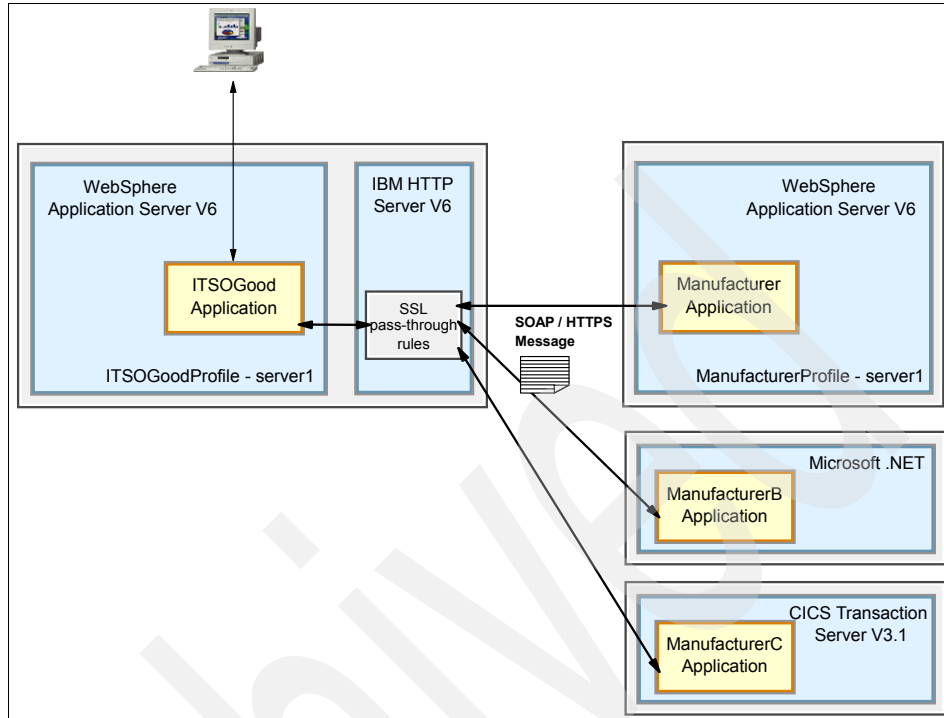


Figure 10-40 Solution topology

The runtime guidelines in this chapter describe how to prepare WebSphere Application Server profiles for the ITSOGood and Manufacturer applications, and the HTTP server. The guidelines describe how to configure these profiles to use the WS-Security integrity and confidentiality settings that we defined in the development guidelines.

Optionally, you can implement the ManufacturerB and ManufacturerC applications as well. To do this you will need access to a Microsoft .NET server and a CICS Transaction Server V3.1 region. The connection between the ITSOGood applications and the ManufacturerB and ManufacturerC applications will not be secured using WS-Security settings in this sample (although in a production environment this should be configured too).

Note: You do not need to configure the ManufacturerB and ManufacturerC servers to build a working end-to-end sample application.

Instructions for configuring the ManufacturerB and ManufacturerC servers are described in:

- ▶ Appendix B, “Microsoft .NET Web services” on page 483
- ▶ Appendix C, “CICS Transaction Server Web services” on page 513

10.4.2 Configuring WebSphere Application Server profiles

This section describes how to create server profiles for the ITSOGood and Manufacturer servers. It assumes the use of WebSphere Application Server V6.0.2, and that you have this product already installed.

Because this scenario describes an extended enterprise scenario, these two profiles should, in theory, be located on physically different machines, in different organizations, and connect using the internet. However, for the purposes of this sample, both profiles can be installed on the same physical machine with the same network address. The instructions assume that a single machine is used, but these instructions can equally be applied to a real extended enterprise scenario.

Creating the ITSOGoodProfile

The ITSOGoodProfile server profile will be used to host the ITSOGood enterprise application. This correlates to the Exposed Direct Connection pattern generic profile product mapping as indicated by the circle in Figure 10-41.

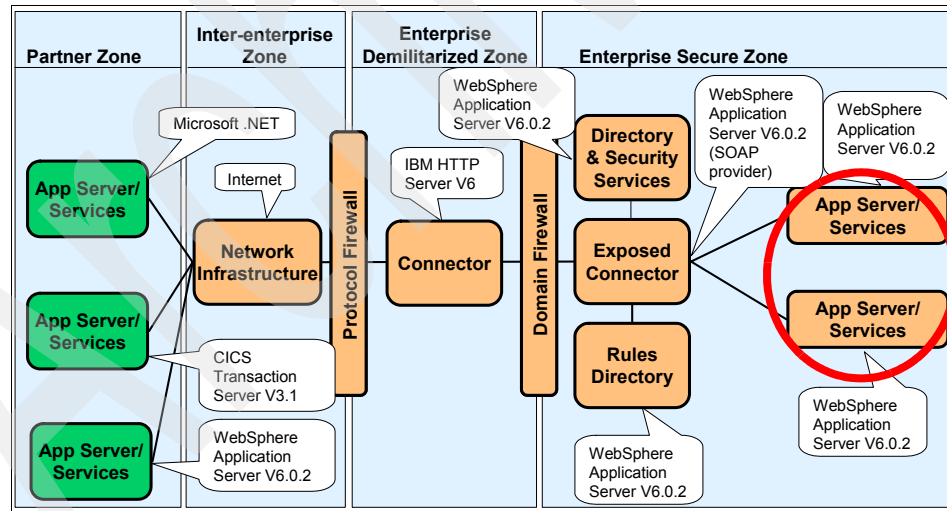


Figure 10-41 ITSOGoodProfile in the Exposed Direct Connection product mapping

To create a WebSphere Application Server profile for use by the ITSOGood application, perform the following steps on the machine with WebSphere Application Server V6.0.2 installed:

1. Start the WebSphere Application Server Profile Creation Wizard. On a Windows system, this can either be done from the Start menu (**Start** → **Programs** → **IBM WebSphere** → **Application Server V6** → **Profile creation wizard**) or by running the `pctWindows.exe` command from the `<WAS_HOME>\bin\ProfileCreator` directory.
2. Navigate through the Process creation wizard to create a new application server profile, with the following attributes:
 - a. Set profile name to `ITS0GoodProfile`.
 - b. Assign a node name of `ITS0GoodNode`.
 - c. Leave the port value assignments to default (for example the Administrative console port should be 9060). The application is hard coded to use port 9080.

Creating the ManufacturerProfile

The ManufacturerProfile server profile will be used to host the Manufacturer enterprise application. This correlates to the Exposed Direct Connection pattern generic profile product mapping as indicated by the circle in Figure 10-42.

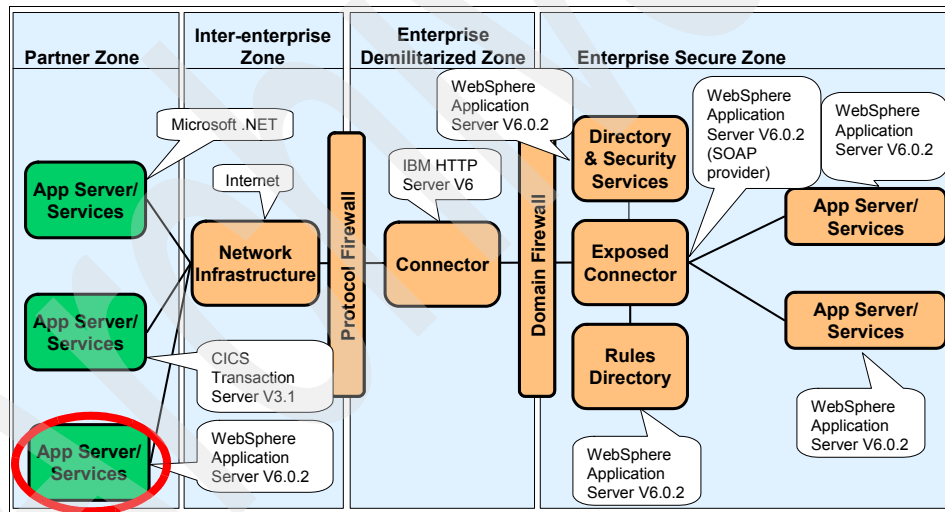


Figure 10-42 ManufacturerProfile in the Exposed Direct Connection product mapping

To create a WebSphere Application Server profile for use by the Manufacturer application, use the Profile creation wizard again to create another application server profile, this time specifying the following attributes:

- ▶ Set the profile name to `ManufacturerProfile`.
- ▶ Assign a node name of `ManufacturerNode`.

- ▶ Leave the port value assignments at their defaults. Notice that the Profile creation wizard assigns ports numbers that do not clash with the previous profile ITSOGoodProfile. For example, the Administrative console port should now be 9061. This allows both profiles to be running on the same machine at the same time.

Starting the application servers

You need both application server profiles running to configure them for this scenario. To start the profiles, follow these steps:

1. From a Command Prompt, navigate to the bin directory of the profile, which by default is located at <WAS_HOME>\profiles\ITSOGoodProfile\bin.
2. Run the command `startserver server1` to start the ITSOGoodProfile server.
3. Navigate to <WAS_HOME>\profiles\ManufacturerProfile\bin and run the command `startserver server1` to start the ManufacturerProfile server.

Once started, you can access the administrative consoles for each server profile as follows:

- ▶ Access the ITSOGoodProfile administrative console using:
`http://localhost:9060/ibm/console`
- ▶ Access the ManufacturerProfile administrative console using:
`http://localhost:9061/ibm/console`

10.4.3 Hosting the WSDL files

Each Web service client in the sample application attempts to retrieve WSDL files that contain port type and binding information. Import statements dictate the location of these files. For example, the SCMSampleUI enterprise application contains a WSDL file that tries to retrieve the Retailer port type and binding by using the following import:

```
<wsdl:import location="http://appsrv1a.itso.ral.ibm.com/wsdl/Retailer.wsdl"
namespace="http://www.ws-i.org/SampleApplications/SupplyChainManagement/200
2-08/Retailer.wsdl"/>
```

As a result, it is necessary to host an HTTP server where these files can be retrieved. This HTTP server host must be assigned the host address of `appsrv1a.itso.ral.ibm.com`.

Assuming IBM HTTP Server V6 (shipped with WebSphere Application Server V6) is installed, complete the following steps:

1. Assign the host name `appsrv1a.itso.ral.ibm.com` to map to your local machine by opening the hosts file.

For Windows hosts open:

```
<WINDOWS_HOME>\system32\drivers\etc\hosts
```

For Linux/Unix hosts open:

```
/etc/host.conf
```

2. Add the following statement to the hosts file. This ensures that all requests sent to appsrv1a.itso.ral.ibm.com are redirected to your local machine.

```
127.0.0.1 appsrv1a.itso.ral.ibm.com
```

3. In addition, add statements which will assign host addresses for the ITSOGood and Manufacturer services. If you are hosting the ITSOGood and Manufacturer servers on the same physical machine, map these to 127.0.0.1 as shown below:

```
127.0.0.1 itsogood.itso.ral.ibm.com
```

```
127.0.0.1 manufacturera.itso.ral.ibm.com
```

4. [Optional] If you intend to install ManufacturerB and ManufacturerC, add an entry in the hosts file pointing each host name to the IP address of the Microsoft .NET server and CICS Transaction Server region:

```
<ip_address_of_.NET> manufacturerb.itso.ral.ibm.com
```

```
<ip_address_of_CICS> manufacturerc.itso.ral.ibm.com
```

5. We installed IBM HTTP Server V6. The WSDL files to be hosted on this HTTP server are provided with the additional materials that are supplied with this book. For information about how to obtain the additional materials see Appendix A, "Additional material" on page 481. Copy the contents of the \DirectConnection\wsdl directory from the additional material to the following directory:

```
<HTTP_SERVER_HOME>\htdocs\en_US\wsdl
```

6. Make sure that the HTTP server is started, then test that the WSDL is available by entering the following URL into a Web browser:

```
http://appsrv1a.itso.ral.ibm.com/wsdl/Retailer.wsdl
```

The contents of the WSDL file for the Retailer Web service should be displayed.

10.4.4 Installing the applications

In this section we install the enterprise applications ITSOGood and Manufacturer into the relevant application servers.

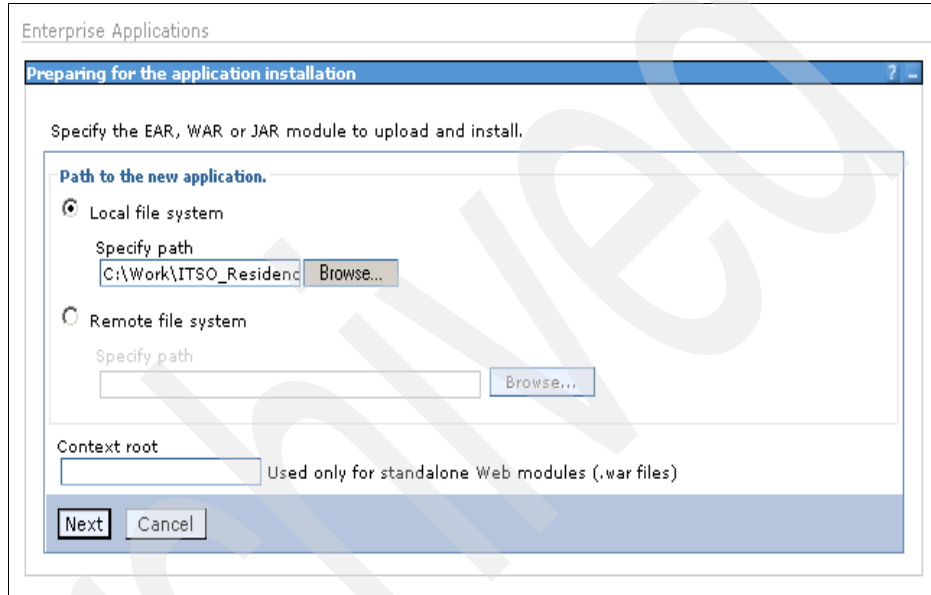
Installing the ITSOGood application

1. Using a Web browser, start the WebSphere administrative console for the server where the ITSOGood application will be deployed. If you have

configured your hosts file correctly, you should be able to use the following URL:

`http://itsogood.itso.ra1.ibm.com:9060/ibm/console`

2. Log in, then select **Applications**, and click **Install New Application**.
3. In the page that opens, as shown in Figure 10-43, enter the location of the */TSOGood.ear* file on your local file system by either entering it directly or by clicking **Browse** and navigating to the file in the open file dialog.



Enterprise Applications

Preparing for the application installation

Specify the EAR, WAR or JAR module to upload and install.

Path to the new application.

☒ Local file system

Specify path

C:\Work\ITSO_Residenc Browse...

☐ Remote file system

Specify path

Context root

Used only for standalone Web modules (.war files)

Next Cancel

Figure 10-43 Specify enterprise application to install

4. Click **Next**.
5. On the next page that opens, accept the defaults and click **Next**.
6. The next page to open is the first page of the installation wizard. The application is fully configured and deployed so that you should use the defaults on each page. Either click **Next** until the final page, or click the last step that is labelled **Summary**.
7. Click **Finish**. When the application has been installed, a screen similar to Figure 10-44 on page 222 is displayed.

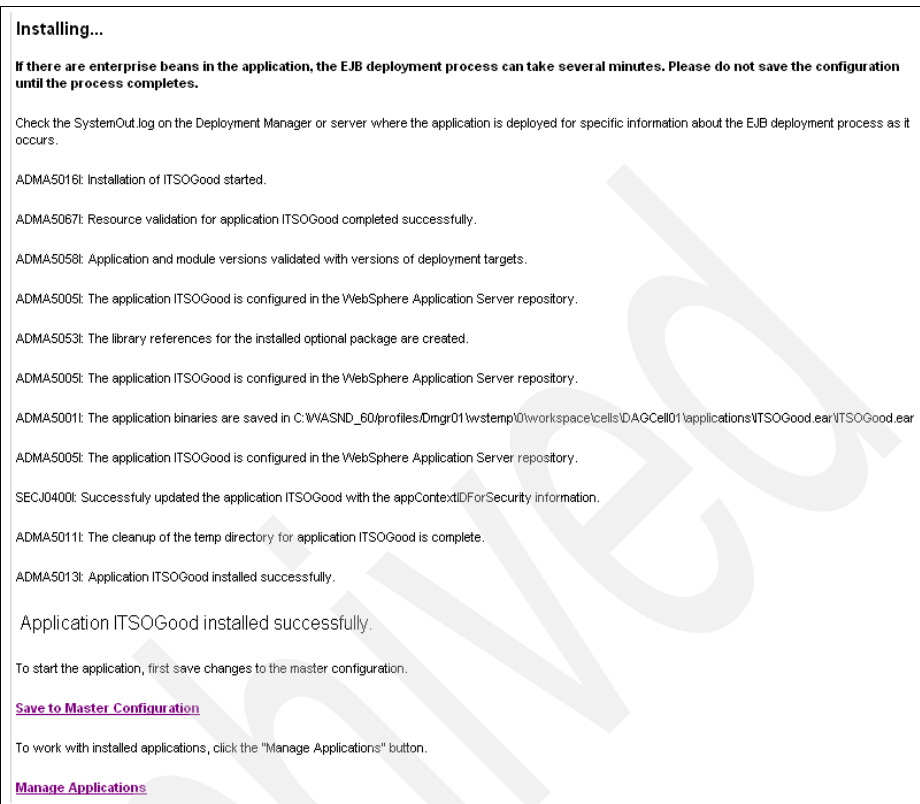


Figure 10-44 Application installation finished page

8. Save the configuration.

Installing the Manufacturer application

1. Follow the same procedure as in "Installing the ITSOGood application" on page 220 to install the Manufacturer enterprise application, this time using the administrative console at the following URL:

`http://manufacturer.a.itso.ra1.ibm.com:9061/ibm/console`

2. Log in, then select **Applications**, and click **Install New Application**. This time install Manufacturer.ear.
3. Once installed, save the changes to the configuration,

10.4.5 Securing the application server using Global Security

We secure the applications using Global Security settings on WebSphere Application Server through the administrative console. Global Security must be turned on in order to use WS-Security.

Perform the following steps on the ITSGoodProfile server:

1. Log in to the administrative console, and click **Security** → **Global security**.
2. Under the User registries category (top right), select **Local OS**.
3. Enter a user ID and password under General Properties. The user ID and password you enter need to be a real user ID and password for your local operating system. This user ID must have administrator privileges. Click **OK**.
4. Return to the main panel, select the **Enable global security** option, and then clear **Enforce Java 2 security**.
5. Select **Local OS** from the Active user registry list. This means that the local operating system security repository will be used with Global Security. The panel should now look similar to Figure 10-45.

Global security

Specifies the global security configuration for a managed domain. The following steps are required to turn on security: 1. Configure the desired user registry listed under User registries and set its properties. 2. Select the Enable global security option on this panel. 3. Select the configured user registry type from the Active user registry option on this panel.

Configuration

General Properties

- ☒ Enable global security
- ☐ Enforce Java 2 security
- ☐ Enforce fine-grained JCA security
- ☐ Use domain-qualified user IDs
- Cache timeout: 600 seconds
- ☒ Issue permission warning
- Active protocol: CSI and SAS
- Active authentication mechanism: Simple WebSphere Authentication Mechanism (SWAM)
- Active user registry: Local OS (single, stand-alone server or sysplex and root administrator only)
- ☐ Use the Federal Information Processing Standard (FIPS)

User registries

- [Custom](#)
- [LDAP](#)
- [Local OS](#)

Authentication

- [Authentication mechanisms](#)
- [Authentication protocol](#)
- [JAAS Configuration](#)

Authorization

- [Authorization providers](#)

Additional Properties

- [Custom properties](#)

Apply OK Reset Cancel

Figure 10-45 Global security configuration in the administrative console

6. Click **OK** and save the configuration.
7. Restart the application server. Global Security should now be on, which means you will need to provide a real user ID and password to log on to the administrative console. Use the user ID and password you specified in the Local OS setting.

Repeat this process to enable Global Security for the ManufacturerProfile profile.

10.4.6 Configuring an HTTP server for SSL pass-through

In the generic profile of the Exposed Direct Connection runtime pattern, a Connector is required in the demilitarized zone. In the product mapping we have selected, this Connector is implemented with IBM HTTP Server V6, as indicated in Figure 10-46.

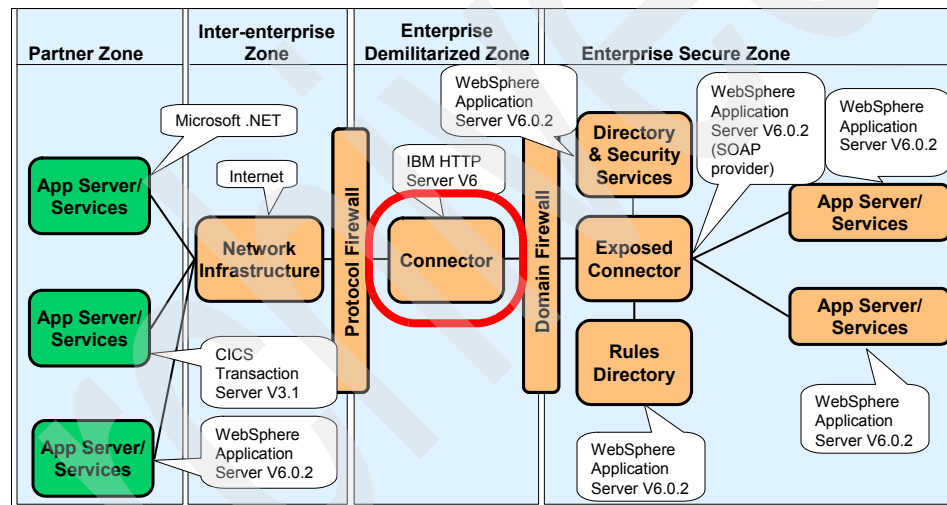


Figure 10-46 Use of IBM HTTP Server as a Connector node

The purpose of the HTTP server in this implementation is to serve as a connector between the applications in the Enterprise Secure Zone and those in the Inter-enterprise Zone.

The HTTP server will contain SSL pass-through rules: receiving requests from Web service clients over SOAP/HTTP and forwarding them to the relevant Manufacturer Web service endpoint URL using SOAP/HTTPS. This is shown for the first Manufacturer in Figure 10-47 on page 225.

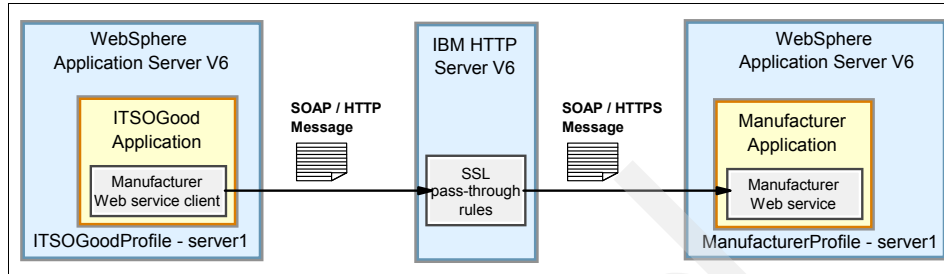


Figure 10-47 Use of an IBM HTTP Server for SSL pass-through

For convenience, you can use the same IBM HTTP Server that has been configured to host WSDL files, as described in 10.3.4, “Configuring WS-Security integrity” on page 188.

Create a keystore for use by the HTTP server

The secured connection between the HTTP server and the application server requires a certificate key store to store its own private and public key files as well as the public certificate from the Web container key file.

In this section, we reuse a precreated key store named `ih.s.kdb`. It is found with other essential files in the additional material supplied with this redbook at the following location:

```
\DirectConnection\keystore\ihssl.zip
```

After you have obtained `ihssl.zip`, perform the following:

1. On the system where the IBM HTTP Server is installed, create a directory named `C:\keys` (assuming you are using Windows).
2. Unzip all the contents of `ihssl.zip` into the newly created directory.
3. Log in to the administrative console for the `ManufacturerProfile` server profile.
4. You need to determine the TCP/IP port assigned to SSL incoming connections. To do this, select **Servers** → **Application servers**. Click **server1**. Under Communications, expand **Ports** and note the value of `WC_defaulthost_secure` (see Figure 10-48 on page 226).

Communications		
Ports		
Port Name	Port	details
BOOTSTRAP_ADDRESS	2810	
SOAP_CONNECTOR_ADDRESS	8881	
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	9404	
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	9405	
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	9406	
WC_adminhost	9061	
WC_defaulthost	9081	
DCS_UNICAST_ADDRESS	9354	
WC_adminhost_secure	9044	
WC_defaulthost_secure	9444	
SIB_ENDPOINT_ADDRESS	7277	
SIB_ENDPOINT_SECURE_ADDRESS	7287	
SIB_MQ_ENDPOINT_ADDRESS	5559	
SIB_MQ_ENDPOINT_SECURE_ADDRESS	5579	
ORB_LISTENER_ADDRESS	9101	

Figure 10-48 Port used to receive SSL requests

If you want to create your own self-signed certificate and use it in the key store instead of using the one supplied with this redbook, follow the steps listed in the WebSphere Application Server Infocenter:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.ihs.doc/info/aes/ae/tihs_certsselfsigned.html

If have created your own self-sign certificate, import this certificate into the server-based key file. The key file is used by the application server to store private keys for client-certificate authentication purposes. The default location and file name for WebSphere Application Server V6 is:

<WAS_HOME>/profiles/<server_name>/etc/DummyServerKeyFile.jks.

Configuring SSL pass-through in the HTTP server

We need to modify the HTTP server configuration file to enable SSL, specify the keystore to use, and add ProxyPass statements for each Manufacturer request that will flow through the HTTP server.

Perform the following steps:

1. Open the HTTP server configuration file `httpd.conf` in a text editor. You can find it in the `<HTTP_SERVER_HOME>\conf\` directory.
2. Add the statements shown in Example 10-4 on page 227 to the end of the configuration file. These entries indicate to the HTTP server that incoming HTTP requests are to be encrypted using the keys in the indicated key store file before being routed to the various Manufacturer Web services based on the distinct URIs.

Example 10-4 Modifications to the httpd.conf file

```
SSLProxyEngine on
Keyfile "C:\keys\ihs.kdb"
ProxyPass /Manufacturer/ https://manufacturer.a.itso.ral.ibm.com:9444/Manufacturer/
ProxyPass /ManufacturerB/ https://manufacturer.b.itso.ral.ibm.com/ManufacturerB/
ProxyPass /ManufacturerC/ https://manufacturer.c.itso.ral.ibm.com/ManufacturerC/
```

Note: Example 10-4 assumes that the value of the SSL port on WebSphere Application Server for ManufacturerProfile is 9444. Be sure to use the correct port number in your environment based on the value of the WC_defaulthost_secure port name, as determined in the previous section.

3. Append the entries shown in Example 10-5 to httpd.conf to load the various code modules necessary for SSL-based routing to work.

Example 10-5 Loading SSL modules to the httpd.conf file

```
LoadModule ibm_ssl_module modules/mod_ibm_ssl.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_connect_module modules/mod_proxy_connect.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule proxy_ftp_module modules/mod_proxy_ftp.so
```

4. Save and close httpd.conf.
5. Restart the HTTP server so the new changes take effect.

10.4.7 Changing the Web service client bindings configuration

Web service clients bind to a specific port endpoint URL. This endpoint URL can be specified in a number of places, including in the WSDL file of the Web service, and in the Java code of the Web service client.

We can use WebSphere Application Server to override the endpoint URL used by Web service clients. We override the Web service clients in the ITSOGood enterprise application that make calls to the Manufacturers. We override each endpoint URL to point to the HTTP server which contains SSL pass-through rules to each Manufacturer (see Figure 10-47 on page 225).

Perform the following steps in the administrative console of the ITSOGoodProfile server:

1. From the administrative console where the ITSOGood enterprise application is installed, select **Applications**, and click **Enterprise Applications**.
2. From the list of installed applications click **ITSOGood**. The application configuration screen opens.

3. Click **EJB Modules**. The server-module installation binding for an EJB module screen is displayed.
4. In the list of EJB Modules click **WarehouseEJB.jar**. The selected EJB Module configuration screen is displayed.
5. Click **Web services client bindings** and the screen shown in Figure 10-49 opens.

Web Service	EJB	WSDL Filename	Preferred Port Mappings	Port Information
ManufacturerService	WarehouseSoapBindingImpl	Use default (META-INF/wsdl/Manufacturer_Impl.wsdl)	Edit...	Edit...
LoggingFacilityService	WarehouseSoapBindingImpl	Use default (META-INF/wsdl/LoggingFacility_Impl.wsdl)	Edit...	Edit...
ManufacturerBService	WarehouseSoapBindingImpl	Use default (META-INF/wsdl/ManufacturerB_Impl.wsdl)	Edit...	Edit...
ManufacturerCService	WarehouseSoapBindingImpl	Use default (META-INF/wsdl/ManufacturerC_Impl.wsdl)	Edit...	Edit...

Figure 10-49 Web services client bindings

6. The Web service name ManufacturerA is ManufacturerService, ManufacturerB is ManufacturerBService and ManufacturerC is ManufacturerCService. Click **Edit** under the Port Information column for ManufacturerService.
7. The Port Information screen allows us to override the endpoint URL. Set the Overridden Endpoint URL to:

`http://itsogood.itso.ral.ibm.com/Manufacturer/services/Manufacturer`

This means that the Web service client for the Manufacturer (located in the WarehouseEJB project of ITSOGood) will use this URL to contact the Web service. This URL points to the HTTP server, where the request will be redirected to the external Manufacturer Web service. See Figure 10-50 on page 229.

Enterprise Applications > ITSOGood > EJB Modules > WarehouseEJB.jar > Web services client bindings > ManufacturerService:WarehouseSoapBindingImpl

Specifies a request timeout, an overridden endpoint URL, and an overridden binding namespace can be set for a port. The timeout determines how many seconds to wait for a request. A value of zero disables the timeout. The current endpoint and binding namespace can be overridden.

Port	Request Timeout (seconds)	Overridden Endpoint URL	Overridden Binding Namespace
Manufacturer		http://itsogood.itso.ral.ibm.com/Manufacturer/s	

Apply OK Reset Cancel

Figure 10-50 Overriding the endpoint URL for the Manufacturer

- Click **OK**.
- If you have created implementations of ManufacturerB and ManufacturerC, you need to repeat this process to override these endpoint URLs as indicated in Table 10-3.

Table 10-3 Overridden endpoint URLs for Manufacturers B and C

Web service name	Overridden endpoint URL
ManufacturerBService	http://itsogood.itso.ral.ibm.com/ManufacturerB/ManufacturerB.asmx
ManufacturerCService	http://itsogood.itso.ral.ibm.com/ManufacturerC/ManufacturerC

- Save your changes to the configuration.

10.4.8 Testing the scenario

This section describes how to test the scenario. For more information about the sample application, see Chapter 8, “Business scenario used in this book” on page 137.

To test the sample application, perform the following steps:

- Enter the following URL in a Web browser to start the ITSOGood sample application:

http://itsogood.itso.ral.ibm.com:9080/SCMSampleUI/

This assumes the unsecured HTTP transport port number is 9080. If this is not the case, use the URL above with the appropriate HTTP transport port number.

The Supply Chain Management Sample Application should start, as shown in Figure 10-51.



Supply Chain Management Sample Application

Enter Customer data. Then place an order.

Parameter	Value
Customer Name	A Shopper
Customer Number	A12345-9876543-xyz
Customer Address	123 ITSO Customer Lane
Configurator Options	<input checked="" type="radio"/> Use all local endpoints - no configuration options



Figure 10-51 SCM Sample application

2. Click **Place New Order**. This generates a Web service call to the Retailer Web service. The Shopping Cart screen displays a list of ten products, as shown in Figure 10-52 on page 231.

Shopping Cart

Enter quantities for products to order. Then Submit Order.

Quantity	Product Number	Name	Description
<input type="text"/>	605001	TV, Brand1	24in, Color, Advanced Velocity Scan Modulation, Stereo
<input type="text"/>	605002	TV, Brand2	32in, Super Slim Flat Panel Plasma
<input type="text"/>	605003	TV, Brand3	50in, Plasma Display
<input type="text"/>	605004	Video, Brand1	S-VHS
<input type="text"/>	605005	Video, Brand2	HiFi, S-VHS
<input type="text"/>	605006	Video, Brand3	s-vhs, mindv
<input type="text"/>	605007	DVD, Brand1	DVD-Player W/Built-In Dolby Digital Decoder
<input type="text"/>	605008	DVD, Brand2	Plays DVD-Video discs, CDs, stereo and multi-channel SACDs, and CD-RWs, 27MHz/10-bit video DAC,
<input type="text"/>	605009	DVD, Brand3	DVD Player with SmoothSlow forward/reverse; Digital Video Enhance Text; Custom Parental Control (20-disc); Digital Cinema Sound mode
<input type="text"/>	605010	TV, Brand4	Designated invalid product code that is allowed to appear in the catalog to be ordered

Figure 10-52 SCM Sample product listing

- Place an order that will trigger the Warehouse to replenish its stock level. The Warehouse replenishes stock by sending a stock request order to a Manufacturer.

The Warehouse stock level is stored in the `org.ws_i.www.Impl.WarehouseImpl` class in the WarehouseEJB project. For example, the stock level for the first three products is shown in Table 10-4.

Table 10-4 Warehouse stock level

Product number	Current level	Minimum level	Maximum level
605001	10	5	25
605002	7	4	20
605003	15	10	50

If the current stock level falls below the minimum stock level, the stock is reordered so that, after the reorder arrives, the stock is at the maximum level. For example, if you order six items of product number 605001, it reduces the current stock level to four ($10 - 6 = 4$). A reorder is made for 21 new items to bring the stock level back to the maximum desired stock level.

Each Manufacturer only manufactures certain products. For example Manufacturer A manufactures products 605001, 605004, and 605007.

Place an order for **6** items of product 605001 and click **Place Order**. This will trigger a re-order of product 605001 from Manufacturer A (the Manufacturer hosted in the WebSphere Application Server ManufacturerProfile server). In our scenario this Manufacturer is in an extended enterprise, hence this will test our WS-Security and SSL settings.

4. The order status screen, as shown in Figure 10-53, shows which orders were placed and which orders were not placed due to insufficient stock.

Product Number	Quantity	Price	Comment
605001	6	1799.70	in stock from Warehouse

[Track Order](#) [Configure](#)

Figure 10-53 SCM Sample order status page

5. Click **Track Order** to see the entries that were written to the LoggingFacility. Figure 10-54 shows the results of an order in which product 605001 was shipped.

Service ID	Event ID	Description	Vendor
Retailer.submitOrder	UC1-5	Order placed by A12345-9876543-xyz for 605001	
WarehouseA.ShipGoods	UC2-2-1	WarehouseA will determine its ability to ship product(s) 605001.	
WarehouseA.ShipGoods	UC2-2-2	WarehouseA is able to ship 605001.	
Retailer.submitOrder	UC1-9	Processing of the order from A12345-9876543-xyz has finished normally	

[Order Status](#) [Configure](#)

Figure 10-54 SCM Sample track order page

6. To confirm the successful invocation of the Manufacturer service, check the SystemOut.log of the ITSOGoodProfile application server. The Warehouse writes a message saying that a Manufacturer was invoked. The message should look like this:

Warehouse: Response from Manufacturer --> Manufacturer_A has received and processed a request.

You should also see the following message in the SystemOut.log file of the ManufacturerProfile application server:

Manufacturer A: Processing Purchase Order

7. To start a new order, click **Configure**. At this point, all state is lost, and the Warehouse stock levels return to their default values. If you have configured the Microsoft .NET and CICS Transaction Server manufacturers, try ordering products 605002 and 605003 to place orders with Manufacturer B and Manufacturer C.

10.4.9 Viewing SOAP messages using the TCP/IP Monitor

You can confirm that the WS-Security integrity and confidentiality settings have been applied to a SOAP message by viewing the contents of the SOAP message. The easiest way to do that is to use the TCP/IP Monitor provided with WebSphere Application Server.

To configure the TCP/IP Monitor, perform the following on the machine hosting ITSOGood:

1. In a Command Prompt, set the PATH environment variable so the java.exe command can be run (substituting <WAS_HOME> for the path where WebSphere Application Server is installed):

```
set PATH=<WAS_HOME>\java\bin;%PATH%
```

2. Navigate to the <WAS_HOME>\lib directory:

```
cd <WAS_HOME>\lib
```

3. Launch the TCP/IP Monitor:

```
java -classpath webservices.jar com.ibm.ws.webservices.engine.utils.tcpmon
```

4. The TCP/IP Monitor should launch in a new window. The monitor works as a proxy. You send TCP/IP requests to it, and it forwards them on to another destination. The advantage of sending messages through the TCP/IP Monitor is it shows you the content of that message. This is very useful for inspecting SOAP messages.

Enter a Listen Port of **81**. Select **Act as a Listener**. Specify Target Hostname as **itsogood.itso.ral.ibm.com** and Target Port as **80**. See Figure 10-55 on page 234.

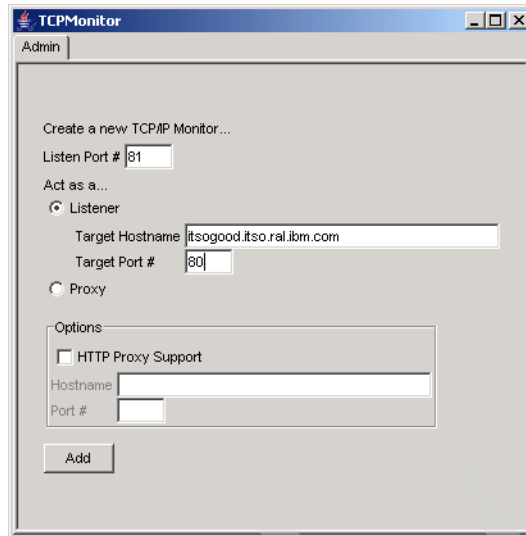


Figure 10-55 TCP/IP Monitor configuration

5. Click **Add** to add a new tab at the top of the TCP/IP Monitor called **Port 81**. Click this tab to view any TCP/IP messages that are received on port 81 by the TCP/IP Monitor. Currently, it shows no messages (Figure 10-56).

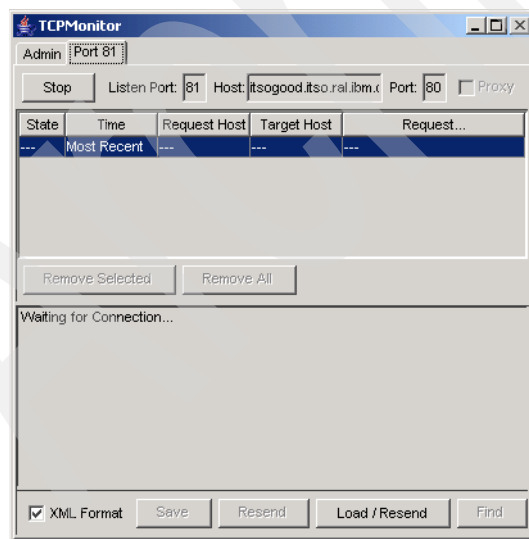


Figure 10-56 TCP/IP Monitor waiting for messages

We use the TCP/IP Monitor to intercept messages sent between the Manufacturer Web service client in ITSOGood and the HTTP server, as shown in Figure 10-57.

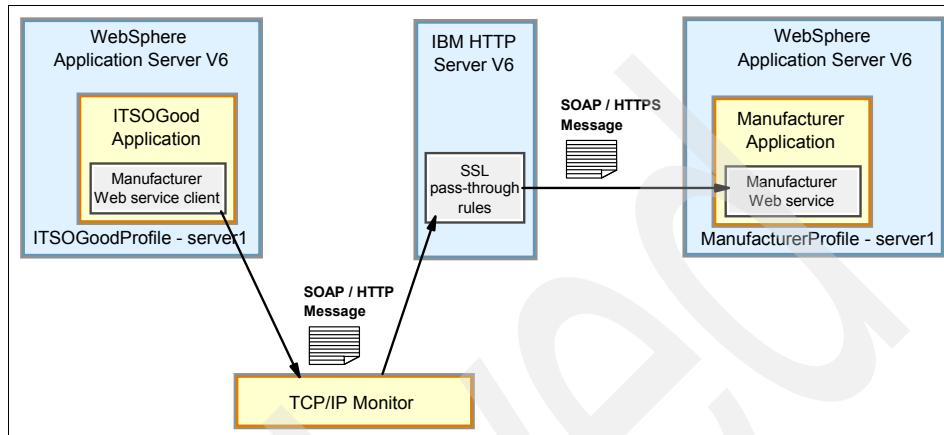


Figure 10-57 Redirecting the SOAP/HTTP message through the TCP/IP Monitor

To achieve this, you must update the Web service client binding for the Manufacturer Web service client to point to the TCP/IP Monitor. Perform the following steps:

1. Log in to the administrative console of the ITSOGoodProfile server.
2. Access the Web service client binding of the Manufacturer Web service. Click **Applications** → **Enterprise Applications** → **ITSOGood** → **EJB Modules** → **WarehouseEJB.jar** → **Web services client bindings**, then click **Edit** under the Port Information column for ManufacturerService.
3. Set the Overridden Endpoint URL to:

```
http://itsogood.itso.ral.ibm.com:81/Manufacturer/services/Manufacturer
```
4. Click **OK**, then restart the ITSOGood enterprise application for the change to take effect.

Test the scenario again, as described in 10.4.8, “Testing the scenario” on page 229. This time you should see the content of the SOAP message sent to the Manufacturer, and the response from the Manufacturer in the TCP/IP Monitor. Notice that it uses WS-Security (Figure 10-58 on page 236).

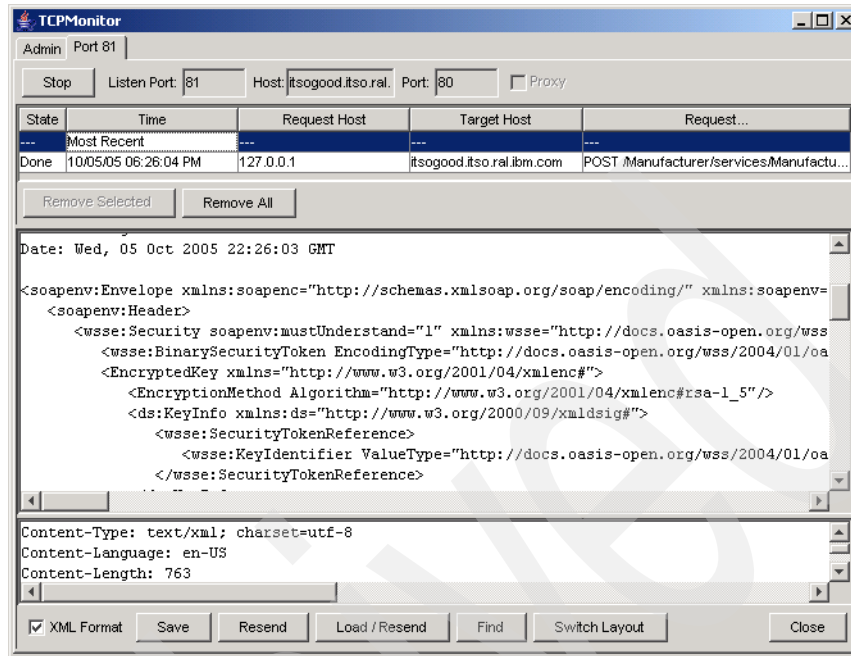


Figure 10-58 Viewing a SOAP message in the TCP/IP Monitor

Exposed Direct Connection runtime pattern: SOA profile

In this chapter, we specialize the Exposed Direct Connection pattern for the SOA environment using the SOA Profile.

This chapter builds upon the scenario described in Chapter 10, “Exposed Direct Connection runtime pattern: generic profile” on page 157. We define how to achieve a point-to-point connection between Web services located in different enterprises by using an Enterprise Service Bus (ESB) and an Exposed ESB Gateway.

This chapter uses the service integration bus of WebSphere Application Server V6 to implement the ESB, and uses the Web services gateway of WebSphere Application Server Network Deployment V6 to implement the Exposed ESB Gateway.

This chapter also describes how to add integrity and confidentiality to SOAP messages sent between enterprises by configuring WS-Security in the Exposed ESB Gateway component.

11.1 Business scenario

The business scenario implemented in this chapter is similar to that defined in Chapter 10, “Exposed Direct Connection runtime pattern: generic profile” on page 157, shown in Figure 11-1.

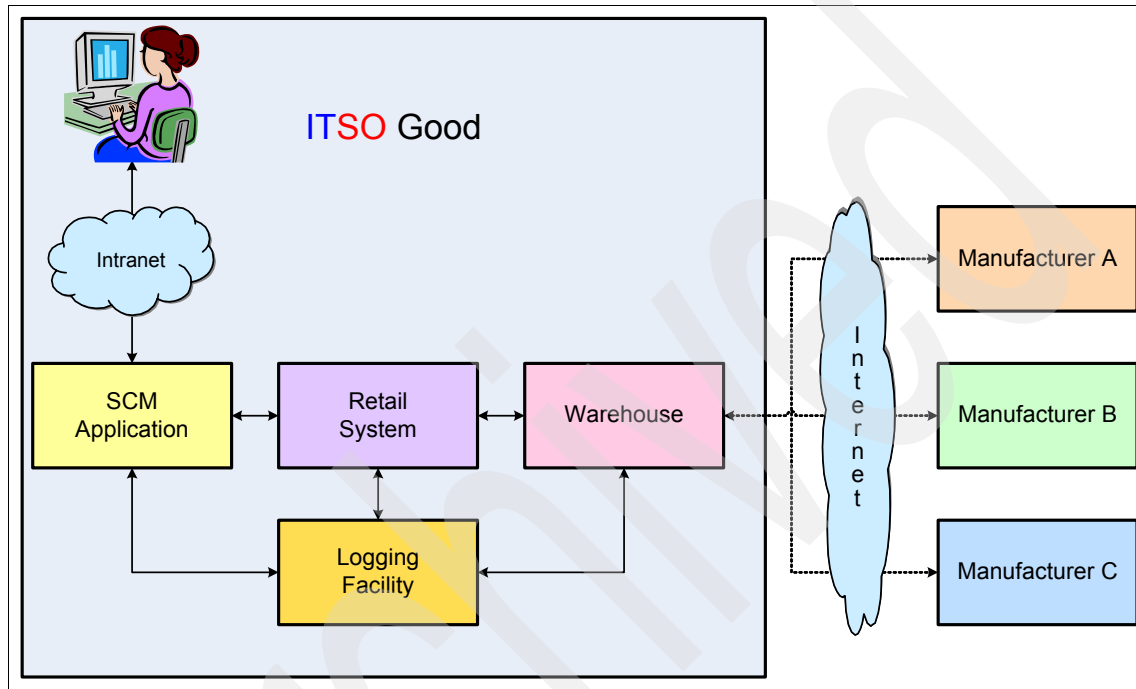


Figure 11-1 High-level business context of the scenario

Having exposed the retail and warehouse functions as services, the next step in the SOA transformation for ITSO Good is to build an infrastructure that can manage these services on an enterprise scale.

The SOA transformation for ITSO Good is a more strategic one driven by the business requirement to align their IT services more closely with the business. This also requires an infrastructure component that would provide them the required flexibility to integrate their business aligned services across the enterprise.

The ITSO Good business strategy plan is to expose many more business functions within their enterprise as services. As more and more business functions are exposed as services, it is vitally important that this infrastructure have the capability to support the management of SOA on an enterprise scale.

Service in an SOA environment: A *service* represents a reusable business function that is defined by explicit implementation-independent interfaces, loosely bound and is invoked through communication protocols that stress location transparency and interoperability.

Successfully implementing an SOA requires applications and infrastructure that can support the SOA principles. In the previous chapter, we SOA-enabled the applications by exposing the business functions as services. Now in its quest to build a comprehensive SOA solution, ITSO Good wants to build a basic SOA infrastructure that addresses the following requirements:

- ▶ Build an SOA infrastructure that provides the ability to integrate and manage the Retail, Warehouse, and other new services that are expected to be exposed within the enterprise.
- ▶ To provide access to external services of the Manufacturing partner systems in a controlled and secured manner. This will provide ITSO Good a single point of control to implement and manage its corporate security policy with regards to accessing partner systems.

11.2 Design guidelines

This section analyzes the infrastructure requirements described in the business scenario. It uses the Patterns for e-business to select the appropriate runtime pattern and also discusses the various architectural decisions involved in implementing the solution.

11.2.1 Analyze IT infrastructure requirements

ITSO Good requires a middleware component that can support its SOA infrastructure requirements. So, what are these requirements for an SOA infrastructure?

This has been the subject of much debate among the different vendors in the SOA marketplace, often for competitive reasons. Taking a neutral and customer-centric view, we look at some of the basic and commonly agreed upon aspects of an SOA infrastructure.

SOA infrastructure requirements

The goal of ITSO Good is to build a flexible and durable infrastructure that makes it easy to integrate the business aligned services created in an SOA.

The basic requirements for a service oriented infrastructure are:

- ▶ *Distributed* is the ability to integrate services that are distributed across the enterprise and even beyond in the extended enterprise comprising partners and suppliers.
- ▶ *Flexibility/Interoperability* enable interaction through services that are defined by explicit implementation-independent interfaces and are loosely coupled.
- ▶ *Communication* provides support for communication protocols that stress location transparency and interoperability.
- ▶ *Integration* provides the ability to integrate services seamlessly across the heterogeneous systems and also provide support for other integration paradigms in the SOA ecosystem like message-driven and event-driven based integration.
- ▶ *Mediation services* support service routing and substitution, protocol and message transformations, and other message processing services.
- ▶ *Quality of service* includes security, transactionality, and availability.
- ▶ *Leverage existing IT investments* supports both service-enabled applications, such as Web services enabled, as well as existing applications based on traditional Enterprise Application Integration (EAI) standards and technologies, while providing a consistent service model.
- ▶ *Management* provides a central point for managing and monitoring the deployed services and also to provide the ability to integrate with systems management tools.

The infrastructure requirements above are implemented as services themselves, as *Infrastructure Services*. And more importantly these services need to be nonintrusive to the business aligned services.

Note: The requirements listed here are not exhaustive in nature. We have only thrown light upon some of the basic features of an SOA infrastructure in this chapter. There are also other aspects such as Autonomic capabilities and Infrastructure Intelligence, which are provided in some of the products available in the market today.

Relevant infrastructure requirements

The infrastructure requirements that are relevant to the given scenario are:

- ▶ *Distributed* is the ability to integrate the Retail and Warehouse services that are distributed across the enterprise to the Manufacturing services of the external partner systems.

- ▶ *Flexibility/Interoperability* enable interaction through services that are defined by explicit implementation-independent interfaces and are loosely coupled.
- ▶ *Communication* provides support for communication protocols that stress location transparency and interoperability.
- ▶ *Integration* provides the ability to integrate services seamlessly across the Manufacturing partners' heterogeneous systems. The three Manufacturers that need to be integrated with the Warehouse service of ITSO Good are implemented using CICS Transaction Server, Microsoft .NET and WebSphere Application Server.
- ▶ *Mediation services* support service routing and substitution.

11.2.2 Selecting a pattern

We use the Patterns for e-business to determine the appropriate Runtime pattern to apply to this scenario. Described here is a step-by-step approach used to navigate the Patterns for e-business asset catalog:

1. Business pattern

We select the Extended Enterprise business pattern because the given scenario requires interactions between the business processes in the Warehouse and Manufacturer systems that reside in separate enterprises.

2. Application pattern

The Warehouse and Manufacturer systems are required to interact on a one-one basis representing point-point connections. So, we select the Exposed Direct Connection application pattern. This pattern has two variations:

- Message Connection Variation
- Call Connection Variation

As the business scenario requires the proposed solution to support real-time request/reply message flows to partner processes, we select the Call Connection variation.

3. Runtime pattern

The selection of the Application pattern provides us with the Runtime patterns for the proposed solution. Because this solution requires building an SOA infrastructure, we select the *SOA profile* of the Exposed Direct Connection runtime pattern to customize the solution to the SOA environment.

Figure 11-2 shows the level 0 decomposition of the SOA profile of the Exposed Direct Connection runtime pattern, mapped on to the Exposed Direct Connection application pattern.

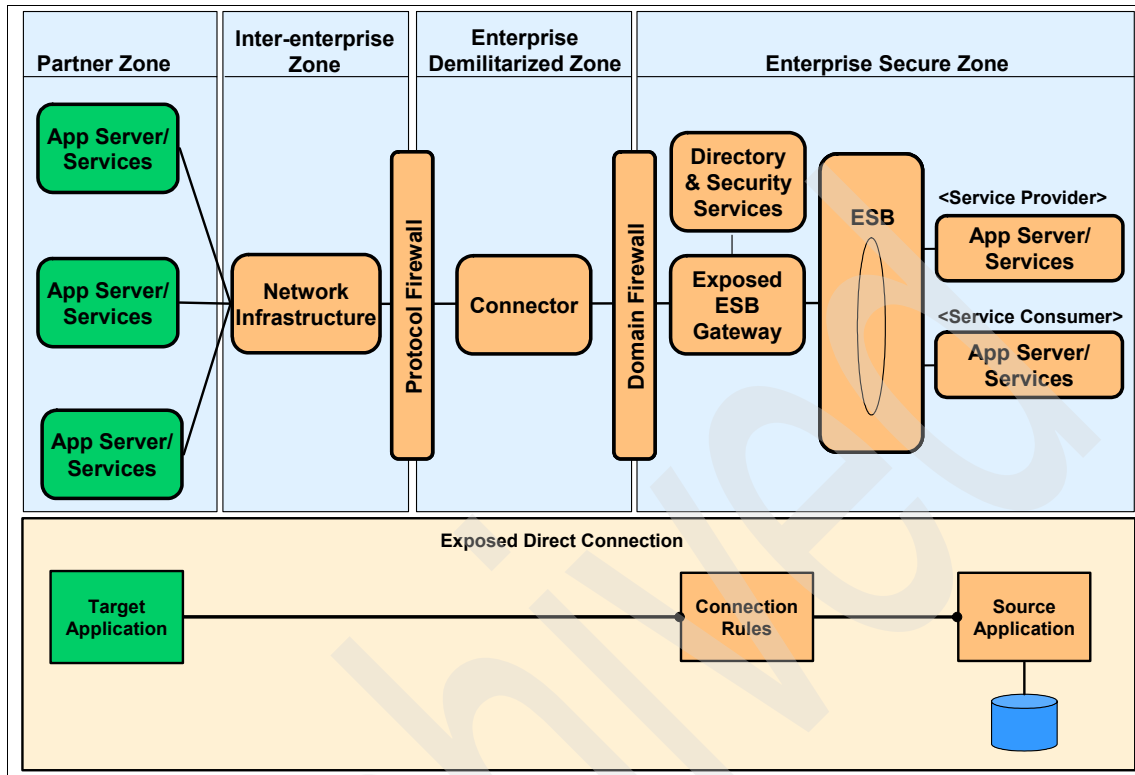


Figure 11-2 Exposed Direct Connection::Runtime pattern = SOA profile

11.2.3 Analyze design options

As shown in the Runtime pattern in Figure 11-2, the Enterprise Service Bus (ESB) is truly emerging as a middleware infrastructure component that supports the implementation of SOA within an enterprise. The need for an ESB can be seen by considering how it supports the infrastructure requirements of the SOA implementation by:

- ▶ Decoupling the consumer's view of a service from the actual implementation of the service.
- ▶ Decoupling technical aspects of service interactions (mediation services).
- ▶ Integrating and managing services in the enterprise.

The true value of the ESB concept, however, is to enable the infrastructure for SOA in a way that reflects the needs of today's enterprise: to provide suitable service levels and manageability, and to operate and integrate in a heterogeneous environment.

Note: A more detailed analysis of the various ESB capabilities is outside the scope of this redbook. In this redbook we focus on only those ESB capabilities that are relevant to the various scenarios discussed.

For more information about ESB capabilities, consult *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346.

Relevant ESB capabilities

This scenario exploits the following ESB capabilities:

- Communications

This means routing of requests from service consumers to the relevant service provider based on endpoint definitions.

- Integration

This means protocol transformation to allow decoupling of the protocol that is used between the service consumers and service providers. This allows service consumers to invoke services that are exposed using a different protocol (for example, SOAP/HTTP to SOAP/JMS).

- Security

Security services such as confidentiality, authentication and authorization are provided by a combination of secure protocols such as HTTP/S, and central control by the Exposed ESB Gateway and the ESB.

- Service interaction

The services are defined using WSDL and made available across enterprises.

Extending the ESB with the Exposed ESB Gateway for this business-to-business scenario fulfils the following requirements:

- Addressing of remote services

Remote services are defined in WSDL and the definitions are made available across enterprises.

- Security over the internet

The layering of the solution into an HTTP server, Exposed ESB Gateway, ESB and applications allows for a robust security environment to be configured.

- Restricting service access to authorized consumers

Internal service consumers can only access those external services that are provided by the Exposed ESB Gateway.

Architectural decision: implementing an ESB

There are various technology options available today to implement an ESB, ranging from the sophisticated and proprietary EAI products to the less mature open-standards based products. They differ in their degree of support for the various ESB capabilities, that enable you to build an SOA infrastructure.

Table 11-1 summarizes the requirements and looks at the various alternatives available for implementing an ESB.

Table 11-1 Architectural decision: Implementing an ESB

Subject Area	Implementing an ESB
Issue Or Problem Statement	To select the product/technology to be used for implementing an ESB that satisfies the following capabilities: 1. Provide service routing and addressing. 2. Provide support for communication protocols that stress location transparency and interoperability. In other words, the infrastructure should allow clients to invoke services in a manner independent of the service location and the communication protocol involved. 3. Ability to integrate services seamlessly across heterogeneous systems. 4. Support loosely coupled services that are defined through implementation-independent interfaces.
Assumptions	None.
Motivation	To build a basic SOA infrastructure that satisfies the current requirement and is based on open and interoperable standards. This would provide us with the required flexibility and extensibility to migrate to a more sophisticated SOA infrastructure as the standards and technologies around them mature over time.
Alternatives	Implement the ESB using: Option1: Service integration bus component of WebSphere Application Server V6.0.2 Option2: WebSphere Business Integration Message Broker.

Subject Area	Implementing an ESB
Decision	The service integration bus component of WebSphere Application Server V6.0.2 will be used to implement the ESB.
Justification	Given that the integration of the services within the ITSO Good enterprise and also their integration with the external Manufacturing partner systems is based on interoperable Web services standards, the Service Integration Bus component of WebSphere Application Server V6.0.2 would be an ideal choice because it provides extensive support to Web services based interaction. This would provide us with the basic SOA infrastructure required to start. We could then migrate it to a more sophisticated infrastructure that would involve a combination of the service integration bus and WebSphere Business Integration Message Broker in a federated manner.

WebSphere Business Integration Message Broker as an ESB

WebSphere Business Integration Message Broker is a mature messaging middleware that satisfies most of the ESB capabilities:

- ▶ Supports high volume and complex data transformations, routing decisions and data validation.
- ▶ Integration with heterogeneous systems through the use of WebSphere Business Integration Adapters and WebSphere MQ bridges.
- ▶ Support for Web services in terms of service routing, SOAP message transformation and protocol translation. However V5 does lack the sophistication of Web services support that might be required in an ESB implementation which makes extensive use of these standards like WS-Security, and so forth.

Building an ESB that is based entirely on WebSphere Business Integration Message Broker is an option when there is a need to interoperate with systems that are not only Web services-based, but also quality-of-service requirements demand the use of mature middleware.

WebSphere Application Server as an ESB

Building an ESB using the service integration bus component of WebSphere Application Server is an option when Web services support is critical and the service provider and consumer environment is predominantly built on J2EE.

The role of the Exposed ESB Gateway

In our scenario, the Warehouse service requires access to the external manufacturing service through the ESB. Access to these external services requires us to provide additional control (service routing) and security measures. This is achieved by using the Exposed ESB Gateway component as depicted in Figure 11-2 on page 242.

An Exposed ESB Gateway makes the services of one organization available to others, and vice versa, in a controlled and secure manner. Although this might require capabilities such as trading partner provisioning and management, which are distinct from ESB capabilities, the *intent* of this component is different from the intent of the ESB, which is to provide a service infrastructure *within* an organization. For both these reasons, the Exposed ESB Gateway is likely to be integrated to, but not part of, the ESB. This component allows us to provide more sophisticated control over external service access by providing additional security models and additional transformations of protocol, data formats, or models of delivery assurance.

Architectural decision: Securing the Web service interaction

In this section we look into how the additional business requirements of this scenario influences the security solution that was put in place in Chapter 10, “Exposed Direct Connection runtime pattern: generic profile” on page 157.

Securing the Web service interaction between the Warehouse and Manufacturing services has been discussed in detail in the previous chapter. What changes in the given scenario in this chapter is the need to provide a controlled and secured access to these external services from within the enterprise, through the Exposed ESB Gateway component. This means that we have to move these Web service security features (WS-Security) from within the internal applications of the enterprise to the Exposed ESB Gateway component so that they can be centrally managed and governed in accordance with the corporate security policy.

Architectural decision: Maintaining an audit trail

When business interactions span between enterprises in any typical B2B integration scenario, it is important to ensure that an effective audit trail mechanism is put in place through which the access and usage of external services is monitored.

The key drivers that influence the auditing requirements are:

- ▶ Regulatory requirements such as the Sarbanes-Oxley Act in the U.S., and industries such as health care, utilities and banking mean that business activities are heavily regulated in most countries.
- ▶ To provide assurance that the business transactions are recorded in an accurate manner.
- ▶ To handle *nonrepudiation* of any claims made by partners.

In our business scenario, this means that the external manufacturing partners should be unable to claim that a replenishment order *was not* requested when in fact it was. That is, it ensures nonrepudiation of the transaction by the partner. Digitally signed receipt acknowledgements of messages can be demanded depending on the auditing requirements.

In an enterprise system, audit-trails are not only maintained for events related to user activity or business transactions but they are also maintained for other types of events such as:

- ▶ Configuration events that are produced by the enterprise infrastructure to monitor health
- ▶ Security events that are generated to help detect external intrusions

Having identified the key drivers that influence the auditing requirements of an enterprise, we can now look at some of the design considerations that are involved in providing an auditing solution.

Common design considerations for an auditing solution are:

- ▶ Distributed versus centralized collection and storage of audit trails and events:
In the distributed model, audit data typically remains in the system where the data is generated. With the centralized approach, data is sent to a central collection and data storage and logging facility. Having a centralized repository of audit data makes it easier to ensure compliance with regulatory requirements and also makes it easier to administer changes necessitated by the security policies of an enterprise at a single central location.
- ▶ Common representation of audit data and events

One of the common issues faced when these audit trails are maintained locally across various heterogeneous systems in an enterprise is that they are represented in different formats. This makes it difficult to correlate events generated across different systems for a given user activity. The Common Base Event (CBE) specification is an effort to standardize the representation of these audit data and events. For more information about CBE and related standards, refer to resources at the following links:

<http://www.ibm.com/developerworks/library/specification/ws-cbe/>

Also, products like WebSphere Partner Gateway have support for a separate nonrepudiation repository.

For the business scenario considered in this chapter, we implement a simple auditing solution that store the events in a centralized repository using the Logging Facility as shown in Figure 11-1 on page 238.

The architecture decision discussed in Table 11-2 evaluates the options for implementing an effective audit trail mechanism.

Table 11-2 Architectural Decision: Handling the Auditing Requirement

Subject Area	Implementing the audit trail for B2B interactions.
Issue Or Problem Statement	Need to put in an effective audit mechanism to monitor and track the usage of the Manufacturing services by the Warehouse. Need to decide which component in the proposed solution would handle this responsibility.
Assumptions	None.
Motivation	Ensure compliance to industry regulations.
Alternatives	<ol style="list-style-type: none"> 1. Log and store audit events in a distributed manner at the Retail, Warehouse and SCM application systems respectively. 2. Log and store audit events in a centralized <i>Logging Facility</i>.
Decision	The audit trail for the interactions/ with the external manufacturing partner systems would be maintained in the centralized <i>Logging Facility</i> .
Justification	Storing these events in a centralized repository facilitates the correlation of events generated across the different systems in response to a online customer request. It also allows for easier administration of security policies related to auditing as well as compliance to federal regulatory requirements.

11.2.4 Products

The SOA profile of the Exposed Direct Connection pattern differs from the Generic profile in its use of an ESB and an Exposed ESB Gateway in its Runtime pattern as shown in Figure 11-2 on page 242. So in this section we look at the products available to implement the ESB and the Exposed ESB Gateway component.

Product implementation options

Product choices for this scenario are based on:

- ▶ Design decisions discussed in 11.2.3, “Analyze design options” on page 242
- ▶ Extended Enterprise capabilities of the products
- ▶ Products that are currently available

ESB component

We can use the following currently available products to implement the ESB component in the given scenario:

- ▶ WebSphere Application Server Network Deployment V6.0.2
- ▶ WebSphere Message Integration Message Broker

For this scenario, the service integration bus in WebSphere Application Server Network Deployment V6.0.2 meets all of the requirements and hence is the product of choice.

Exposed ESB Gateway component

We can use the following currently available products to implement the ESB component in the given scenario:

- ▶ Web services gateway component available in WebSphere Application Server Network Deployment V6.0.2.
- ▶ WebSphere Partner Gateway.

In our scenario, there is no requirement for advanced functions such as Partner Management and provisioning that are provided by the WebSphere Partner Gateway product. Because the Web services gateway component in WebSphere Application Server Network Deployment V6.0.2 is used to implement the Exposed ESB Gateway.

Product mapping selected

The complete product mapping for this scenario is shown in Figure 11-3 below:

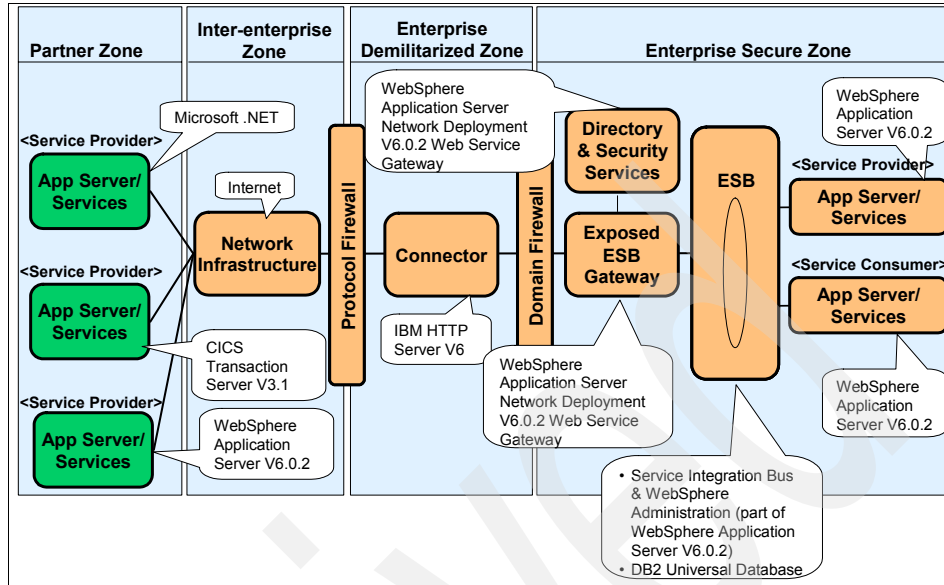


Figure 11-3 Exposed Direct Connection:: Product mappings

In this Product mapping, the WebSphere Application Server V6.0.2 Network Deployment was used for all services within the ITSO Good enterprise. The Manufacturing services of the three external partner systems are implemented using CICS Transaction Server V3.1, WebSphere Application Server V6.0.2 and Microsoft .NET respectively.

The ESB is implemented using the service integration bus component provided in the WebSphere Application Server Network Deployment and uses a DB2 Universal database internally for holding the SDO repository.

The ESB component routes service requests from within the enterprise to these external Manufacturing partner systems through the Exposed ESB Gateway component implemented using the Web Service Gateway component of WebSphere Application Server Network Deployment V6.0.2.

11.3 Development guidelines

The scenario in this book is based on the WS-I sample application. This section discusses how the scenario makes use of the Exposed Direct Connection runtime pattern using the SOA profile.

11.3.1 Exposed Direct Connection interaction: SOA profile

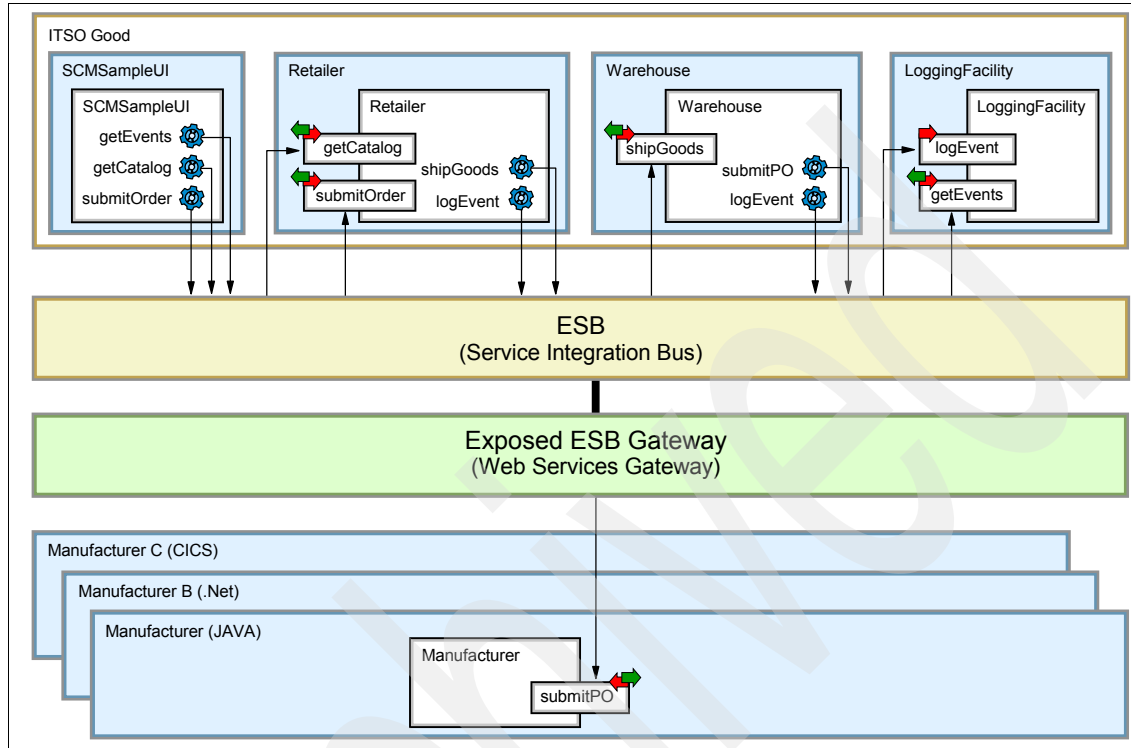


Figure 11-4 Scenario implementation using the Exposed Direct Connection:: SOA profile

As shown in Figure 11-4, the SOA profile is implemented by introducing the ESB and the Exposed ESB Gateway component. The ESB integrates all the services calls at an enterprise level. Any interaction with an extended enterprise are managed through the Exposed ESB Gateway (Web Services Gateway).

In the Exposed Direct Connection pattern Generic profile where services consumer and providers are within the enterprise, services consumers invoke the service provider directly.

In the SOA profile of the Exposed Direct Connection pattern, where services consumer and providers are within the enterprise, services consumer invoke an inbound service on the ESB. This inbound service is responsible for sending the request to an outbound service, which then forwards the request to service provider.

Where the service provider is in an extended enterprise, services consumer invoke an inbound service on the ESB. The ESB is responsible for sending the

request to the Exposed ESB Gateway, which then invokes the service provider. The Exposed ESB Gateway provides extended enterprise capabilities such as security.

11.4 Runtime guidelines

This section takes you through the steps that are involved for configuring the sample application using the Exposed Direct Connection SOA profile pattern. It assumes the following products are already installed:

- ▶ WebSphere Application Server Network Deployment V6.0.2
- ▶ DB2 Universal Database V8.2

This section describes the following activities:

- ▶ Creating the basic infrastructure
- ▶ Create and configure a service integration bus
- ▶ Create and configure a Web services gateway
- ▶ Create and configure a service integration bus link
- ▶ Enabling security on the Web services gateway

In this chapter we describe how to build an SOA implementation of the Exposed Direct Connection scenario progressively. To allow the user to test each of the activities listed easily, we use unsecure versions of the ITSOGood and Manufacturer applications when creating and configuring the service integration bus, Web services gateway, and the service integration bus link.

Finally we replace the unsecure version of the Manufacturer application with a version of the application which has security constraints applied to incoming messages, and describe how to enable the Web services gateway to support WS-Security.

11.4.1 Solution topology

As in the previous chapter, to represent the complete business scenario the sample application is divided into four subapplications:

- ▶ ITSOGood contains the SCMSampleUI, Retailer, Warehouse, and LoggingFacility services.
- ▶ Manufacturer, ManufacturerB, ManufacturerC are three individual services, each packaged separately, and deployed to three different enterprises.

As described in the Product mapping in “Product mapping selected” on page 249, we use WebSphere Application Server Network Deployment V6.0.2 to host the ITSOGood and Manufacturer applications, Microsoft .NET to host the

ManufacturerB application, and CICS Transaction Server to host the ManufacturerC application. This is shown in Figure 11-5.

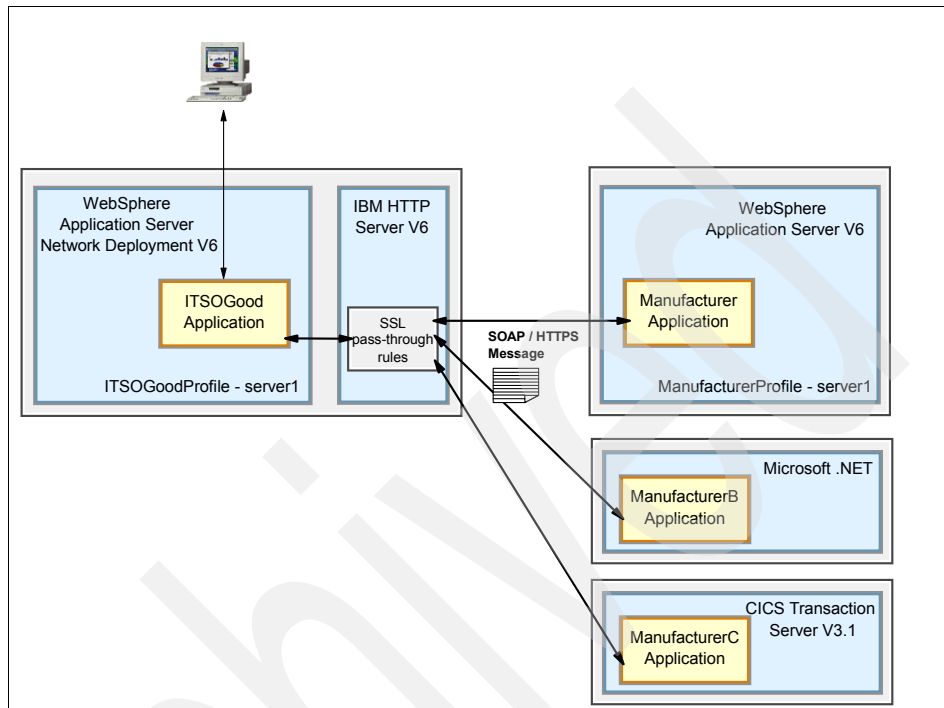


Figure 11-5 Solution topology

The runtime guidelines in this chapter describe how to prepare the ITSOGoodProfile server profile as an ESB and an Exposed ESB Gateway. This includes the configuration of WS-Security integrity and confidentiality which in this scenario is defined in the Exposed ESB Gateway rather than in individual enterprise applications.

The ESB component is primarily implemented in the service integration bus feature of WebSphere Application Server Network Deployment V6.0.2. The Exposed ESB Gateway component is implemented in the Web services gateway feature of WebSphere Application Server Network Deployment V6.0.2.

Optionally, you can implement the ManufacturerB and ManufacturerC applications as well. To do this, you will need access to a Microsoft .NET server and a CICS Transaction Server V3.1 region. The connection between the ITSOGood applications and the ManufacturerB and ManufacturerC applications will not be secured using WS-Security settings in this sample (although in a production environment this should be configured too).

Note: You do not need to configure the ManufacturerB and ManufacturerC servers to build a working end-to-end sample application.

Instructions for configuring the ManufacturerB and ManufacturerC servers are described in:

- ▶ Appendix B, “Microsoft .NET Web services” on page 483
- ▶ Appendix C, “CICS Transaction Server Web services” on page 513

11.4.2 Creating the basic infrastructure

If you have completed the previous scenario (Chapter 10, “Exposed Direct Connection runtime pattern: generic profile” on page 157) you can use that configuration as the basis for this scenario.

However, if you have not completed the previous scenario, you need to complete the following steps:

1. Create WebSphere Application Server server profiles for ITSOGoodProfile and ManufacturerProfile by following the instructions in 10.4.2, “Configuring WebSphere Application Server profiles” on page 217.
2. Configure an HTTP server to perform:
 - Hosting of WSDL files used by the enterprise applications, as described in 10.4.3, “Hosting the WSDL files” on page 219.
 - Configuration of SSL pass-through as described in 10.4.6, “Configuring an HTTP server for SSL pass-through” on page 224.

After you have completed these steps, or if you are reusing the configuration you built for the previous scenario, you need to make the following additions to build the basic infrastructure:

- ▶ Add the ITSOGoodProfile server profile to a WebSphere Application Server Network Deployment deployment manager. This is required to enable the Web services gateway feature.
- ▶ Install ITSOGood and Manufacturer enterprise applications that do not contain WS-Security settings. In this scenario the WebSphere Application Server administrative console is used to define these WS-Security configuration settings rather than the enterprise applications themselves.

Creating a deployment manager

To enable Web services support, you need to create and start a deployment manager, and federate the ITSOGoodProfile server to this deployment manager.

Perform the following steps:

1. Start the WebSphere Application Server profile creation wizard. On a Windows system, select **Start** → **Programs** → **IBM WebSphere** → **Application Server Network Deployment v6** → **Profile creation wizard**.
2. When the profile creation wizard starts, select **Create a deployment manager profile**. Specify the following settings in this profile:
 - Profile name: Dmgr01
 - Node name: ITSOGoodCellManager01
 - Host name: itsogood.itso.ral.ibm.com
 - Cell name: ITSOGoodCell01
3. Start the deployment manager by running:

```
<WAS_HOME>\bin\startManager -profileName Dmgr01
```
4. Federate the ITSOGoodProfile server to the deployment manager node:

```
<WAS_HOME>\bin\addNode itsogood.itso.ral.ibm.com -profileName ITSOGoodProfile
```
5. Start the ITSOGoodProfile server:

```
<WAS_HOME>\bin\startServer server1 -profileName ITSOGoodProfile
```

Installing ITSOGood and Manufacturer applications

The ITSOGood and Manufacturer enterprise applications used in the previous scenario contain settings for WS-Security integrity and confidentiality. At runtime WebSphere Application Server uses these settings to determine the levels of WS-Security to apply. The downside to this approach is that it requires each application to specify its security configuration at build-time, and this configuration needs to be replicated for all enterprise applications that need to use security.

In this scenario, the WS-Security configuration is configured through the WebSphere Application Server Network Deployment administrative console. Therefore, we need to replace the ITSOGood and Manufacturer enterprise applications that contain WS-Security settings used in the previous scenario with new enterprise applications that do not contain WS-Security settings.

Perform the following steps:

1. Log in to the WebSphere Application Server Network Deployment administrative console of the deployment manager:

```
http://itsogood.itso.ral.ibm.com:9062/ibm/console
```

Note: This URL uses a port of 9062. The HTTP port assigned to your deployment manager might be different.

2. Click **Applications** → **Enterprise Applications**.
3. Click **Install New Application** to install a new version of ITSOGood.
4. Browse to the path where the ITSOGood enterprise application can be found. You can find this enterprise application in the additional material that is supplied with this redbook at:

\DirectConnectionSOA\ears\ITSOGood_NoSec.ear

5. Click **Next** click **Step 7 Summary**, then click **Finish** to install the new enterprise application.
6. When it is installed, save the configuration, and start the new ITSOGood enterprise application.

Repeat this process for the Manufacturer enterprise application on the ManufacturerProfile server. Remember to complete the following:

1. Log in to the administrative console:

<http://manufacturera.itso.ra1.ibm.com:9061/ibm/console>

Note: This URL uses a port of 9061. The HTTP port assigned to your ManufacturerProfile server may be different

2. Remove the existing Manufacturer enterprise application, if it is already installed from the previous scenario.
3. Install the new Manufacturer enterprise application from:

\DirectConnectionSOA\ears\Manufacturer_NoSec.ear

4. Start the new Manufacturer enterprise application.

Finally, you need to override the Web service client binding in the ITSOGood enterprise application so that it points to the HTTP server that forwards the request on to the Manufacturer.

Perform the following in the administrative console of the deployment manager:

1. Select **Applications**, and click **Enterprise Applications**.
2. From the list of installed applications click **ITSOGood** → **EJB Modules** → **WarehouseEJB.jar** → **Web services client bindings**.
3. Click **Edit** under the Port Information column for ManufacturerService.
4. The Port Information screen allows us to override the endpoint URL. Set the Overridden Endpoint URL to:

<http://itsogood.itso.ra1.ibm.com/Manufacturer/services/Manufacturer>

This means that the Web service client for the Manufacturer (located in the WarehouseEJB project of ITSOGood) will use this URL to contact the Web

service. This URL points to the HTTP server, where the request will be redirected to the external Manufacturer Web service. See Figure 11-6.

Enterprise Applications > ITSOGood > EJB Modules > WarehouseEJB.jar > Web services client bindings > ManufacturerService:WarehouseSoapBindingImpl

Specifies a request timeout, an overridden endpoint URL, and an overridden binding namespace can be set for a port. The timeout determines how many seconds to wait for a request. A value of zero disables the timeout. The current endpoint and binding namespace can be overridden.

Port	Request Timeout (seconds)	Overridden Endpoint URL	Overridden Binding Namespace
Manufacturer	0	http://itsogood.itso.ral.ibm.com/Manufacturer/s	

Apply OK Reset Cancel

Figure 11-6 Overriding the endpoint URL for the Manufacturer

5. Click **OK**, then save your changes.
6. Restart the ITSOGood enterprise application to ensure the change to the Web services client binding is in effect.

You should now have a working end-to-end scenario that does not use WS-Security:

- ▶ If you want to test this scenario, follow the instructions in 10.4.8, “Testing the scenario” on page 229.
- ▶ You can use the TCP/IP Monitor to confirm the SOAP messages do not contain WS-Security integrity and confidentiality entries by following the instructions in 10.4.9, “Viewing SOAP messages using the TCP/IP Monitor” on page 233.

11.4.3 Create and configure a service integration bus

In this section, you create a service integration bus that implements many of the functions required of the ESB component in the SOA profile of the Exposed Direct Connection pattern (Figure 11-7).

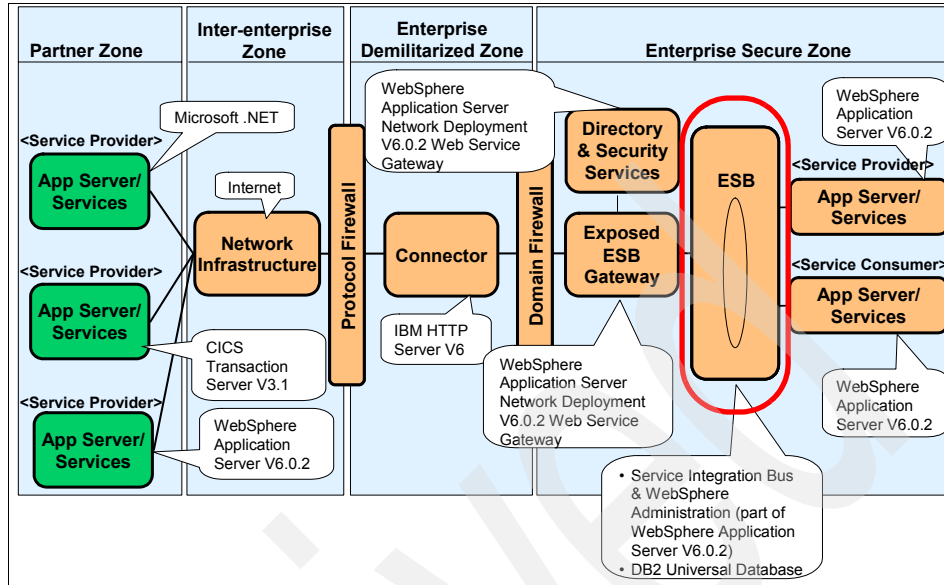


Figure 11-7 Exposed Direct Connection:: Product mappings

In this section you perform all of the steps necessary to configure a service integration bus to redirect Web services requests to appropriate Web services endpoints.

Creating a bus

The first step in the process is to create a new service integration bus. We use the Web services support of the service integration bus to act as an intermediary between the Web services invocation. In the context of this scenario, the service integration bus is a component of the ESB.

Create a bus on the ITSOGoodProfile server:

1. In the deployment manager administrative console, expand **Service integration** and click **Buses**.
2. Click **New**. In the field labeled Name, enter ESBBus, as shown in Figure 11-8 on page 259.
3. Uncheck the **Secure** check box, because we do not require any security checks on nodes within the ITSOGood enterprise. For the other values, accept defaults.

Buses > New

A service integration bus supports applications using message-based and service-oriented architectures interconnected servers and clusters that have been added as members of the bus. Applications connect to the messaging engines associated with its bus members.

Configuration

General Properties

* Name
ESBBus

UUID
E1A75F995CCDEEBF

Description

Security

☐ Secure

Inter-engine authentication alias
(none)

Mediations authentication alias
(none)

Inter-engine transport chain

☐ Discard messages

The additional properties will not be available until general properties for this item are saved.

Topology

- Bus members
- Messaging engines
- Foreign buses

Destination resources

- Destinations
- Mediations

Services

- Inbound Services
- Outbound Services

Additional Properties

- Custom properties

Related Items

- J2EE Connector Architecture (J2C) authentication entries

Figure 11-8 New bus details entry page

- Click **OK**, and the bus is created. Save the changes.

Adding a bus member

Creating a bus just creates an administrative entity. It does not create any resources for messaging. To create resources, we need to add a bus member, which has the effect of creating a messaging engine. The messaging engine is used by the bus to forward requests and responses between service providers and service consumers.

To add a bus member, follow these steps:

- Click **ESBBus** to show its properties. Under Topology, click **Bus members**.
- Click **Add**. Accept the defaults, and click **Next**.

3. This page is a details summary page. Click **Finish**. The server is added as a member of the bus, and a messaging engine created.
4. Save the changes.

Installing the Service Data Objects (SDO) repository

The service integration bus Web services support stores the WSDL and schemas for Web services in the SDO repository. When WebSphere Application Server is installed it does not install the SDO repository, so this step must be performed manually.

To install the SDO repository, follow these steps:

1. In a command prompt window, navigate to the <WASND_HOME>/bin directory, where <WASND_HOME> is the directory where you installed WebSphere Application Server Network Deployment.
2. Run the following command to create the SDO repository enterprise application on the ITSOGoodProfile server:

```
wsadmin -f installSdoRepository.jacl ITSOGoodNode server1
```

3. After the script completes successfully, the SDO repository has been installed. You can confirm the installation by examining the installed enterprise applications. You should see a new enterprise application added and started that is called SDO Repository.

Configuring the SDO repository

The SDO repository supports a wide variety of databases. By default, the SDO repository uses embedded Cloudscape. However, in this scenario, we use DB2 Universal Database.

To create a database, follow these steps:

1. Open the DB2 Command Window by clicking **Start → Programs → IBM DB2 → Command Line Tools → Command Window**.
2. From the DB2 Command Window enter the following:
 - a. db2 create database sdodb
 - b. db2 connect to sdodb
 - c. db2 create schema sdorep
 - d. db2 create table sdorep.bytestore (name varchar(250) not null, bytes blob(1G), timestamp1 bigint not null)
 - e. db2 disconnect sdodb
3. Now with the databases and tables created, configure the connectivity to them from WebSphere Application Server. Create a directory structure into

which the client JAR files for DB2 Universal Database should be placed. This is to ensure that WebSphere Application Server can connect to the database through the drivers supplied by the database.

- a. From a Command Prompt or from Windows Explorer navigate to <WAS_HOME> and create a subdirectory name db2udb.
- b. Copy these following three jar files from <DB2_HOME>\java to <WAS_HOME>\db2udb:

```
db2jcc.jar  
db2jcc_license_cu.jar  
db2jcc_license_cisuz.jar
```

4. The data source used by the SDO repository needs to have a component-managed authentication alias. An authentication alias is used to allow the same user ID and password combination to be used in many different places. In this case the DB2 database has security configured so you need to specify the same user ID and password as created during the DB2 install. Complete the following steps to create an alias:
 - a. In the deployment manager administrative console navigation panel click **Security** → **Global security**
 - b. Under Authentication expand **JAAS Configuration** and click **J2C Authentication data** → **New**.
 - c. Enter the setting values as shown in Figure 11-9 on page 262.

Global security

[Global security > J2EE Connector Architecture \(J2C\) authentication data entries](#)

Specifies a list of user IDs and passwords for Java 2 connector security to use.

Configuration

General Properties

* Alias
DB2ESBAlias

* User ID
db2admin

* Password

Description
DB2 Alias for SDO and ME

Apply OK Reset Cancel

Figure 11-9 Setting the J2C authentication data

- Alias

This field contains the name by which this alias will be known in the admin console. This alias can be anything you want, but in this case we specify the name DB2ESBAlias.

- User ID

You must specify a value for the user ID that will be used to log in. Specify the same value as the ID created when installing DB2. In this case we specify the name db2admin.

- Password

This is the password associated with the user ID. A value must be specified. Specify the same value as the password created when installing DB2.

- d. Click **OK** and save the changes to the master configuration by clicking **Save**.

5. Creating a JDBC provider for IBM DB2 Universal Database V8.2

The next step is to configure the service integration bus to access the SDO repository database using DB2. To do this, you need to define a new JDBC provider. In the administration console, complete the following steps to create a JDBC provider:

- a. In the navigation panel click **Resources** → **JDBC Providers**.
- b. Clear the **Node** entry field and click **Apply**,
- c. With the cell scope now applied, click **New** to create the JDBC provider.
- d. Next, some general information about the type of database and the connection mechanism is needed as shown in Figure 11-10. It should be noted that the pull-down boxes are disabled until the values in the preceding boxes have been entered.

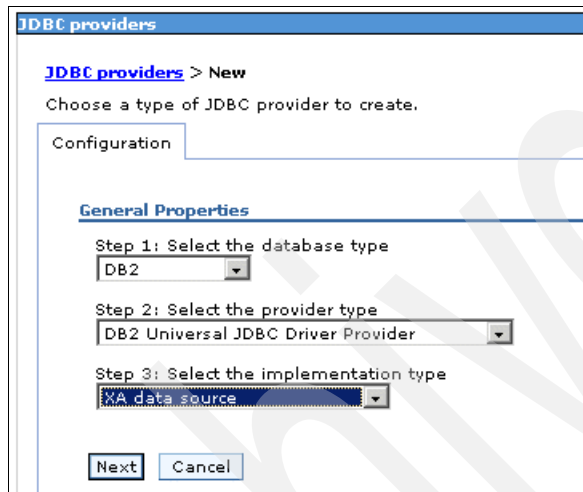


Figure 11-10 Specifying properties for DB2 JDBC provider

- i. Setup the database type.
This is used to specify the type of database to which the JDBC provider will connect. In this case, choose **DB2**.
- ii. Select the provider type.
This is used to specify how the database will be accessed. In this case, choose **DB2 Universal JDBC Driver Provider**.
- iii. Select the implementation type.
This is used to specify how the provider will be implemented. In this case choose **XA data source**.
- e. Click **Next**.
- f. Modify the following JDBC provider properties panel:
 - i. Remove the existing **Class path** entries and add the three DB2 classes you copied to the <WAS_HOME>\db2udb directory in the previous section.
 - ii. Clear the **Native library path** field.

- iii. The completed panel is shown in Figure 11-11. Accept all other defaults and click **OK**.

Note: Figure 11-11 is restricted to showing two of the three DB2 classes. Ensure you enter all three. Also, ensure that you enter the class path entries using the forward slash to delimit directories.

JDBC providers > New

JDBC providers are used by the installed applications to access data from databases.

Configuration

General Properties

* Scope
cells:ITSO Server01Cell01:nodes:ITSO Server01Node01

* Name
DB2 Universal JDBC Driver Provider (XA)

Description
XA DB2 Universal JDBC Driver-compliant Provider. Datasources created under this provider support the use of XA to perform 2-phase commit

Class path
c:\jar
C:\WAS\AppServer\db2udb\db2jc
c_license_cisuz.jar
C:\WAS\AppServer\db2udb\db2jc
c_license_cu.jar

Native library path

* Implementation class name
com.ibm.db2.jcc.DB2XADataSource

The additional properties will not be a general properties for this item are:

Additional Properties

- Data sources
- Data sources (Version 4)

Apply OK Reset Cancel

Figure 11-11 Specifying additional properties for DB2 JDBC provider

- g. Save the changes to the master configuration by clicking **Save**.
6. Create the JDBC data sources.

The next step is to create the JDBC data sources for accessing the SDO and messaging engine tables in DB2. Assuming that you are at the JDBC providers panel from the preceding step, complete the following steps to create the JDBC data sources:

- a. Click **DB2 Universal JDBC Driver Provider (XA)** → **Data sources** → **New**.

- b. You now need to enter the details for the new data source used for the SDO repository. The panel is a long, scrolling, browsing panel. The top half should look like Figure 11-12.

General Properties

* Scope
cells:ITSOServer01Cell01:nodes:ITSOServer01Node01

* Name
SDODB Datasource

JNDI name
jdbc/com.ibm.ws.sdo.config/SdoRepository

☒ Use this Data Source in container managed persistence (CMP)

Description
DB2 Universal Driver Datasource

Category

Data store helper class name

☒ Select a data store helper class

Data store helper classes provided by WebSphere Application Server

- DB2 Universal data store helper
(com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper)
- DB2 for iSeries data store helper
(com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper)

Figure 11-12 Specifying the SDO data source parameters - part 1

The bottom half of the same panel should look similar to Figure 11-13 on page 266.

☐ Specify a user-defined data store helper
Enter a package-qualified data store helper class name

Component-managed authentication alias
Component-managed authentication alias
ITSGoodCellManager01/DB2ESBAlias

Authentication alias for XA recovery
☒ Use component-managed authentication alias
☐ Specify:

Container-managed authentication
Container-managed authentication alias (deprecated in V6.0, use resource reference authentication settings instead)
(none)
Mapping-configuration alias (deprecated in V6.0, use resource reference authentication settings instead)
(none)

DB2 Universal data source properties
* Database name
SDODB
* Driver type
4
Server name
itsogood.itso.ral.ibm.com
Port number
50000

Apply OK Reset Cancel

Figure 11-13 Specifying the SDO data source parameters - part 2

- Name

The name is just an administrative entity that only has meaning within the administrative console. This can be specified as anything you like.

In our example we used SDODB Datasource.

- JNDI Name

The JNDI name is the location from where applications pick the data source. Specify a value of:

`jdbc/com.ibm.ws.sdo.config/SdoRepository`

- Component-managed authentication alias

This is the alias that will be used when making connections to the database where the application managed authentication is being used by the application.

Select the value that ends in **DB2ESBAlias**.

- DB2 Universal data source properties

Enter the path to the SDO DB2 database.

- i. Database name

The name of the SDO database. Enter SD0DB.

- ii. Driver type

Change to driver type 4.

Note: Type 4 JDBC drivers are direct-to-database pure Java drivers (*thin* drivers).

- iii. Server name

The host name where the DB2 server is running. Enter
itsogood.itso.ral.ibm.com

- c. Leave all other values as default. Click **OK**.

- d. Save the changes to the master configuration by clicking **Save**.

- e. To ensure that the data sources have been created successfully check the datasource and click **Test connection**. You should see confirmation of a successful connection.

Note: If the data source connection fails, then try restarting the deployment manager, node and the application server.

7. Return to the command prompt window, then navigate to <WAS_HOME>\bin. Change the SDO data source backend type by running the following command:

```
wsadmin -f installSdoRepository.jacl -editBackendId DB2UDB_V82
```

Installing the Web services applications

The service integration bus Web services support is packaged in four different applications. To get support for SOAP over HTTP, it is only necessary to install three of them. The fourth package is required for SOAP over JMS support. You need to install only three applications for this scenario. To install these applications, follow these steps:

1. Install the resource adapter first. To install the resource adapter navigate to <WAS_HOME>\bin and execute the following command:

```
wsadmin -f <WAS_HOME>/util/sibwsInstall.jacl INSTALL_RA -installRoot  
<WAS_HOME> -nodeName ITSOGoodNode
```

Important: Replace <WAS_HOME> with the directory where you installed WebSphere Application Server. The second <WAS_HOME> must have elements in the path separated by a forward slash (/) even on Windows system. So a path of C:\WAS\AppServer becomes C:/WAS/AppServer.

2. Install the Web services support application:

```
wsadmin -f <WAS_HOME>/util/sibwsInstall.jacl INSTALL -installRoot  
<WAS_HOME> -nodeName ITSGoodNode -serverName server1
```

3. Although the Web services support has been installed, you cannot use it until at least one endpoint listener application is installed. There are two endpoint listener applications: one for SOAP over HTTP and one for SOAP over JMS. For this scenario, you need to install the SOAP over HTTP application only. Install this application by running the following command:

```
wsadmin -f <WAS_HOME>/util/sibwsInstall.jacl INSTALL_HTTP -installRoot  
<WAS_HOME> -nodeName ITSGoodNode -serverName server1
```

Creating the endpoint listener

Next, you need to create an endpoint listener, which will use the endpoint listener application at runtime. Endpoint listeners listen for incoming Web service requests and forward them onto the relevant inbound serve. Inbound services are bound to an endpoint listener when they are created.

1. From the administrative console of the deployment manager, expand **Servers** and click **Application servers**.
2. Click **server1**.
3. Under Additional Properties, click **Endpoint Listeners**.
4. Click **New**.
5. Create and endpoint listener using the dialog box as shown in Figure 11-14 on page 269.

Application servers

[Application servers](#) > [server1](#) > [Endpoint Listeners](#) > **New**

An endpoint listener receives requests from service requester applications within a specific application server or cluster.

Configuration

General Properties

* Name
SOAPHTTPChannel1

Description

* URL root
http://itsogood.itso.ral.ibm.c

* WSDL serving HTTP URL root
psrv1a.itso.ral.ibm.com/wsd

Additional Properties

■ Connection Properties

Apply OK Reset Cancel

Figure 11-14 Creating an endpoint listener

In this dialog box, enter the following information:

- ▶ The Name, which is the name of the endpoint listener, must have the name SOAPHTTPChannel1.
- ▶ The URL root is the base URL for Web service requests into this endpoint listener. The URLs that are used for making Web service requests to the service integration bus will have this root at the beginning.

Set this to:

http://itsogood.itso.ral.ibm.com:9080/wsgwsoaphttp1

You can replace 9080 with the correct port number for your server.

- ▶ WSDL serving HTTP URL root, which is the location of the HTTP URL that is serving your Web service WSDL. Enter a value of:

http://appsrv1a.itso.ral.ibm.com/wsd1

6. Click **Apply**.
7. Under Additional Properties click **Connection Properties**.
8. Click **New** to create a new connection property.
9. Select **ESBBus** from the Bus Name list and click **OK**.
10. Save the changes.

Creating the outbound services

Outbound services define Web service requests that leave the service integration bus and are received by a Web service provider. We need to define an outbound service for every service that we will route through the service integration bus. These services are illustrated in Figure 11-15.

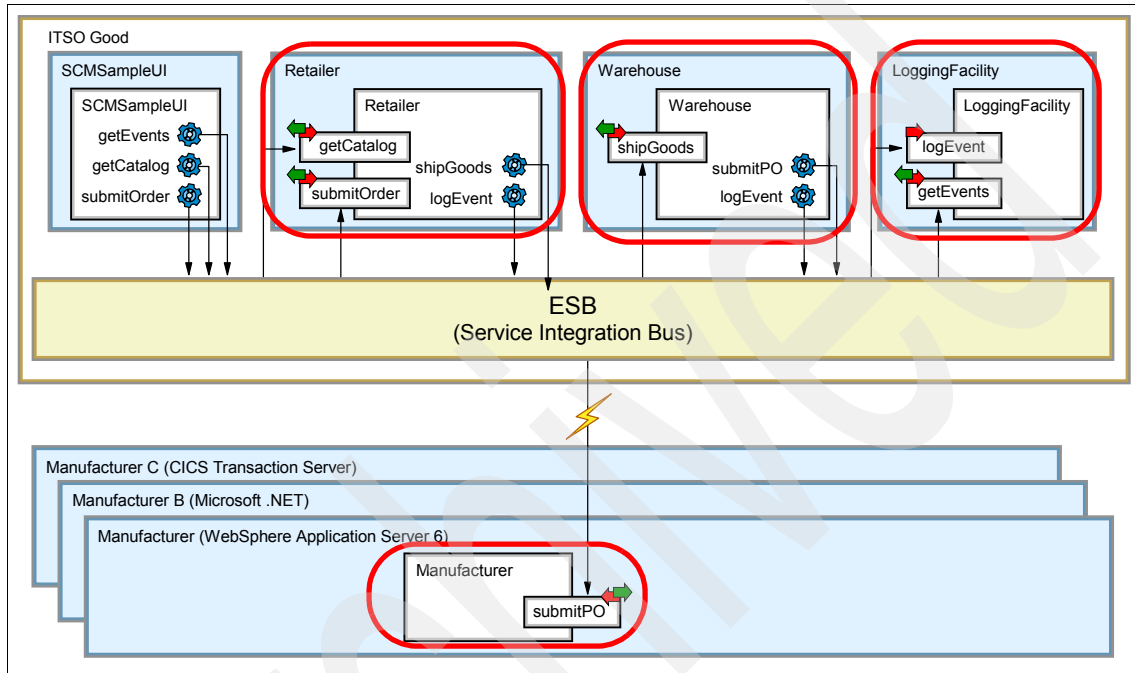


Figure 11-15 Web services that require an outbound service defined in the bus

Note that in Figure 11-15 you can see we have not yet implemented an Exposed ESB Gateway, therefore the service integration bus contacts each Manufacturer service directly. We will add an Exposed ESB Gateway to access the Manufacturers later in the chapter.

To define an outbound service for the LoggingFacilityService Web service, perform the following tasks:

1. From the administrative console, expand **Service Integration** and click **Buses**.
2. Click **ESBBus**.
3. Under Services, click **Outbound Services**.
4. Click **New**.

5. The first page of the wizard (Figure 11-16) requires you to specify a URL or UDDI repository where a WSDL definition of the service can be found. In this case, use a URL. The URL option allows you to specify an HTTP URL or a file system path. Enter the following URL and click **Next**:

`http://appsrv1a.itso.ral.ibm.com/wsdl/LoggingFacility_Impl.wsdl`

The screenshot shows a wizard window titled "New outbound service". It contains a sidebar with five steps: Step 1 (selected), Step 2, Step 3, Step 4, and Step 5. The main area is titled "Locate the target service WSDL" and contains the instruction "Select the location of the WSDL that describes the service provider". There are two radio buttons: "URL" (selected) and "UDDI". Below them is a text field for "WSDL location" containing the URL "http://appsrv1a.itso.ral.ibm.com/wsdl/LoggingFacility_Ir". At the bottom, there is a dropdown menu for "WSDL UDDI registry" set to "(none)". "Next" and "Cancel" buttons are at the bottom left.

Figure 11-16 Web service definition selection

6. The next page displays the available services that are defined in the WSDL file. You can select the service for which you want to create an outbound service. In this case, there is only one service to select, **LoggingFacilityService**. Click **Next**.
7. The next page (Figure 11-17 on page 272) displays the ports that are defined for the selected service. There is only one port in our service, so select **LoggingFacility** and click **Next**.

New outbound service

An outbound service represents a WSDL-described service.

Step 1: Locate the target service WSDL
Step 2: Select service
→ **Step 3: Select Ports**
Step 4: Name the outbound service and destinations
Step 5: Select assigned bus members for port destinations and the port selection mediation, if required.

Select Ports

Select the ports that are to be enabled for this service

Select	Port name	Binding	Port address
<input checked="" type="checkbox"/>	LoggingFacility	http://schemas.xmlsoap.org/soap/http	http://localhost:9080/LoggingFacility

Previous Next Cancel

Figure 11-17 Port selection page

8. On this page you can change the name of the outbound service, service destination name and port destination name. You can also specify a port selection mediation. Accept the defaults and click **Next**.
9. On this page you can select the bus member you wish to assign the outbound service. Accept the default and click **Finish**. The outbound service is created.
10. Repeat these steps to define outbound services for the other Web services, as shown in Table 11-3.

Table 11-3 Outbound service definitions

Web service	WSDL location
ManufacturerService	http://appsrv1a.itso.ral.ibm.com/wSDL/Manufacturer_Impl.wSDL
ManufacturerBService	http://appsrv1a.itso.ral.ibm.com/wSDL/ManufacturerB_Impl.wSDL
ManufacturerCService	http://appsrv1a.itso.ral.ibm.com/wSDL/ManufacturerC_Impl.wSDL
RetailerService	http://appsrv1a.itso.ral.ibm.com/wSDL/Retailer_Impl.wSDL
WarehouseService	http://appsrv1a.itso.ral.ibm.com/wSDL/Warehouse_Impl.wSDL

11. Save the changes. You should see six outbound services defined, as shown in Figure 11-18 on page 273.

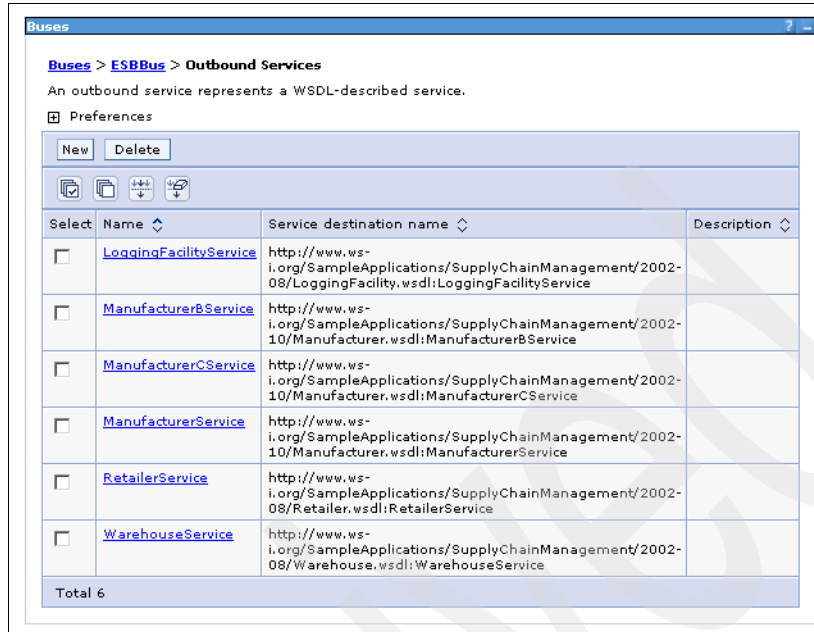


Figure 11-18 All outbound services defined

12.A Web service destination and port destination are created for each outbound service. You can see these destinations by selecting **Destinations** in the bus details page.

Tip: If the WSDL file referenced in the WSDL location field is updated at any time after creation of an outbound service, you must click **Reload** on the Outbound service panel and save the configuration. The reason for this is the values from the WSDL file are saved to the SDO repository and the file is then not referenced directly. Changing the WSDL file means the reference in the SDO repository must also be reloaded.

Creating the inbound services

Inbound services define Web service requests that are received by the service integration bus. These requests are then routed to the appropriate outbound service. We need an inbound service definition for each outbound service.

To define an inbound service for the LoggingFacilityService Web service, perform the following tasks:

1. From the bus details page for ESBBus under Services, click **Inbound Services**.

- Click **New**.
- Select the service destination name and supply the template WSDL service definition, as shown in Figure 11-19.

Buses Close page

New Inbound service

An inbound service describes the Web service enablement of a service destination. It provides the configuration of endpoint listeners within a port.

→ **Step 1: Select the service destination and template WSDL location**

Step 2: Select service from template WSDL

Step 3: Name the inbound service and select endpoint listeners

Step 4: Define UDDI publication properties

Select the service destination and template WSDL location

Select the service destination that is to be enabled for Web service access, and the location of the WSDL that defines the portType service.

* Service destination name

Template WSDL location type

☒ URL
☐ UDDI

* Template WSDL location

Template WSDL UDDI registry

Figure 11-19 Service destination and template WSDL settings page

- Service destination name, which is the destination on which the inbound service requests should be placed. In this case, you want to specify the Web service destination that was created for the LoggingFacility outbound service.

Select the following:

`http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/LoggingFacility.wsdl:LoggingFacilityService`

- Template WSDL location, which specifies the WSDL definition of the Web service to be invoked. While the WSDL that is used by client applications will be slightly different, it is based on this WSDL. In this scenario, you can specify the WSDL of the Web service endpoint that will ultimately be invoked after the request has been routed through the bus:

Enter the following:

`http://appsrv1a.itso.ral.ibm.com/wsdl/LoggingFacility_Impl.wsdl`

- Click **Next**.
- Select the template WSDL service that should be used. Our WSDL has only one entry so accept the default, and click **Next**.
- Rename the inbound service and specify which endpoint listener is to be used, as shown in Figure 11-20 on page 275.

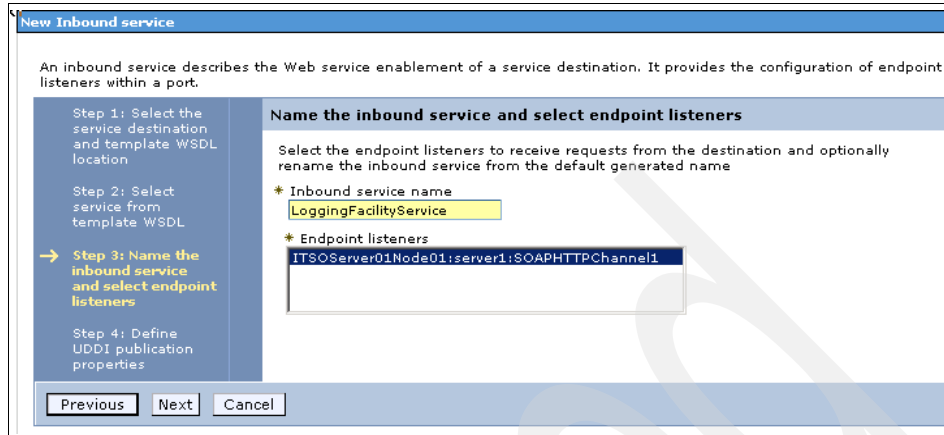


Figure 11-20 Specify inbound service name and endpoint listener

- Inbound Service name is the service in the WSDL. It affects the code that is generated by the application development tooling. By default, the name is based on the service destination name with InboundService at the end. Enter **LoggingFacilityService**.
 - Endpoint listener defines what mechanism is used to get Web service requests into the inbound service. There is only one endpoint listener available, for **SOAPHTTPChannel1**.
7. Click **Next**.
 8. Specify UDDI specific properties. Because you are not using UDDI, you can accept the defaults and click **Finish**.
 9. The default port name for the inbound service is based on the endpoint listener name followed by the phrase InboundPort. In this case, the inbound port name is **SOAPHTTPChannel1InboundPort**. Because our clients are calling a port called **LoggingFacility**, the clients would be unable to invoke the service. To fix this issue:
 - a. From the inbound service listing page, click **LoggingFacilityService**.
 - b. Under Additional Properties, click **Inbound Ports**.
 - c. Click the port name that ends with **SOAPHTTPChannel1_InboundPort**.
 - d. Modify the inbound port name as shown in Figure 11-21 on page 276. Change the name to **LoggingFacility** and click **OK**.

General Properties

* Name
LoggingFacility

Description

* Endpoint listener
ITSO Server01Node01:server1:SOAPHTTPChannel1

Template port name
LoggingFacility

JAX-RPC handler list

Figure 11-21 Inbound port details page

10. Repeat these steps to create the other inbound service definitions. Use Table 11-4, Table 11-5, Table 11-6 on page 277, Figure 11-7 on page 277, and Table 11-8 on page 277 for guidance.
11. When all inbound services are defined, save the changes.

Table 11-4 Inbound service settings for *ManufacturerService*

Field	Value
Service destination name	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-2008/Manufacturer.wsdl:ManufacturerService
Template WSDL location	http://appsrv1a.itso.ral.ibm.com/wsdl/Manufacturer_Impl.wsdl
Inbound service name	ManufacturerService
Endpoint listener	SOAPHTTPChannel1
Inbound port name	Manufacturer

Table 11-5 Inbound service settings for *ManufacturerBService*

Field	Value
Service destination name	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-2008/Manufacturer.wsdl:ManufacturerBService
Template WSDL location	http://appsrv1a.itso.ral.ibm.com/wsdl/ManufacturerB_Impl.wsdl
Inbound service name	ManufacturerBService
Endpoint listener	SOAPHTTPChannel1

Field	Value
Inbound port name	ManufacturerB

Table 11-6 Inbound service settings for ManufacturerCService

Field	Value
Service destination name	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-2008/Manufacturer.wsdl:ManufacturerCService
Template WSDL location	http://appsrv1a.itso.ral.ibm.com/wsdl/ManufacturerC_Impl.wsdl
Inbound service name	ManufacturerCService
Endpoint listener	SOAPHTTPChannel1
Inbound port name	ManufacturerC

Table 11-7 Inbound service settings for RetailerService

Field	Value
Service destination name	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-2008/Retailer.wsdl:RetailerService
Template WSDL location	http://appsrv1a.itso.ral.ibm.com/wsdl/Retailer_Impl.wsdl
Inbound service name	RetailerService
Endpoint listener	SOAPHTTPChannel1
Inbound port name	Retailer

Table 11-8 Inbound service settings for WarehouseService

Field	Value
Service destination name	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-2008/Warehouse.wsdl:WarehouseService
Template WSDL location	http://appsrv1a.itso.ral.ibm.com/wsdl/Warehouse_Impl.wsdl
Inbound service name	WarehouseService

Field	Value
Endpoint listener	SOAPHTTPChannel1
Inbound port name	<i>Warehouse</i>

Overriding Web services client bindings

Currently, all Web services calls are point-to-point connections. Now that you have created the inbound services on the service integration bus, you have to change your Web services consumers to point to the inbound services on the service integration bus. For example Figure 11-23 shows how a Web service client for Retailer needs to be changed to point to the service integration bus.

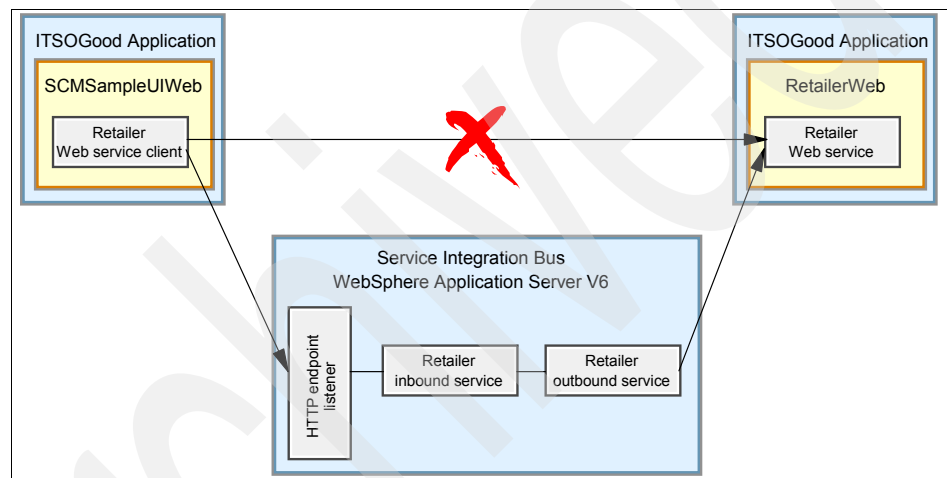


Figure 11-22 Overriding the Web service client bindings for the Retailer Web service

This can be achieved by changing the endpoint for the Web services provider in the code or by overriding the Web services client binding through the admin console. In this chapter, we implement the latter.

To override Web services client binding, follow these steps:

1. In the administrative console, click **Service integration** → **Buses**, then click **ESBBus**.
2. Under Services, click **Inbound Services** and then **RetailerService**.
3. Under Additional Properties, click **Publish WSDL files to ZIP file**.
4. On this page, click **RetailerService.zip**. Click **Open** on the file dialog that opens.

5. If you have a program with ZIP files, a list of WSDL files will open. Select and open the service WSDL file in a text editor. In our case, this is the **ESBBus.RetailerServiceService.wsdl** file.

6. Make note of the *location* text:

`http://itsogood.itso.ral.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/ESBBus/RetailerService/Retailer`

Now that you know the endpoint for your inbound service, you will override the Web service client binding in the enterprise application's Web and EJB modules.

7. Click **Applications** → **Enterprise Application**, and then click **ITSOGood** and **Web modules**.
8. Click **SCMSampleUIWeb.war** and the under Additional Properties click **Web services client bindings**.
9. On the Web services client bindings screen, click **Edit..** under the **Port Information** column for **RetailerService** Web service.
10. Enter the following in the **Overrriden Endpoint URL** field for **Retailer** port with the endpoint for the **RetailerService** inbound service.

`http://itsogood.itso.ral.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/ESBBus/RetailerService/Retailer`

This is the endpoint URL taken from ESBBus.RetailerServiceService.wsdl. See Figure 11-23.

The screenshot shows the 'Enterprise Applications' console. The breadcrumb trail is: Enterprise Applications > ITSOGood > Web module > SCMSampleUIWeb.war > Web services client bindings > RetailerService. Below the breadcrumb trail is a text area explaining that a request timeout, an overridden endpoint URL, and an overridden binding namespace can be set for a port. The main table is titled 'Port Information for Web service RetailerService'. It has four columns: Port, Request Timeout (seconds), Overrriden Endpoint URL, and Overrriden Binding Namespace. The 'Retailer' port is listed with an empty request timeout field, the endpoint URL 'http://itsogood.itso.ral.ibm.com:9080/wsgwsoa', and an empty binding namespace field. At the bottom are buttons for Apply, OK, Reset, and Cancel.

Port	Request Timeout (seconds)	Overrriden Endpoint URL	Overrriden Binding Namespace
Retailer		http://itsogood.itso.ral.ibm.com:9080/wsgwsoa	

Figure 11-23 Overriding the endpoint URL for RetailerService

11. Repeat these steps to update the following Web service client bindings so they point to the relevant inbound service:
 - SCMSampleUIWeb
 - LoggingFacilityService Web service

- RetailerWeb
 - LoggingFacilityService Web service
 - WarehouseService Web service
- WarehouseEJB
 - LoggingFacilityService Web service
 - ManufacturerService Web service
 - ManufacturerBService Web service
 - ManufacturerCService Web service

Testing sample application and the service integration bus

At this stage, you can test your configuration. This test does not use an Exposed ESB Gateway to contact the Manufacturers, and does not use WS-Security. However, it is a way to validate that you have correctly implemented the previous steps.

Figure 11-24 shows how the ESB currently connects to the Manufacturers. The HTTP server is used to ensure that SOAP messages sent to each Manufacturer are sent over SOAP/HTTPS.

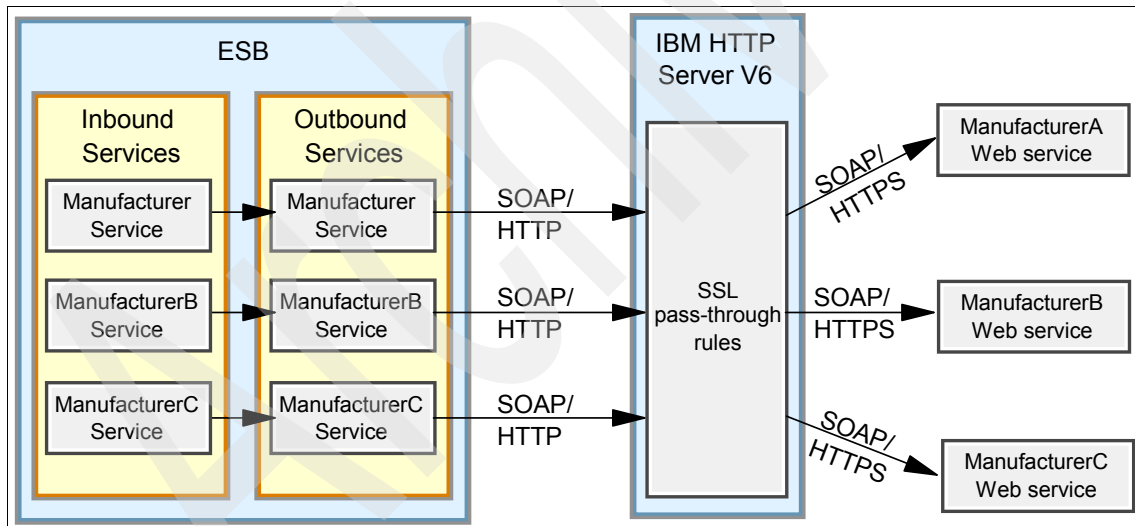


Figure 11-24 Implementation of Manufacturers without an Exposed ESB Gateway

Note: If you have only implemented ManufacturerA, the Warehouse will only be able to replenish stock for 605001, 605004, and 605007.

1. When all configuration has been completed, restart the application server.

2. You can test the sample application by opening a Web browser and entering:
`http://itsogood.itso.ral.ibm.com:9080/SCMSampleUI/`
3. For instructions on how to test the application server, see 10.4.8, “Testing the scenario” on page 229.

11.4.4 Create and configure the Web service gateway

The Web services gateway is essentially a tool that gives administrators an easy way to map Web services across a service integration bus. It enables the mapping of different protocols. In our scenario, one protocol (SOAP/HTTP) is used for the Web service invocation. A SOAP/HTTP request comes into the gateway from the ESB (another service integration bus) and is then retargeted to an external service, also over SOAP/HTTP.

Figure 11-25 on page 282 shows the flow of request and response messages through the bus, and highlights the different steps required for configuring a gateway service as follows:

1. Create an endpoint listener and reply destination and connect it to the bus.
2. Create a gateway instance on the bus.
3. Create a gateway service on the gateway instance, including creating and configuring:
 - a. Request and response destinations for the gateway
 - b. An outbound service and an outbound service destination
 - c. An outbound port and an outbound port destination
 - d. An inbound port
4. Export the Web service gateway WSDL.
5. Connect the client (in our case the ESB) to the gateway service.

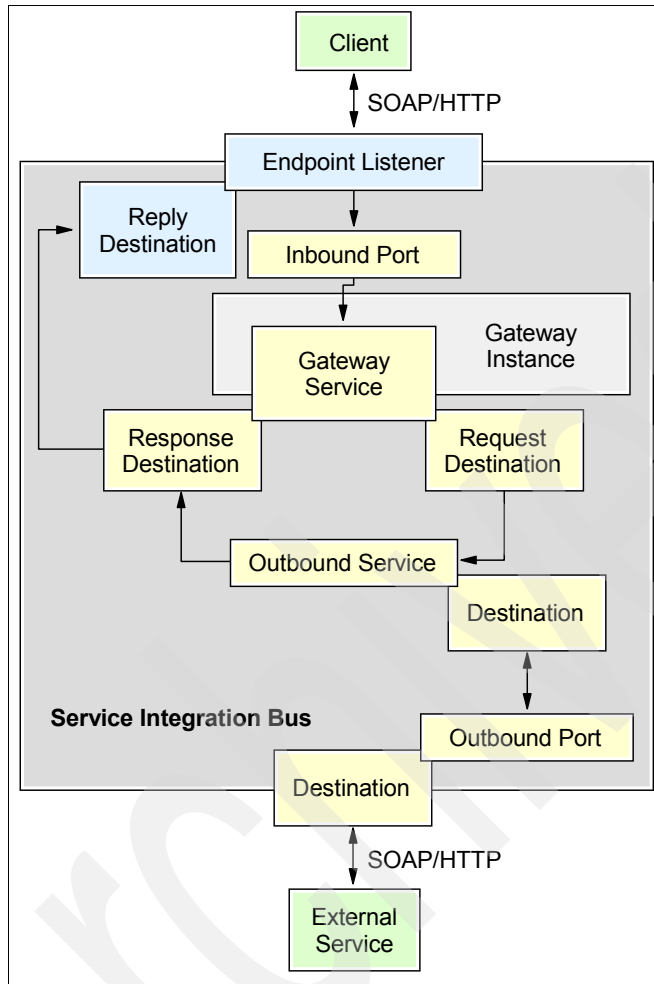


Figure 11-25 Overview of Web services gateway configuration

Creating a bus

We first need to create a new service integration bus. We will then create a Web service gateway instance to run on this bus. In the context of this scenario the Web services gateway is the Exposed ESB gateway component of the Direct Connection runtime pattern: SOA profile.

In the ITSOGood server profile, create a bus named `ExposedESBGatewayBus` and a bus member as described in “Creating a bus” on page 258 and “Adding a bus member” on page 259.

Create an endpoint listener

We need to create a new endpoint listener as an entry point into the gateway. “Creating the endpoint listener” on page 268 describes the steps required to set up an endpoint listener. Use the following settings on the Endpoint Listeners page in the WebSphere Application Server Network Deployment administrative console to configure the new endpoint listener:

- ▶ Name: SOAPHTTPChannel2
- ▶ URL Root: `http://itsogood.itso.ra1.ibm.com:9080/wsgwsoaphttp2`
- ▶ WSDL serving HTTP URL Root: `http://appsrv1a.itso.ra1.ibm.com/wsdl`
- ▶ Additionally, create a new connection property for this new endpoint listener and set it to the `ExposedESBGatewayBus`.

Create a gateway instance

Create a gateway instance on the bus. The gateway instance allows you to partition gateway and proxy services into logical groups for ease of management.

1. Expand **Service Integration** and click **Buses**.
2. Click **ExposedESBGatewayBus**.
3. Under Additional Properties, click **Web service gateway instances**.
4. Click **New**.
5. Specify the settings for the gateway instance as shown in Figure 11-26 on page 284.

Figure 11-26 Create a gateway instance

Enter the following values then click **OK**:

- Name, which is an arbitrary name for the gateway instance. Enter a value of ExposedESBGateway.
- Gateway namespace, also an arbitrary name. Enter a value of http://esb.gateway.
- Default proxy WSDL URL, which is the generic template WSDL file supplied with WebSphere Application Server Network Deployment. Enter http://itsogood.itso.ral.ibm.com:9080/sibws/proxywsdl/ProxyServiceTemplate.wsdl.

6. Save the changes.

Create a gateway service

The gateway service is associated with a single external service, defined by the WSDL supplied when creating the gateway service. To create the gateway service:

1. Expand **Service Integration** and click **Buses**.
2. Click **ExposedESBGatewayBus**.

3. Under Additional Properties, click **Web service gateway instances**.
4. Click **ExposedESBGateway**.
5. Under Additional Properties, click **Gateway services**.
6. Click **New**.
7. Select **WSDL-defined web service provider** as the type of target service, shown in Figure 11-27.

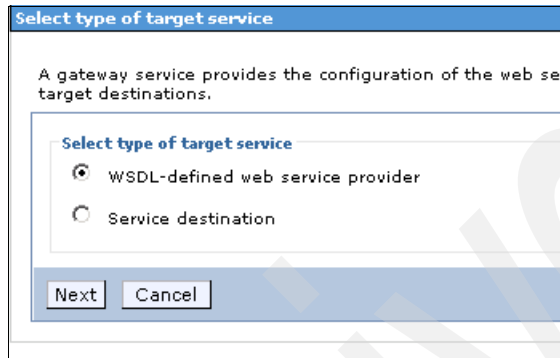


Figure 11-27 Selecting the type of target service

8. Click **Next**.

Configuring the request and response destinations

9. Enter the gateway service and destination names as shown in Figure 11-28 on page 286.

New gateway service

A gateway service provides the configuration of the web service enablement of a service destination, all information that maps to one or more target destinations.

→ **Step 1: Specify gateway service name, service destinations and mediations**

Step 2: Locate the target service WSDL

Step 3: Select service

Step 4: Select ports

Step 5: Name the service and port destinations

Step 6: Select points for service and port destinations

Step 7: Select endpoint listeners

Step 8: Define

Specify gateway service name, service destinations and mediations

Provide the names of the objects that are to be used to support the gateway at the runtime.

* Gateway service name

Gateway request destination name

Request mediation

Request mediation bus member

Gateway response destination name

Response mediation

Response mediation bus member

Figure 11-28 Specifying the gateway service and service destination names

Some of the values can keep their defaults. The following list are the values that you must enter:

- Gateway service name: ManufacturerGatewayService (This is an arbitrary name.)
- Gateway request destination name: ManufacturerGatewayRequestDestination. This destination processes the request messages for the gateway.
- Gateway response destination name: ManufacturerGatewayResponseDestination. This destination processes the response messages for the gateway.

Note: In WebSphere Application Server Network Deployment, a *destination* is defined as a virtual location within a service integration bus. Applications can attach as producers, consumers or both to exchange messages.

10. Click **Next**.

Configuring an outbound service and destination

11. Specify the location of the target service WSDL using the page displayed in Figure 11-29. We are using a URL location type.

The screenshot shows the 'New gateway service' configuration page. On the left, a sidebar lists five steps: Step 1: Specify gateway service name, service destinations and mediations; Step 2: Locate the target service WSDL (highlighted with a yellow arrow); Step 3: Select service; Step 4: Select ports; and Step 5: Name the service and port destinations. The main content area is titled 'Locate the target service WSDL' and contains the instruction 'Select the location of the WSDL that describes the service provider'. Below this, there are two radio buttons for 'WSDL location type': 'URL' (selected) and 'UDDI'. A text field for 'WSDL location' contains the URL 'http://appsrv1a.itso.ral.ibm.c'. Below that, a dropdown menu for 'WSDL UDDI registry' is set to '(none)'.

Figure 11-29 Locating the target service WSDL

Enter the following value for the WSDL location:

`http://appsrv1a.itso.ral.ibm.com/wsdl/Manufacturer_Impl.wsdl,`

12. Click **Next**.

13. We now select the service in the WSDL to configure. The Manufacturer application only exposes one service, so we can select the default value, shown in Figure 11-30.

The screenshot shows the 'New gateway service' configuration page at Step 3: Select service. The sidebar on the left shows Step 2: Locate the target service WSDL as the previous step. The main content area is titled 'Select service' and contains the instruction 'Select a service that is defined in the WSDL'. Below this, there is a dropdown menu for 'Service' with the value '{http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl'.

Figure 11-30 Selecting the service

14. Click **Next**.

Selecting an outbound port and port destination

15. We can now select the port to be enabled. As shown in Figure 11-31, for our application, we select the default port.

New gateway service

A gateway service provides the configuration of the web service enablement of a service destination, along with the information that maps to one or more target destinations.

Step 1: Specify gateway service name, service destinations and mediations
Step 2: Locate the target service WSDL
Step 3: Select service
→ **Step 4: Select ports**

Select ports

Select the ports that are to be enabled for this service

Select	Port name	Binding	Port address
<input checked="" type="checkbox"/>	Manufacturer	http://schemas.xmlsoap.org/soap/http	http://localhost:9080/Manufacturer

Figure 11-31 Selecting the port

16. Click **Next**.

17. The external service is configured as an outbound service, and we associate its destination with the gateway service destination created earlier. The page used to name these destinations is shown in Figure 11-32.

New gateway service

A gateway service provides the configuration of the web service enablement of a service destination, along with the information that maps to one or more target destinations.

Step 1: Specify gateway service name, service destinations and mediations
Step 2: Locate the target service WSDL
Step 3: Select service
Step 4: Select ports
→ **Step 5: Name the service and port destinations**
Step 6: Select points for service and port destinations

Name the service and port destinations

Optionally rename the outbound service and the service and port destination default generated names if required, provide the name of the port selected if any, and identify the default port.

* Outbound service name
ManufacturerOutboundService

* Service destination name
ManufacturerOutboundServiceDestination

Port selection mediation
(none)

Default port	Port name	Port destination name
<input checked="" type="radio"/>	Manufacturer	* ManufacturerOutboundPortDestination

Figure 11-32 Naming the destinations

18. Some of the values can keep their defaults. The following lists the values that you must enter:

- Outbound service name: ManufacturerOutboundService
- Service destination name: ManufacturerOutboundServiceDestination
- Port destination name: ManufacturerOutboundPortDestination

19. Click **Next**.

Configuring an inbound port

20. To select the bus member, accept the default and click **Next**, as shown in Figure 11-33.

The screenshot shows a web-based configuration interface titled "New gateway service". It includes a sidebar with a list of steps: Step 1 (Specify gateway service name, service destinations and mediations), Step 2 (Locate the target service WSDL), Step 3 (Select service), Step 4 (Select ports), Step 5 (Name the service and port destinations), and Step 6 (Select points for service and port destinations). Step 6 is highlighted with a yellow arrow. The main content area is titled "Select points for service and port destinations" and contains the text "Each port destination needs to be assigned to a bus member". Below this text is a table with two columns: "Port Name" and "Bus member". The "Port Name" column has a single entry "Manufacturer". The "Bus member" column has a dropdown menu with the selected value "ITSOServer01Node01:server1".

Port Name	Bus member
Manufacturer	ITSOServer01Node01:server1

Figure 11-33 Selecting the bus member

21. We can now select the endpoint listener we created earlier by accepting the default value and clicking **Next**. The page for selecting the endpoint listener is shown in Figure 11-34 on page 290.

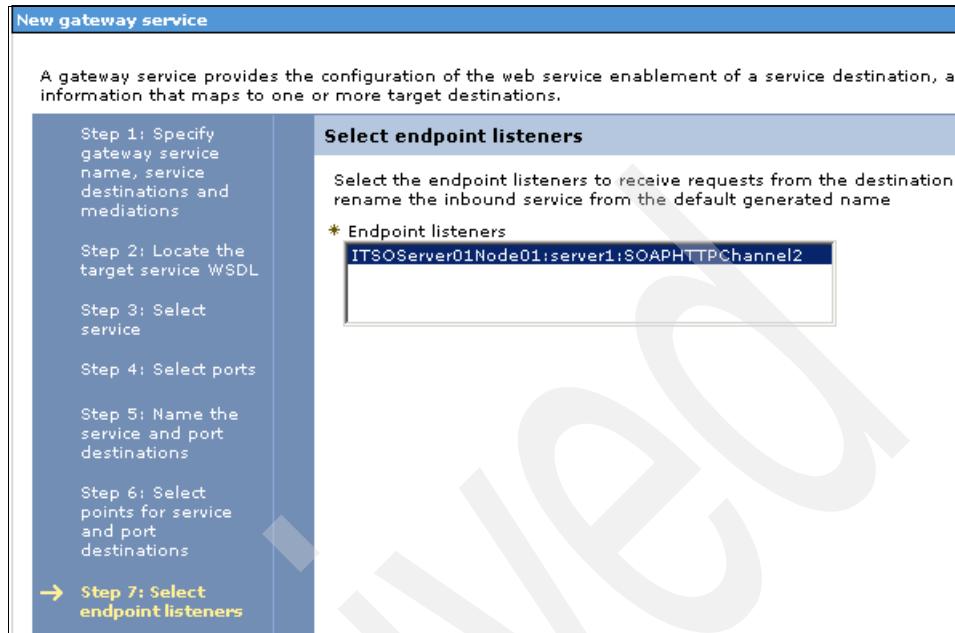


Figure 11-34 Selecting the endpoint listener

22. For our scenario, we will not be publishing to a UDDI registry, so just click **Finish** on the final page.

23. Save the changes.

Renaming the inbound port

Rename the inbound port by following these steps:

1. Expand **Service Integration** and click **Buses**.
2. Click **ExposedESBGatewayBus**.
3. Under Services, click **Inbound Services**.
4. Click on **ManufacturerGatewayService**.
5. Under Additional Properties, click **Inbound Ports**.
6. Replace ITSOServer01Node01_server1_SOAPHTTPChannel2_InboundPort with **ManufacturerGatewayInboundPort** (Figure 11-35 on page 291) then click **OK**.

The screenshot shows a configuration window titled "Buses". The breadcrumb path is "Buses > ExposedESBGatewayBus > Inbound Services > ManufacturerGatewayService". The selected item is "ITSOServer01Node01_server1_SOAPHTTPChannel2_InboundPort". A description states: "An inbound port describes the Web service enablement of a service destination on a s associated configuration." The "Configuration" tab is active. Under "General Properties", the "Name" field is highlighted and contains "ManufacturerGatewayInboundPort". The "Description" field is empty. The "Endpoint listener" dropdown is set to "ITSOServer01Node01:server1:SOAPHTTPChannel2". Other dropdowns for "Template port name", "JAX-RPC handler list", "Security request binding", "Security response binding", and "Security configuration" are all set to "(none)". At the bottom are buttons for "Apply", "OK", "Reset", and "Cancel".

Buses

Buses > ExposedESBGatewayBus > Inbound Services > ManufacturerGatewayService

ITSOServer01Node01_server1_SOAPHTTPChannel2_InboundPort

An inbound port describes the Web service enablement of a service destination on a s associated configuration.

Configuration

General Properties

* Name
ManufacturerGatewayInboundPort

Description

* Endpoint listener
ITSOServer01Node01:server1:SOAPHTTPChannel2

Template port name
Manufacturer

JAX-RPC handler list
(none)

Security request binding
(none)

Security response binding
(none)

Security configuration
(none)

Apply OK Reset Cancel

Figure 11-35 Renaming the inbound port

7. We have now configured a service integration bus to contain a Web service gateway as shown in Figure 11-36 on page 292.
8. Save your changes.

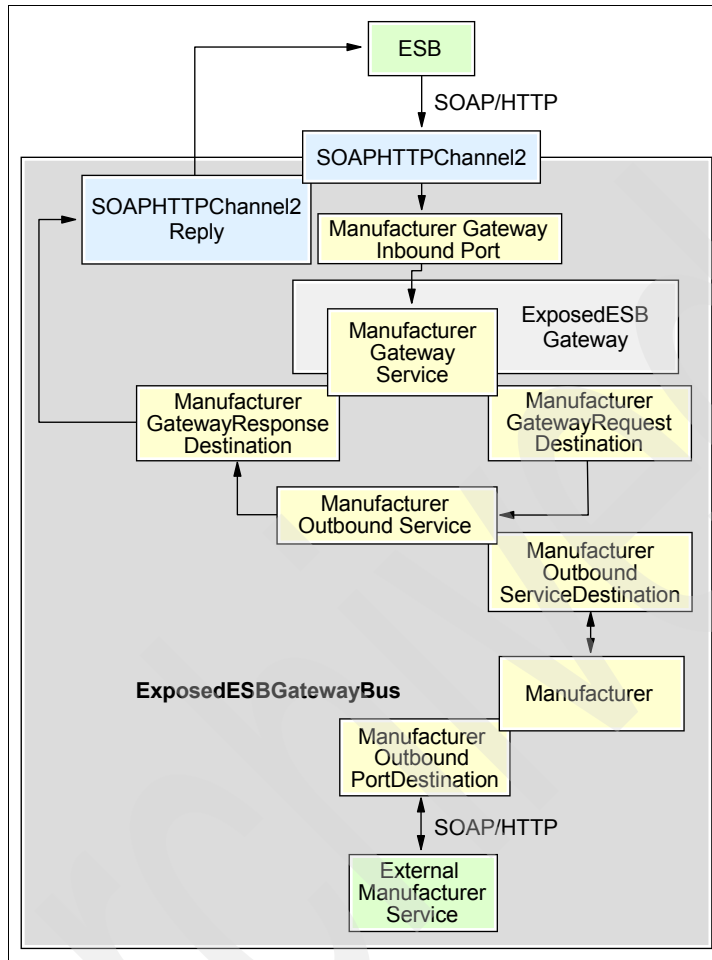


Figure 11-36 Our Web service gateway configuration

Exporting the gateway WSDL

We need to export the WSDL that defines the inbound service for the Manufacturer Web service gateway service, so that the ESB can be configured to access this service.

1. Expand **Service Integration** and click **Buses**.
2. Click **ExposedESBGatewayBus**.
3. Under Services, click **Inbound Services**.
4. Click **ManufacturerGatewayService**.
5. Click **Publish WSDL files to ZIP file**.

6. On this page, click **ManufacturerGatewayService.zip**. Click **Open** on the file dialog that appears.
7. A list of WSDL files as shown in Figure 11-37 will open in the program associated with ZIP files. Extract these files to the IBM HTTP Server WSDL directory (<HTTP_SERVER_HOME>/htdocs/en_US/wsd1), so that this new gateway WSDL will be served by the HTTP server.

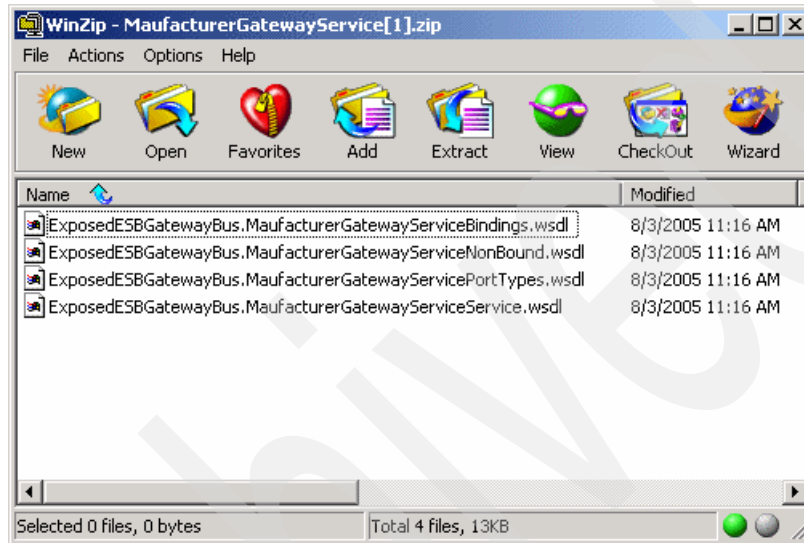


Figure 11-37 Exported WSDL files

Creating Gateway services for the other Manufacturers

We now need to create GatewayServices in a similar manner for ManufacturerB and ManufacturerC. To do so apply the steps and naming conventions from "Create a gateway service" on page 284.

You do not need to complete these steps if you are not implementing ManufacturerB and ManufacturerC.

11.4.5 Connecting the ESB and the Exposed ESB Gateway

When the ESB (implemented in the service integration bus) needs to communicate with the Manufacturer Web services, it delegates this task to the Exposed ESB Gateway (implemented in the Web services gateway). Therefore we have to connect the outbound service for each Manufacturer in the ESB to the relevant service in the Exposed ESB Gateway.

There are two approaches to do this:

- ▶ Create a new outbound service on the ESB that points to the relevant Manufacturer inbound service on the Exposed ESB Gateway. This uses SOAP/HTTP messages to communicate between the ESB and Exposed ESB Gateway. This solution could also be implemented using SOAP/JMS.
- ▶ Create a service integration bus link between the ESB and the Exposed ESB Gateway, and have the ESB place a message directly on the relevant outbound service destination on the Exposed ESB Gateway. This uses messaging to communicate between the ESB and the Exposed ESB Gateway.

Both scenarios are discussed in this section. You can choose to implement just one of these options, or you might prefer to try each option in turn.

Connecting using new outbound services

In this scenario, new outbound services are created in the ESB that point to the relevant Manufacturer Gateway service on the Exposed ESB Gateway. This is shown in Figure 11-38.

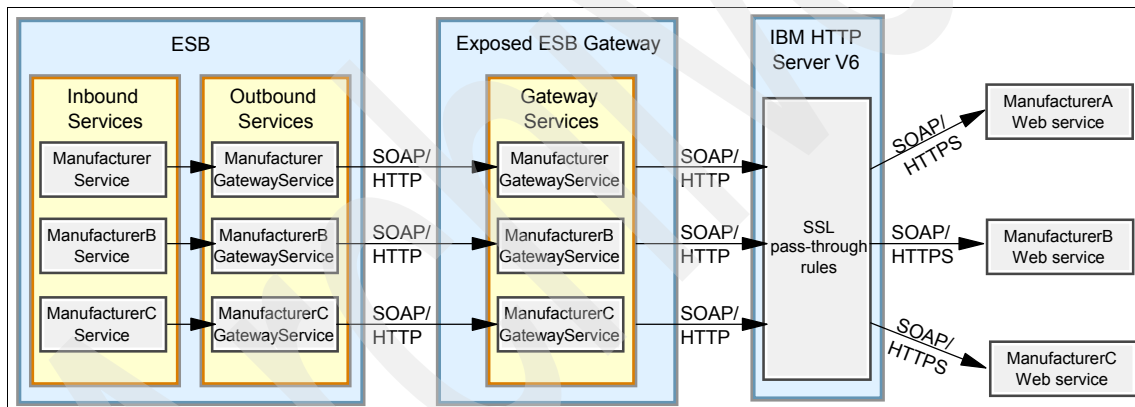


Figure 11-38 Connecting the ESB and Exposed ESB Gateway using SOAP/HTTP

This implementation has the advantage of being easy to implement, as the only change is to create new outbound services in the ESB component, and redirect the inbound services to point to them. However, the disadvantage is that this implementation uses SOAP/HTTP or SOAP/JMS to communicate between the ESB and Exposed ESB Gateway which in many cases may not be as efficient as using a service integration bus link.

If you want to implement this method for connecting the ESB and Exposed ESB Gateway you will need to complete the following steps:

1. Creating new outbound services on the ESB
2. Reconfiguring the inbound services
3. Testing the new outbound services

Creating new outbound services on the ESB

The existing outbound services we have defined for the Manufacturers point directly to the HTTP server, which forwards them on to the relevant Web service. We need to create new outbound services that instead points to the relevant Manufacturer Gateway service.

1. Follow the instructions in “Creating the outbound services” on page 270 to create a new outbound service in ESBBus, which points to the ManufacturerGatewayService. To do this, you use the WSDL generated for the ManufacturerGatewayService. Use the values in Table 11-9.

Table 11-9 Values for ManufacturerGatewayService

Field	Value
WSDL location	http://appsrv1a.itso.ral.ibm.com/wsdl/ExposedESBGatewayBus.ManufacturerGatewayServiceService.wsdl
Outbound service name	ManufacturerGatewayService
Service destination name	ManufacturerGatewayDestination
Port destination name	ManufacturerGatewayPortDestination

2. You should now see a new outbound service defined in the outbound service list for ESBBus called ManufacturerGatewayService (Figure 11-39 on page 296).

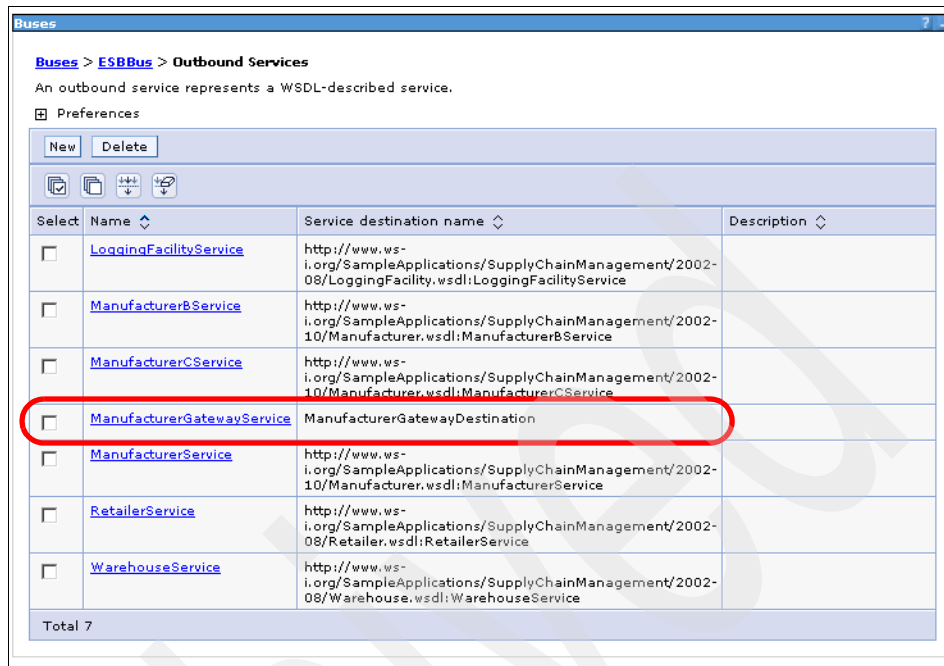


Figure 11-39 Definition of the ManufacturerGatewayService outbound service

3. Define similar outbound services to point to ManufacturerBGatewayService and ManufacturerCGatewayService.

Reconfiguring the inbound services

We now want to configure the ManufacturerService inbound service to point at this new outbound destination.

1. Expand **Service Integration** and click **Buses**.
2. Click **ESBBus**.
3. Under Services, click **Inbound Services**.
4. Click **ManufacturerService** and select the new outbound service, **ManufacturerGatewayDestination** from the Service destination name drop down list shown below in Figure 11-40 on page 297.

General Properties	
* Name	ManufacturerService
Service destination name	MaufacturerGatewayDestination
	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/LoggingFacility.wsdl:LoggingFacilityService
	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/LoggingFacility.wsdl:LoggingFacilityService
	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Retailer.wsdl:RetailerService
	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Retailer.wsdl:RetailerService
	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Warehouse.wsdl:WarehouseService
	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Warehouse.wsdl:WarehouseService
	ITSO Server01Node01.server1.SOAPHTTPChannel1Reply
	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl:ManufacturerService
	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl:ManufacturerService
	ManufacturerGatewayDestination
	ManufacturerGatewayPortDestination

Figure 11-40 Updated inbound service

- Click **OK** and save the changes. The list of inbound services now looks like Figure 11-41.

Select	Name	Service destination name	Published
<input type="checkbox"/>	LoggingFacilityService	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/LoggingFacility.wsdl:LoggingFacilityService	No
<input type="checkbox"/>	ManufacturerService	MaufacturerGatewayDestination	No
<input type="checkbox"/>	RetailerService	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Retailer.wsdl:RetailerService	No
<input type="checkbox"/>	WarehouseService	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Warehouse.wsdl:WarehouseService	No
Total 4			

Figure 11-41 List of inbound services

- Repeat this process for ManufacturerBService and ManufacturerCService so that they point to the new Gateway outbound services you have defined for them.

Testing the new outbound services

Run the SCMSampleUI sample application again to verify that the scenario is working:

- Open a Web browser and enter the URL:
http://localhost:9080/SCMSampleUI/
- Navigate through the application, and place an order for nine items of product 605001. This generates a stock replenishment request for ManufacturerA.

3. The application should complete successfully. To confirm that the Web service gateway was used, look in the SystemOut.log file of the ITSOGoodProfile application server profile for the line shown in bold in Example 11-1.

Example 11-1 Output from a successful invocation of ManufacturerA using the Web services gateway

```
SystemOut      0 Warehouse: Calling Manufacturer A
ServletWrapper A SRVE0242I: [sibwshttp2.ITS0Server01Node01.server1] [/wsgwsoaphttp2]
[SIBus_HTTPRouter]: Initialization successful.
SystemOut      0 Warehouse: Response from Manufacturer --> Manufacturer_A has received and
processed a request
```

Connecting using a service integration bus link

In this scenario, the ESB connects to the Exposed ESB Gateway using a service integration bus link. Both the ESB and the Exposed ESB Gateway run within their own service integration bus (the Exposed ESB Gateway uses a Web services gateway on top of its service integration bus) and these buses can be linked together. When the ESB needs to call a Manufacturer Gateway service on the Exposed ESB Gateway, it uses this service integration bus link to place a message on the relevant Manufacturer Gateway service destination. This is shown in Figure 11-42.

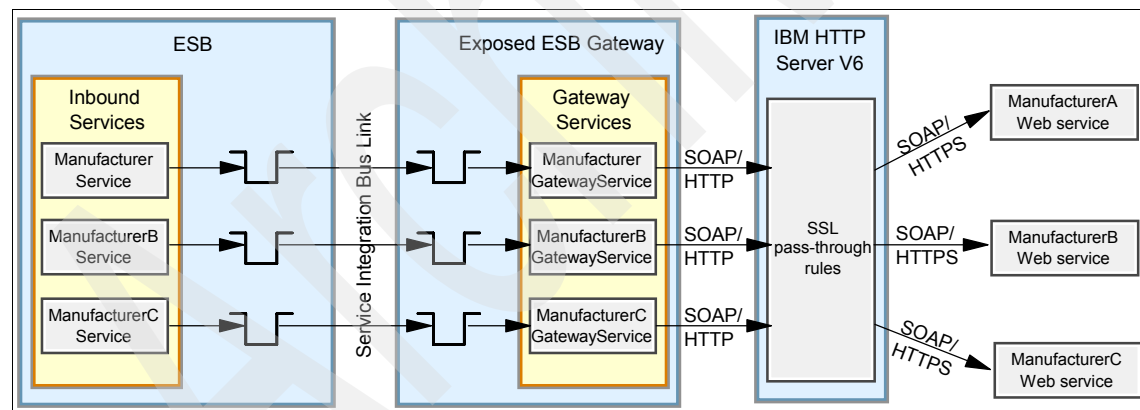


Figure 11-42 Connecting the ESB and Exposed ESB Gateway using a service integration bus link

This implementation has the advantage of the possibility of increased performance for messages sent between the ESB and Exposed ESB Gateway. Messages are sent directly between buses rather than converting them to SOAP over HTTP or JMS messages. The disadvantage of this approach is that it requires more configuration than the previous implementation.

If you wish to implement this method for connecting the ESB and Exposed ESB Gateway you need to complete the following steps:

Note: Remember that this is an alternative way to connect the ESB and Exposed ESB Gateway. If you have already connected the ESB and Exposed ESB Gateway using the instructions in “Connecting using new outbound services” on page 294, then it is not necessary to implement these steps to build a working implementation.

1. Creating a foreign bus
2. Creating the service integration bus link
3. Defining destinations which route to the other bus
4. Reconfiguring the inbound services
5. Testing the service integration bus link

Creating a foreign bus

A foreign bus represents another service integration bus in another cell (or within the same cell) or a WebSphere MQ network, with which a service integration bus can exchange messages. Messages are routed to a foreign bus either directly or through a link between the buses. The service integration bus link allows a link to be established between messaging engines on two service integration buses.

To create a foreign bus, follow these steps:

1. Expand **Service Integration** and click **Buses**.
2. Click on **ESBBus**.
3. Under Topology, click **Foreign buses**.
4. Click **New**.
5. A four-step wizard opens. You first enter the name of the foreign bus. It is important to ensure that the name that you enter is the name of the Exposed ESB Gateway bus. So, specify a value of `ExposedESBGatewayBus`, as shown in Figure 11-43 on page 300, and click **Next**.

Figure 11-43 New foreign bus wizard

6. In the next page, you select the routing type. There are three options:

- Direct, service integration bus link
- Direct, WebSphere MQ link
- Indirect

The default is Direct, service integration bus link, which is the option we need. So, just click **Next**.

7. In the next page, you specify the user ID to be used for inbound and outbound message authentication. These setting are not needed for this scenario, so click **Next**.

8. The last page is a summary page. Click **Finish**, and the foreign bus is created.

9. Save the changes.

10. Repeat this process to create a foreign bus called ESBBus on the ExposedESBGatewayBus. Make sure you define this foreign bus for the ExposedESBGatewayBus bus, and specify a foreign bus name of ESBBus.

Creating the service integration bus link

The next step is creating the service integration bus link. You must create a service integration bus link for each bus. This link connects a messaging engine on one bus to a messaging engine on another. To create this link:

1. Expand **Service Integration** and click **Buses**.
2. Click on **ESBBus**.
3. Under Topology, click **Messaging engines**.
4. Click the messaging engine name of **ITSOGoodNode.server1-ESBBus**.

5. Under Additional Properties, click **Service integration bus link**.
6. Click **New**.
7. In the next page, shown in Figure 11-44, you set up the service integration bus link.

The screenshot shows a web-based configuration interface for 'Buses'. The breadcrumb navigation at the top reads 'Buses > ESBBus > Messaging engines > ITSGoodNode'. Below this, a subtitle reads 'Links between this messaging engine and message bus'. The 'Configuration' tab is selected. The 'General Properties' section contains several fields: 'Name' (SIBLink), 'UUID' (240D9F9926ABDBD0), 'Description' (empty), 'Foreign bus name' (ExposedESBGatewayBus), 'Remote messaging engine name' (ITSGoodNode.server1-ExposedESBGatewayBus), 'Target inbound transport chain' (empty), 'Bootstrap endpoints' (localhost:7276), 'Authentication alias' ((none)), and 'Initial state' (Started). At the bottom are buttons for 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 11-44 Creating a new service integration bus link.

Complete only the mandatory information in this page:

- Name is an administrative entity. Enter **SIBLink**.
- Foreign bus name is the foreign bus to which this messaging engine is linked. This is a drop-down list and contains a single entry. Select **ExposedESBGatewayBus**.
- **Remote messaging engine name**, which is the name of the messaging engine on the foreign bus to which this messaging engine is connected.

Enter the name of the messaging engine on the foreign bus. In Figure 11-44, we entered `ITS0GoodNode.server1-ExposedESBGatewayBus`.

- Bootstrap endpoints specify where to find the messaging engine. It is a comma-separated list of entries. Each entry consists of up to three parts. If one part is missing, that part assumes a default value. The parts are separated with a colon:
 - Host num is the name of the host.
 - Port number is the port number on which the remote messaging engine is listening. This setting defaults to 7276. You can determine the port number of a messaging engine by clicking **Servers** → **Application Servers** → **server1** → **Ports** and by noting the value of the `SIB_ENDPOINT_ADDRESS` port name.
 - Protocol name, which is the symbolic name of the messaging protocol that is used. There are currently two: `BootstrapBasicMessaging` and `BootstrapSecureMessaging`. The default is `BootstrapBasicMessaging`.

In this case, we are using the default values, so simply enter `localhost:7276`.

8. Click **OK**. The service integration bus link has been configured.
9. Save the changes.

You must repeat this process for the `ExposedESBGatewayBus` bus to create a service integration bus link, also called `SIBLink`, to the ESB bus.

Defining destinations which route to the other bus

Configure three destinations (one for each Manufacturer) which will be used to forward messages to the Exposed ESB Gateway. Create the following queue type destinations:

- ▶ `ManufacturerServiceDestination`
- ▶ `ManufacturerBServiceDestination`
- ▶ `ManufacturerCServiceDestination`

Perform the following tasks:

1. Expand **Service Integration** and click **Buses**.
2. Click on **ESBBus**.
3. Under Destination Resources, click **Destinations** and click **New**.
4. Accept the default value of Queue and click **Next**.
5. Enter `ManufacturerServiceDestination` as the Identifier and click **Next**.
6. Click **Next** to accept the default bus member.
7. Click **Finish** on the summary panel and save the changes.
8. Repeat this process to define queue type destinations called `ManufacturerBServiceDestination` and `ManufacturerCServiceDestination`.

You can now set the forward routing paths for each destination to point to the relevant Gateway service destination on the Exposed ESB Gateway. Perform the following:

1. Under the list of destinations for ESBBus, click the **ManufacturerServiceDestination** queue.
2. On the page that appears as shown in Figure 11-45, set up a default forward routing path.

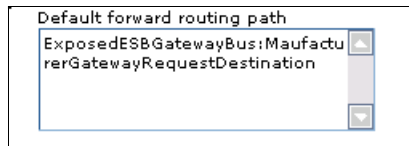


Figure 11-45 Configuring the forward routing path

A default forward routing path is applied to messages sent to a destination if the forward routing path of that message is available. Thus, messages that are sent to the **ManufacturerServiceDestination** can be routed elsewhere, which in our case is the Exposed ESB Gateway bus. The format of this box is a comma-separated list of qualified destination names. A qualified destination name consists of a bus name and a destination name separated by a colon. The bus name is optional if the destination is on the current bus.

Enter the following in the Default forward routing path:

ExposedESBGatewayBus:ManufacturerGatewayRequestDestination

3. Click **OK**. Requests to the **ManufacturerService** will now be forwarded to the Exposed ESB Gateway Bus.
4. Repeat these steps for the **ManufacturerBService** and **ManufacturerCService** destinations, entering the forward routing paths as shown in Table 11-10.

Table 11-10 Inbound service destination default forward routing paths

Destination	Default forward routing path
ManufacturerBServiceDestination	ExposedESBGatewayBus:ManufacturerBGatewayRequestDestination
ManufacturerCServiceDesination	ExposedESBGatewayBus:ManufacturerCGatewayRequestDestination

5. Save your changes.

Reconfiguring the inbound services

We need to update each **Manufacturer** inbound service to use the destinations we have defined in the previous step. Perform the following steps:

1. Expand **Service Integration** and click **Buses**.
2. Click on **ESBBus**.
3. Under Services, click **Inbound Services**.
4. Click **ManufacturerService**.
5. Under Service destination name drop down list, select the destination that is called **ManufacturerServiceDestination** and click **OK**.
6. Repeat these steps for the ManufacturerBService and ManufacturerCService inbound services, using the service destination name shown in Table 11-11.

Table 11-11 Inbound service mappings to destinations

Inbound service name	Service destination name
ManufacturerBService	ManufacturerBServiceDestination
ManufacturerCService	ManufacturerCServiceDestination

7. Save your changes.

Testing the service integration bus link

We can now test that the service integration bus link allows us to communicate with the Manufacturers using the Exposed ESB Gateway. Perform the following:

1. To start the service integration bus link, restart the application server. When the application server has restarted, check the SystemOut.log file of the ITSGoodProfile server profile for the message shown in Example 11-2.

Example 11-2 Successful service integration bus links

CWSIT0032I: The inter-bus connection SIBLink from messaging engine ITSGoodNode.server1-ESBBus in bus ESBBus to messaging engine ITSGoodNode.server1-ExposedESBGatewayBus in bus ExposedESBGatewayBus started.

2. Open a Web browser and enter the URL:
<http://itsogood.itso.ral.ibm.com:9080/SCMSampleUI/>
3. Navigate through the application, and place an order for nine items of product 605001 to generate a stock replenishment request for ManufacturerA.
4. The application should complete successfully.

11.4.6 Adding WS-Security to the Web service gateway

In our Extended Enterprise scenario, we must ensure that the information we are sending to the partner Manufacturer enterprise is secure. We can achieve this by using the support provided by the Web service gateway for WS-Security. This will

mean the connection between the Warehouse Web service client and the Manufacturer Web service is secured during WS-Security as shown logically in Figure 11-46. (Note that Figure 11-46 does not show the ESB and Exposed ESB Gateway as middle tiers between the Warehouse and Manufacturer).

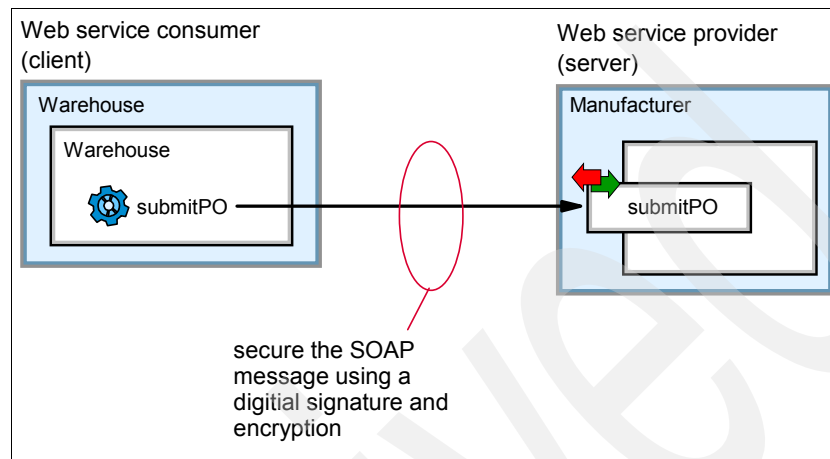


Figure 11-46 Using WS-Security for calls to the Manufacturer

Although the Web service gateway provides support for a number of different facets of WS-Security, we have identified that of primary concern in our scenario are integrity and confidentiality.

Note: This scenario differs from Chapter 10, “Exposed Direct Connection runtime pattern: generic profile” on page 157 because that scenario required us to change the Web service consumer in Rational Application Developer to add additional deployment descriptors for WS-Security. In this scenario, the Web service consumer remains unchanged, and we define the WS-Security settings as an administrative task in WebSphere Application Server.

We need to complete the following steps to secure the transmission of messages from the Warehouse to the Manufacturer:

- Create a WS-Security configuration
- Create a WS-Security binding
- Configure the WS-Security configuration and binding for integrity
- Configure the WS-Security configuration and binding for confidentiality
- Apply the WS-Security resources to the Web service gateway bus
- Enable global security

WS-Security resources can be applied to both inbound and outbound services. They can be classified as:

- ▶ Request consumer
Used on requests from a client to an inbound service.
- ▶ Request generator
Used when generating requests from an outbound service to a target Web service.
- ▶ Response consumer
Used on responses from a target Web service to an outbound service.
- ▶ Response generator
Used when generating responses from an inbound service to a client.

Figure 11-47 illustrates these security resource types.

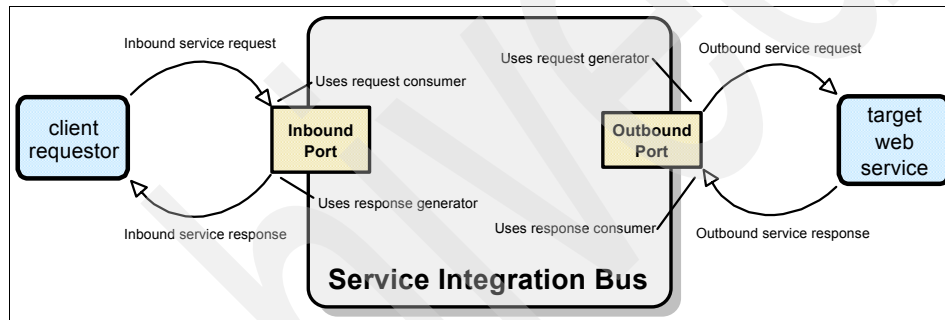


Figure 11-47 WS-Security resources used in the service integration bus

For this scenario we configure the outbound services on the Web service gateway responsible for communicating with the different Manufacturers as request generators. Our scenario does not require the message returned from a Manufacturer to the Warehouse to be secure.

Note: We only illustrate the steps required to create a secure request generator. Similar concepts can be applied when creating secure request consumers, response generators and response consumers.

We need to create both a WS-Security binding and a WS-Security configuration. The binding and configuration can then be applied to each of the outbound services on the Web service gateway instance. Before beginning to configure WS-Security we need to copy the keystore files to the file system. As we are only securing the outbound service, we only require the client keystore files:

- ▶ client.jks
- ▶ client_rsa.cer

You can find these files in the additional material supplied with this redbook in the `DirectConnectionSOA\keystore` directory. The files should be copied to `${WAS_INSTALL_ROOT}\etc\ws-security\samples\ITS0Good` where `${WAS_INSTALL_ROOT}` is the WebSphere Application Server installation directory.

Important: When implementing WS-Security in the bus, you have to ensure that you obtain WS-Security information, such as binding information and key stores, from the owning parties of the client when securing *inbound* services and the target Web service when securing *outbound* services.

Installing the secure Manufacturer application

In order to use WS-Security we need to use a Web service that is expecting a secured SOAP message. We described how to build a secured Manufacturer application in Chapter 10, “Exposed Direct Connection runtime pattern: generic profile” on page 157. Perform the following to install it:

1. Log in to the ManufacturerProfile administrative console at the following URL:
`http://manufacturer.a.itso.ra1.ibm.com:9061/ibm/console`
2. Select **Applications**, and click **Enterprise Applications**.
3. We need to uninstall the unsecured Manufacturer, so check the **Manufacturer** enterprise application and click **Stop**. After it is stopped, check Manufacturer again and click **Uninstall**. After it is uninstalled, save your changes.
4. Now install the secured Manufacturer application. Under Applications click **Install New Application**. Install **Manufacturer_Sec.ear** which you can find in the `DirectConnectionSOA\ears` directory of the additional material supplied with this book.
5. When the .ear file is installed, save the changes to the configuration.
6. Check the newly installed **Manufacturer** enterprise application and click **Start**.

Creating a WS-Security configuration

Figure 11-48 on page 308 gives a conceptual view of the elements required when creating and configuring a WS-Security configuration for integrity and confidentiality.

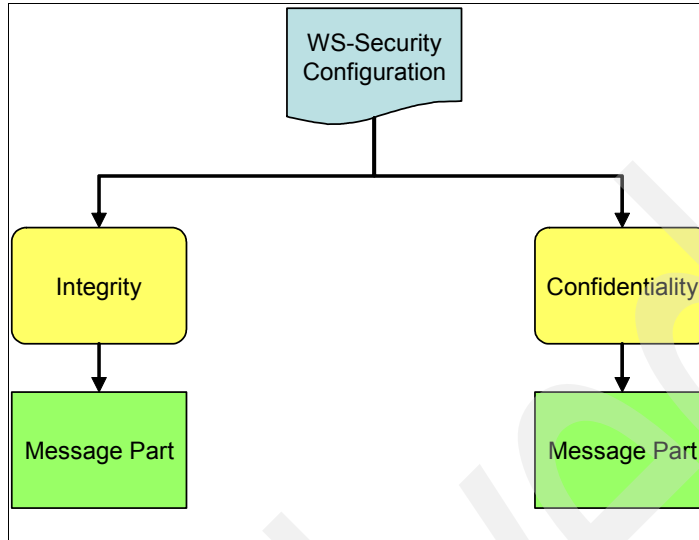


Figure 11-48 WS-Security configuration elements

To create the WS-Security configuration using the WebSphere Application Server Network Deployment administrative console for ITSOGoodProfile:

1. Expand **Service Integration** and expand **Web services**.
2. Click **WS-Security configurations** and click **New**.
3. Accept the default of security version 1.0 and click **Next**.
4. Select the **Outbound** security type option and click **Next**.
5. Enter `ManufacturerSecurityConfig` as the unique name for the configuration in the WS-Security configuration type panel as shown in Figure 11-49 on page 309 and click **Next**.

New WS-Security configuration

Use this wizard to create a new inbound or outbound WS-Security configuration.

Step 1: Select security version

Step 2: Specify service type

→ **Step 3: Specify WS-Security configuration type.**

Step 4: Summary

Specify WS-Security configuration type.

Service type
Outbound

* Name
ManufacturerSecurityConfig

Actor URI

Previous Next Cancel

Figure 11-49 WS-Security configuration type panel

- Click **Finish** on the Summary panel and save the changes. The list of WS-Security configurations now looks similar to Figure 11-50.

WS-Security configurations

WS-Security configurations for inbound and outbound services.

⊕ Preferences

New Delete

⊞ ⊞ ⊞ ⊞

Select	Name	Service type	Security version
<input type="checkbox"/>	ManufacturerSecurityConfig	Outbound	1.0

Total 1

Figure 11-50 WS-Security configurations

Creating a WS-Security binding

Figure 11-51 on page 310 gives a conceptual view of the elements required when creating and configuring a WS-Security binding for integrity and confidentiality.

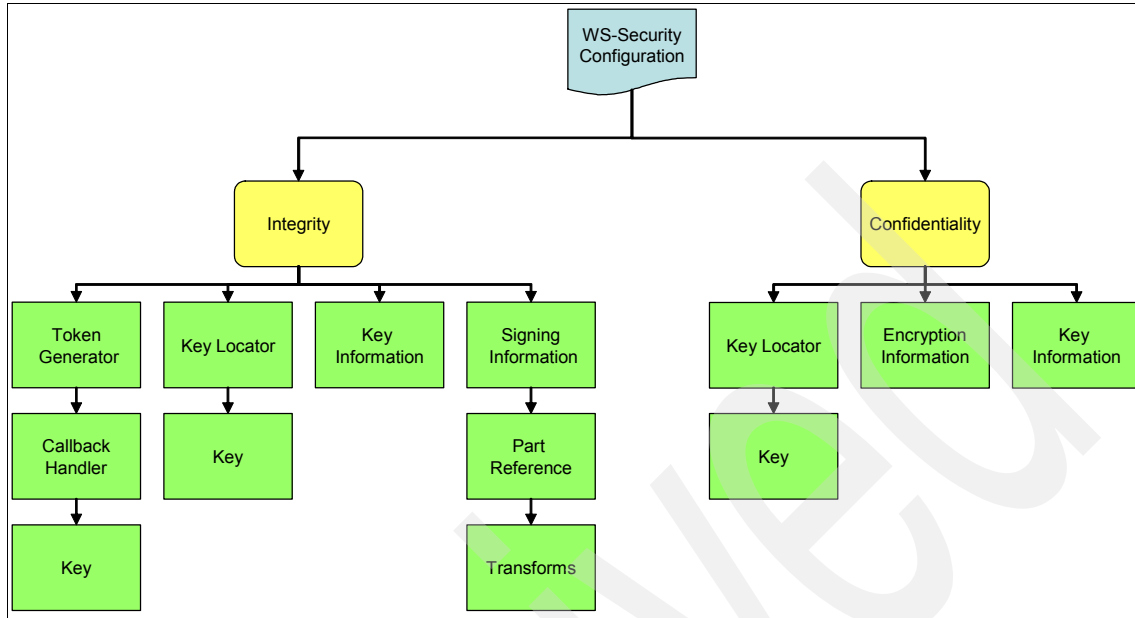


Figure 11-51 WS-Security binding elements

To create the WS-Security binding using the WebSphere Application Server Network Deployment administrative console for ITSOGoodProfile:

1. Expand **Service Integration** and expand **Web services**.
2. Click **WS-Security bindings** and click **New**.
3. Accept the default of security version 1.0 and click **Next**.
4. Select the **Request generator** security binding type option and click **Next**.
5. Enter `ManufacturerSecurityBinding` as the unique name for the binding in the WS-Security binding panel as shown in Figure 11-49 on page 309 and accept the default Web services security namespace,

`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd`.

New WS-Security binding

Use this wizard to configure a new WS-Security binding.

Step 1: Select security version
Step 2: Specify binding type
→ **Step 3: Specify WS-Security binding.**
Step 4: Summary

Specify WS-Security binding.

Binding Type
Request generator

* Name
ManufacturerSecurityBinding

☐ Use defaults

* Web services security namespace
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-s

Previous Next Cancel

Figure 11-52 Creating the WS-Security binding

6. Click **Next**.
7. Click **Finish** on the Summary panel and save the changes. The list of WS-Security bindings now looks similar to Figure 11-53.

WS-Security bindings

WS-Security bindings for consumption and generation of requests and responses.

⊕ Preferences

New Delete

⊞ ⊞ ⊞ ⊞

Select	Name	Binding Type	Security version
<input type="checkbox"/>	ManufacturerSecurityBinding	Request generator	1.0

Total 1

Figure 11-53 WS-Security bindings

Now we need to edit the configuration and binding to provide support for integrity and confidentiality.

Updating the WS-Security configuration for integrity

First we need to create an integrity constraint and associated message part for the configuration.

1. Expand **Service Integration** and expand **Web services**.

2. Click **WS-Security configurations**.
3. From the WS-Security configuration panel, click **ManufacturerSecurityConfig**. The Configuration panel for this security configuration is displayed as shown in Figure 11-54.

WS-Security configurations > **ManufacturerSecurityConfig**

WS-Security configuration for an outbound request. This defines WS-Security requirements for the request generated and response consumed from the target. The objects created may be applied to one or more outbound ports.

Configuration

General Properties

WS-Security version
1.0

Service type
Outbound WS-Security configuration

* Name
ManufacturerSecurityConfig

Actor URI

Apply OK Reset Cancel

Request generator

- ☐ Actor
- ☐ Integrity
- ☐ Confidentiality
- ☐ Security Token
- ☐ Add time stamp
- ☐ Properties

Response consumer

- ☐ Required integrity
- ☐ Required confidentiality
- ☐ Required security token
- ☐ Caller
- ☐ Add time stamp
- ☐ Properties

Figure 11-54 WS-Security configuration

4. Under Request generator, click **Integrity** and then click **New** to create a new integrity constraint.
5. Specify the name and order of the constraint. Enter `int_body` as the value for name and `1` for value of order and click **OK**.
6. Figure 11-55 on page 313 shows the list of integrity constraints. Click **Save** to apply the changes.

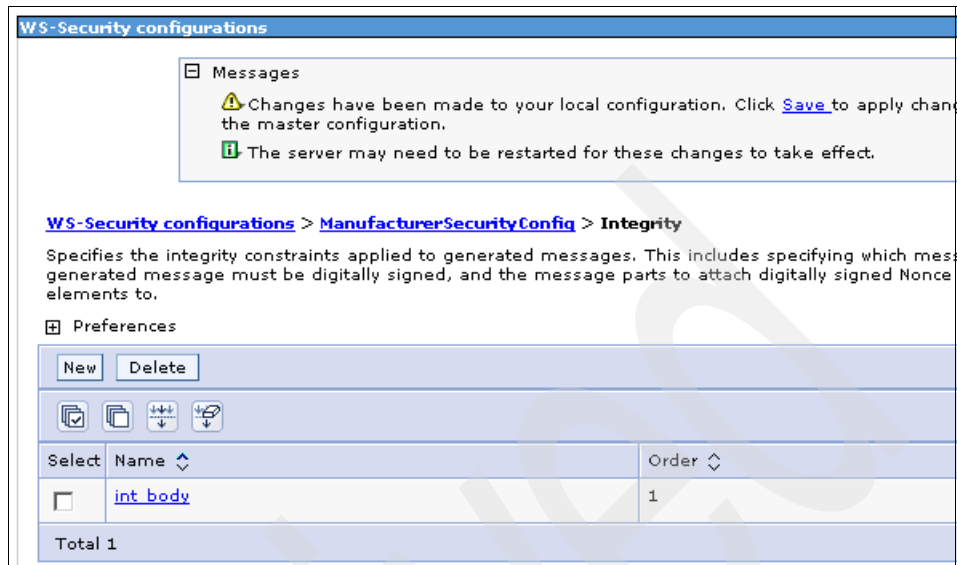


Figure 11-55 Integrity constraints list

7. Click the Integrity name **int_body**.
8. Under **Additional Properties**, click **Message parts** and click **New** to create a new message part.
9. Figure 11-56 on page 314 shows the Message part panel. Enter the following values:
 - Name is an arbitrary name for the message part. Enter a value of IntegrityMP.
 - Dialect specifies the message dialect to use. Accept the default value of <http://www.ibm.com/websphere/webservices/wssecurity/dialect-was>.
 - Keyword identifies the message part in a way defined by the chosen dialect. Enter body. This specifies the SOAP body.

WS-Security configurations

[WS-Security configurations](#) > [ManufacturerSecurityConfig](#) > [Integrity](#) > [int_body](#) > [Message parts](#) >

Identifies a specific message part according to the specified dialect and keyword.

Configuration

General Properties

* Name
IntegrityMP

* Dialect
http://www.ibm.com/websphere/webservices/wssecurity/dialect-was

* Keyword
body

Apply OK Reset Cancel

Figure 11-56 Message part settings for integrity

10. Click **OK** and save the changes.

Updating the WS-Security binding for integrity

We need to configure the binding to define the location of keys, encryption settings and signing parameters.

To configure the WS-Security binding for integrity we need to:

- ▶ Create a token generator, callback handler and associated key.
- ▶ Create a key locator
- ▶ Create key information
- ▶ Create signing information

Creating the token generator

Follow these steps to create the token generator:

1. Expand **Service Integration** and expand **Web services**.
2. Click **WS-Security bindings**.
3. From the WS-Security binding panel, click **ManufacturerSecurityBinding**.
The Configuration panel for this security binding is displayed in Figure 11-57 on page 315.

WS-Security bindings

[WS-Security bindings](#) > **ManufacturerSecurityBinding**

WS-Security binding for the generation of outbound request to a target.

Configuration

General Properties

WS-Security version

1.0

Binding Type

Request generator

* Name

ManufacturerSecurityBinding

☐ Use defaults

* Web services security namespace

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd

Apply

OK

Reset

Cancel

Additional Properties

■ [Signing information](#)

■ [Encryption information](#)

■ [Token generators](#)

■ [Key information](#)

■ [Key locators](#)

■ [Collection certificate store](#)

■ [Properties](#)

Figure 11-57 WS-Security binding

- To insert a security token into the message for signing, under Additional Properties, click **Token generators** and click **New** (Figure 11-58 on page 316).

WS-Security bindings

[WS-Security bindings](#) > [ManufacturerSecurityBinding](#) > [Token generators](#) > **gen_dsigtgen**

Specifies the parameters for the token generator. The information is used on the generator side only to generate the security token. Because you can plug-in a custom token generator, you must specify a Java class name.

Configuration

General Properties	Additional Properties
<p>* Token generator name <input type="text" value="gen_dsigtgen"/></p> <p>* Token generator class name <input type="text" value="com.ibm.wsspi.wssecurity.token.X509TokenGenerator"/></p> <p>Part reference name <input type="text"/></p>	<p>■ Callback handler</p> <p>■ Properties</p>
<p>Certificate path</p> <p><input checked="" type="radio"/> None</p> <p><input type="radio"/> Dedicated signing information</p> <p>Certificate store <input type="text" value="SampleCollectionCertStore"/></p>	
<p>Username token</p> <p><input type="checkbox"/> Add nonce</p> <p><input type="checkbox"/> Add timestamp</p>	
<p>Value type</p> <p>Local name <input type="text" value="http://docs.oasis-open.org/w"/></p> <p>URI <input type="text"/></p>	

Figure 11-58 Creating a token generator

5. Figure 11-58 shows the token generator general properties panel. Some of the values can keep their defaults. The following lists the values that you must enter:
 - Token generator name, which specifies the name of the token generator configuration. Enter a value of `gen_dsigtgen`.
 - Token generator class name, which specifies the token generator implementation class name. Enter `com.ibm.wsspi.wssecurity.token.X509TokenGenerator`

- Value type Local name, enter
`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509`

Important: The Help text for the Value type settings is incorrect. The URI value must be entered in the Local name field.

6. Click **Apply**.

Important: Be sure that the values you have entered have been saved correctly. The WS-Security binding and configuration panels appear to sometimes lose the values entered or to set values from lists back to the default value when saving. Exercise caution to be certain that all the values have been saved properly

7. Now under Additional Properties, click **Callback handler**.
8. Figure 11-59 on page 318 shows the callback handler general properties panel. Some of the values can keep their defaults. The following list is the values that you must enter:
- Callback handler class name. Enter
`com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler`
 - Key store Password, enter `client`
 - Key store Path, which specifies the location of the keystore file. Enter
`${WAS_INSTALL_ROOT}\etc\ws-security\samples\ITS0Good\client.jks`
 - Key store type, which specifies the keystore file format type. Select JKS.

WS-Security bindings

☐ Messages

⚠ Changes have been made to your local configuration. Click [Save](#) to apply changes to the master configuration.

🔄 The server may need to be restarted for these changes to take effect.

WS-Security bindings > ManufacturerSecurityBinding > Token generators > gen_dsigtgtgen > Callback handler

Specifies the parameters for the callback handler that are used for generating the token. Because you can plug-in a custom callback handler, you must specify the implementation class name. WebSphere Application Server provides options for identity assertion, basic authentication, and the key store that are passed to the callback handler implementation.

Configuration

General Properties

★ Callback handler class name
com.ibm.wsspi.wssecurity.aut

Identity assertion

☐ Use identity assertion

☐ Use RunAs identity

Basic authentication

User ID

Password

Key store

Password

Path
#{WAS_INSTALL_ROOT}\etc\

Type
JKS

The additional properties will not be available until the general properties for this item are saved.

Additional Properties

- Keys
- Properties

Figure 11-59 Creating a callback handler

9. Click **Apply**.
10. Now under Additional Properties, click **Keys** and click **New**. The Key Configuration panel shown in Figure 11-60 on page 319 is displayed.

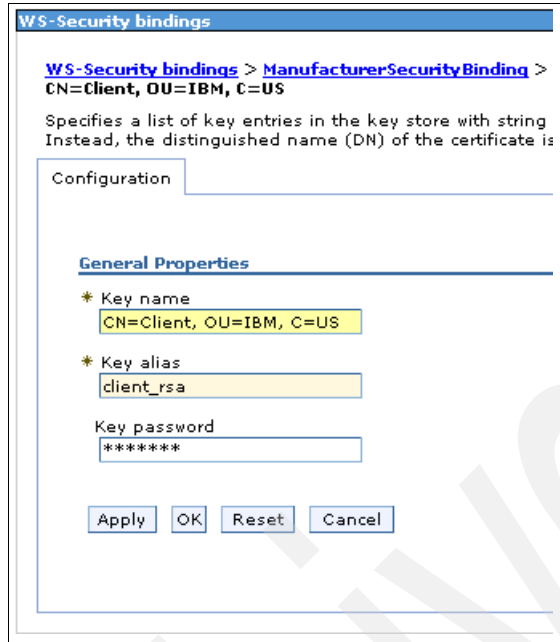


Figure 11-60 Creating a key

11. Enter the following values:

- Key name specifies the name of the key object. The key is used by the request generator signing information to determine which key is used to digitally sign the message. For encryption the key name is used to determine the key used for encryption. Enter `CN=Client, OU=IBM, C=US`.
- Key alias is used by the key locator to find the key within the keystore file. Enter `client_rsa`.
- Key password is the password used to access the key object within the keystore file. Enter `client_rsa`.

12. Click **OK**, and save all the changes to the configuration.

Creating the key locator

Now we will create the key locator specifying how to retrieve a key for signing.

1. Expand **Service Integration** and expand **Web services**.
2. Click **WS-Security bindings**.
3. From the **WS-Security binding** panel, click **ManufacturerSecurityBinding**.
4. Under Additional Properties, click **Key locator** and click **New**.
5. The Key locator panel shown in Figure 11-61 on page 320 is displayed.

WS-Security bindings

[WS-Security bindings](#) > [ManufacturerSecurityBinding](#) > [Key locators](#) > **gen_dsiglocator**

Specifies a list of key locator configurations that retrieve the key for signature and encryption. You can use a key locator class to retrieve keys from other types of repositories. The default implementation retrieves keys from the local key store.

Configuration

General Properties	Additional Properties
<p>* Key locator name gen_dsiglocator</p> <p>* Key locator class name com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator</p>	<p>■ Keys</p> <p>■ Properties</p>
<p>Key store</p> <p>Password *****</p> <p>Path \${WAS_INSTALL_ROOT}\etc\</p> <p>Type JKS</p>	

Apply OK Reset Cancel

Figure 11-61 Creating a key locator

6. Enter the following values:
 - For Key locator name, enter gen_dsiglocator.
 - For Key locator class name, enter com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator.
 - For Key store Password, enter client.
 - Key store Path specifies the location of the keystore file. Enter \${WAS_INSTALL_ROOT}\etc\ws-security\samples\ITSOGood\client.jks.
 - Key store type specifies the keystore file format type. Select **JKS**.
7. Click **Apply**.
8. Now under Additional Properties, click **Keys** and click **New**.
9. Enter the following values:
 - Key name CN=Client, OU=IBM, C=US
 - Key alias client_rsa
 - Key password client_rsa

10. Click **OK**, and save all the changes to the configuration.

Creating the key information

Now, we create the key information for specifying a signature by X.509 certificate.

1. Expand **Service Integration** and expand **Web services**.
2. Click **WS-Security bindings**.
3. From the **WS-Security binding** panel, click **ManufacturerSecurityBinding**.
4. Under Additional Properties, click **Key information** and click **New**. The key information panel as shown in Figure 11-62 on page 322 is displayed.

WS-Security bindings

[WS-Security bindings](#) > [ManufacturerSecurityBinding](#) > [Key information](#) > **gen_dsigkeyinfo**

Specifies the related configuration needed to generate the key for XML digital signature or XML encryption.

Configuration

General Properties

* Key information name

Key information type

Key locator mapping

Key locator reference

Key name reference

Token reference

Encoding method

Calculation method

Value type

Namespace URI

Local name

Figure 11-62 Creating the key information

5. Some of the values can keep their defaults. The following lists the values that you must enter:
 - Key information name specifies the name for the key information configuration. Enter a value of `gen_dsigkeyinfo`.
 - Key information type specifies how to reference security tokens. Select a value of **Security token reference** from the drop-down list.

- Key locator reference specifies the reference that is used to retrieve the key for digital signature and encryption. Select **gen_dsigklocator** from the drop-down list.
 - Click **Get keys**, and select **CN=Client, OU=IBM, C=US** from the Key name reference drop-down list. The key reference name specifies the name of the key that is used for generating the digital signature and encryption.
 - Token reference is used to specify the name of a token generator that is used for processing a security token. Select **gen_dsigtgen** from the drop down list.
6. Click **OK** and save the changes.

Creating the signing information

Now we create the signing information for specifying a signature by X.509 certificate.

1. Expand **Service Integration** and expand **Web services**.
2. Click **WS-Security bindings**.
3. From the **WS-Security binding** panel, click **ManufacturerSecurityBinding**.
4. Under Additional Properties, click **Signing information** and click **New**.
Figure 11-63 on page 324 shows the Signing information General Properties panel.

The screenshot shows the 'WS-Security bindings' configuration window. The breadcrumb path is 'WS-Security bindings > ManufacturerSecurityBinding > Signing information > sign_body'. Below the path, it states 'Specifies the configuration for the signing parameters.' The 'Configuration' tab is active. The window is divided into two main sections: 'General Properties' and 'Additional Properties'.

General Properties:

- * Signing information name:
- Signature method:
- Canonicalization method:
- Key information signature type:
- Signing key information:

Additional Properties:

- [Part references](#)
- [Canonicalization method properties](#)
- [Signature method properties](#)
- [Properties](#)

At the bottom, there are four buttons: 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 11-63 Creating the signing information

5. Enter the following values:
 - Signing information name specifies the name assigned to the signing configuration. Enter a value of `sign_body`.
 - Signature method specifies the algorithm URI of the signature method. Select a value of `http://www.w3.org/2000/09/xmlsig#rsa-sha1` from the drop-down list.
 - Canonicalization method is the algorithm URI of the canonicalization method. Select `http://www.w3.org/2001/10/xml-exc-c14n#` from the drop down list.
 - Key information signature type specifies how to sign a KeyInfo element if *dsigkey* or *enckey* is specified for the signing part in the deployment descriptor. If no value is selected, the `keyinfo` value will be used by WebSphere Application Server by default. Select **(none)** from the drop-down list.
 - Signing key information specifies a reference to the key information that WebSphere Application Server uses to generate a digital signature. Select **gen_dsigkeyinfo** from the drop-down list.
6. Click **Apply**.

7. Under Additional Properties, click **Part references** and click **New**.
Figure 11-64 shows the Part reference General Properties panel.

The screenshot shows a web-based configuration interface for WS-Security bindings. The breadcrumb trail at the top reads: [WS-Security bindings](#) > [ManufacturerSecurityBinding](#) > [Signing information](#) > [sign_body](#) > [Part references](#). Below this, a descriptive text states: "Specifies a reference to the message parts for XML digital signature and XML encryption defined in the deploy descriptors. In addition, the digest method can be specified for the message parts." The main configuration area is titled "Configuration" and is divided into two tabs: "General Properties" and "Additional Properties". The "General Properties" tab is active and contains three fields: "Part name" (with a yellow highlight and the value "sign_part"), "Part reference name" (with the value "int_body"), and "Digest method algorithm" (a dropdown menu showing "http://www.w3.org/2000/09/xmldsig#sha1"). At the bottom of the "General Properties" tab are four buttons: "Apply", "OK", "Reset", and "Cancel". The "Additional Properties" tab is currently inactive and shows two sub-sections: "Transforms" and "Digest method properties".

Figure 11-64 Creating the part reference

8. Enter the following values:
 - Part name specifies the name assigned to the part reference configuration. Enter a value of `sign_part`.
 - Part reference name specifies the name of the `<integrity>` element for the signed part of the message. Enter a value of `int_body`.
 - Digest method algorithm is the algorithm URI of the digest method used for the signed part that is specified by the part reference. Select **`http://www.w3.org/2000/09/xmldsig#sha1`** from the drop-down list.
9. Click **Apply**.
10. Under Additional Properties, click **Transforms** and click **New**. Figure 11-65 on page 326 shows the Transforms General Properties panel.

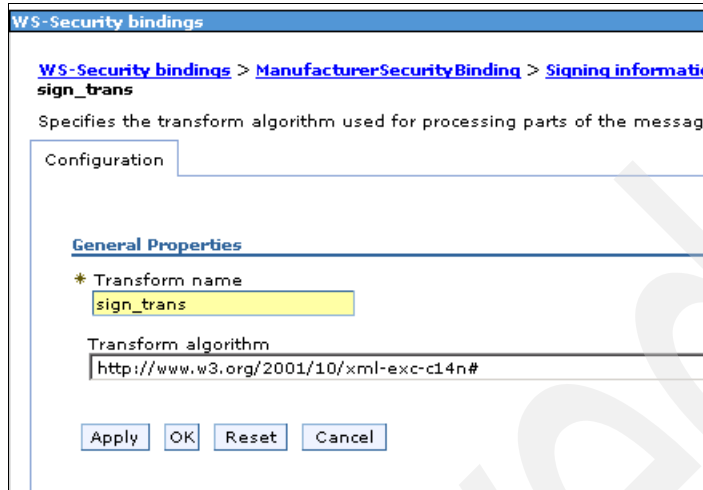


Figure 11-65 Creating the transform

11. Enter the following values:

- Transform name specifies the name assigned to the transform algorithm. Enter a value of `sign_trans`.
- Transform algorithm is the algorithm URI of the transform algorithm. Select **`http://www.w3.org/2001/10/xml-exc-c14n#`** from the drop-down list.

12. Click **OK** and save all changes.

We have now completed updating the WS-Security binding to support integrity.

Updating the WS-Security configuration for confidentiality

Now we need to create a confidentiality constraint and associated message part for the configuration, in the same way as we have previously created the integrity constraint described in , “Updating the WS-Security configuration for integrity” on page 311:

1. Expand **Service Integration** and expand **Web services**.
2. Click **WS-Security configurations**.
3. From the **WS-Security configuration** panel, click **ManufacturerSecurityConfig**.
4. Under Additional Properties, click **Confidentiality** and then click **New** to create a new confidentiality constraint.
5. Specify the name and order of the constraint. Enter `conf_body` as the value for name and 2 for value of order. Click **OK** and save the changes.

6. Now click the Confidentiality name **conf_body**.
7. Under Additional Properties, click **Message parts** and click **New** to create a new message part.
8. Figure 11-66 shows the Message part panel. Enter the following values:
 - Name is an arbitrary name for the message part. Enter a value of ConfidentialityMP.
 - Dialect specifies the message dialect to use. Accept the default value of **http://www.ibm.com/websphere/webservices/wssecurity/dialect-was**
 - Keyword identifies the message part in a way defined by the chosen dialect. Enter bodycontent. This specifies the SOAP body.

The screenshot shows a web browser window titled "WS-Security configurations". The breadcrumb navigation is: [WS-Security configurations](#) > [ManufacturerSecurityConfig](#) > [Confidentiality](#) > [conf_body](#) > [Message parts](#). The main heading is "ConfidentialityMP". Below it, a description reads: "Identifies a specific message part according to the specified dialect and keyword." There is a "Configuration" tab. Under the "General Properties" section, there are three fields: "Name" with the value "ConfidentialityMP", "Dialect" with a dropdown menu showing "http://www.ibm.com/websphere/webservices/wssecurity/dialect-was", and "Keyword" with the value "bodycontent". At the bottom, there are four buttons: "Apply", "OK", "Reset", and "Cancel".

Figure 11-66 Message part settings for confidentiality

9. Click **OK** and save the changes.

Updating the WS-Security binding for confidentiality

We now need to configure the binding to define the location of keys and encryption settings for confidentiality.

To configure the WS-Security binding for confidentiality we need to:

- ▶ Create a key locator
- ▶ Create key information
- ▶ Create encryption information

Creating the key locator

In this section, we create the Key locator, specifying how to retrieve a key for signing.

Note: More information, including diagrams describing the configuration of Key locators and Key information, is described in “Updating the WS-Security binding for integrity” on page 314.

1. Expand **Service integration** and expand **Web services**.
2. Click **WS-Security bindings**.
3. From the WS-Security bindings panel, click **ManufacturerSecurityBinding**.
4. Under Additional Properties, click **Key locator** and click **New**.
5. Figure 11-67 on page 329 shows the Key locator settings for confidentiality. Enter the following values:
 - For Key locator name, enter `gen_encklocator`.
 - For Key locator class name, enter:
`com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator`
 - For Key store Password, enter `client`.
 - For Key store Path, enter:
`${WAS_INSTALL_ROOT}\etc\ws-security\samples\ITSOGood\client.jks`
 - For Key store type, select JKS.

WS-Security bindings

[WS-Security bindings](#) > [ManufacturerSecurityBinding](#) > [Key locators](#)

Specifies a list of key locator configurations that retrieve the key for signature or decryption. The default is to retrieve keys from the local key store.

Configuration

General Properties

* Key locator name

* Key locator class name

Key store

Password

Path

Type

Figure 11-67 Key locator for confidentiality

6. Click **Apply**.
7. Now under Additional Properties, click **Keys** and click **New**.
8. Enter the following values, as shown in Figure 11-68 on page 330:
 - The Key name is CN=Server, OU=IBM, C=US.
 - The Key alias is server_rsa.
 - The Key password should be left blank.

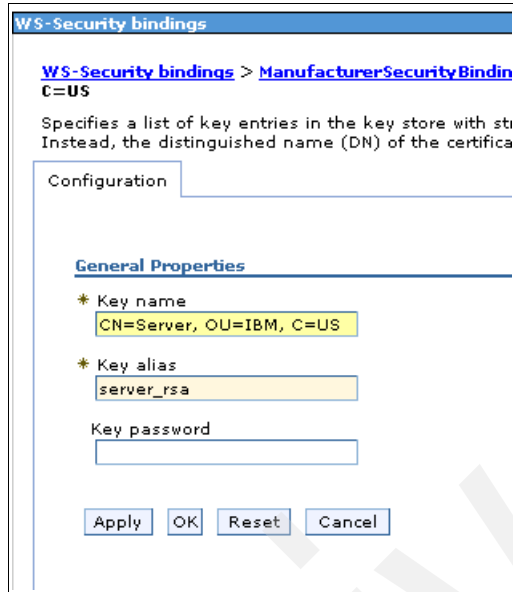


Figure 11-68 Key settings

9. Click **OK**, and save all the changes to the configuration.

Creating the key information

Now we will create the key information.

1. Expand **Service Integration** and expand **Web services**.
2. Click on **WS-Security bindings**.
3. From the WS-Security bindings panel, click **ManufacturerSecurityBinding**.
4. Under Additional Properties, click **Key information** and click **New**.
5. Figure 11-69 on page 331 shows the key information settings for confidentiality. Some of the values on the key information general properties can keep their defaults. The following lists the values that you must enter:
 - For Key information name, enter a value of **gen_enckeyinfo**.
 - For Key information type, select a value of **Key identifier** from the drop-down list.
 - For Key locator reference, select **gen_encklocator** from the drop-down list.
 - Click **Get keys**, and select **CN=Server, OU=IBM, C=US** from the Key reference name drop-down list.

WS-Security bindings

[WS-Security bindings](#) > [ManufacturerSecurityBinding](#) > [Key information](#) > [gen_encke](#)

Specifies the related configuration needed to generate the key for XML digital signature

Configuration

General Properties

* Key information name

Key information type

Key locator mapping

Key locator reference

Key name reference

Token reference

Encoding method

Calculation method

Value type

Namespace URI

Local name

Figure 11-69 Key information for confidentiality

6. Click **OK** and save the changes.

Creating the encryption information

Create the encryption information for body content encryption.

1. Expand **Service Integration** and expand **Web services**.
2. Click **WS-Security bindings**.
3. From the WS-Security bindings panel, click **ManufacturerSecurityBinding**.

4. Under Additional Properties, click **Encryption information** and click **New**.
5. Figure 11-70 on page 332 shows the Encryption information settings for confidentiality. Enter the following values:
 - Encryption information name specifies the name for the encryption information. Enter a value of `enc_body`.
 - Data encryption algorithm specifies the algorithm URI of the data encryption method. Select a value of **`http://www.w3.org/2001/04/xmlenc#tripledes-cbc`** from the drop-down list.
 - Key encryption algorithm specifies the algorithm URI of the key encryption method. Select **`http://www.w3.org/2001/04/xmlenc#rsa-1_5`** from the drop-down list.
 - Encryption key information specifies the key information reference that is used for encryption. Select **`gen_enckeyinfo`** from the drop-down list.
 - Part reference name specifies the name of the `<confidentiality>` element for the generator binding. Enter the value `conf_body`.

The screenshot shows a dialog box titled "WS-Security bindings". Inside, there is a breadcrumb trail: "WS-Security bindings > ManufacturerSecurityBinding > Encrypti". Below this, a description reads: "Specifies the configuration for the XML encryption and decryption specified, the application server only accepts elements encrypted". There is a "Configuration" tab selected. Under the "General Properties" section, the following fields are visible:

- "Encryption information name" with a text box containing "enc_body".
- "Data encryption algorithm" with a dropdown menu showing "http://www.w3.org/2001/04/xmlenc#tripledes-cbc".
- "Key encryption algorithm" with a dropdown menu showing "http://www.w3.org/2001/04/xmlenc#rsa-1_5".
- "Encryption key information" with a dropdown menu showing "gen_enckeyinfo".
- "Part reference name" with a text box containing "conf_body".

At the bottom of the dialog are four buttons: "Apply", "OK", "Reset", and "Cancel".

Figure 11-70 Encryption information for confidentiality

6. Click **OK** and save the changes.

The details of the WS-Security resources are saved by WebSphere Application Server Network Deployment into the `sibws-wssecurity.xml` file. This file is at the cell level for both the deployment manager and the application server.

Applying WS-Security resources to the Web service gateway

After you have created and configured your WS-Security resources, you can then apply them to services within the Web service gateway. To do this using the WebSphere administrative console:

1. Expand **Service Integration** and click **Buses**.
2. Click on **ExposedESBGatewayBus**.
3. Under Services, click **Outbound Services** and then click **ManufacturerOutboundService**.
4. Under Additional Properties click **Outbound Ports** and then click **Manufacturer**.
5. On the Outbound Port panel shown in Figure 11-71 on page 334, select the WS-Security resources you want to apply. From the Security request binding drop down list, select **ManufacturerSecurityBinding (1.0)**.
6. From the Security configuration drop down list, select **ManufacturerSecurityConfig (1.0)**.

Buses

[Buses](#) > [ExposedESBGatewayBus](#) > [Outbound Services](#) > [ManufacturerOutboundPortDestination](#)

An outbound port represents a single port for a WSDL-defined Web service.

Configuration

General Properties

* **Name**

Description

* **Port destination name**

* **Port destination point**

Binding namespace

Endpoint address

JAX-RPC handler list

Security request binding

Security response binding

Security configuration

Authenticating proxy host name

Authenticating proxy port number

Authenticating proxy Authorization Alias

Figure 11-71 Applying the security resources

7. Click **OK** and save all changes.

The Web service gateway is now configured to use the WS-Security resources we have defined.

Note: In our scenario we are only securing messages sent to the Java implementation of the Manufacturer. The messages sent to the .NET and CICS Manufacturer implementations will remain unsecured.

Testing with WS-Security

This section describes how to test the scenario. For more information about the sample application, see Chapter 8, “Business scenario used in this book” on page 137.

For full instructions on testing the sample application see 10.4.8, “Testing the scenario” on page 229. In summary, perform the following steps:

1. Restart the ITSOGoodProfile application server so the WS-Security settings take effect.
2. Enter the following URL in a Web browser to start the ITSOGood sample application:

`http://itsogood.itso.ral.ibm.com:9080/SCMSampleUI/`

This assumes the unsecured HTTP transport port number is 9080. If this is not the case, use the URL above with the appropriate HTTP transport port number.

3. Order six items of product 605001. This will trigger the Warehouse to contact ManufacturerA to replenish its stock for this product.
4. To confirm the successful invocation of the Manufacturer service, check the SystemOut.log of the ITSOGoodProfile application server. The Warehouse writes a message saying that a Manufacturer was invoked. The message should look like this:

Warehouse: Response from Manufacturer --> Manufacturer_A has received and processed a request.

You should also see the following message in the SystemOut.log file of the ManufacturerProfile application server:

Manufacturer A: Processing Purchase Order

To confirm the SOAP message is using the WS-Security settings you have defined, use the TCP/IP Monitor. Use of the TCP/IP Monitor is fully described in 10.4.9, “Viewing SOAP messages using the TCP/IP Monitor” on page 233. In this scenario, you need to modify the outbound service for the Manufacturer in the Exposed ESB Gateway.

In summary, perform the following tasks:

1. In a Command Prompt, set the PATH environment variable so the java.exe command can be run (substituting <WAS_HOME> for the path where WebSphere Application Server is installed):

```
set PATH=<WAS_HOME>\java\bin;%PATH%
```

2. Navigate to the <WAS_HOME>\lib directory:

```
cd <WAS_HOME>\lib
```

3. Launch the TCP/IP Monitor:

```
java -classpath webservices.jar com.ibm.ws.webservices.engine.utils.tcpmon
```

4. The TCP/IP Monitor should launch in a new window. Enter a Listen Port of **81**. Select **Act as a Listener** and specify Target Hostname as **itsogood.itso.ral.ibm.com** and Target Port as **80**.

5. Click **Add**. This adds a new tab at the top of the TCP/IP Monitor called **Port 81**. Click this tab to see any TCP/IP messages that are received on port 81 by the TCP/IP Monitor.

6. Open the Manufacturer_Impl.wsdl file in a text editor. Find this file installed in the HTTP server directory at: <HTTP_SERVER_HOME>\htdocs\en_US\wsdl

7. Change the following line in Manufacturer_Impl.wsdl so it uses port 81 as shown below, then save the change:

```
<wsdlsoap:address  
  location="http://itsogood.itso.ral.ibm.com:81/Manufacturer/services/Manu  
  facturer"/>
```

8. Reload this WSDL file for the Manufacturer outbound service on the Exposed ESB Gateway. In the ITSOGoodProfile administrative console click **Service integration** → **Buses** → **ExposedESBGatewayBus** → **Outbound Services** → **ManufacturerOutboundService** and click **Reload WSDL**. This reloads the port change made in Manufacturer_Impl.wsdl from the HTTP server.

9. Save your changes, then run the sample application again. You should see the SOAP message sent from the Warehouse to the Manufacturer in the TCP/IP Monitor, and that the request message has WS-Security applied (Figure 11-72 on page 337).

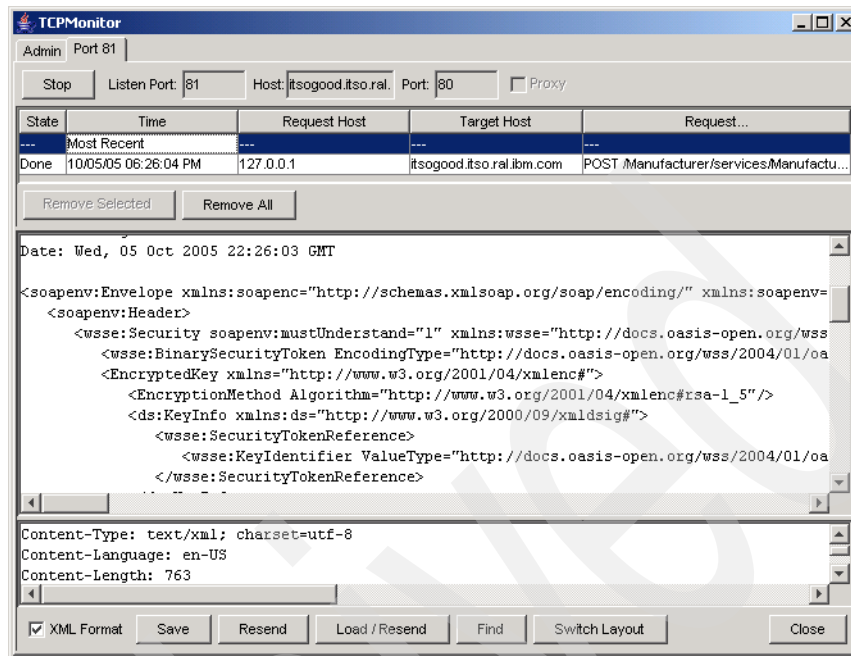


Figure 11-72 Viewing a SOAP message in the TCP/IP Monitor

Archived

Exposed Broker runtime pattern: generic profile

This chapter describes how to architect, develop, and implement the generic profile of the Exposed Broker runtime pattern.

In this chapter we describe how to add broker functionality, where a Broker node receives a single request and sends responses to multiple targets in other enterprises. To do this, we:

- ▶ Develop mediations using the mediation API of WebSphere Application Server.
- ▶ Describe how to deploy the mediations to a Web services gateway,
- ▶ Describe how to test the application.

12.1 Business scenario

The business scenario implemented in this chapter builds on the business scenario discussed in Chapter 10, “Exposed Direct Connection runtime pattern: generic profile” on page 157.

Having successfully integrated their Warehouse systems with the external Manufacturing partners, ITSO Good would now like to scale up their operations and also plans to improve the overall quality of service to the customer.

ITSO Good’s IT Operations team have come up with the following additional requirement for the supply chain management solution to align effectively with the business decision to scale up their operations:

- The distribution logic and rules within the Warehouse need to be separated from the application logic to provide the application services the required flexibility to scale independently.

Figure 12-1 displays the high-level business context of our business scenario.

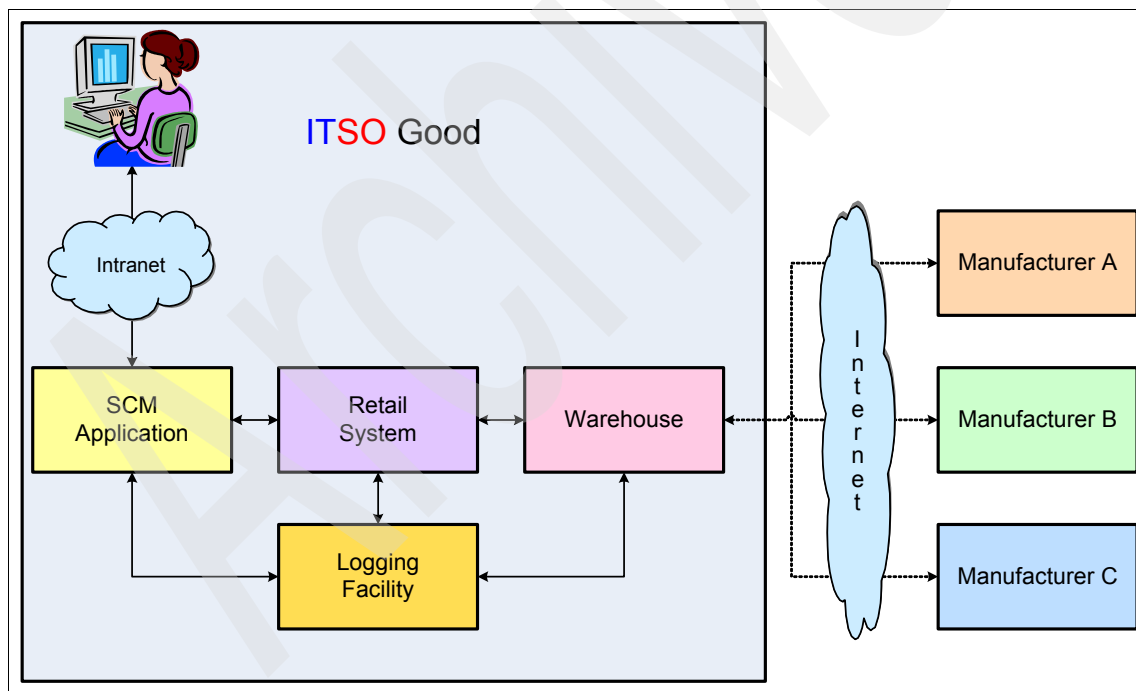


Figure 12-1 High-level business context of the scenario

12.2 Design guidelines

In this section, we analyze the business requirements and apply the Patterns for e-business to determine the appropriate Runtime pattern for the solution. We then discuss the various design options available to us in implementing the solution and also look at the product mappings.

12.2.1 Analyze business requirements

The business scenario requires the distribution rules to be separated from the application logic within the Warehouse system for the order replenishment process. For example, if a customer order results in the product stock falling below the threshold level, then the Warehouse raises a single replenishment request with the products that need to be replenished to a Broker component. The *Broker*, in turn, brokers the request into multiple replenishment orders and sends them to each Manufacturer partner application required to fulfill the replenishment orders.

The Broker additionally has to handle the decomposition and recomposition of the SOAP messages exchanged across the organization boundary in a seamless manner so as to make it look like a single interaction for the Warehouse and in turn the customer placing the order.

To summarize, the Broker component has to address the following requirements in the given scenario:

- ▶ Service/message routing
- ▶ Decomposition/recomposition of SOAP messages
- ▶ Message transformation, if required
- ▶ Maintain business rules related to the above aspects

12.2.2 Selecting a pattern

We use the Patterns for e-business to determine the appropriate Runtime pattern to apply to this scenario. This section describes a step-by-step approach used to navigate the Patterns for e-business asset catalog:

1. Business pattern

We select the Extended Enterprise business pattern because the given scenario requires interactions between the business processes in the Warehouse and Manufacturer systems that reside in separate enterprises.

2. Application pattern

Because the source application (Warehouse) initiates an interaction that is to be distributed to multiple target partner applications concurrently, we choose the Exposed Broker application pattern, as shown in Figure 12-2.

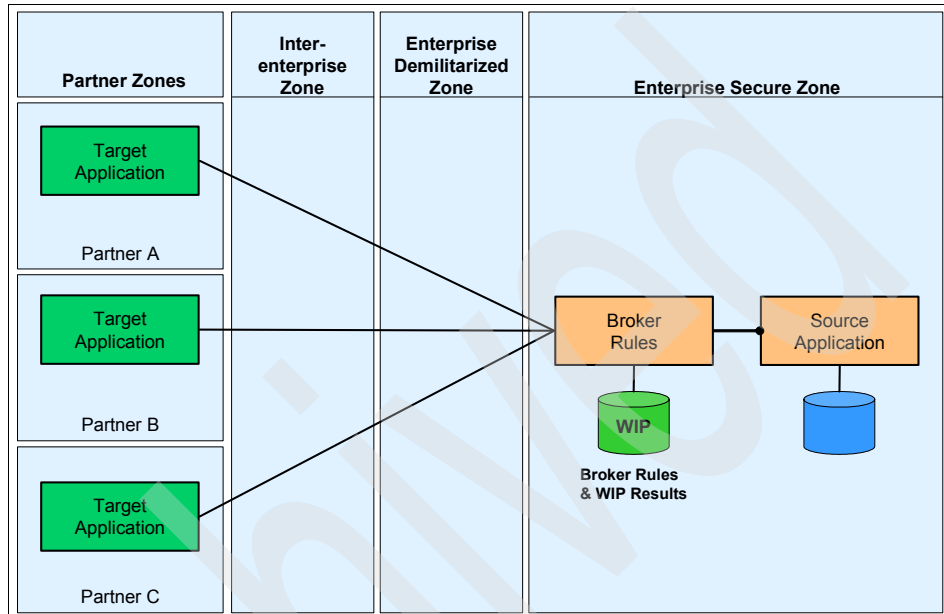


Figure 12-2 Exposed Broker application pattern

3. Runtime pattern

The selection of the Application pattern provides us with the possible runtime patterns for the proposed solution. Because the business requirement does not mandate an SOA infrastructure, we select the *generic profile* of the Exposed Broker runtime pattern.

Figure 12-3 on page 343 shows the level 0 decomposition of the generic profile for the Exposed Broker runtime pattern, mapped to the Exposed Broker application pattern.

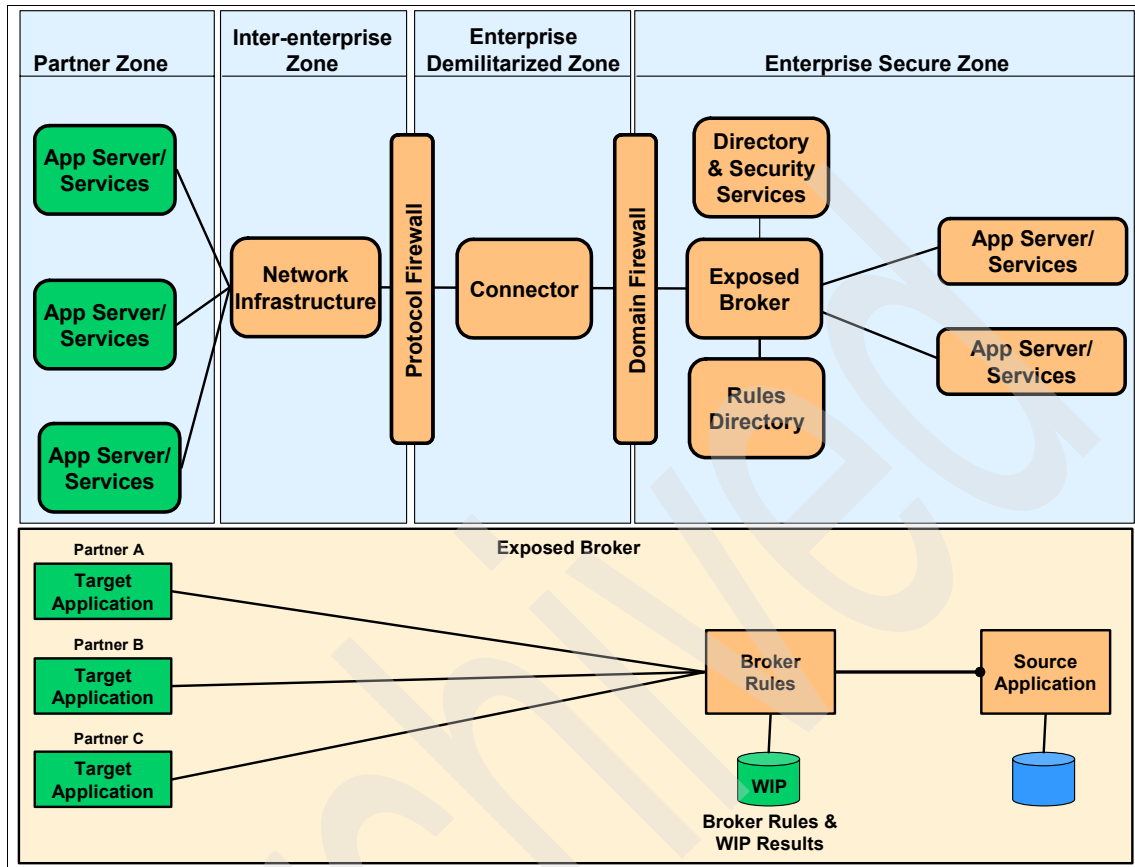


Figure 12-3 Exposed Broker::Runtime pattern = generic profile

12.2.3 Analyze design options

This section discusses the architectural decisions that we made and their implementation options for the given business scenario using the Exposed Broker runtime pattern.

Most of the design decisions we made in the previous chapters also apply to the Broker interactions:

- ▶ “Architectural decision: integration options” on page 163, describing the use of Web services to integrate with the external partner services.
- ▶ The design guidelines in the “Securing Web services” on page 165.
- ▶ The architectural decision related to securing the Web services in “Architectural decision: Securing the Web service interaction” on page 170

Additional design decisions that are specific to this scenario are discussed in the following sections.

Architectural decision: designing the broker component

The architectural decisions for designing the broker component are discussed in Table 12-1.

Table 12-1 Architectural decision: designing the broker component

Decision title	The most suitable operational topology
Issue or Problem statement	<p>To externalize the routing and distribution rules for the interaction with the external partner applications by using a broker. The broker would have to satisfy the following requirements:</p> <ul style="list-style-type: none"> ▶ Service / Message routing ▶ Decomposition / recomposition of the SOAP messages. ▶ Message transformation, if required. ▶ Maintain business rules related to the above aspects. <p>Since the Broker component would be the one interacting with the partner services across the boundary, it should also handle,</p> <ul style="list-style-type: none"> ▶ Securing the Web service interaction between the Warehouse and the external Manufacturing service.
Assumptions	None.
Motivation	
Alternatives	<p>1) WebSphere Business Integration Message Broker.</p> <p>2) Web services gateway component in WebSphere Application Server Network Deployment.</p>
Decision	<p>The Web services gateway component in WebSphere Application Server Network Deployment will be used to implement the Broker component. It would handle</p> <ul style="list-style-type: none"> ▶ Security by using the WS-Security features. ▶ Decomposition/recomposition of the SOAP messages using the mediation support available in the service integration bus where the Web services gateway runs.

Justification	<p>WebSphere Business Integration Message Broker would be an ideal choice if there is a need to handle high-volume of messages, reliable messaging and other sophisticated capabilities of integration.</p> <p>Given the lack of clarity on these non-functional aspects and in order to keep the solution simple, we would use the Web services gateway. The solution could be migrated to WebSphere Business Integration Message Broker at a later time.</p>
---------------	--

12.2.4 Products

In this section we look at the products available to implement the various components in the Exposed Broker runtime pattern.

Product implementation options

Product choices for this scenario are based on:

- ▶ Design decisions made in 12.2.3, “Analyze design options” on page 343.
- ▶ Extended Enterprise capabilities of the products
- ▶ Products currently available

We can use the following currently available products to implement the Broker component in the given scenario:

- ▶ Web services gateway component in WebSphere Application Server Network Deployment.
- ▶ WebSphere Business Integration Message Broker

For this scenario, the Web services gateway component in WebSphere Application Server Network Deployment meets all of the requirements and, as a result, is the product of choice.

The complete product mapping for this scenario is shown in Figure 12-4 on page 346.

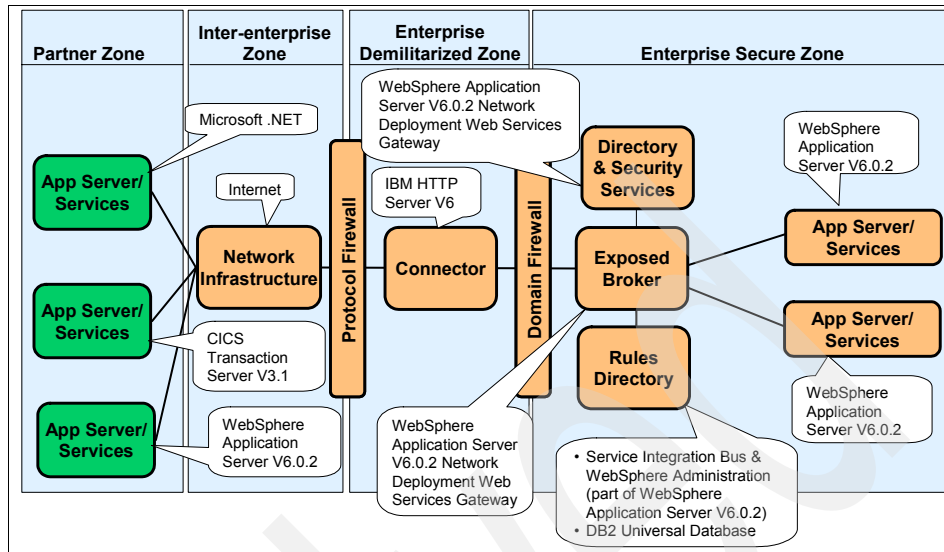


Figure 12-4 Exposed Broker :: Product mappings

In this Product mapping, WebSphere Application Server V6.0.2 Network Deployment was used for all services within the ITSO Good enterprise. The Manufacturing services of the three external partner systems are implemented using CICS Transaction Server V3.1, WebSphere Application Server V6.0.2 and Microsoft .NET respectively.

The Exposed Broker is implemented using the Web services gateway component in WebSphere Application Server Network Deployment.

12.3 Development guidelines

The scenario this book uses is based on the WS-I sample application, as described in Chapter 8, "Business scenario used in this book" on page 137. We use a modified version of this sample application called the *ITSO Good sample application*.

This section discusses how the scenario uses the Exposed Broker runtime pattern. In order to complete this section, you need to use Rational Application Developer V6.0.1 or later.

To follow the steps in this section we provide you the required projects in a zip file called *BrokerGenericProjects.zip*. You need to unzip the file and import the projects into your Rational Application Developer workspace. You can find the files you need under \BrokerGeneric\projects directory in the additional

material provided along with this book. For information about how to obtain the additional materials, see Appendix A, “Additional material” on page 481.

Perform the following activities to setup and test successfully the sample application with Exposed Broker runtime pattern:

- Understand the scenario implementation.
- Import provided projects into your Rational Application Developer workspace.
- Create the Broker EJB project.
- Code the Broker by implementing mediation handlers.
- Assign and export the mediation handlers.
- Export the EAR files.

12.3.1 Scenario implementation: Exposed Broker runtime pattern

Figure 12-5 shows the interactions that are made by each component in the ITSO Good sample application.

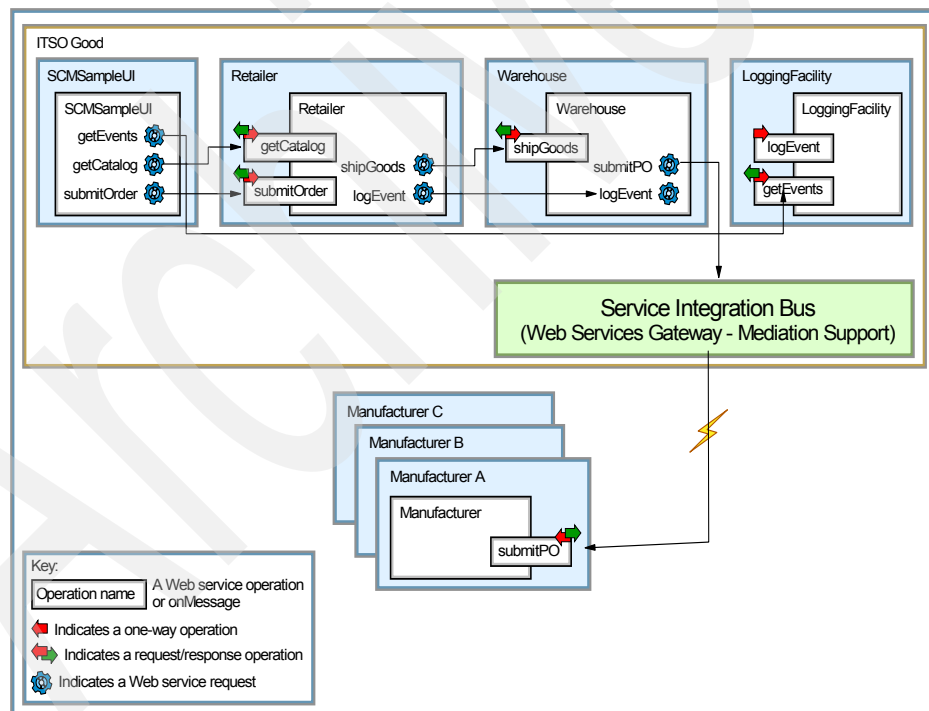


Figure 12-5 Exposed Broker scenario

In this scenario, we move the business routing logic supporting which Manufacturers the Warehouse should call. We move this logic from the

Warehouse application itself into a mediation running in the service integration bus. By moving this logic out of the Warehouse application, we have a more flexible architecture that allows changes to the business routing logic without having to change the applications themselves.

How the scenario works

This is the runtime process:

1. When replenishing stock, the Warehouse sends a single request to a Manufacturer listing all of the products that need to be restocked. This call is intercepted by the service integration bus mediation.
2. The mediation determines which Manufacturers need to be invoked. The mediation generates and sends new SOAP request messages for each Manufacturer needed.
3. The Manufacturers each return a SOAP response.
4. This response is received by the mediation. The mediation waits for all responses, and then packages these responses into a single SOAP response message and send this message back to the Warehouse.
5. The Warehouse is not aware that multiple SOAP messages were required to complete the stock replenishment.

12.3.2 Mediations

In this scenario there are multiple Manufacturers to be called depending on the purchase order. A broker could be implemented by using JAX-RPC handlers, but JAX-RPC handlers do not natively support the generation of multiple SOAP messages from a single SOAP message.

Mediations in the service integration bus of WebSphere Application Server V6 supports features such as aggregation and disaggregation. We decided to implement the broker using these mediations.

A WebSphere Application Server V6 *mediation handler* processes messages that are between production by one application and consumption by another application. Mediations provide functionality that allows customization of the messaging behavior of the service integration bus, which can include processing such as:

- ▶ Transforming a message from one format to another
- ▶ Routing messages to one or more targets that were not specified by the sending application
- ▶ Augmenting messages by adding data from a data source
- ▶ Distributing messages to multiple target destinations

A mediation is associated with a destination. A *destination* is a virtual location in a service integration bus that can be used to exchange messages by the applications connected to the service integration bus.

When a mediation is applied to a destination, it becomes a mediated destination that has two parts: premediated and postmediated. Applications send messages to the *premediated* part and receive them from the *postmediated* part. The mediation receives messages from the premediated part, transforms them in some way, then places them on the *postmediated* part. In this way, the mediation controls the progress of the messages to their intended target destination.

The behavior of a mediation is defined by a mediation handler list which contains mediation handlers and can be identified by:

- ▶ A unique name
- ▶ A description of the message processing provided by the mediation
- ▶ A set of properties that control behavior during message processing

A *mediation handler list* is a collection of mediation handlers that are invoked in sequence. A *mediation handler* is a Java program that performs the function of a mediation and can be deployed in a mediation handler list. The unique name for a mediation handler list is determined by the programmer who deployed the mediation.

A mediation is configured for a particular destination in a service integration bus. The physical location of the destination is referred to as a *mediation point*. The message processing by the mediation is started when the mediation point receives a messages from the messaging runtime.

Mediation APIs

Several application programming interfaces (APIs) are provided to allow you to work with the message context and code mediations.

▶ MediationHandler

This interface defines the method which is invoked by the mediation runtime. The method returns boolean `true` if the message passed into this method should continue along the handler list. Otherwise, it returns `false`. The API has just one method handle, `handle()`, which is used by the runtime to invoke a mediation.

In addition to the context information that is passed from one handler to another, it can return a reference to an `SIMessage` and an `SIMediationSession`. The `SIMessage` is the service integration bus representation of the message that is processed by the `MediationHandler`. The `SIMediationSession` is a handle to the runtime resources.

► **MessageContext**

This interface abstracts the message context that is processed by a handler in the handle method. The MessageContext interface provides methods to manage a property set. The API has two methods:

- `getSIServiceMessage()` is a method to get the service integration bus representation of the message being mediated
- `getSession()` is a method to get an `SIMediationSession` object, which is a handle to the core runtime.

► **SIMessage**

This interface is the public interface to a service integration bus message for use by mediations. The SIMessage interface has many methods which allow you to work with the message properties, header contents, routing path, metadata, and others.

In particular, the method `getDataGraph()` returns the SDO data graph which contains the SIMessage content in a tree representation. This method allows you to work directly with the individual fields in the message payload.

Forward and reverse routing paths define a sequential list of intermediary bus destinations that messages must pass through to reach a target bus destination. A routing path is used to apply the mediations configured on several destinations to messages sent along the path. The methods `getForwardRoutingPath()`, `setForwardRoutingPath()`, `getReverseRoutingPath()`, and `setReverseRoutingPath()` allow you to get and set the contents of the forward routing path and reverse routing path for this SIMessage.

► **SIMediationSession**

This interface defines the methods for querying and interacting with the service integration bus, and also includes methods that provide information about where the mediation is invoked from, and the criteria that are applied before the message is mediated.

The API has these methods:

- `getBusName()` returns the name of the bus upon which the mediation is associated.
- `getDestinationName()` returns the name of the destination with which the mediation is associated.
- `getDiscriminator()` returns the discriminator that is defined in the mediation definition.
- `getMediationName()` returns the name of the mediation that is being executed.

- `getMessageSelector()` returns the message selector that is defined in the mediation definition.
- `getMessagingEngineName()` returns the name of the messaging engine from which the mediation was invoked.
- `getSIDestinationConfiguration()` returns the `SIDestinationConfiguration` object associated with the destination that is specified by `destinationName` or `destinationAddress`.
- `receive()` receives an `SIMessage` from the service integration bus.
- `send()` sends a copy of an `SIMessage` to the service integration bus in addition to the message that is returned by the message interface.

SDO DataGraphs

A message published in one format (for instance, a Web services SOAP message) can be routed to a service provider that requires another format (Java beans, for example), using the Java API for XML-based RPC (JAX-RPC). Equally, the routing could be in the other direction. If the message is operated on by a mediation as it passes through the bus, in either direction, the mediation must be able to operate on the message regardless of the underlying format. This is achieved by using a common message model for the data mediators to use. The model is called SDO DataGraph and it gives an abstract view of the message, allowing you to concentrate on the information being conveyed (such as the parameters of the request, and the data of the response) without having to worry about the packaging of that information.

SDO is based on the concept of data graphs. In the data graphs architecture, a mediation retrieves a data graph from a message, transforms the data graph, and applies the data graph changes back to the data source. A *data graph* is a collection of tree-structured or graph-structured data objects.

In general, a graph that is generated from a message is a tree structure. The service presents a standard SDO data graph representation of the message payload, whatever the format of the incoming message's payload. A data object holds a set of named properties, each of which contains either a primitive-type value or a reference to another Data Object. The Data Object API provides a dynamic data API for manipulating these properties.

Routing paths

A *routing path* defines a sequential list of intermediary bus destinations that messages must pass through to reach a target bus destination. A routing path is used to apply the mediations configured on several destinations to messages sent along the path.

A forward routing path identifies a list of bus destinations that a message should be sent to from the producer to the last destination from which receivers retrieve messages. The reverse routing path is constructed automatically for request/reply messages, and identifies the list of destinations that any reply message should be sent to from the receiver back to the producer. Use of reverse routing path enables a reply message to take a different route back to the producer, and therefore have more mediations applied.

When a message arrives at a destination in the path, mediations can manipulate the entries in the forward routing path, to change the sequence of destinations through which messages pass. If a mediation manipulates the forward routing path, and the reverse routing path has been set (for a request message that expects a reply), then the mediation is responsible for making any corresponding changes to the reverse routing path.

A destination without mediations can be included in a routing path to provide a future option to apply a mediation assigned to that destination.

12.3.3 Developing a mediation handler class

Mediations are implemented as mediation handlers. A mediation handler executes some specific message processing at runtime, for example, transforming a message format or routing a message to a particular destination. A *mediation handler* is a Java program framework to which you add the code that performs the mediation function.

This section describes how to use IBM Rational Application Developer V6 to create the mediation handler. This product provides support for developing mediation handler code and adding mediation handlers to the J2EE deployment descriptors. The Application Server Toolkit that is provided with WebSphere Application Server also provides support for developing mediation handlers.

In Rational Application Developer, a mediation handler class can be defined either in a Java project or an EJB project. This section describes how to create mediation handlers in an EJB project. However, the steps are very similar if you want to create a Java project, because you simply define a target server for either a Java project or an EJB project and the server runtime plug-in sets the classpath correctly.

To create a mediation handler, start Rational Application Developer and perform the following steps:

1. From Rational Application Developer, click **File** → **Import**. In the Import dialog, select **Project Interchange** and click **Next**.

2. On the From zip file field, click **Browse** to select the project interchange to import.
3. Navigate to the file in the open file dialog and open **BrokerGenericProjects.zip** file. This file is found in the additional material supplied with this redbook in the \BrokerGeneric\projects directory.
4. The Import Projects window is populated with all available projects as shown on Figure 12-6. Click **Select All** and then *deselect* projects **ManufacturerBroker** and **ManufacturerBrokerEJB**.
5. Click **Finish**. The projects are imported.

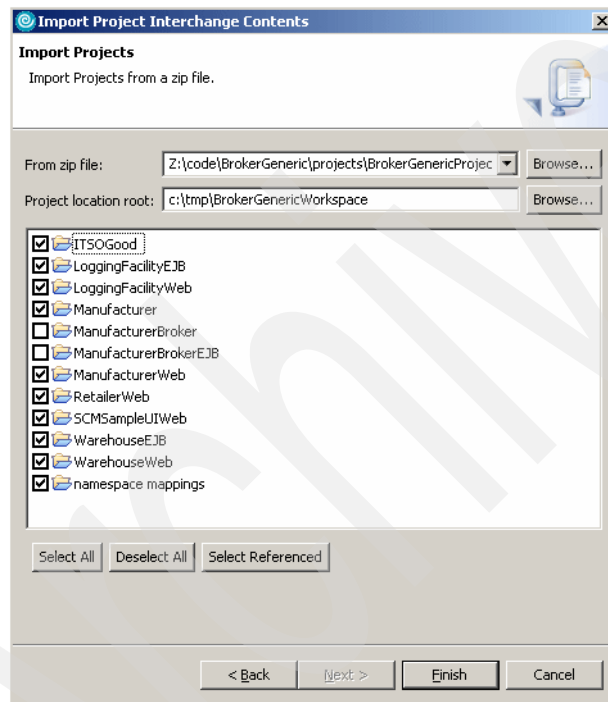


Figure 12-6 Import projects

6. Right-click in the Project Explorer and select **New** → **Project**. In the New Project dialog box expand **EJB** and select **EJB Project**. Click **Next**.
7. In the New EJB Project dialog, as shown on Figure 12-7 on page 354, enter the name of the EJB ManufacturerBrokerEJB. Click **Show Advanced** and enter ManufacturerBroker in the EAR project field. We do not need an EJB client, so *deselect* **Create an EJB client JAR project to hold the client interfaces and classes**. Click **Finish**.

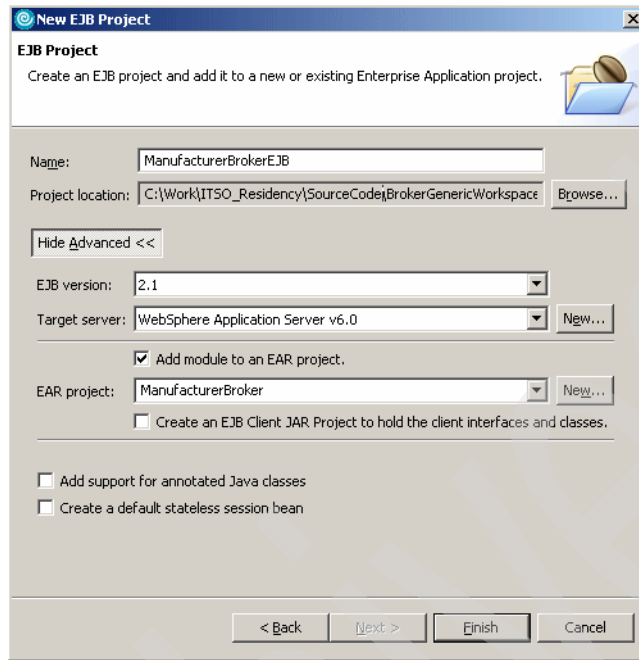


Figure 12-7 Create a new EJB project

8. Create a mediation handler class. The mediation handler class implements the `com.ibm.websphere.sib.mediation.handler.MediationHandler` interface. In the Project Explorer view, expand the **ManufacturerBrokerEJB** project and right click **ejbModule**. Select **New** → **Class**. In the New Java Class dialog (Figure 12-8 on page 355), enter `com.ibm.itsogood.broker` in the Package field. Enter `RequestMediator` in the Name field. Click **Add** and add interface **MediationHandler**. Click **Finish**.

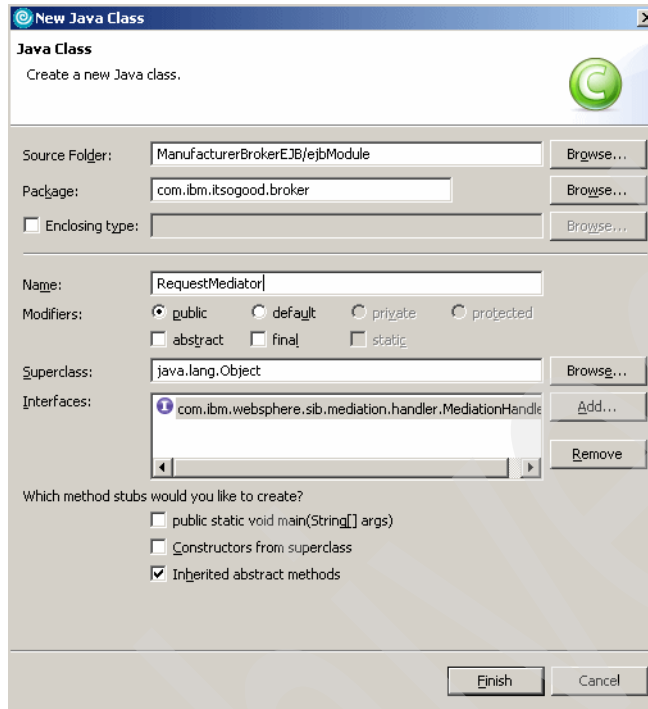


Figure 12-8 Create a Java class

9. Create a ResponseMediator class. Repeat step 8 on page 354 using the same package and interface name, but name the class ResponseMediator.
10. In addition to RequestMediator and ResponseMediator classes, we need to import another Java class, called DataGraphHolder, to hold a data graph and routing path. The source code for this class is located in the \BrokerGeneric\projects directory from the additional material that accompanies this book. To import the source code:
 - a. In the Package Explorer view, right-click **com.ibm.itsogood.broker** package and select **Import**.
 - b. Select **File System** and click **Next**.
 - c. Browse to the directory where you downloaded the additional material, select **DataGraphHolder.java**, and click **Finish**.

Adding code to the mediations

Now that we have created the mediation handler classes, we need to add code to them. The mediations perform the following functions:

► Request mediation

The request mediation separates the list of purchase orders sent by the Warehouse into multiple purchase orders, one for each Manufacturer. The request mediation creates a new Web Service SOAP request for each purchase order and sends it to the relevant Manufacturer.

► Response mediation

The response mediation receives the SOAP response messages from each Manufacturer and sends one single SOAP response message back to the Warehouse.

Request mediation code

The RequestMediator class must implement the `handle()` method. The `handle()` method provides access to the service integration bus in WebSphere Application Server V6 to send and receive messages. It is invoked by the arrival of a message on the destination that it has been configured to mediate.

When the mediation is invoked, the method gets a handle to the `MessageContext` interface. The mediation abstracts the message context that is processed by a handler in the `handle()` method. The `MessageContext` interface provides methods to manage a property set. Message context properties enable handlers in a handler chain to share processing related state.

The request mediation:

1. Receives the SOAP request from Warehouse.
2. Gets the name of the log queue, which is defined as a property of the mediation within WebSphere Application Server V6. The log queue is used to hold control data that indicates how many messages were sent out to Manufacturers. This information is used by the request mediation to indicate how many responses it should expect.
3. Parses the request message and builds new messages to send to relevant Manufacturer.
4. Sends these messages to the Manufacturers.
5. Sends a control message to the log queue.

In Rational Application Developer, perform the following steps to modify the `handle()` method of the RequestMediator class:

1. Open the RequestMediator.java file in the Java Editor. By default, the `handle()` method returns `false`. You must change that to `true`. Example 14-1 shows the RequestMediator class definition.

Example 12-1 RequestMediator class

```
public class RequestMediator implements MediationHandler
{
    public boolean handle(MessageContext arg0) throws MessageContextException
    {
        // Enter handle() code here
        return true;
    }
}
```

2. Enter the code shown in Example 12-2 in the `handle()` method.

Example 12-2 ResponseMediator `handle()` method

```
System.out.println("ManufacturerBroker REQUEST Mediator: Starting ...");
try {
    // cast message context to SIFMessageContext
    SIFMessageContext contextMessage = (SIFMessageContext) arg0;
    // get the session
    SIFMediationSession session = contextMessage.getSession();
    // get message
    SIFMessage message = contextMessage.getSIFMessage();
    // get message context property logQueueName
    String logQueueName = (String) arg0.getProperty("logQueueName");
    // store the original message in the DataGraphHolder
    DataGraphHolder dgHolder = new DataGraphHolder(message);
    // call separateMessage() to separate the messages for each manufacturer
    DataGraphHolder[] dgHolderArray = separateMessages(contextMessage,
dgHolder);
    // get the number of messages that have been separated
    int messageCount = dgHolderArray.length;

    for (int i = 0; i < messageCount; i++) {
        // Clone the message
        SIFMessage clonedMessage = (SIFMessage) message.clone();
        // Copy the API message id from the incoming message.
        clonedMessage.setApiMessageId(message.getApiMessageId());
        // Override the message body.
        clonedMessage.setDataGraph(dgHolderArray[i].getDataGraph(),
dgHolderArray[i].getFormat());
        // set the forward routing path
        if (dgHolderArray[i].getFrp() != null) {
            clonedMessage.setForwardRoutingPath(dgHolderArray[i].getFrp());
        }
        // Send the message in the current global unit of work.
        session.send(clonedMessage, false);
    }

    // create a new datagraph to be sent to logQueue
```

```

        DataGraph logMessageBody = message.getNewDataGraph("JMS:text");
        // record the number of messages sent
        logMessageBody.getRootObject().setString("data/value", "" +
messageCount);
        message.setDataGraph(logMessageBody, "JMS:text");
        // set forward routing path
        List frp = new ArrayList(1);

        frp.add(SIDestinationAddressFactory.getInstance().createSIDestinationAddress(log
queueName, true));
        // send control message to logQueue
        message.setForwardRoutingPath(frp);
    } catch (Exception e) {
        e.printStackTrace();
        throw new MessageContextException(e.getMessage());
    }

    System.out.println("ManufacturerBroker REQUEST Mediator: Finished.");
    return true;

```

3. Create a new method in the RequestMediator class named `saparateMessages()`. The method signature must be similar to this:

```

private DataGraphHolder[] separateMessages(SIMessageContext
messageContext, DataGraphHolder inDataGraphHolder) throws Exception

```

4. Enter the code shown Example 12-3 in the `separateMessages()` method.

Example 12-3 RequestMediator separateMessages() method

```

// create data graph holders for each manufacturer
DataGraph dgManufacturerA = inDataGraphHolder.getNewDataGraph();
DataGraph dgManufacturerB = inDataGraphHolder.getNewDataGraph();
DataGraph dgManufacturerC = inDataGraphHolder.getNewDataGraph();
// Delete items from copied datagraphs
DataObject manufacturerArootNode = dgManufacturerA.getRootObject();
DataObject manufacturerAPONode =
manufacturerArootNode.getDataObject("Info/body/PurchaseOrder");
DataObject manufacturerAItemsNode =
manufacturerArootNode.getDataObject("Info/body/PurchaseOrder/items");
manufacturerAItemsNode.delete();
DataObject manufacturerBrootNode = dgManufacturerB.getRootObject();
DataObject manufacturerBPONode =
manufacturerBrootNode.getDataObject("Info/body/PurchaseOrder");
DataObject manufacturerBItemsNode =
manufacturerBrootNode.getDataObject("Info/body/PurchaseOrder/items");
manufacturerBItemsNode.delete();
DataObject manufacturerCrootNode = dgManufacturerC.getRootObject();
DataObject manufacturerCPONode =
manufacturerCrootNode.getDataObject("Info/body/PurchaseOrder");

```

```

        DataObject manufacturerCItemsNode =
manufacturerCrootNode.getDataObject("Info/body/PurchaseOrder/items");
        manufacturerCItemsNode.delete();
        // Create empty items entries
        manufacturerAPONode.createDataObject("items");
        manufacturerBPONode.createDataObject("items");
        manufacturerCPONode.createDataObject("items");
        manufacturerAItemsNode =
manufacturerArootNode.getDataObject("Info/body/PurchaseOrder/items");
        manufacturerBItemsNode =
manufacturerBrootNode.getDataObject("Info/body/PurchaseOrder/items");
        manufacturerCItemsNode =
manufacturerCrootNode.getDataObject("Info/body/PurchaseOrder/items");

        // Navigate to items of the input datagraph
        DataObject rootNode = inputDataGraphHolder.getDataGraph().getRootObject();
        DataObject itemListNode =
rootNode.getDataObject("Info/body/PurchaseOrder/items");
        // Get List of items from items
        List items = itemListNode.getList("Item");
        // Loop thru items list from input datagraph, update itemList of
appropriate
        // manufacturer datagraph
        Iterator it = items.iterator();
        DataObject newItem = null;
        while (it.hasNext()) {
            DataObject itemNode = (DataObject) it.next();
            String compare = itemNode.getString("ID");
            System.out.println("ManufacturerBroker REQUEST Mediator: Processing
product " + compare + ".");
            if (compare.equals("605001") || compare.equals("605004") ||
compare.equals("605007")) {
                newItem = manufacturerAItemsNode.createDataObject("Item");
                newItem.setString("ID", itemNode.getString("ID"));
                newItem.setString("qty", itemNode.getString("qty"));
                newItem.setString("price", itemNode.getString("price"));
            } else if (compare.equals("605002") || compare.equals("605005") ||
compare.equals("605008")) {
                newItem = manufacturerBItemsNode.createDataObject("Item");
                newItem.setString("ID", itemNode.getString("ID"));
                newItem.setString("qty", itemNode.getString("qty"));
                newItem.setString("price", itemNode.getString("price"));
            } else if (compare.equals("605003") || compare.equals("605006") ||
compare.equals("605009")) {
                newItem = manufacturerCItemsNode.createDataObject("Item");
                newItem.setString("ID", itemNode.getString("ID"));
                newItem.setString("qty", itemNode.getString("qty"));
                newItem.setString("price", itemNode.getString("price"));
            } else

```

```

        System.out.println("Warehouse RequestMediator: ### Invalid item");
    }
    // If manufacturer has items in items add it to datagraph holder
    List graphs = new ArrayList(3);
    List newitems = manufacturerAItemsNode.getList("Item");
    SIDestinationAddress destination = null;

    if (newitems.size() > 0) {
        List frp = new ArrayList(1);
        destination = SIDestinationAddressFactory.getInstance()

.createSIDestinationAddress("ManufacturerOutboundServiceDestination", false);
        frp.add(0, destination);
        graphs.add(new DataGraphHolder(dgManufacturerA,
inDataGraphHolder.getFormat(), frp));
    }
    newitems = manufacturerBItemsNode.getList("Item");
    if (newitems.size() > 0) {
        List frp = new ArrayList(1);
        destination = SIDestinationAddressFactory.getInstance()

.createSIDestinationAddress("ManufacturerBOutboundServiceDestination", false);
        frp.add(0, destination);
        graphs.add(new DataGraphHolder(dgManufacturerB,
inDataGraphHolder.getFormat(), frp));
    }
    newitems = manufacturerCItemsNode.getList("Item");
    if (newitems.size() > 0) {
        List frp = new ArrayList(1);
        destination = SIDestinationAddressFactory.getInstance()

.createSIDestinationAddress("ManufacturerCOutboundServiceDestination", false);
        frp.add(0, destination);
        graphs.add(new DataGraphHolder(dgManufacturerC,
inDataGraphHolder.getFormat(), frp));
    }

    return (DataGraphHolder[]) graphs.toArray(new DataGraphHolder[0]);
}

```

5. Right-click in the Java Editor and select **Source** → **Organize Imports**. This adds import statements for each of the classes used in the source code. You are challenged which imports to select in the Organize Imports window. At this point select the following:
 - java.util.List
 - java.util.Iterator
6. Save RequestMediator.java. It should contain no errors.

Response mediation code

The response mediation class must also implement the `handle()` method. The response mediation:

1. Receives the SOAP responses from Manufacturers.
2. Get the names of the log queue and the temporary storage queue. These names are defined as properties of the mediation within WebSphere Application Server V6.
3. Read the control message from log queue.
4. If the messages count on a log queue is greater than one, then:
 - a. Decrement count field on the control message, rewrite the control message to the log queue and put a SOAP response message from the Manufacturer on a temporary storage queue.
5. If the message count on a log queue is equal to one, then:
 - a. Read messages from the temporary storage queue.
 - b. Build a single SOAP response message for the Warehouse service.
 - c. Send the response message back to the Warehouse service.

In Rational Application Developer, perform the following steps to modify the response mediation `handle()` method:

1. Open the `ResponseMediator.java` file with the Java Editor. By default, the `handle()` method returns `false`. You must change that to `true` (Example 12-4).

Example 12-4 ResponseMediator class definition

```
public class ResponseMediator implements MediationHandler
{

    public boolean handle(MessageContext arg0) throws MessageContextException
    {
        // TODO Auto-generated method stub
        return true;
    }
}
```

2. Enter the code shown in Example 12-5 into the `handle()` method:

Example 12-5 ResponseMediator handle() method

```
System.out.println("ManufacturerBroker RESPONSE Mediator: Started.");
try {
    // Convert to an SIFMessageContext
    SIFMessageContext messageContext = (SIFMessageContext) arg0;
    // Get the SIFMediationSession
```

```

        SIMediationSession mediationSession = messageContext.getSession();
        // Get the message
        SIMessage message = messageContext.getSIMessage();
        // Get the name of the log queue
        String logQueueName = (String) arg0.getProperty("logQueueName");
        // Get the name of the queue used to store already received messages
        String tmpStorageQueueName = (String)
arg0.getProperty("tmpStorageQueueName");
        // Get the message count log message.
        SIMessage logMessage = mediationSession.receive(logQueueName, 0, null,
"SI_MessageID='" + message.getCorrelationId() + "'", false);
        // Get the message body as a datagraph, changes to the datagraph affect
message body.
        DataGraph logMsgDataGraph = logMessage.getDataGraph();
        // Get the body of the message
        String body = logMsgDataGraph.getRootObject().getString("data/value");
        int number = Integer.parseInt(body);

        if (number == 1) { // Last message from manufacturers
            // receive messages from temStorageQueue
            SIMessage receivedMessage = null;
            int numberOfManufacturers = 0;
            String respMsg = "ManufacturerBroker RESPONSE Mediator: Message from
Manufacturer: ";

            // process last response message, which is the current message.
            DataObject msgRootNode = message.getDataGraph().getRootObject();
            DataObject bodyNode = msgRootNode.getDataObject("Info/body");
            String ackPO = getAckPO(bodyNode);
            System.out.println(respMsg + ackPO);
            if (!ackPO.startsWith("FAILED"))
                numberOfManufacturers++;

            // process messages from temp storage queue
            while ((receivedMessage =
mediationSession.receive(tmpStorageQueueName, 0, null, "SI_CorrelationID='" +
message.getCorrelationId() + "'", false)) != null) {
                DataObject receivedRootNode =
receivedMessage.getDataGraph().getRootObject();
                DataObject receivedBodyNode =
receivedRootNode.getDataObject("Info/body");
                ackPO = getAckPO(receivedBodyNode);
                System.out.println(respMsg + ackPO);
                if (!ackPO.startsWith("FAILED"))
                    numberOfManufacturers++;
            }

            // set response message to Warehouse

```

```

        bodyNode.setString("ackP0", "ManufacturerBroker has submitted Purchase
Orders to " + numberOfManufacturers + " Manufacturer(s).");
    } else {
        // decrement message count and resend to the log queue.
        logMsgDataGraph.getRootObject().setString("data/value", "" + (number -
1));
        List frp = new ArrayList(1);
        SIDestinationAddress logQueueDestination =
SIDestinationAddressFactory.getInstance().createSIDestinationAddress(logQueueNa
me, true);
        frp.add(logQueueDestination);
        logMessage.setForwardRoutingPath(frp);
        // send new message to logQueue
        mediationSession.send(logMessage, false);
        // route the message to the temporary storage queue until the
lastcmessage arrives.
        frp = new ArrayList(1);
        SIDestinationAddress tempQDestination = SIDestinationAddressFactory
.getInstance()
.createSIDestinationAddress(tempStorageQueueName, true);
        frp.add(tempQDestination);
        message.setForwardRoutingPath(frp);
    }
} catch (Exception e) {
    System.out.println("ManufacturerBroker RESPONSE Mediator FAILED: " +
e.getMessage());
    e.printStackTrace();
    throw new MessageContextException(e.getMessage());
}

System.out.println("ManufacturerBroker RESPONSE Mediator: Finished.");
return true;

```

3. Add the code shown in Example 12-6 to define a new method to the RequestMediator class:

Example 12-6 getAckPO() method

```

private String getAckPO(DataObject node)
{
    String result = null;
    try {
        result = node.getString("ackP0");
    } catch(Exception e) {
        result = "FAILED to call a Manufacturer";
    }
}

```

```
}  
  
    return result;  
}
```

4. Right-click in the Java Editor and select **Source** → **Organize Imports**. This adds import statements for each of the classes used in the source code. You are challenged which imports to select in the Organize Imports window. At this point, select the following:
 - java.util.List
5. Save ResponseMediator.java. It should contain no errors.

12.3.4 Assigning and exporting the mediation handlers

To use the mediation handler classes, you must assign them to mediation handler lists, then export the code to an enterprise application project so it can be imported into WebSphere Application Server.

Assigning the mediation handlers

To assign the mediation handlers, follow these steps:

1. In the Project Explorer view, expand **EJB Projects** → **ManufacturerBrokerEJB**, and double-click **Deployment Descriptor**. The EJB Deployment Descriptor editor opens.
2. In the EJB Deployment Descriptor editor, click **Mediation Handlers**. This section allows you to define the mediation handlers.
3. In the Mediation Handlers section, click **Add** to add a request mediation handler.
4. In the Define Mediation Handler window, set the mediator name to RequestMediator and use the Browse button to set the Handler class to **com.ibm.itsogood.broker.RequestMediator**. Click **Finish**.
5. In the Mediation Handlers section, click **Add** to add a response mediation handler.
6. In the Define Mediation Handler window (Figure 12-9 on page 365), set the mediator name to ResponseMediator and use the Browse button to set the Handler class to **com.ibm.itsogood.broker.ResponseMediator**. Click **Finish**.

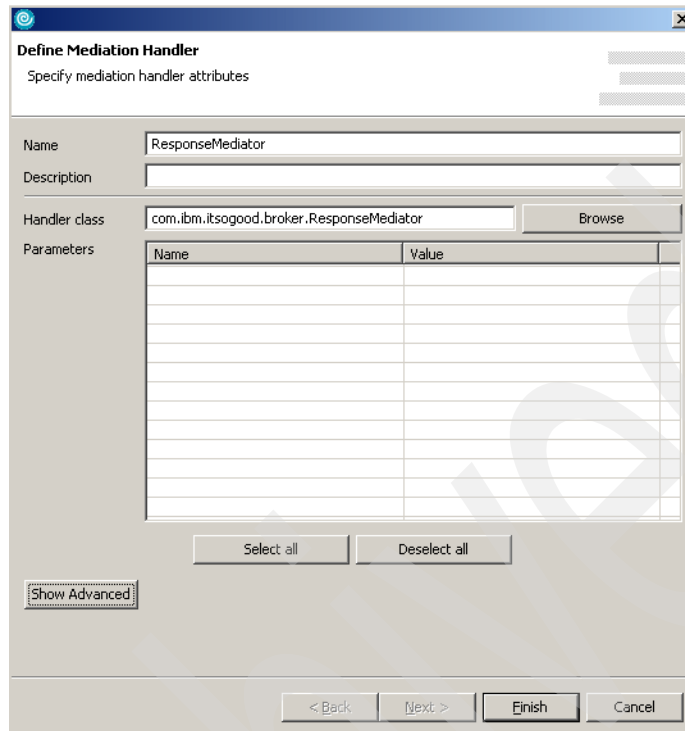


Figure 12-9 Defining mediation handler

7. Save the EJB Deployment Descriptor editor.

Exporting the mediation EAR file

To export the EAR file, complete these tasks:

1. In the Project Explorer view, expand **Enterprise Applications**, right-click **ManufacturerBroker** and select **Export** → **EAR File**.
2. In the Export wizard, enter a destination where you want to save the EAR file and click **Finish**.

12.4 Runtime guidelines

This section takes you through the steps that are involved for configuring the Exposed Broker runtime pattern using WebSphere Application Server V6. It describes the following tasks:

- ▶ Building the solution topology
- ▶ Configuring the service integration bus and Web services gateway

- ▶ Installing and configuring the applications
- ▶ Running and using the ITSO Good sample application
- ▶ An overview of the steps to add WS-Security to the completed solution

12.4.1 Solution topology

As in the previous chapters, to represent the complete business scenario the sample application is divided into four subapplications:

- ▶ ITSOGood contains the SCMSampleUI, Retailer, Warehouse, and LoggingFacility services.
- ▶ Manufacturer, ManufacturerB, ManufacturerC are three individual services, each packaged separately, and deployed to three different enterprises.

As described in the Product mapping in “Product implementation options” on page 345, we use WebSphere Application Server Network Deployment V6.0.2 to host the ITSOGood and Manufacturer applications, Microsoft .NET to host the ManufacturerB application, and CICS Transaction Server to host the ManufacturerC application. This is shown in Figure 12-10.

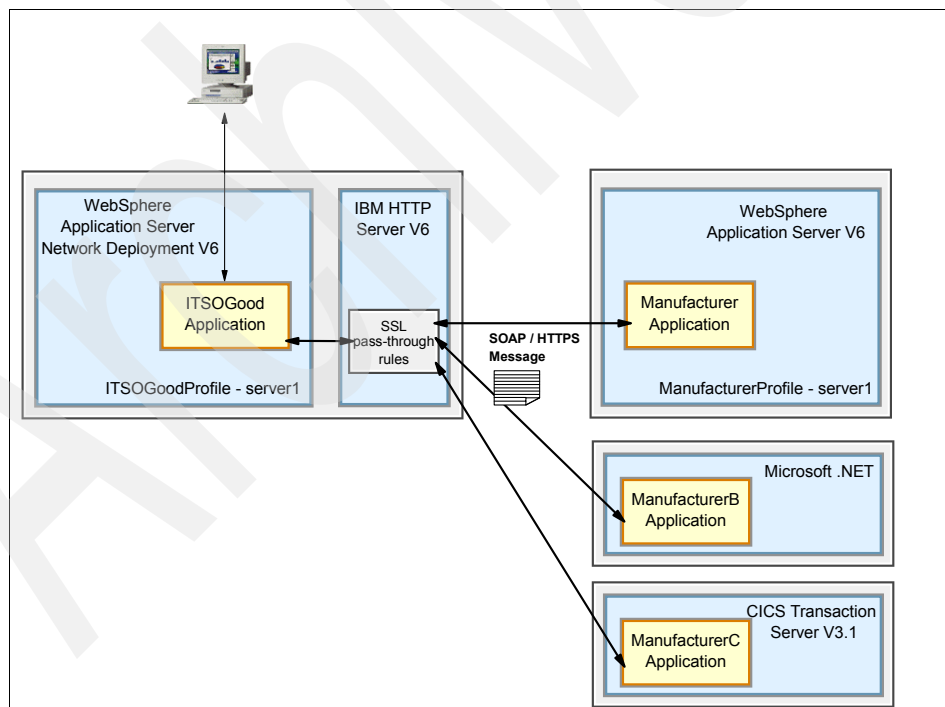


Figure 12-10 Solution topology

The runtime guidelines in this section describe how to prepare the ITSOGoodProfile server profile as an Exposed Broker. This includes describing how to add mediations, configure a service integration bus for Web services, and defining a Web services gateway.

The ESB component is primarily implemented in the service integration bus feature of WebSphere Application Server Network Deployment V6.0.2. The Exposed ESB Gateway component is implemented in the Web services gateway feature of WebSphere Application Server Network Deployment V6.0.2.

You need to implement either one or both additional manufacturers: ManufacturerB and ManufacturerC. At least two manufacturers are required to be able to demonstrate the Exposed Broker capability. To do this, you need access to a Microsoft .NET server, a CICS Transaction Server V3.1 region, or both.

Instructions for configuring the ManufacturerB and ManufacturerC servers are described in:

- ▶ Appendix B, “Microsoft .NET Web services” on page 483
- ▶ Appendix C, “CICS Transaction Server Web services” on page 513

12.4.2 Creating the basic infrastructure

To prepare the environment, you need to complete the following steps:

- ▶ Create WebSphere Application Server server profiles for ITSOGoodProfile and ManufacturerProfile by following the instructions in 10.4.2, “Configuring WebSphere Application Server profiles” on page 217.
- ▶ Configure an HTTP server to perform:
 - Hosting of WSDL files used by the enterprise applications, as described in 10.4.3, “Hosting the WSDL files” on page 219.
 - Configuration of SSL pass-through as described in 10.4.6, “Configuring an HTTP server for SSL pass-through” on page 224.

Important: If you are setting up this scenario in a host that is connected to the Internet, you need to follow these instructions:

1. Add the following statement to the hosts file:

```
127.0.0.1 www.ws-i.org
```

2. From the additional material, copy the contents of the \BrokerGeneric\SampleApplications directory to the following directory in the HTTP server:

```
<HTTP_SERVER_HOME>\htdocs\en_US\SampleApplications
```

- ▶ Add the ITSOGoodProfile server profile to a WebSphere Application Server Network Deployment deployment manager. This is required to enable the Web services gateway feature. Instructions for completing this task are described in “Creating a deployment manager” on page 254.
- ▶ Install ITSOGood and Manufacturer enterprise applications that do not contain WS-Security settings. In this scenario, use the WebSphere Application Server administrative console to define these WS-Security configuration settings rather than the enterprise applications themselves. Instructions for completing this task are described in “Installing ITSOGood and Manufacturer applications” on page 255.

After completing these steps, you should be able to test a working end-to-end ITSOGood sample application, as described in 10.4.8, “Testing the scenario” on page 229.

12.4.3 Configuring the service integration bus

This section describes how to configure a service integration bus for use with Web services, defines the service integration bus, then configures the Web services gateway.

These resources all run on the ITSOGoodProfile application server.

Perform the following tasks to create an SDO repository, and prepare WebSphere Application Server Network Deployment to use the service integration bus with Web services:

1. Install an SDO repository as described in “Installing the Service Data Objects (SDO) repository” on page 260.
2. Configure the SDO repository database as described in “Configuring the SDO repository” on page 260.
3. Install the Web services applications for the service integration bus as described in “Installing the Web services applications” on page 267.

When you have completed these installation tasks, you are ready to proceed with the remainder of the configuration.

Creating a service integration bus

Create a service integration bus by performing the following tasks:

1. From your Web browser, access and log into the Network Deployment administrative console at:

`http://itsogood.itso.ral.ibm.com:9062/ibm/console`

Assuming port 9062 has been assigned as the administrative port. Enter your correct port number if it is different.

2. Expand **Service integration** and click **Buses**.
3. Click **New**. In the field labeled Name, enter BUZLITEAR. *Deselect* the **Secure** option.
4. Click **OK**, and the bus is created. Save the configuration.

Adding a bus member

To add a bus member, follow these steps:

1. Expand **Service integration** and click **Buses**.
2. Click **BUZLITEAR** to show its properties. Under Additional Properties, click **Bus members**.
3. On the Bus members page click **Add**.
4. Accept the defaults, and click **Next**.
5. The next page that opens is the summary of your selections. Click **Finish**. The server is added as a member of the bus, and a messaging engine is created.
6. Save the configuration.

Creating the endpoint listener

Next, you need to create an endpoint listener (which will use the endpoint listener application at runtime). Endpoint listeners listen for incoming Web service requests and forward them onto the relevant inbound service. Inbound services are bound to an endpoint listener when they are created.

1. From the administrative console of the deployment manager, expand **Servers** and click **Application servers**.
2. Click **server1**.
3. Under Additional Properties, click **Endpoint Listeners**.
4. Click **New**.
5. Create an endpoint listener using the dialog box as shown in Figure 12-11 on page 370.

Application servers

[Application servers](#) > [server1](#) > [Endpoint Listeners](#) > **New**

An endpoint listener receives requests from service requester applications within a specific application server or cluster.

Configuration

General Properties

* Name
SOAPHTTPChannel1

Description

* URL root
http://itsogood.itso.ral.ibm.c

* WSDL serving HTTP URL root
psrv1a.itso.ral.ibm.com/wsd

Additional Properties

■ Connection Properties

The additional properties will not be available until the general properties for this item are saved.

Apply OK Reset Cancel

Figure 12-11 Creating an endpoint listener

6. In this dialog box, enter the following information:
 - Name is the name of the endpoint listener. It must have the name SOAPHTTPChannel1.
 - URL root is the base URL for Web service requests into this endpoint listener. The URLs that are used for making Web service requests to the service integration bus will have this root at the beginning. Set this to:
 http://itsogood.itso.ral.ibm.com:9080/wsgwsoaphttp1
 You can replace 9080 with the correct port number for your server.
 - WSDL serving HTTP URL root is the location of the HTTP URL that is serving your Web service WSDL. Enter a value of:
 http://apsrv1a.itso.ral.ibm.com/wsd1
7. Click **Apply**.
8. Under Additional Properties click **Connection Properties**.
9. Click **New** to create a new connection property.
10. Select **BUZLITEAR** from the Bus Name list and click **OK**.
11. Save the changes.

12.4.4 Creating the gateway service

The gateway service is associated with a single external service, defined by the WSDL, which is supplied during the creation of the gateway service.

To create the gateway service:

1. From the administrative console, expand **Service Integration** and click **Buses**.
2. Click **BUZLITEAR**.
3. Under Additional Properties, click **Web service gateway instances**.
4. The Web service gateway services page appears. Click **New**.
5. In the General Properties, as shown in Figure 12-12 on page 372, enter the following:
 - a. A unique name for the instance. In our case GatewayInstance.
 - b. Enter a gateway namespace. This is the namespace that will be used in all gateway-generated WSDL. It is a good practice to use a namespace that you are happy with from the start, because changing it later will require you to redeploy any associated gateway services. We use the namespace: `http://itsogood.gateway`.
 - c. Enter a default proxy WSDL URL, which is the generic template WSDL file supplied with WebSphere Network Deployment. Enter:

`http://itsogood.itso.ra1.ibm.com:9080/sibws/proxywsdl/ProxyServiceTemplate.wsdl`.

Buses

[Buses](#) > [BUZLITEAR](#) > [Web service gateway instances](#) > [New](#)

A web services gateway allows you to configure specific endpoint listeners and deploy gateway and proxy services to those listeners.

Configuration

General Properties

* Name
GatewayInstance

Description

* Gateway namespace
http://itsogood.gateway

* Default proxy WSDL URL
http://itsogood.itso.ral.ibm.c

Additional Properties

The additional properties will not be available until the general properties for this item are saved.

☒ Gateway services
☐ Proxy services

Buttons: Apply, OK, Reset, Cancel

Figure 12-12 Web service gateway creation

6. Click **Apply**.
7. Under Additional Properties, click **Gateway services**.
8. On the Gateway services page, click **New**.
9. Under Select type of target service, select **WSDL-defined web service provider** as the type of target service and click **Next**.
10. On the next wizard page (Figure 12-13 on page 373) enter the gateway service and destinations names.

→ Step 1: Specify gateway service name, service destinations and mediations

Step 2: Locate the target service WSDL

Step 3: Select service

Step 4: Select ports

Step 5: Name the service and port destinations

Step 6: Select points for service and port destinations

Step 7: Select endpoint listeners

Step 8: Define UDDI publication properties

Specify gateway service name, service destinations and mediations

Provide the names of the objects that are to be used to support the gateway service in the runtime.

* Gateway service name

Gateway request destination name

Request mediation

Request mediation bus member

Gateway response destination name

Response mediation

Response mediation bus member

Figure 12-13 Specify gateway service name and destinations

11. Mediations will not be configured at this time, so leave them set to (none). The following lists the values that you must enter:
 - Gateway service name: ManufacturerGatewayService.
 - Gateway request destination name:
 ManufacturerGatewayRequestDestination.
 - Gateway response destination name:
 ManufacturerGatewayResponseDestination.
 - Click **Next**.
12. On the next page, specify the location of the target service WSDL. We are using a URL location type. Under WSDL location type, select **URL**. Enter `http://appsrv1a.itso.ra1.ibm.com/wsd1/Manufacturer_Impl.wsd1` for the WSDL location. Click **Next**.
13. On the next page, select the service in the WSDL that you want to configure. The Manufacturer application only exposes one service. Click **Next**.
14. Select the port **Manufacturer** and click **Next**.
15. On the next page, name the service and destinations, as shown in Figure 12-14 on page 374.

Step 5: Name the service and port destinations

Optionally rename the outbound service and the service and port destinations from the default generated names if required, provide the name of the port selection mediation, if any, and identify the default port.

* Outbound service name
ManufacturerOutboundService

* Service destination name
rOutboundServiceDestination

Port selection mediation
(none)

Default port	Port name	Port destination name
<input checked="" type="radio"/>	Manufacturer	* ManufacturerOutboundPortDe

Previous Next Cancel

Figure 12-14 Naming the destinations

16. Enter the following values:

- Outbound service name: ManufacturerOutboundService
- Service destination name: ManufacturerOutboundServiceDestination
- Port destination name: ManufacturerOutboundPortDestination
- Click **Next**.

17. Each port destination needs to be assigned to a bus member. Select the bus member and click **Next**.

18. Select the Endpoint listener and click **Next**.

19. For our scenario, we will not be publishing to a UDDI registry. Click **Finish**.

20. Save the configuration.

Creating additional Manufacturer outbound services

Outbound services define Web service requests that leave the service integration bus and are received by a service provider.

To define an outbound service for the ManufacturerB Web service:

1. From the administrative console, expand **Service Integration** and click **Buses**.
2. Click **BUZLITEAR**.

3. Under Services, click **Outbound Services**. Notice there is already one outbound service defined, ManufacturerOutboundService, which was created as a result of defining the gateway service.
4. Click **New**.
5. The first page of the wizard requires you to specify a URL or UDDI repository where a WSDL definition of the service can be found. In this case, use a URL. The URL option allows you to specify an HTTP URL or a file system path. Enter the following URL and click **Next**.

```
http://appsrv1a.itso.ral.ibm.com/wsdl/ManufacturerB_Impl.wsdl
```
6. The next page displays the available services that are defined in the WSDL file. There is only one service to select. Click **Next**.
7. The next page displays the ports that are defined for the selected service. There is only one port in ManufacturerB service. Select **ManufacturerB** and click **Next**.
8. On the next page, we change the name of the outbound service and destinations. Enter the following values then click **Next**:
 - Outbound service name: ManufacturerBOutboundService
 - Service destination name: ManufacturerBOutboundServiceDestination
 - Port destination name: ManufacturerBOutboundPortDestination
9. On the next page you can select the bus member you want to assign the outbound service. Accept the default and click **Finish**. The outbound service is created.
10. Repeat these steps for the ManufacturerC service, using the URL locations and name of outbound service and destinations that are specified in Table 12-2 and Table 12-3.

Table 12-2 Web services and WSDL Locations

Web service	WSDL location
ManufacturerCService	http://appsrv1a.itso.ral.ibm.com/wsdl/ManufacturerC_Impl.wsdl

Table 12-3 Outbound service and destinations names

Outbound Service	ManufacturerCOutboundService
Service Destination	ManufacturerCOutboundServiceDestination
Port Destination	ManufacturerCOutboundPortDestination

11. Save the configuration.

12.4.5 Installing and defining the mediation application

This section describes how to install the necessary enterprise applications into the ITSOGoodProfile server, and how to configure the mediation.

Install the mediation, and modified ITSOGood application

This solution requires a modified version of the ITSOGood enterprise application, where the Warehouse application no longer contains the logic to determine which Manufacturer to invoke for a given product.

Perform the following in the Network Deployment administrative console:

1. Expand **Applications** and click **Enterprise Applications**.
2. If an enterprise application named ITSOGood is already installed, you must remove it. Check **ITSOGood** then click **Uninstall**. When it is uninstalled, save the changes.
3. Select **Applications**, and click **Install New Application**. We can now install the ITSOGood enterprise application with the modified Warehouse.
4. Enter the location of the ITSOGood.ear file. This EAR file is in the additional material, provided with this book, under \BrokerGeneric\ears directory. Click **Next**.
5. On the next page that appears, accept the defaults, and click **Next**.
6. The next page to open is the first page in the wizard. Click **Next** until the final page, or click the last step that is labelled **Summary**.
7. Click **Finish**.
8. Save the changes and start the application.
9. Repeat these steps to install ManufacturerBroker.ear which contains the mediation handler code that you exported in 12.3.4, "Assigning and exporting the mediation handlers" on page 364. Once installed, save the changes and start the application.

The ITSOGood and ManufacturerBroker enterprise applications should now be started (Figure 12-15 on page 377).

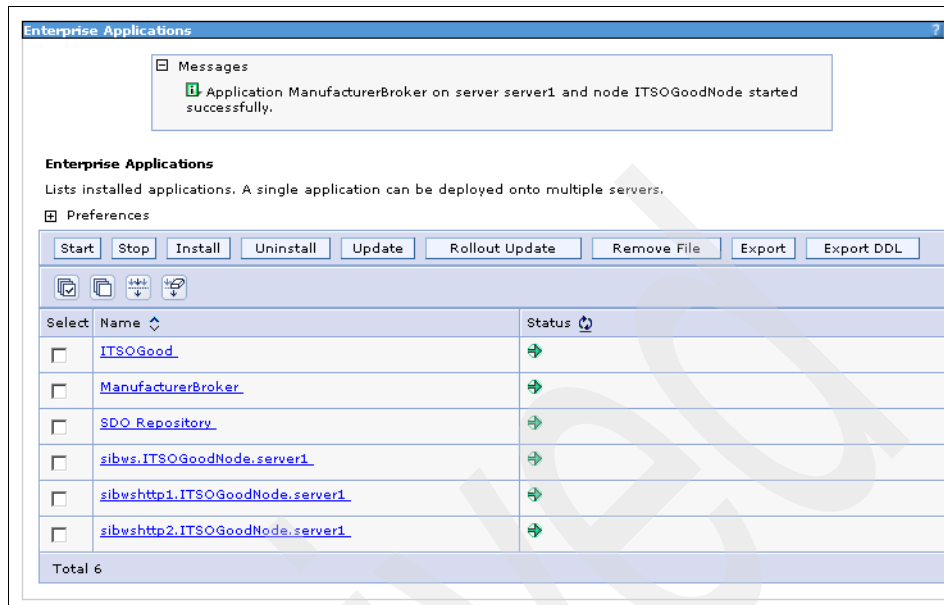


Figure 12-15 ITSOGood and ManufacturerBroker enterprise applications

Defining mediations

After the mediation application (ManufacturerBroker) has been installed, you need to define a mediation to the deployed handler list. Use the administrative console to complete the following steps:

1. Navigate to the mediations pane. Click **Service integration** → **Buses** → **BUZLITEAR** → **Mediations**.
2. Click **New** to add a new mediation.
3. In the new mediation page (Figure 12-16 on page 378), specify the relevant properties for the mediation.

Configuration

General Properties

* Mediation name
RequestMediator

UUID
D44CDD0FE17DF109

Description

* Handler list name
RequestMediator

☐ Global transaction

☐ Allow concurrent mediation

Selector

Discriminator

Apply OK Reset Cancel

Figure 12-16 Defining a new destination mediation

- a. Set the following properties:
 - Mediation name is a name for the mediation that is unique to the service integration bus. This name is used to identify the mediation for administrative purposes. Set this to RequestMediator.
 - Handler list name was determined by the programmer who deployed the mediation handler. Set this to RequestMediator.
 - Global transaction, if selected, starts a global transaction for each message mediated by the mediation. Select this box.
- b. Other properties that you can set are:
 - Description is a description of the behavior of the mediation.
 - Allow concurrent mediation mediates multiple messages at the mediated destination.
 - Selector controls which messages are mediated.
- c. Click **OK**.
4. Create a second mediation called ResponseMediator, specifying a handler list of ResponseMediator, and selecting **Global transaction**.
5. Click **Save** to save the changes to the master configuration.

Mediating a destination

Mediating a destination associates a mediation with a selected service integration bus destination. At runtime, the mediation applies some message processing to the messages handled by the service integration bus destination.

Note: You can only mediate a destination with a single mediation at a time. You can mediate more than one destination with the same mediation.

You need to mediate two destinations, as listed in Table 12-4.

Table 12-4 Mediations and associated destinations

Mediation	Destination to be mediated
RequestMediator	ManufacturerGatewayRequestDestination
ResponseMediator	ManufacturerGatewayResponseDestination

To mediate a destination, use the administrative console to complete the following steps:

1. Click **Service integration** → **Buses** → **BUZLITEAR**. Under Destination Resources, click **Destinations**.
2. Check the bus destination that you want to mediate (in our case, **ManufacturerGatewayRequestDestination**), and click **Mediate**.
3. In the Select mediation wizard, set the mediation to apply to this destination. In our case, select **RequestMediator**. Click **Next**.
4. Select the bus member where the mediation is installed (by default there is only one), and click **Next**.
5. Click **Finish** to mediate the destination.

Repeat these steps to assign the ResponseMediator mediation to the ManufacturerGatewayResponseDestination destination. When complete, you should see two mediations that are defined in the destinations list, as shown in Figure 12-17. Save the changes to the configuration.

<input type="checkbox"/>	ManufacturerGatewayRequestDestination	Web service		RequestMediator
<input type="checkbox"/>	ManufacturerGatewayResponseDestination	Web service		ResponseMediator

Figure 12-17 Mediations assigned to destinations

Configuring context properties for a mediation

The mediation context is used in conjunction with the message header information to ensure that messages are processed correctly by a mediation.

This scenario needs context properties for both the mediations. Table 12-5 shows the properties that are needed for the RequestMediator and ResponseMediator mediation.

Table 12-5 Mediation context properties

Name	Context Type	Context Value
logQueueName	String	logQ
tmpStorageQueueName	String	tmpStorageQ

To configure context information for a mediation, use the administrative console to complete the following steps:

1. Find the destination you want to mediate. Click **Service integration** → **Buses** → **BUZLITEAR**. Then, under Destination resources, click **Mediations** and choose the mediation in which you are interested. We will start with **RequestMediator**.
2. Under Additional Properties, click **Context properties**.
3. Click **New** and specify the properties for the context information, the name, the context type, and the context value. To start with define the logQueueName context property. Click **OK**.
4. Follow these steps for all of the context properties that are listed in Table 12-5 on page 380, remember to define these context properties for both RequestMediator and ResponseMediator.
5. When complete, save the configuration.

12.4.6 Creating additional destinations

In this section we define additional queue destinations into the BUZLITEAR bus. The queues are used by the mediation handler to control the separation and aggregation of the messages.

Follow these steps to create the queue destinations:

1. From the administrative console, click **Service integration** → **Buses** → **BUZLITEAR**.
2. Under Destination resources, click **Destinations**.
3. Click **New**.

4. In the page that opens, as shown in Figure 12-18, you can select the type of destination to be created. Accept the default of **Queue**, and click **Next**. This launches the Queue creation wizard.

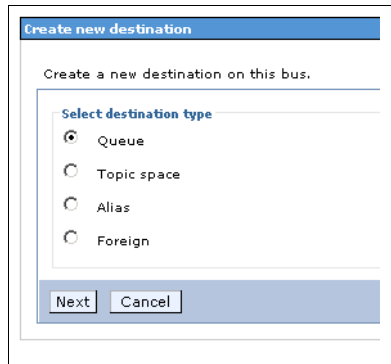


Figure 12-18 Destination type selection page

5. The first page of the wizard, as shown in Figure 12-19, asks for an identifier and description. The identifier is the name by which the destination will be exposed to applications. Enter `logQ`, and click **Next**.

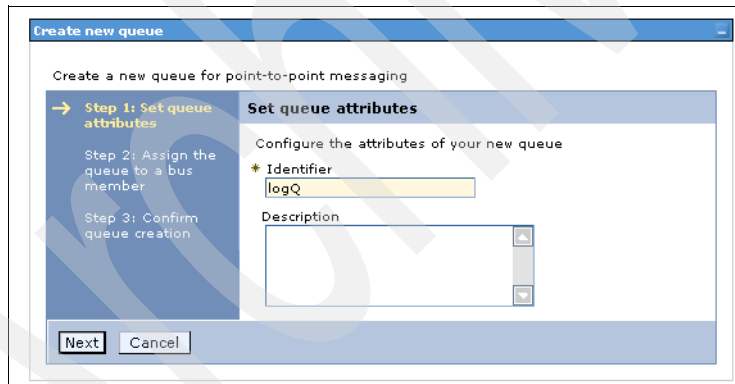


Figure 12-19 New destination wizard

6. The next page allows you to specify to which bus member to assign the destination. There is only one bus member in this scenario, so accept the default. Click **Next**.
7. The final page is the summary. Click **Finish**, and the destination is created.
8. We need to create an additional queue type destination. Repeat these steps to create a queue destination called `tmpStorageQ`.
9. Save the changes.

12.4.7 Changing the Warehouse endpoint URL

By default, the Warehouse module, in ITSOGood application, directly invokes the Manufacturer application. We need to change the endpoint URL of the Warehouse application to invoke the inbound service `ManufacturerGatewayService`.

To find out what is the endpoint URL of the `ManufacturerGatewayService`, export the WSDL that defines the inbound service for the Manufacturer Web service gateway service. From the administrative console, follow these steps:

1. Expand **Service Integration** and click **Buses**.
2. Click **BUZLITEAR**.
3. Under Services, click **Inbound Services**.
4. Click on **ManufacturerGatewayService**.
5. Under Additional Properties, click **Publish WSDL files to ZIP file**.
6. On the next page, click **ManufacturerGatewayService.zip**. Click **Open** on the file dialog box that opens.
7. A list of WSDL files opens in the program you have associated with ZIP files.
8. Open the WSDL file named **BUZLITEAR.ManufacturerGatewayServiceService.wsdl**. The service definition should look similar to Figure 12-7:

Example 12-7 BUZLITEAR.ManufacturerGatewayServiceService.wsdl

```
<wsdl:service name="ManufacturerGatewayService">
  <wsdl:port name="ITSOGoodNode_server1_SOAPHTTPChannel1_InboundPort"
binding="sibusbinding:ITSOGoodNode_server1_SOAPHTTPChannel1_InboundPortBinding">
  <soap:address
location="http://itsogood.itso.ra1.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/BUZLITEAR/Manufact
urerGatewayService/ITSOGoodNode_server1_SOAPHTTPChannel1_InboundPort"/>
  </wsdl:port>
</wsdl:service>
```

9. Copy the value of the `soap:address location` element to the clipboard, which is:

```
http://itsogood.itso.ra1.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/BUZLITEAR/ManufacturerGatewayService/ITSOGoodNode_server1_SOAPHTTPChannel1_InboundPort
```

To change the endpoint URL in the client bindings configuration, follow these steps:

1. From the console, expand **Applications**, and click **Enterprise Applications**.

2. In the list of installed applications, click **ITSOGood**. The application configuration screen opens.
3. Click **EJB Modules**. The server-module installation binding for an EJB module screen opens.
4. In the list of EJB Modules, click **WarehouseEJB.jar**. The selected EJB Module configuration screen opens.
5. Click the link **Web Services client bindings** and the screen shown in Figure 12-20 opens.

Web Service	EJB	WSDL Filename	Preferred Port Mappings	Port Information
ManufacturerService	WarehouseSoapBindingImpl	Use default (META-INF/wsdl/Manufacturer_Impl.wsdl)	Edit...	Edit...
LoggingFacilityService	WarehouseSoapBindingImpl	Use default (META-INF/wsdl/LoggingFacility_Impl.wsdl)	Edit...	Edit...

Figure 12-20 Web Services client bindings

6. In the row for ManufacturerService, click **Edit** under the Port Information column.
7. The Port Information for Web services screen opens. In the Overridden Endpoint URL field paste the endpoint URL:

`http://itsogood.itso.ral.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/BUZLITEAR/ManufacturerGatewayService/ITSOGoodNode_server1_SOAPHTTPChannel1_InboundPort`

8. Click **OK**, then save the configuration.
9. Restart the application server for all the changes to take effect.

12.4.8 Testing the scenario

This section describes how to test the scenario using the ITSO Good sample application. Full instructions on the ITSO Good sample application can be found in 10.4.8, “Testing the scenario” on page 229.

In order to test the Broker capabilities of the mediation, we need to create an order than requires two Manufacturers to be contacted. Be sure all of the Manufacturers you have implemented are running before continuing. Perform the following tasks:

1. Enter the following URL in a Web browser to start the ITSOGood sample application:

`http://itsogood.itso.ral.ibm.com:9080/SCMSampleUI/`

This assumes the unsecured HTTP transport port number is 9080. If this is not the case, use the URL above with the appropriate HTTP transport port number.

2. Click **Place Order**.
3. We need to place an order that will require the Warehouse to replenish stock from one or more Manufacturers. The mediation you have coded receives restock requests from the Warehouse and forwards them on to appropriate Manufacturers. It selects the Manufacturer to use for a given product based on the product number. This is shown in Table 12-6.

Table 12-6 Relationship between product number and Manufacturers

Product number	Manufacturer used
605001 605004 605007	ManufacturerA
605002 605005 605008	ManufacturerB
605003 605006 605009	ManufacturerC

Let us assume we want to trigger calls to ManufacturerA and ManufacturerB. To do this, we need to order sufficient stock to require the Warehouse to replenish these products. For example, the stock level for the first three products is shown in Table 12-7.

Table 12-7 Warehouse stock level

Product number	Current level	Minimum level	Maximum level
605001	10	5	25
605002	7	4	20
605003	15	10	50

Order **six** items of 605001 and **six** items of 605002. This will reduce both products below their minimum stock level, and require stock to be reordered from ManufacturerA and ManufacturerB. Click **Submit Order**.

4. To confirm the successful invocation of the manufacturers, check the SystemOut.log of the application server where ITSOGood application is running. The message should look as shown in Example 12-8 on page 385.

Example 12-8 SystemOut.log messages

```
Warehouse: Calling Manufacturer Broker. Number of Items to order = 2
ManufacturerBroker REQUEST Mediator: Starting...
ManufacturerBroker REQUEST Mediator: Processing product 605001.
ManufacturerBroker REQUEST Mediator: Processing product 605002.
ManufacturerBroker REQUEST Mediator: Finished.
ManufacturerBroker RESPONSE Mediator: Started.
ManufacturerBroker RESPONSE Mediator: Finished.
ManufacturerBroker RESPONSE Mediator: Started.
ManufacturerBroker RESPONSE Mediator: Message from Manufacturer: Manufacturer_A has received
and processed a request.
ManufacturerBroker RESPONSE Mediator: Message from Manufacturer: Manufacturer_B has received
and processed a request.
ManufacturerBroker RESPONSE Mediator: Finished.
Warehouse: ManufacturerBroker has submitted Purchase Orders to 2 Manufacturer(s).
```

12.4.9 Adding WS-Security to the solution

At this stage, the connection between the Warehouse and the Manufacturers is using HTTP/S, but not WS-Security. To add WS-Security, follow the instructions in 11.4.6, “Adding WS-Security to the Web service gateway” on page 304.

Archived

Exposed Router runtime pattern: SOA profile

This chapter describes how to configure WebSphere Partner Gateway V6 to implement the Exposed ESB Gateway. It describes how to use the SOAP pass through feature of WebSphere Partner Gateway to communicate with the Extended Enterprise.

13.1 Business scenario

The business scenario that is implemented in this chapter is shown in Figure 13-1.

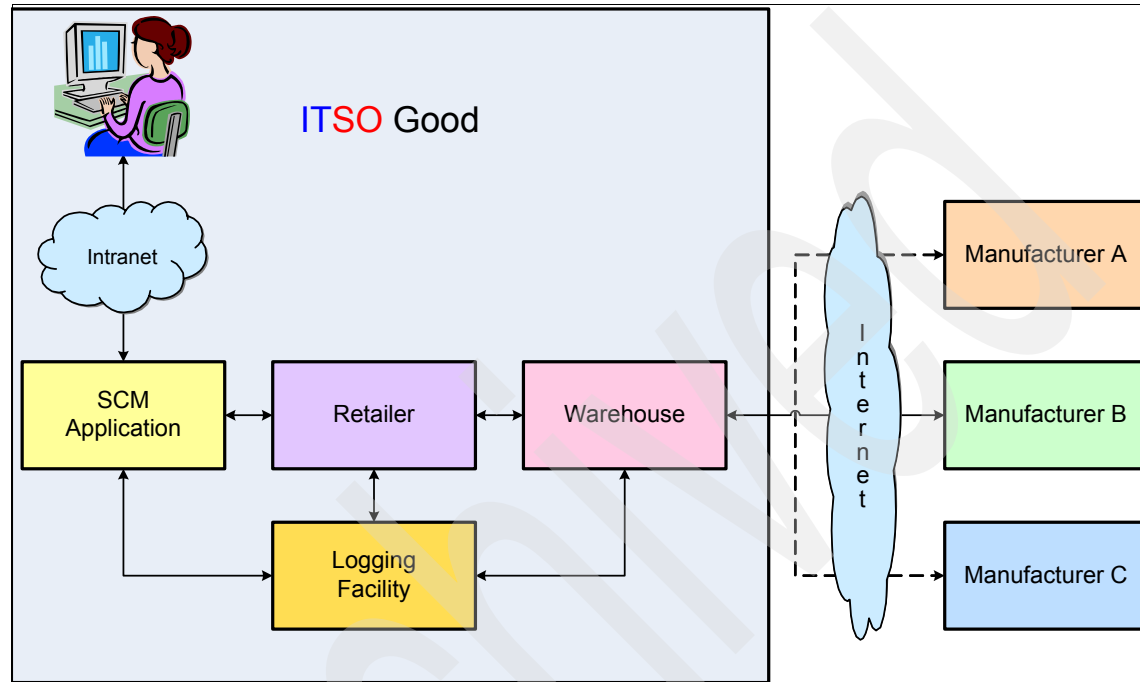


Figure 13-1 High-level business context

The expanding product lines of the Manufacturers presents ITSO Good with the option of having multiple Manufacturers supplying the same product. And in the face of stiff pricing competition within the market, ITSO Good have quickly decided to enter into a volume-based agreements with their small, medium, and large-scale manufacturing partners.

As a result, ITSO Good have come up with the following additional business requirements:

1. The Warehouse systems should route the customer orders based on the product quantity (volume) to different manufacturing partners.
2. ITSO Good also plans to enter into agreements with some more manufacturing partners to support their growing order volumes, therefore they would like the proposed IT solution to have the flexibility to maintain partner profile information such as trade volumes and locations.

13.2 Design guidelines

In this section, we analyze the business requirements and apply Patterns for e-business to determine the appropriate runtime pattern for the solution. We then discuss the various design options available to us in implementing the solution and also look at the product mappings.

13.2.1 Analyze business requirements

The business scenario requires the replenishment order request from the Warehouse application to be forwarded to, at most, one of the multiple partner Manufacturer applications and services. This requires moving the routing rules from the individual applications within the enterprise to the integration middleware component.

The benefits of moving the routing logic out to the integration middleware component are:

- ▶ It allows the integration of multiple, diverse partner applications.
- ▶ It provides routing services, relieving the source application from being aware of the target application.
- ▶ The use of a central router minimizes the impact of changes in location of the target application.

The solution also needs to support the definition of trading partner profiles delineating preferences, entitlements and contacts.

13.2.2 Selecting a pattern

We use the Patterns for e-business to determine the appropriate Runtime pattern to apply to this scenario.

Described here is a step-by-step approach used to navigate the Patterns for e-business asset catalog:

1. Business pattern

We select the Extended Enterprise business pattern because the given scenario requires interactions between the business processes in the Warehouse and Manufacturer systems that reside in separate enterprises.

2. Application pattern

Because the source application (Warehouse) initiates an interaction that is forwarded to, at most, one of many partner (Manufacturer) applications, we choose the Router variation of the Broker application pattern as shown in the Figure 13-2 on page 390.

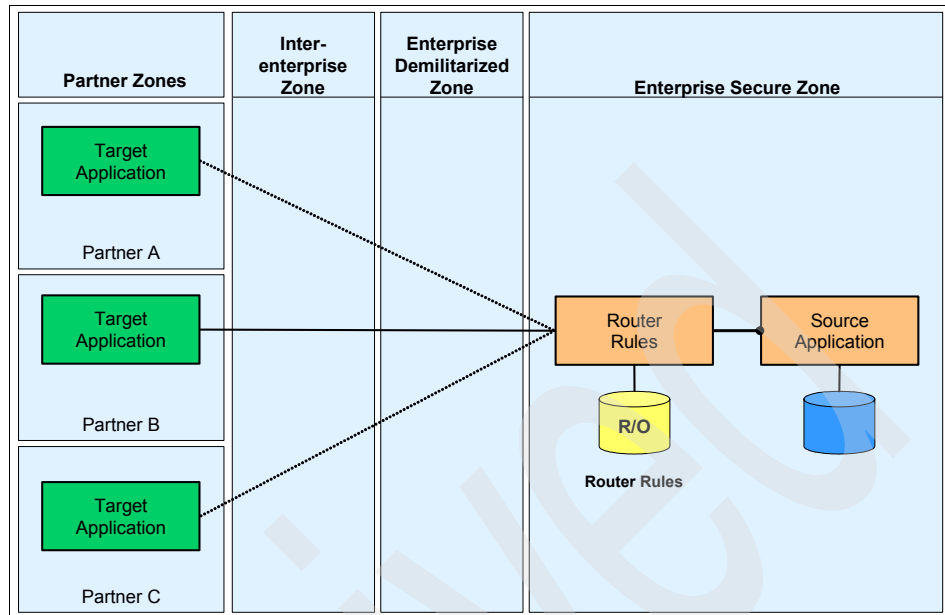


Figure 13-2 Exposed Broker application pattern:: Router variation

3. Runtime pattern

The selection of the application pattern provides us with the possible runtime patterns for the proposed solution. Because this solution requires building an SOA infrastructure, we select the SOA profile to customize the solution to the SOA environment.

Figure 13-3 on page 391 shows the level 0 decomposition of the SOA profile of the Exposed Router runtime variation, mapped on to the Exposed Router application variation.

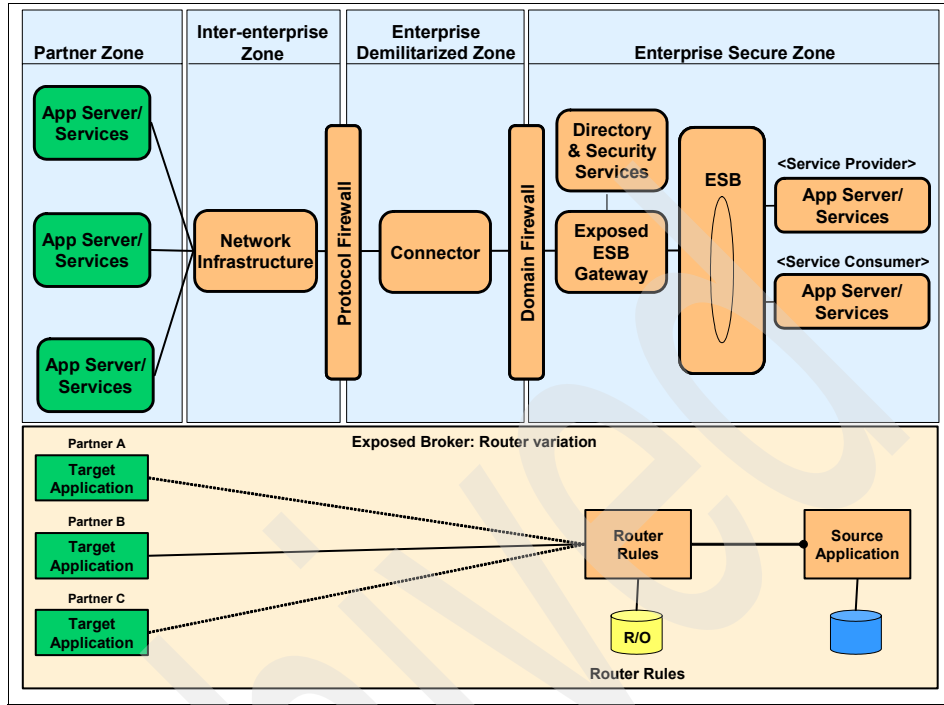


Figure 13-3 Exposed Router runtime pattern = SOA profile

13.2.3 Analyze design options

This section discusses the architectural decisions that we made and their implementation options for the given business scenario using the Exposed Router runtime variation.

Most of the design decisions made in the previous chapters, also apply to the Router interactions:

- “Architectural decision: integration options” on page 163 discusses the use of Web services to integrate with the external partner services.
- “Architectural decision: implementing an ESB” on page 244 discusses the service integration bus component of WebSphere Application Server is used to implement the ESB.
- “Architectural decision: Maintaining an audit trail” on page 246 discusses the Exposed ESB Gateway component handles the auditing aspects of the interactions with the extended enterprise.

There are also additional design decisions that are specific to this scenario, which are discussed in the following sections.

Architectural decision: implementing the Exposed ESB Gateway

Table 13-1 summarizes the requirement for the Exposed ESB Gateway component and looks at the various alternatives that are available.

Table 13-1 Architectural decision: implementing the Exposed ESB Gateway

Decision title	The most suitable operational topology
Issue or Problem statement	Product selection for implementing the Exposed ESB Gateway component that satisfies the following requirement: 1. Provides a controlled and secured access to external services. 2. Partner profile management.
Assumptions	None.
Motivation	Keeping in mind that ITSO Good has plans to expand and enter into agreements with some more partner Manufacturers. The ability to support different business protocols such as EDI, RosettaNet, and AS1/AS2 could prove to be a differentiating factor.
Alternatives	1. Web services gateway component of the WebSphere Application Server Network Deployment V6.0. 2. WebSphere Partner Gateway.
Decision	The WebSphere Partner Gateway is used to implement the Exposed ESB Gateway component.

Justification	<p>Web services gateway :</p> <ul style="list-style-type: none"> ► Provides extensive support for Web services and its related standards such as WS-Security. ► Provides controlled and secured access to external Web services. ► Provides mediation capabilities for business-to-business integrations using Web services. ► Lacks the support for traditional EAI integration and related business protocols. <p>WebSphere Partner Gateway:</p> <ul style="list-style-type: none"> ► It is fully scalable and designed to support the diverse protocol, document-processing, and security requirements of large and small companies alike. ► Mediation capabilities to support message routing, transformation and validation. It can also handle digital signature verification. ► Partner profile management. ► Non-repudiation via authentication and audit. ► Provides only pass-through support for SOAP requests. <p>In conclusion, by delegating the Web services message level security to the ESB as discussed in the Related Decision, we can use WebSphere Partner Gateway to support Web service interactions using the SOAP pass-through support and be able to leverage on its extended business-to-business functions.</p>
Related Decision	Refer to the “Architectural decision: securing the Web service interaction” on page 393.

Attention: While evaluating the product options for the Exposed ESB Gateway component, the business criticality of the various factors (such as security, auditing, partner profile management, support for traditional EAI integration, and so forth) involved should be weighed against each other carefully. This could also be influenced by the business and IT strategy of the organization and their long-term plans.

Architectural decision: securing the Web service interaction

WebSphere Partner Gateway provides the Exposed ESB Gateway capabilities for additional control and monitoring of access to external services using the administrative facilities in its Community console. It also provides security

features for authentication using either SSL or basic authentication. It additionally provides a nonrepudiation repository.

However, WebSphere Partner Gateway lacks support for the WS-Security extensions of Web services and only provides SOAP pass-through support. Therefore if they are required, these features must be delegated to another product. The WebSphere Application Server service integration bus provides support for securing Web service interactions using WS-Security and is used in this scenario to provide the Web services security support.

13.2.4 Products

In this section, we examine the products that could be used to implement the ESB and Exposed ESB nodes of the Exposed Router runtime variation.

Product implementation options

Product choices for this scenario are based on:

- ▶ Design decisions that we made in 13.2.3, “Analyze design options” on page 391
- ▶ Extended Enterprise capabilities of the products
- ▶ Products that are currently available

ESB component

We can use the following currently available products to implement the ESB component in the given scenario:

- ▶ Service integration bus in WebSphere Application Server Network Deployment V6.0.2 or WebSphere Application Server V6.0.2.
- ▶ WebSphere Business Integration Message Broker V5

For this scenario, the service integration bus in WebSphere Application Server Network Deployment V6.0.2 meets all of the requirements and, as a result, is the product of choice.

Exposed ESB Gateway component

We can use the following currently available products to implement the Exposed ESB Gateway component in the given scenario:

- ▶ Web services gateway component available in the WebSphere Application Server Network Deployment V6.0.2 package.
- ▶ WebSphere Partner Gateway V6.

In the given scenario, the requirement for advanced functions such as partner management and provisioning makes WebSphere Partner Gateway product an ideal choice.

Product mapping selected

The complete product mapping for this scenario is shown in Figure 13-4.

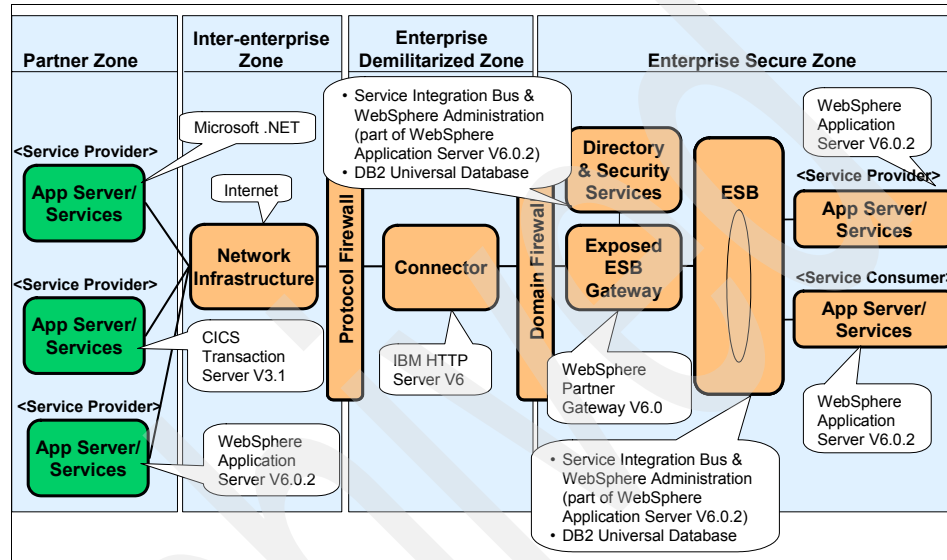


Figure 13-4 Exposed Router variation: SOA profile product mapping

This Product mapping uses WebSphere Application Server V6.0.2 to host the Application Services in the Enterprise Secure Zone.

The ESB is run in a service integration bus within WebSphere Application Server Network Deployment V6.0.2, providing service location transparency between service consumers and service providers within the enterprise. With the Network Deployment offering, you can implement a scalable clustering of multiple WebSphere Application Server servers.

A local DB2 Universal Database database is used to store the SDO repository.

The WebSphere Partner Gateway acts as the Exposed ESB Gateway node providing a standard, consistent interface for the internal processes to access external processes. The use of an Exposed ESB Gateway minimizes the disruption caused by changes in the external partner infrastructure.

In the Directory and Security services node, the service integration bus within WebSphere Application Server Network Deployment V6.0.2 is configured to

secure all transactions to the external Partner Zone to use WS-Security integrity and confidentiality.

The IBM HTTP Server V6 acts as an Adapter Connector by providing protocol transformation to SOAP/HTTPS thus delivering transport level security between the enterprise and the partner organizations.

13.3 Development guidelines

There are no specific development guidelines to build this scenario. In this section we simply introduce the scenario used in this chapter.

13.3.1 Scenario implementation: Exposed Router SOA profile interaction

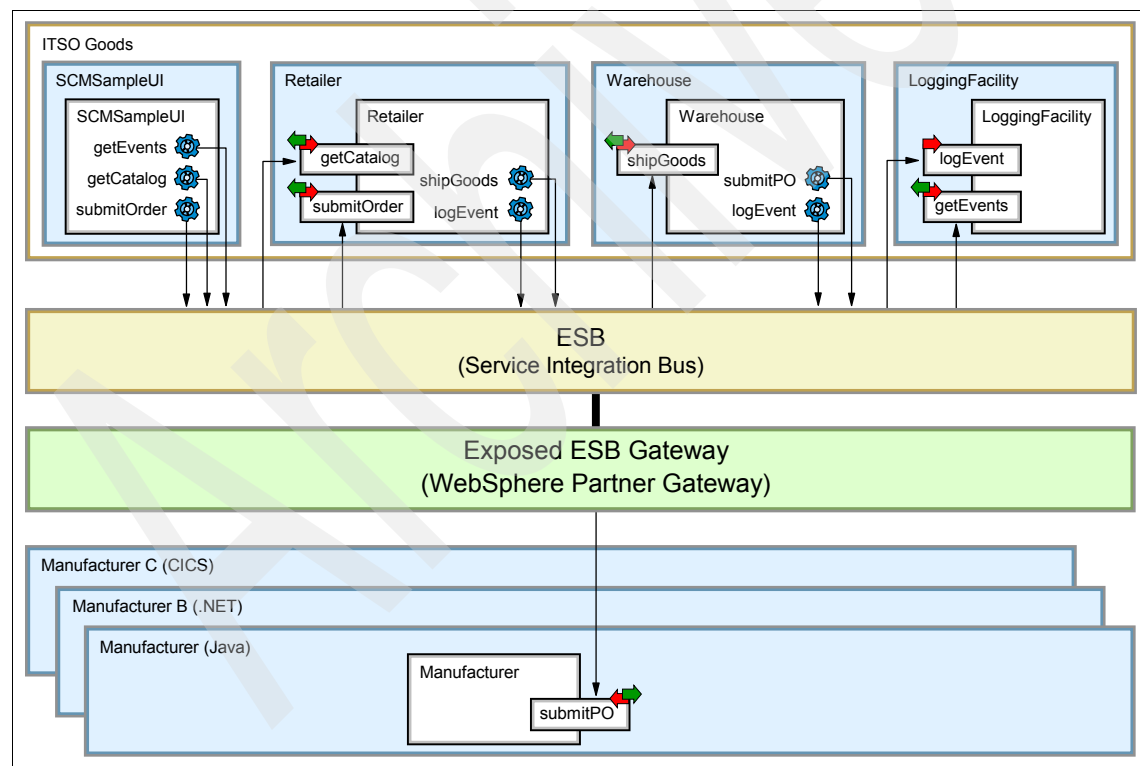


Figure 13-5 Scenario implementation using the Exposed Router: SOA profile

As shown in Figure 13-5 on page 396, the SOA profile is implemented by introducing the ESB and the Exposed ESB Gateway component. The ESB integrates all the service calls at an enterprise level and provides the mediation logic built on business agreements and terms. Any interaction with an extended enterprise is managed through the Exposed ESB Gateway component, which we have implemented using WebSphere Partner Gateway V6.

Note: The link between the ESB and Exposed ESB Gateway will be configured using instructions in the Runtime guidelines section.

13.4 Runtime guidelines

This section describes how to configure the runtime aspects of this scenario. It uses the basic infrastructure of Chapter 11, “Exposed Direct Connection runtime pattern: SOA profile” on page 237 as a starting point. In this section, you add a WebSphere Partner Gateway configuration to act as the Exposed ESB Gateway, and define new outbound services in the ESB to point to WebSphere Partner Gateway configuration for each Manufacturer call.

13.4.1 Solution topology

As in the previous chapters, to represent the complete business scenario the sample application is divided into four subapplications:

- ▶ ITSOGood contains the SCMSampleUI, Retailer, Warehouse, and LoggingFacility services.
- ▶ Manufacturer, ManufacturerB, ManufacturerC are three individual services, each packaged separately, and deployed to three different enterprises.

Calls to the three Manufacturer enterprise applications are sent to WebSphere Partner Gateway V6. This application uses a SOAP pass-through to send SOAP messages to an HTTP Server, which adds HTTP/S security to the SOAP message and forwards the request on to the relevant Manufacturer. This is shown in Figure 13-6 on page 398.

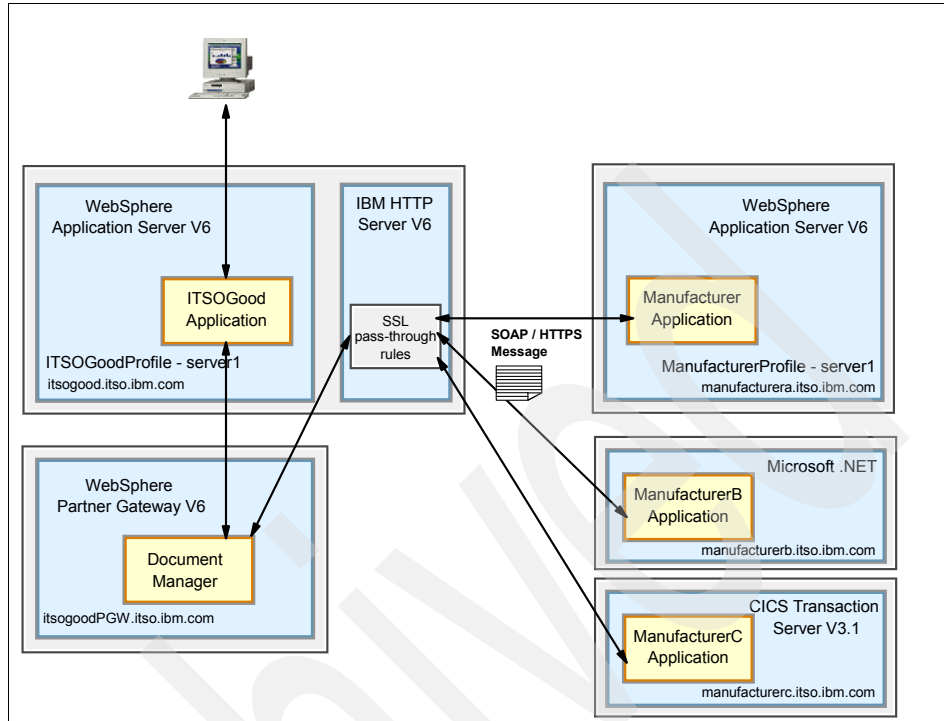


Figure 13-6 Solution topology

This section describes in detail how to configure WebSphere Partner Gateway to communicate with ManufacturerA. Optionally, you can implement the ManufacturerB and ManufacturerC applications as well. To do this you need access to a Microsoft .NET server and a CICS Transaction Server V3.1 region.

Note: You do not need to configure the ManufacturerB and ManufacturerC servers to build a working end-to-end sample application.

Instructions for configuring the ManufacturerB and ManufacturerC servers are described in:

- ▶ Appendix B, “Microsoft .NET Web services” on page 483
- ▶ Appendix C, “CICS Transaction Server Web services” on page 513

13.4.2 Creating the basic infrastructure

You need to have built the following infrastructure before you can continue with this chapter:

- ▶ Complete the steps in 11.4.2, “Creating the basic infrastructure” on page 254. This describes how to build WebSphere Application Server server profiles called ITSOGoodProfile and ManufacturerProfile, how to add a Network Deployment manager, an HTTP Server, and install the necessary enterprise applications.
- ▶ Complete the steps in 11.4.3, “Create and configure a service integration bus” on page 257 to create a configure a service integration bus and an SDO repository.
- ▶ You have installed and started a basic WebSphere Partner Gateway V6 Advanced or Enterprise Edition configuration.

13.4.3 Scenario implementation overview

In this scenario, you configure a WebSphere Partner Gateway Community Manager and a Community Participant for each manufacturer. You add an HTTP target, Gateways, and a Document Flow Definition for each manufacturer.

WebSphere Partner Gateway supports interactions between the Community Manager and Community Participants through the use of targets. In the following scenario, only the target representing a HTTP URI is setup. The HTTP target is used to enable SOAP over HTTP pass through between the Manager and Participants.

WebSphere Partner Gateway comprises three major runtime components: Receiver, Document Manager, and Community Console. In our scenario the service integration bus is configured to send SOAP documents over HTTP to the Receiver service. The Receiver stores the documents in a file system for the Document Manager to process. The **Document Manager** then polls the file system for documents to process and then routes the document to predefined destinations. The interactions in this scenario between the components are shown in Figure 13-7 on page 400.

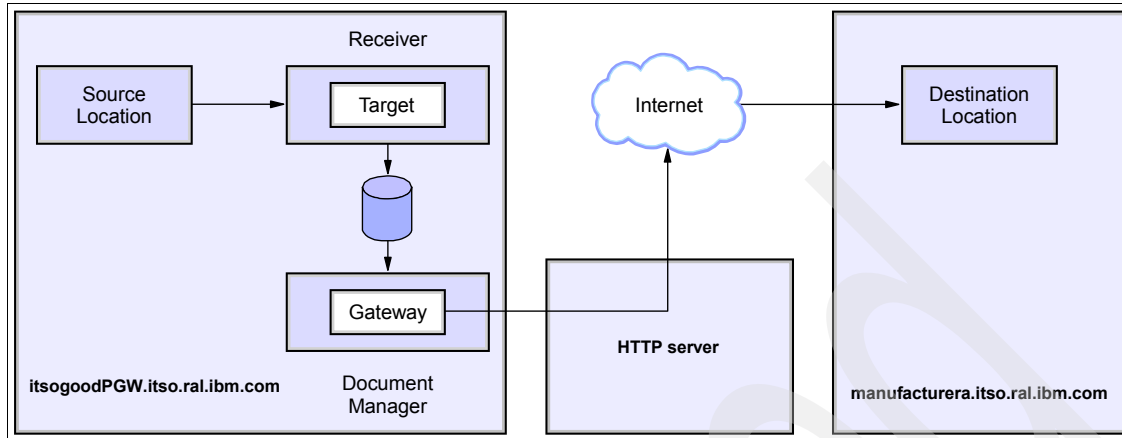


Figure 13-7 Use of WebSphere Partner Gateway in the ITSOGood sample application scenario

The Community Console provides a Web-based graphical user interface for configuring, administering and monitoring the trading community and their activities.

13.4.4 Configuring WebSphere Partner Gateway

You need to complete the following tasks to configure the WebSphere Partner Gateway V6 configuration:

- ▶ Creating the ITSOGood Community Manager profile
- ▶ Creating the Manufacturer Community Participant profile
- ▶ Configuring an HTTP target
- ▶ Setting up the HTTP gateways for the connection
- ▶ Creating document flow definitions
- ▶ Creating interactions
- ▶ Enabling B2B capabilities
- ▶ Activating participant connections
- ▶ Extracting the WSDL file for the Partner Gateway Manufacturer
- ▶ Configuring the hosts file

When you have completed these steps, you will have configured a synchronous SOAP/HTTP pass through configuration using WebSphere Partner Gateway.

Creating the ITSOGood Community Manager profile

The Community Manager is typically the company that owns the WebSphere Partner Gateway server and that uses the server to communicate with participants. The Community Manager is also considered a participant of the hub and has a profile, gateways, and B2B capabilities.

To create a Community Manager (there is only one Community Manager for each WebSphere Partner Gateway configuration), perform the following:

1. Start the Partner Gateway Console service:

```
<Gateway_installation_directory>/bin/bcgStartServer bcgconsole
```

2. Start the Partner Gateway Receiver service:

```
<Gateway_installation_directory>/bin/bcgStartServer bcgreceiver
```

3. Start the Partner Gateway Document Manager service:

```
<Gateway_installation_directory>/bin/bcgStartServer bcgdocmgr
```

4. In a Web browser, access the Community Console (Figure 13-8) using the following URL:

```
http://itsogoodPGW.itso.ra1.ibm.com:58080/console
```

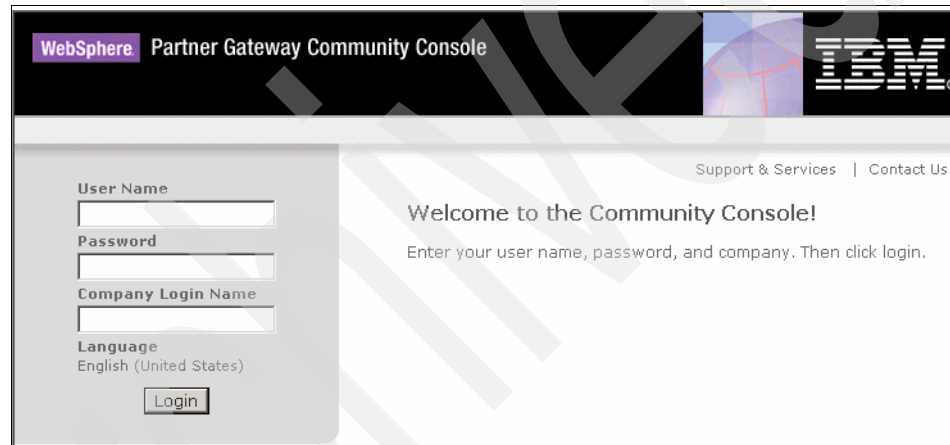


Figure 13-8 WebSphere Partner Gateway Community Console login

5. A default company called Operator is set up. One user ID, hubadmin, belongs to this company, and its expired password is Pa55word. Enter this information in the login section. Then click **Login**. If this is the first time you have logged in, you are asked to change the expired password.
6. Click **Account Admin** → **Profiles** → **Community Participant** (Figure 13-9 on page 402).



Figure 13-9 Community Participant screen

7. Click **Create**.
8. Enter the following information to create a new community manager participant).
 - a. For Company Login Name, enter SIB.
 - b. For Participant Display Name, enter ITS0Good. This is the name that appears on the Participant Search list.
 - c. From the Participant Type list, select **Community Manager**.
 - d. Set Status to **Enabled**.
 - e. Set Vendor Type to **Supplier**.
 - f. Click **New** under Business ID.
 - g. With **DUNS** pre-selected as Type, and enter the 555555551 as the nine-digit identifier. Based on this identifier, WebSphere Partner Gateway routes documents to and from the Community Manager.
 - h. Enter the host name address for the ITSOGood Community Manager by performing the following steps:
 - i. Under IP Address, click **New**.
 - ii. Specify **Production** as the Gateway Type.
 - iii. Enter itsogoodPGW as **IP address or Host Name** of the Community Manager
9. Click **Save**. Note the password assigned to the Community Manager. You will need it later for HTTP authentication.

Creating the Manufacturer Community Participant profile

The Community Participant is typically the company that has a trading relationship with other participants and has a profile, gateways, and B2B capabilities.

In this section, we create a Community Participant to represent the Manufacturer. Perform the following tasks:

1. Click **Account Admin** → **Profiles** → **Community Participant**.
2. Click **Create**.
3. For Company Login Name, enter ManufacturerA.
4. For Participant Display Name, enter ManufacturerA. This is the name that appears on the Participant Search list.
5. From the Participant Type list, select **Community Participant**.
6. Set Status to **Enabled**.
7. Set Vendor Type to **Distributor**.
8. Click **New** under Business ID.
9. Specify an Identifier of 999999992. Based on this identifier, WebSphere Partner Gateway will route SOAP messages to and from this Community Participant.
10. Enter the IP address for the Manufacturer service by performing the following steps:
 - a. Under IP Address, click **New**.
 - b. Set Gateway Type to **Production**.
 - c. Enter itsogood as the IP address or Host Name.
11. The profile should match the settings shown in Figure 13-10.

Account Admin Viewers | Tools | Hub Admin | Community Participant Simulator | System Administration

Profiles | Participant Connections | Alerts | Exclusion List

Community Participant | Gateways | B2B Capabilities | Certificates | Users | Groups | Contacts | Addresses

Language Locale: en_US | Format Local

Profile: New Participant

Company Login Name *

Participant Display Name *

Participant Type *

Status ☒ Enabled ☐ Disabled

Vendor Type

Web Site

Business ID

Type	Identifier	Remove
<input type="text" value="DUNS"/>	<input type="text" value="999999992"/>	<input type="checkbox"/>

IP Address or Host Name

Gateway Type	IP Address or Host Name	Remove
<input type="text" value="Production"/>	<input type="text" value="itsogood"/>	<input type="checkbox"/>

Figure 13-10 Manufacturer Community Participant profile

12. Click **Save**.

Optionally, create Community Participant profiles for ManufacturerB and ManufacturerC. To do this repeat the steps above, using the relevant manufacturer name. Set ManufacturerB an identifier of 888888883 and ManufacturerC an identifier of 777777773.

Configuring an HTTP target

A target is an instance of the Receiver configured for a particular deployment. Documents received at a target on the hub originate from Community Participants or from a Community Manager back-end application. In this scenario, the SOAP messages originate from the ITSOGood Community Manager.

As we will be sending SOAP messages over HTTP, we need to configure an HTTP target. We will be sending a request/response SOAP message through the HTTP target, therefore we need to configure the HTTP target to enable

synchronous HTTP requests. This will ensure our Web service client receives a response message from WebSphere Partner Gateway.

Perform the following:

1. Click **Hub Admin > Hub Configuration > Targets** to display the Targets List page.
2. From the Target List page, click **Create Target**.
3. In the Target Details section, set the Target Name to **Manufacturer HTTP Target**. The name you enter here will be displayed on the Targets list.
4. Indicate the status of the target as **Enabled**. The target is now ready to accept documents. A target that is disabled cannot accept documents.
5. Select **HTTP/S** from the Transport list.
6. In the Target Configuration section, specify the Gateway Type as **Production**. The gateway type defines the nature of the transmission.
7. Enter the URI for the HTTP/S target as **/bcgreceiver/sib**. Data sent to the Gateway server will be received at this target.
8. In the Handlers section, set Configuration Point Handlers to **syncCheck**. This indicates we would like to configure a handler for synchronous messages.
9. In the Handler Selection section, select **com.ibm.bcg.server.sync.SoapSyncHdler** and click **Add** to move it to the Configured List. This is the synchronous handler specifically for SOAP messages, and ensures the HTTP target configuration will stay open to allow a Web service client to receive a synchronous response.
10. The setting for the HTTP target should match those shown in Figure 13-11.

Target Details Welcome, Hub Adm

Target Name *

Status ☒ Enabled ☐ Disabled

Description

Transport *

Target Configuration

Gateway Type: *

URI: *

Sync Routing: *Global Http/S Transport Attributes*

Maximum Synchronous Timeout (Seconds)	300
Max Sync Sim Conn	100

Handlers

Configuration Point Handlers:

Handler Selection

Available List	Configured List
<p>Selected handler:</p> <p>com.ibm.bcg.server.sync.RnifSyncHdlr</p> <p>com.ibm.bcg.server.sync.As2SyncHdlr</p> <p>com.ibm.bcg.server.sync.XmlSyncHdlr</p> <p>com.ibm.bcg.server.sync.RnifSyncHdlr</p> <p>com.ibm.bcg.server.sync.DefaultAsynchronousSyncCheckHandler</p>	<p>Selected handler:</p> <p>com.ibm.bcg.server.sync.SoapSyncHdlr</p> <p>com.ibm.bcg.server.sync.SoapSyncHdlr</p>

Figure 13-11 HTTP target definition

11. Click **Save**.

Setting up the HTTP gateways for the connection

Next, set up an HTTP gateway so that documents can be sent to the participant's IP address. Perform the following:

1. Click **Account Admin > Profiles > Community Participant**.
2. Click **Search** without entering any search criteria to display a list of all participants.
3. Click the View details icon to display the ManufacturerA Community Participant profile.
4. Click **Gateways**.
5. Click **Create**.
6. From the Gateway List page, enter a Gateway Name of ManufacturerSOAP.
7. Enter the Description of the gateway as Gateway to Manufacturer App.

8. In the Gateway Configuration section of the page, perform the following steps:
 - a. Select **HTTP/1.1** from the Transport list.
 - b. In the Address field, enter the following as the URI where the document will be delivered:

http://itsogood.itso.ral.ibm.com/Manufacturer/services/Manufacturer

Note: This URI points to the HTTP server on the itsogood.itso.ral.ibm.com machine. The HTTP server will forward this request on to the manufacturera.itso.ral.ibm.com server and encrypt the HTTP message using HTTPS. This flow is shown in Figure 13-6 on page 398.

9. Click **Save**.
10. Set this gateway as the default:
 - a. From the Gateway List click **View Default Gateways**.
 - b. Set Production to **ManufacturerSOAP** as shown in Figure 13-12.

Gateway Type	Current Default Gateway
Production	ManufacturerSOAP
Test	No gateway selected
CPS Participant	No gateway selected
CPS Manager	No gateway selected

Figure 13-12 Setting the default gateway

- c. Click **Save**.
11. Optionally, you can create Gateways for ManufacturerB and ManufacturerC using the values in Table 13-2 on page 408.

Table 13-2 Gateway settings for ManufacturerB and ManufacturerC

Gateway Name	Address
ManufacturerBSOAP	http://itsogood.itso.ral.ibm.com/ManufacturerB/ManufacturerB.asmx
ManufacturerCSOAP	http://itsogood.itso.ral.ibm.com/ManufacturerC/ManufacturerC

12. An HTTP gateway is also required for the Community Manager:

- a. Click **Account Admin** → **Profiles** → **Community Participant**. Click **Search** and view the ITSOGood Community Manager.
- b. Click **Gateways** → **Create** and create a gateway with the following values, then click **Save**:
 - Gateway Name: ITSOGoodSOAP
 - Transport: **HTTP/1.1**
 - Address: http://

Note: The Address field is a required field, but the value will not be used at runtime, that is why http:// is specified.

- c. From the Gateway List click **View Default Gateways**, set Production to **ITSOGoodSOAP**, and click **Save**.

Creating document flow definitions

Document flow definitions specify the types of document that will be processed by WebSphere Partner Gateway. To set up the document flow definition, you either upload the WSDL (Web Service Definition Language) files that define the Web service, or you enter the equivalent document flow definitions manually through the Community Console.

Here, we setup a document flow definition by uploading a zipped archive containing multiple WSDL and XSD files. The ZIP file contains all of the WSDL and XSD file necessary to contact the Manufacturer. Perform the following:

1. Click **Hub Admin** → **Hub Configuration** → **Document Flow Definition**.
2. Click **Upload/Download Packages**.
3. Set the WSDL Package radio button to **Yes**.
4. In the File field, browse to locate the file manufacturera.zip. You can find this file in the \RouterSOA directory in the additional material supplied with this redbook. This file contains the Manufacturer_Impl.wsd1 file, and all of the WSDL and XSD files that Manufacturer_Impl.wsd1 directly and indirectly imports.

5. Set Web Service Public URL to the public URL to access this Web service through WebSphere Partner Gateway. In our case this is:
- `http://itsogoodPGW.itso.ra1.ibm.com:57080/bcgreceiver/sib?to=999999992`
6. Set Commit to database to **Yes**.
7. Click **Upload**. If the upload was successful, the Messages text box should contain text that reads: Upload successful. No committed. Data committed, as shown in Figure 13-13.

Account Admin | Viewers | Tools | **Hub Admin** | Community Participant Simulator | System

Hub Configuration | Console Configuration

Event Codes | Targets | **Document Flow Definition** | XML Formats | Actions | Fixed Workflow

Language Local

Upload/Download Packages

Provide valid xml document in 'zip' format or valid 'wsdl' file for Up

WSDL Package: Yes ☒ No ☐

File:

Web Service Public URL: (required)

Commit to database: Yes ☒ No ☐

Overwrite data: Yes ☐ No ☒

Messages:

NOTE: To enable this web service you will need to create a Gateway for this URL (the web service provider endpoint), through the management console, and set up Participant Connections with it as the destination gateway: 'http://itsogood.itso.ra1.ibm.com/Manufacturer/services/Manufacturer'

Upload successful. No warnings. Data committed.

Figure 13-13 Successful upload of WSDL

8. Optionally, repeat these steps if you wish to configure ManufacturerB and ManufacturerC using the settings in Table 13-3.

Table 13-3 Upload packages for ManufacturerB and ManufacturerC

File	Web Service Public URL
manufacturerb.java	<code>http://itsogoodPGW.itso.ra1.ibm.com:57080/bcgreceiver/sib?to=888888882</code>
manufacturerc.java	<code>http://itsogoodPGW.itso.ra1.ibm.com:57080/bcgreceiver/sib?to=777777772</code>

Creating interactions

Interactions indicate to WebSphere Partner Gateway the actions to perform on a document.

To create an interaction, perform the following steps:

1. Click **Hub Admin** → **Hub Configuration** → **Document Flow Definition**.
2. Click **Manage Interactions**.
3. Click **Create Interaction**.
4. An interaction consists of two parts: a source and a target. You must select a document flow definition from both the source and the target to create the interaction:
 - a. Under the Source section, expand **Package: None** → **Protocol: Web Service** → **Document Flow** → **Activity** and select the **Action: Purchase Order** radio button as shown in Figure 13-14.

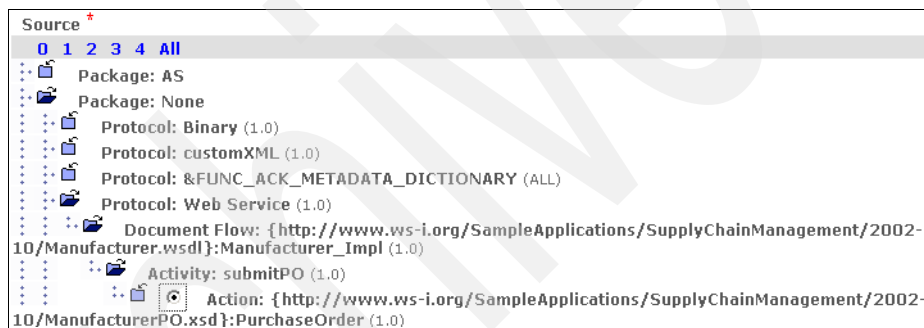


Figure 13-14 Selecting the source document flow definition interaction

- b. Select the same action in the target section (Figure 13-15).



Figure 13-15 Selecting the target document flow definition interaction

- Set Action to **Pass Through**. Note that Pass Through is the only valid option supported in WebSphere Partner Gateway for a Web service interaction.
- Click **Save**.
- Optionally, repeat this process for ManufacturerB and ManufacturerC.

Enabling B2B capabilities

In order to create a connection, the Web services B2B capabilities for the document flow definition must be enabled. Perform the following:

- Click **Account Admin** → **Profiles** → **Community Participant**.
- Click **Search** to display all participants, then **View Details for ITSOGood**.
- Click **B2B Capabilities**.
- In the B2B Capabilities window, click the **icon** under Set Source and Set Target for Package: None. This should set the Enabled row to Enabled.
- Expand **Package: None**, and keep expanding and enabling the source and target for the Manufacturer_Impl document flow, as shown in Figure 13-16.

<div> Account Admin Viewers Tools Hub Admin Community Participant Simulator System Administration </div>						
<div> Profiles Participant Connections Alerts Exclusion List </div>						
<div> Community Participant Gateways B2B Capabilities Certificates Users Groups Contacts Addresses </div>						
<div> Language Locale: en_US Format Locale: en_US Time Zone: </div>						
<div> Profile : Hub Operator : B2B Capabilities </div>						
Set Source	Set Target	Enabled	Edit	Document Flow Definition		
				0	1	2
				Package: AS		
		Enabled		Package: None		
				Protocol: Binary (1.0)		
				Protocol: customXML (1.0)		
				Protocol: &FUNC_ACK_METADATA_DICTIONARY (ALL)		
		Enabled		Protocol: Web Service (1.0)		
		Enabled		Document Flow: { http://www.ws-i.org/SampleApplicati		
		Enabled		Activity: submitPO (1.0)		
		Enabled		Action: { http://www.ws-i.org/SampleApplica		

Figure 13-16 Enabling the Web service document flow for the Community Manager

- Optionally, enable the document flows for ManufacturerB and ManufacturerC.
- Repeat these steps to enable the Manufacturer_Impl document flow for the ManufacturerA participant (and optionally for ManufacturerB and ManufacturerC).

Activating participant connections

Participant connections contain the information necessary for the proper exchange of each document flow. A document cannot be routed unless a connection exists between the Community Manager and one of its participants.

The system automatically creates connections between the Community Manager and participants based on their B2B capabilities. An administrator will have to search for these connections and then activate them.

Use the following procedure to perform a basic search for connections and then activate the connections:

1. Click **Account Admin** → **Participant Connections**. The Manage Connections page is displayed.
2. Select **ITSOGood** as the Source and select **ManufacturerA** as the target.
3. Click **Search**.
4. A connection should be shown (Figure 13-17).

The screenshot shows the 'Manage Connections' page in the Account Admin interface. The page has a navigation bar with 'Account Admin' selected, and sub-navigation links for 'Viewers', 'Tools', 'Hub Admin', 'Community Participant Simulator', and 'System Administration'. Below the navigation bar, there are tabs for 'Profiles', 'Participant Connections' (selected), 'Alerts', and 'Exclusion List'. The page title is 'Manage Connections'. On the right, there is a 'Welcome, Hub Admin' message and a language/locale/time zone selector. The main content area has a search form with 'Source' and 'Target' dropdowns. 'ITSOGood' is selected for Source and 'ManufacturerA' for Target. There are 'Search' and 'Reset' buttons. Below the search form, there is a table with columns for 'Enabled', 'B2B Capabilities', 'Connection Details', and 'B2B Capabilities'. The table contains one row with a connection details box. The details box shows the following information: Package: None (N/A), Protocol: Web Service (1.0), Document Flow: (http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl);Manufacturer_Impl (1.0), Action: submitPO (1.0), and a link to the document flow. An 'Activate' button is located next to the details box.

Figure 13-17 Connection awaiting activation

5. Click **Activate** to activate this connection. You should see the connection is now activated.
6. Select **ManufacturerA** as the source and **ITSOGood** as the target. Click **Search**, then **Activate** this connection as well.

7. Optionally, repeat this process for ManufacturerB and ManufacturerC.

Extracting the WSDL file for the Partner Gateway Manufacturer

The Manufacturer Web service should be invoked using the WebSphere Partner Gateway connection. WebSphere Partner Gateway creates a WSDL file which describes how to invoke this Web service. You need to locate and save this WSDL file so it can be used by the service integration bus when it attempts to invoke the manufacturers.

Perform the following tasks:

1. Click **Hub Admin** → **Hub Configuration** → **Document Flow Definition**.
2. Expand **Package: None** → **Protocol: Web Service** and click the **View Document Flow** action for the **Manufacturer_Impl** document flow.
3. In the Update Document Flow Definitions window, you should see the file **Manufacturer_Impl.public.wsdl**, as shown in Figure 13-18.

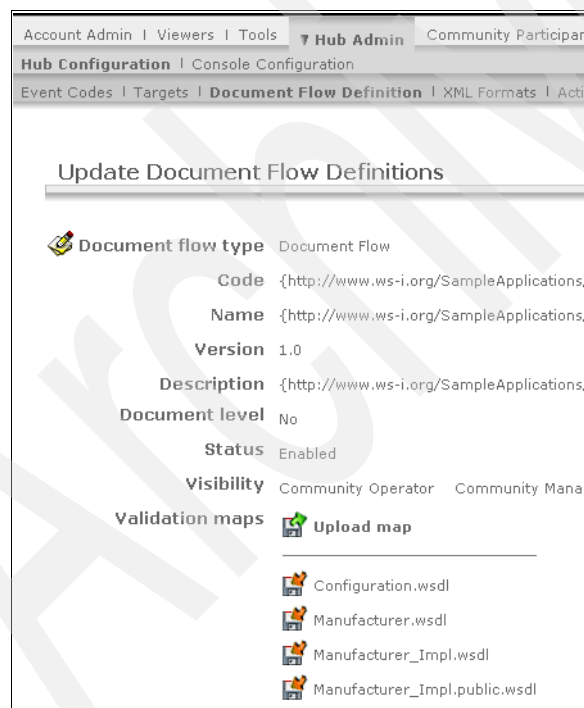


Figure 13-18 *Manufacturer_Impl.public.wsdl*

4. Click **Manufacturer_Impl.public.wsdl** to open it. Ensure the following line points to the port on which the Receiver is listening (by default it is port 57080):

```
<wsdlsoap:address
location="http://itsogoodPGW.itso.ral.ibm.com:57080/bcgreceiver/sib?to=9999
99992"/>
```

5. Save this file to the HTTP Server on the itsogood.itso.ral.ibm.com machine in the following directory:

```
<HTTP_SERVER_HOME>\htdocs\en_US\wsdl
```

6. Optionally, repeat this process for ManufacturerB and ManufacturerC.

Configuring the hosts file

Finally you need to configure the hosts file on your system. Perform the following steps:

1. Open the hosts files in a text editor. On a Windows operating system this will be located in the <WINDOWS_HOME>\system32\drivers\etc directory.

2. Add the following entry:

```
127.0.0.1 itsogoodPGW.itso.ral.ibm.com itsogoodPGW
```

3. Add a second entry that points to the IP address where the itsogood.itso.ibm.com HTTP Server is located. For example, if it were using the IP address of 1.2.3.4 you would enter:

```
1.2.3.4 itsogood.itso.ral.ibm.com itsogood
```

4. Save and close the hosts file.

13.4.5 Configuring WebSphere Application Server

You need to configure the Manufacturer outbound services on the service integration bus to send SOAP over HTTP documents to the WebSphere Partner Gateway configuration. You also need to configure the inbound services to use these new outbound services.

Creating the Manufacturer outbound service

In the administrative console of the ITSOGoodProfile server, perform the following:

1. Click **Service integration** → **Buses** → **ESBBus** → **Outbound Services**.

2. Click **New** to create a new outbound service. Define this outbound service using the WSDL exported in “Extracting the WSDL file for the Partner Gateway Manufacturer” on page 413. Use the following settings:

- WSDL location:

```
http://appsrv1a.itso.ral.ibm.com/wsdl/Manufacturer_Impl.public.wsdl
```

- Outbound service name: ManufacturerPGWService

- Service destination name: `ManufacturerPGWServiceDestination`
 - Port destination name: `ManufacturerPGWServiceDestination`
3. Optionally, repeat this process to define outbound services for `ManufacturerB` and `ManufacturerC`.
 4. Save your changes.

Reconfiguring the Manufacturer inbound service

We want to configure the `ManufacturerService` inbound service to point to this new outbound destination.

1. Click **Service integration** → **Buses** → **ESBBus** → **Inbound Services**.
2. Click **ManufacturerInboundService**.
3. Set the Service destination name to **ManufacturerPGWServiceDestination**.
4. Click **OK**.
5. Optionally, repeat this process to define outbound services for `ManufacturerB` and `ManufacturerC`.
6. Save your changes.

Configuring the hosts file

You need to configure the hosts file on your system to point to the WebSphere Partner Gateway machine. Perform the following:

1. Open the hosts files in a text editor. On a Windows operating system this will be located in the `<WINDOWS_HOME>\system32\drivers\etc` directory.
2. Add an entry that points to the IP address of the machine where the WebSphere Partner Gateway configuration is running. For example if it were using the IP address of 1.2.3.4 you would enter:


```
1.2.3.4 itsogoodPGW.itso.ral.ibm.com itsogoodPGW
```
3. Save and close the hosts file.

Defining HTTP basic authentication

When you invoke the WebSphere Partner Gateway Web service, you need to pass an HTTP basic authentication token for user `SIB`. To do this, perform the following:

1. In the WebSphere Application Server administrative console click **Applications** → **Enterprise Applications** → **ITSOGood** → **EJB Modules** → **WarehouseEJB.jar** → **Web services: Client security bindings**.

2. Click the **Edit** button in the HTTP basic authentication column for the Web service called service/ManufacturerService.
3. Enter a Basic authentication ID of SIB and enter the password assigned to the ITSOGood Community Manager in “Creating the ITSOGood Community Manager profile” on page 400.
4. Click **OK**.
5. Optionally, repeat this process for ManufacturerB and ManufacturerC.

13.4.6 Testing the WebSphere Partner Gateway configuration

This section describes how to test the scenario. For more information about the sample application, see Chapter 8, “Business scenario used in this book” on page 137. Perform the following steps:

1. Enter the following URL in a Web browser to start the ITSOGood sample application:

`http://itsogood.itso.ral.ibm.com:9080/SCMSampleUI/`

This assumes the unsecured HTTP transport port number is 9080. If this is not the case, use the URL above with the appropriate HTTP transport port number.

2. Order **6** items of product 605001 to trigger the Warehouse to contact ManufacturerA to replenish its stock for this product. As a result, it will invoke the WebSphere Partner Gateway configuration.
3. To confirm the successful invocation of the Manufacturer service, check the SystemOut.log of the ITSOGoodProfile application server. The Warehouse writes a message saying that a Manufacturer was invoked. The message should look like this:

Warehouse: Response from Manufacturer --> Manufacturer_A has received and processed a request.

You should also see the following message in the SystemOut.log file of the ManufacturerProfile application server:

Manufacturer A: Processing Purchase Order

4. To confirm the SOAP/HTTP request and response documents flowed through the WebSphere Partner Gateway configuration, perform the following:
 - a. Log on to the WebSphere Partner Gateway Community Console and click **Viewers → Document Viewers**.
 - b. Click **Search** to search for new documents.
 - c. You should see both the request and response documents used to invoke the Manufacturer Web service (Figure 13-19 on page 417).











Participants		Time Stamps	Protocol/Document Flow		Gateway Type	Synchronous	Status
Document ID: -							
Doc Time Stamp: -							
	<input type="checkbox"/> Source: ManufacturerA	In: 12/11/05 2:13:14 PM (0.644 kb)		Web Service (1.0) {http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl}:Manufacturer_Impl: {http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl}:Manufacturer_Impl (1.0)	Production		
	<input type="checkbox"/> Target: ITSGood	Out: 12/11/05 2:13:25 PM (0.5 kb)		Web Service (1.0) {http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl}:Manufacturer_Impl: {http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl}:Manufacturer_Impl (1.0)			
Document ID: -							
Doc Time Stamp: -							
	<input type="checkbox"/> Source: ITSGood	In: 12/11/05 2:13:04 PM (1.508 kb)		Web Service (1.0) {http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl}:Manufacturer_Impl: {http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl}:Manufacturer_Impl (1.0)	Production		
	<input type="checkbox"/> Target: ManufacturerA	Out: 12/11/05 2:13:14 PM (1.398 kb)		Web Service (1.0) {http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl}:Manufacturer_Impl: {http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl}:Manufacturer_Impl (1.0)			

Figure 13-19 Successful invocation and response from the Manufacturer Web service

- Optionally test ManufacturerB by ordering 6 items of product 605002, and ManufacturerC by ordering 6 items of product 605003.

Archived

Exposed Serial Process runtime pattern: generic profile

In this chapter, we discuss the Exposed Serial Process runtime pattern using the generic profile. This chapter describes how to build a WS-BPEL process conforming to this pattern using WebSphere Studio Application Developer Integration Edition.

The Warehouse service from the ITSO Good sample application is implemented as a WS-BPEL process in this chapter. The WS-BPEL process runs in WebSphere Business Integration Server Foundation. The business process must accept Web service calls from the Retailer, and be able to invoke the LoggingFacility and Manufacturer Web services.

14.1 Business scenario

The business scenario implemented in this chapter builds on the business scenario discussed in Chapter 12, “Exposed Broker runtime pattern: generic profile” on page 339.

In the generic broker previous scenario, we can see how externalizing the routing and distribution rules provided ITSO Good with the flexibility required to scale up their application services.

ITSO Good now wants to focus on improving the operational effectiveness of their business through continuous improvement. In particular, they require their business processes to be flexible and responsive to the changes in the market and be able to take advantage of the increased capabilities offered by their partners.

To support these new business goals, ITSO Good have the following additional business requirements:

- ▶ The business processes needs to be flexible and responsive to the changes in the market and be able to take advantage of the increased capabilities offered by their partners.
- ▶ Build the capabilities required to monitor and measure the effectiveness of business processes.

14.2 Design guidelines

In this section, we analyze the business requirements and apply the Patterns for e-business to determine the appropriate Runtime pattern for the solution. We then discuss the various design options available to us for implementing the solution and also look at the product mappings.

14.2.1 Analyze business requirements

The given business scenario requires the need to externalize the process execution logic from the individual application services. This allows us to use a process manager to automate the coordination of business process flow between the Warehouse and the Manufacturing partners of ITSO Good.

As the business process in the given scenario goes across organizational boundaries, the following additional system requirements need to be addressed:

- ▶ Interoperability standards: Standards should be used where possible to minimize future changes required to the source and target applications.

- **Security:** Security is a primary concern when opening business processes to external organizations. As a result, the solution should include robust security mechanisms to protect enterprise resources.

14.2.2 Selecting a pattern

We use the Patterns for e-business to determine the appropriate Runtime pattern to apply to this scenario.

Described here is a step-by-step approach used to navigate the Patterns for e-business asset catalog:

1. Business pattern

We select the Extended Enterprise business pattern because the given scenario requires interactions between the business processes in the Warehouse and Manufacturer systems that reside in separate enterprises.

2. Application pattern

Because the source application (Warehouse) initiates an interaction that is to be distributed to multiple target partner applications in a serial manner, we choose the Exposed Serial Process application pattern.

3. Runtime pattern

Selecting the Application pattern provides us with the possible Runtime patterns for the proposed solution. Because the business requirement does not mandate an ESB infrastructure, we select the *generic profile* of the Exposed Serial Process application pattern.

Figure 14-1 on page 422 shows the level 0 decomposition of the generic profile of the Exposed Serial Process runtime pattern, mapped on to the Exposed Serial Process application pattern.

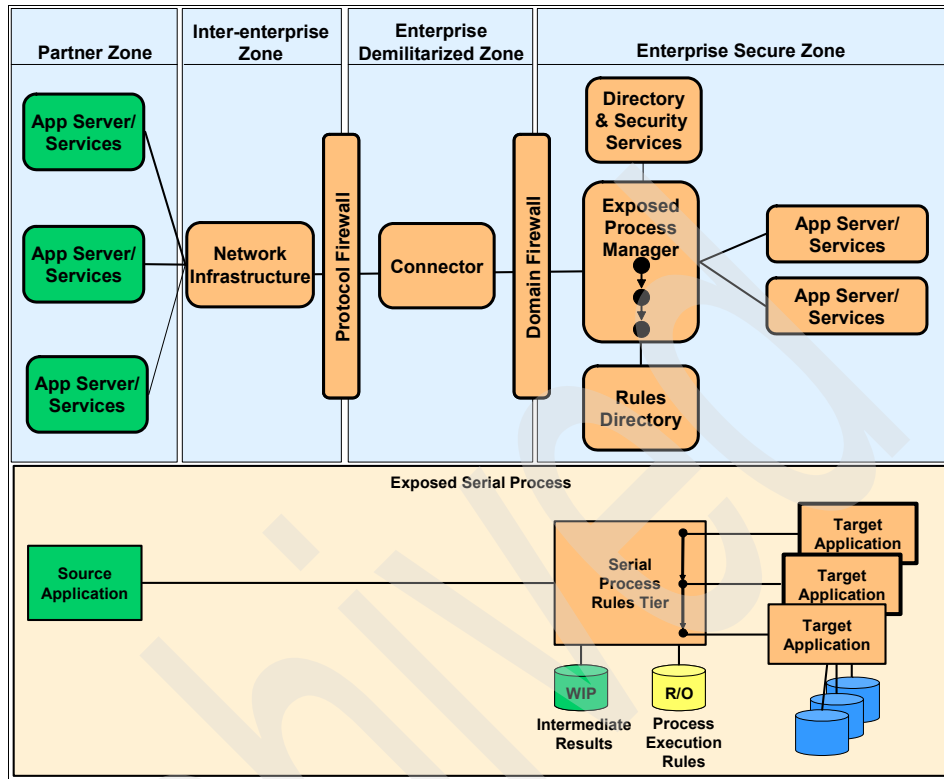


Figure 14-1 Exposed Serial Process::Runtime pattern = generic profile

14.2.3 Analyze design options

This section discusses the architectural decisions that we made and their implementation options for the given business scenario using the Exposed Serial Process runtime pattern.

Most of the design decisions made in the previous chapters, also apply to the Exposed Serial Process interactions:

- ▶ “Architectural decision: integration options” on page 163 for the use of Web services to integrate with the external partner services.
- ▶ The design guidelines in the “Securing Web services” on page 165.
- ▶ The architectural decision related to securing the Web services in “Architectural decision: Securing the Web service interaction” on page 170

There are also additional design decisions that are specific to this scenario, which are discussed in the following sections.

Implementing an Exposed Serial Process

Some of the common capabilities that you find in business process engine implementations are:

- ▶ Service composition: Composing application services to create workflows
- ▶ Process State Management: Interruptible (long-running) processes that their state and status to be persisted between activities
- ▶ Transactional behavior and compensation
- ▶ Support for human interaction, business processes that require human or other manual intervention to complete

Some of the other aspects of designing business process that need to be considered in an extended enterprise scenario are security and interoperability.

Web services are a common standards-based approach to integrate with applications and services of partners that reside outside the enterprise. And WS-BPEL (Business Process Execution Language for Web Services) is fast emerging as an industry standard for choreographing services implemented using Web services together to implement a business process.

WS-BPEL

WS-BPEL is a standard that is used to define business process models by enabling the description of Web services operations, their relationships, and order of execution. By doing so, it extends the Web services interaction model and enables it to support business transactions. WS-BPEL defines an interoperable integration model that can facilitate the expansion of automated process integration in both intra- and interenterprise scenarios.

For the given business scenario, WS-BPEL, in combination with the message level security features of WS-security, provides an ideal combination to implement the serial order process that is initiated at the Warehouse and which involves interaction with external services provided by the partner Manufacturer systems.

For more information about process choreography and WS-BPEL (formerly BPEL4WS), refer to the following resources:

- ▶ IBM Redbook *BPEL4WS Business Processes with WebSphere Business Integration*, SG24-6381.
- ▶ IBM Redbook *WebSphere Business Integration Server Foundation V5.1 Handbook*, SG24-6318.
- ▶ Business Process Execution Language for Web Services V1.1

<http://www.ibm.com/developerworks/library/specification/ws-bpel/>

14.2.4 Products

In this section we look at the products available to implement the various components in the Exposed Broker runtime pattern.

Product implementation options

Product choices for this scenario are based on:

- ▶ Design decisions that we made in 14.2.3, “Analyze design options” on page 422
- ▶ Extended Enterprise capabilities of the products
- ▶ Products that are currently available

Exposed Process Manager component

We can use the following currently available products to implement the Exposed Process Manager component in the given scenario:

- ▶ WebSphere MQ Workflow
- ▶ WebSphere Business Integration Server Foundation
- ▶ WebSphere Process Server

For this scenario, WebSphere Business Integration Server Foundation V5.1 meets all of the requirements and, therefore, is the product of choice.

The complete product mapping for this scenario is shown in Figure 14-2.

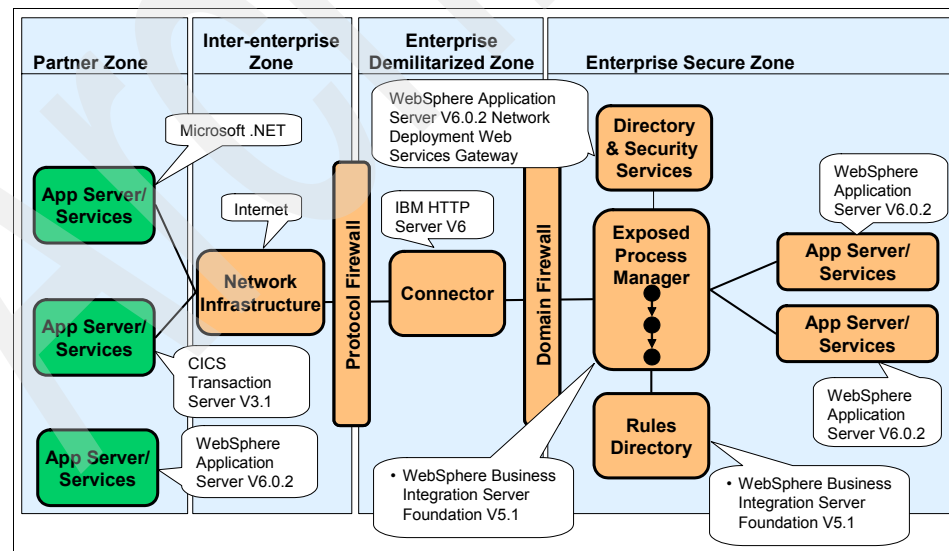


Figure 14-2 Exposed Serial Process: generic profile product mapping

This Product mapping uses WebSphere Application Server V6.0.2 to host the Application Services in the Enterprise Secure Zone.

An application service invokes the automated process instance implemented by the Process Manager node using the Web Services Invocation Framework (WSIF). The Exposed Process Manager implemented using WebSphere Business Integration Server Foundation V5.1 invokes the services provided by the external partner application service.

The Rules Directory node implemented using WebSphere Business Integration Server Foundation V5.1 identifies which external partner organization application service to invoke.

In the Directory and Security services node, the service integration bus within WebSphere Application Server Network Deployment V6.0.2 is configured secure all transactions to the external Partner Zone to use WS-Security integrity and confidentiality.

The IBM HTTP Server V6 acts as an Adapter Connector by providing protocol transformation to SOAP/HTTPS thus delivering transport level security between the enterprise and the partner organizations.

14.3 Development guidelines

This section describes how to implement a serial process (WS-BPEL flow) using Process Choreographer in WebSphere Studio Application Developer Integration Edition.

Note: This section requires the use of WebSphere Studio Application Developer Integration Edition V5.1.1 plus Cumulative Fix 010 or higher installed.

14.3.1 Scenario implementation: Serial process interaction

We use the sample scenario implemented in Chapter 10, “Exposed Direct Connection runtime pattern: generic profile” on page 157 as the starting point for this chapter. We replace the Warehouse Web module with a WS-BPEL process. None of the other applications or modules require any code change.

We use the Process Choreographer tool in WebSphere Studio Application Developer Integration Edition to create and generate deployment code for a business process.

As a result of the limitations in the Business Process Engine (BPE) and WebSphere Studio Application Developer Integration Edition, the WSDL files provided with the WS-I sample application do not work. Therefore, we create Web services clients for Manufacturer services using the WS-I sample WSDL and this service invokes the business process. Our Warehouse business process is not be exposed directly to the Retailer application.

Because there are no WS-Security capabilities in Process Choreographer or WebSphere Business Integration Server Foundation, we invoke the Manufacturer services through Web services gateway, which is responsible for securing the communication to the Manufacturer.

The scenario implementation is shown in Figure 14-3.

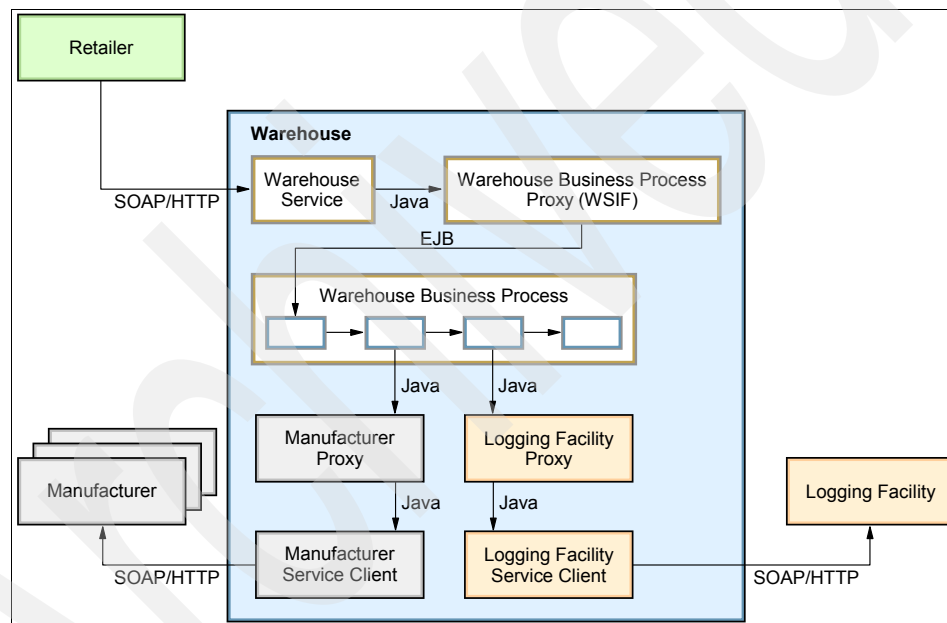


Figure 14-3 Scenario implementation for the Exposed Serial Process

The following components are developed in this chapter and are shown in Figure 14-3:

► Warehouse Service

This component acts as an endpoint for the Retailer application and other clients invoking the Warehouse application. In our scenario, this component is implemented as a J2EE Web module and exposed as a HTTP/SOAP Web service.

► Warehouse Business Process

This WS-BPEL process is a short running process (micro flow) and contains the Warehouse application business logic.

- ▶ Warehouse Business Process Proxy

This Web Services Invocation Framework (WSIF) Java class is the proxy used by the WarehouseService to invoke the Warehouse Business Process.

- ▶ Manufacturer Proxy and Logging Facility Proxy

These are Java classes exposed as services and are invoked from the Warehouse Business Process when calling the Manufacturer and LoggingFacility.

- ▶ ManufacturerGW Service Client and Logging Facility Service Client

Web services client code is generated from the WSDLs supplied from the three Manufacturer's and LoggingFacility. In our scenario, these Web services clients are created in Java projects.

14.3.2 Creating the basic infrastructure

Before beginning the development guidelines, the basic infrastructure must be configured before you can begin building the Warehouse business process.

This scenario is based on the infrastructure built in Chapter 10, "Exposed Direct Connection runtime pattern: generic profile" on page 157. If you have already built this infrastructure, you can reuse it here.

This section describes how to build a Warehouse business process to fit into the topology shown in Figure 14-4 on page 428.

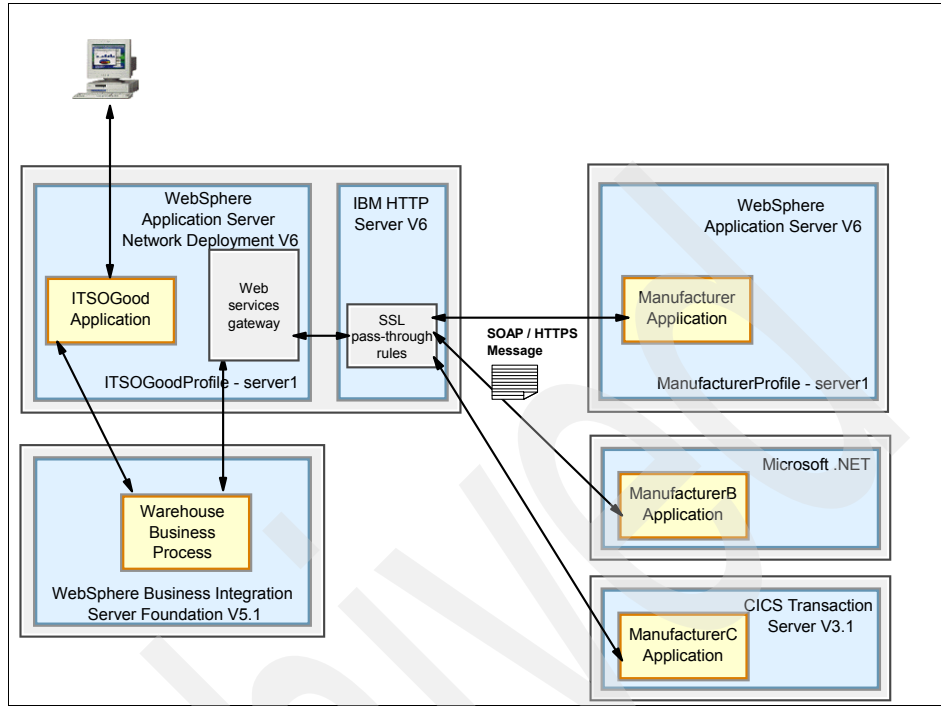


Figure 14-4 Solution topology

Before building the Warehouse business process, complete the following steps to create this infrastructure.

1. Configure the WebSphere Application Server Network Deployment and HTTP Server infrastructure as described in 11.4.2, “Creating the basic infrastructure” on page 254. This creates the following:
 - WebSphere Application Server Network Deployment profiles for ITSOGoodProfile and ManufacturerProfile
 - An HTTP server configured for hosting WSDL and SSL pass-through
 - ITSOGoodProfile added to the deployment manager to enable the Web services gateway feature
 - Non WS-Security versions of the ITSOGood and Manufacturer enterprise applications
2. Configure the service integration bus as described in 11.4.3, “Create and configure a service integration bus” on page 257. Complete only the following steps:
 - “Installing the Service Data Objects (SDO) repository” on page 260
 - “Configuring the SDO repository” on page 260

- “Installing the Web services applications” on page 267
- 3. Configure the Web services gateway infrastructure as described in 11.4.4, “Create and configure the Web service gateway” on page 281.

14.3.3 Configuring WebSphere Studio

Perform the following to configure WebSphere Studio Application Developer Integration Edition.

1. Start WebSphere Studio Application Developer Integration Edition, and select **Window → Preferences**.
2. Click **J2EE**. Under Server Targeting Support select **Enable server targeting support**.
3. On the left side pane, expand **Java** and select **Installed JREs**. Select **WebSphere v5.1 EE JRE** and click **OK**.

14.3.4 Creating Manufacturer and LoggingFacility Web services clients

This section describes how to create Web services clients for the Logging Facility and the three Manufacturers.

Web services clients can be created in Web, EJB or Java projects. In our scenario we create the clients in Java projects.

Creating a new Java project

Follow these steps, to create Java projects from the service clients:

1. In WebSphere Studio Application Developer Integration Edition create a Java project, select **File → New → Project**.
2. From the next window, select **Java** from the left side pane and then select **Next**.
3. Enter the project name `LoggingFacilityServiceClient`.
4. Click **Finish**.
5. Repeat the above steps to create another project with the name `ManufacturerGWServiceClient`.

Importing WSDL files

In order to create the service clients, you must import the XSD and WSDL files used by the LoggingFacility and Manufacturer into WebSphere Studio.

1. Locate the following WSDL and XSD files hosted on the ITSOGood HTTP Server and copy them to the machine where you are running WebSphere Studio Application Developer Integration Edition:

- Configuration.wsdl
- LoggingFacility.wsdl
- LoggingFacility_Impl.wsdl
- LoggingFacility.xsd
- Configuration.xsd
- envelope.xsd
- ManufacturerPO.xsd
- ManufacturerSN.xsd
- ExposedESBGatewayBus.ManufacturerGatewayBinding.wsdl
- ExposedESBGatewayBus.ManufacturerGatewayPortType.wsdl
- ExposedESBGatewayBus.ManufacturerGatewayServiceService.wsdl
- ExposedESBGatewayBus.ManufacturerBGatewayBinding.wsdl
- ExposedESBGatewayBus.ManufacturerBGatewayPortType.wsdl
- ExposedESBGatewayBus.ManufacturerBGatewayServiceService.wsdl
- ExposedESBGatewayBus.ManufacturerCGatewayBinding.wsdl
- ExposedESBGatewayBus.ManufacturerCGatewayPortType.wsdl
- ExposedESBGatewayBus.ManufacturerCGatewayServiceService.wsdl

Tip: <HTTP_Server_home>\htdocs\en_US\wsdl will be the location of directory where these WSDL files are hosted.

2. In WebSphere Studio Application Developer Integration Edition, switch to Package Explorer view in the Business Integration perspective.
3. Right-click the **LoggingFacilityServiceClient** project and select **Import**.
4. Select **File System** and click **Next**.
5. In the From Directory field, enter the path where you saved the WSDL and XSD files in the previous steps. In the left-side pane below the From Directory field you can see the folder. Select the folder and the right-side pane displays all the WSDL and XSD files. Select the following files:
 - Configuration.wsdl
 - Configuration.xsd
 - envelope.xsd
 - LoggingFacility_Impl.wsdl
 - LoggingFacility.wsdl
 - LoggingFacility.xsd
6. In the Into Folder field, enter LoggingFacilityServiceClient\wsdl.
7. Click the **Finish** button.

8. Repeat these steps to import the following files into the folder **ManufacturerGWServiceClient\wsdl**.
 - Configuration.wsdl
 - Configuration.xsd
 - envelope.xsd
 - ExposedESBGatewayBus.ManufacturerGatewayBinding.wsdl
 - ExposedESBGatewayBus.ManufacturerGatewayPortType.wsdl
 - ExposedESBGatewayBus.ManufacturerGatewayServiceService.wsdl
 - ExposedESBGatewayBus.ManufacturerBGatewayBinding.wsdl
 - ExposedESBGatewayBus.ManufacturerBGatewayPortType.wsdl
 - ExposedESBGatewayBus.ManufacturerBGatewayServiceService.wsdl
 - ExposedESBGatewayBus.ManufacturerCGatewayBinding.wsdl
 - ExposedESBGatewayBus.ManufacturerCGatewayPortType.wsdl
 - ExposedESBGatewayBus.ManufacturerCGatewayServiceService.wsdl
 - ManufacturerPO.xsd
 - ManufacturerSN.xsd
9. You can see some compilation errors reported in some of the WSDL files. This is because the import statements in some of these files refer to the HTTP server which cannot be accessed from your WebSphere Studio Application Developer Integration Edition workspace at development time. Change these WSDL files to import the locally stored WSDL files.

To fix these compilation errors, open each of the WSDL file reporting errors and change import path for WSDL. Change the text:

```
location = "http://appsrv1a.itso.ra1.ibm.com/wsdl/<wsdl file>"
```

Replace it with this line:

```
location="<wsdl file>"
```

This should fix all the compilation errors in the workspace.

10. The **LoggingFacility_Impl.wsdl** file needs to point to the LoggingFacility Web service running in the ITSOGoodProfile server. To configure this, perform the following steps:

- a. Open **LoggingFacility_Impl.wsdl** located in the folder **LoggingFacilityServiceClient/wsdl** in the WSDL editor and switch to the Source view.
- b. Change the address location to use **itsogood.itso.ra1.ibm.com**, then save the change:

```
location="http://itsogood.itso.ra1.ibm.com:9080/LoggingFacility/services/LoggingFacility"
```

- c. Add an entry to your hosts file (on a Windows system this is located at `<WINDOWS_HOME>\system32\drivers\etc\hosts`) to point the host name **itsogood.itso.ra1.ibm.com** to the IP address where the ITSOGoodProfile

server is running. For example if it were running at the IP address 1.2.3.4 you would enter:

1.2.3.4 itsogood.itso.ral.ibm.com

- d. This addition to the hosts file also allows the Manufacturer Web service to be found because this Web service contains an address location that also points to itsogood.itso.ral.ibm.com.

Generate Web services client

Generate Web service clients for the LoggingFacility and Manufacturers as follows:

1. In WebSphere Studio Application Developer Integration Edition, right-click **LoggingFacility_Impl.wsdl** in the LoggingFacilityServiceClient project, and select **Web Services** → **Generate Client** as shown in Figure 14-5.

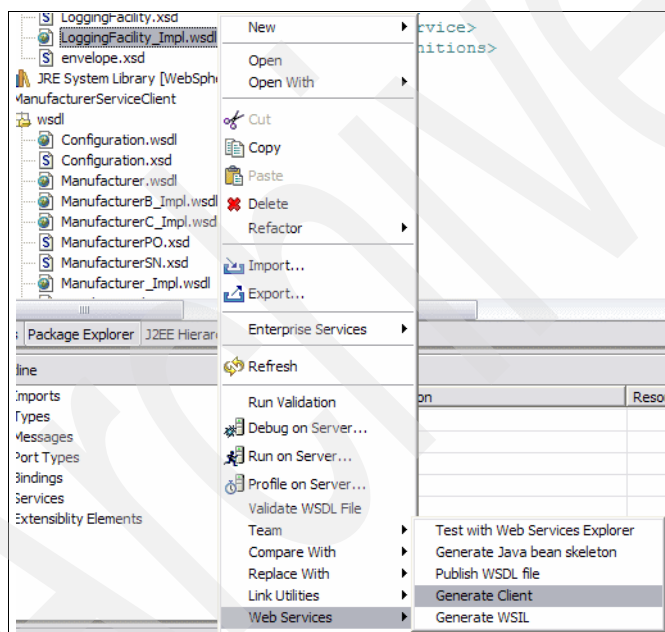


Figure 14-5 Generating Web services client

2. On the WSDL to Java Bean Proxy window, accept all the defaults and click **Next**.
3. The Web services client can be created in a Java, Web, or EJB module. In our scenario we create a Java client (Figure 14-6 on page 433).
 - a. In the Client-Side Environment Selection section, select **Choose server first**.

- b. In the Server panel, expand **Server Types** → **WebSphere version 5.1** and select **Integration Server v5.1**.
- c. In the Web service runtime panel, select **IBM WebSphere V5**.
- d. Select Client Type **Java** and Client Project as **LoggingFacilityClientService**.
- e. Click **Next**.

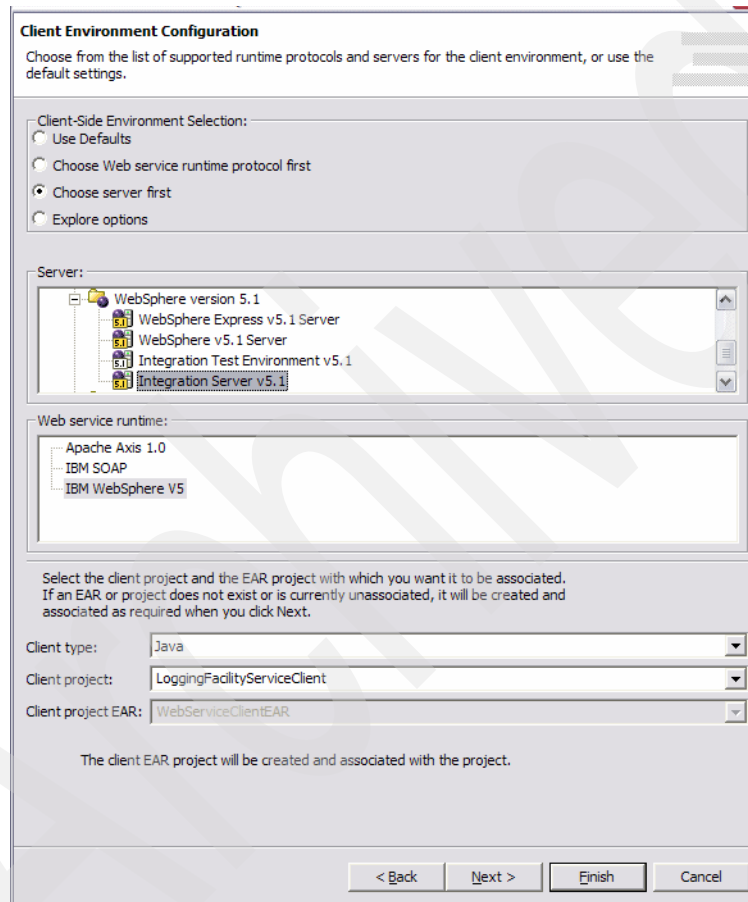


Figure 14-6 Web services client environment configuration

4. On Web Service Selection Page, confirm that the WSDL URI is set to `/LoggingFacilityServiceClient/wsd/LoggingFacility_Impl.wsd` and click **Next**.
5. On Web Service Proxy Page window, accept all defaults and click **Finish**. When the code generation is finished, you can view the generated class

LoggingFacilityServiceLocator in package org.ws_i.www. This class invokes the LoggingFacility service.

6. Repeat the previous steps to create Web services clients for the Manufacturer as shown in Table 14-1, ManufacturerB using Table 14-2, and ManufacturerC using Table 14-3. Ignore any compilation errors.

Table 14-1 Web service client setting for Manufacturer

Field	Value
WSDL file	ExposedESBGatewayBus.ManufacturerGatewayServiceService.wsdl
Client type	Java
Client Project	ManufacturerGWServiceClient
WSDL URI	\\ManufacturerGWServiceClient\wsdl\ExposedESBGatewayBus.ManufacturerCGatewayServiceService.wsdl

Table 14-2 Web service client setting for ManufacturerB

Field	Value
WSDL file	ExposedESBGatewayBus.ManufacturerBGatewayServiceService.wsdl
Client type	Java
Client Project	ManufacturerGWServiceClient
WSDL URI	\\ManufacturerGWServiceClient\wsdl\ExposedESBGatewayBus.ManufacturerCGatewayServiceService.wsdl

Table 14-3 Web service client setting for ManufacturerC

Field	Value
WSDL file	ExposedESBGatewayBus.ManufacturerCGatewayServiceService.wsdl
Client type	Java
Client Project	ManufacturerGWServiceClient
WSDL URI	\\ManufacturerGWServiceClient\wsdl\ExposedESBGatewayBus.ManufacturerCGatewayServiceService.wsdl

If a warning message is displayed as shown in Figure 14-7, then select **Yes to All**. This message is displayed because some of the files being generated already exist in your workspace.

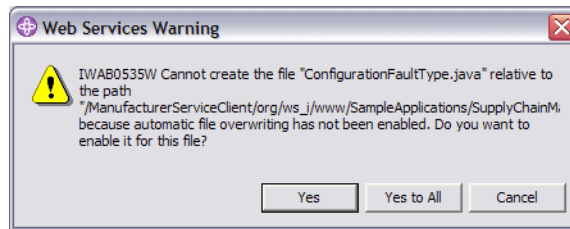


Figure 14-7 Web services warning

7. You might see some compilation errors in the generated code. To fix these errors, delete **ManufacturerPortTypeProxy.java** in the `org.ws_i.www` package because we do not use this class. There should now be no errors or warnings in the Tasks view.

14.3.5 Create Java proxy classes

In this section we create Java proxy classes using modified WSDL files that are compatible with the WS-BPEL process. Our business process invokes these Java classes which perform data mapping and call the Web service clients created in the previous section. These proxy classes are created in the same project as our business process.

Create a new service project

1. In WebSphere Studio Application Developer Integration Edition to create a service project, switch to **Business Integration** perspective and select **File → New → Service Project**.
2. Enter the project name `WarehouseBusinessProcess` and click **Finish**.

Import WSDL and XSD files

Import the WSDL and XSD files required for the proxy classes.

1. Create a new package `com.ibm.itso.wsdl` in the `WarehouseBusinessProcess` project.
2. In WebSphere Studio Application Developer Integration Edition, import the below listed WSDL and XSD files under package `com.ibm.itso.wsdl` in the `WarehouseBusinessProcess` project. These files can be found in the additional materials supplied with this rebook in the `SerialGeneric\supportFiles` directory:

- Configuration.wsdl
- Configuration.xsd
- envelope.xsd
- LoggingFacility.wsdl
- LoggingFacility.xsd
- Manufacturer.wsdl
- ManufacturerPO.xsd

Note: The WSDL and XSD files imported in this step have been modified to use simple data types which are compatible with WebSphere Studio Application Developer Integration Edition.

Create proxy classes

In this section we generate and modify Java classes from the Manufacturer and LoggingFacility WSDL files.

1. Right-click the **Manufacturer.wsdl** in package `com.ibm.itso.wsdl` and select **New** → **Build from Service**.
2. In the Create Service window ensure **Java Service Skeleton** is selected then click **Next**.
3. In the Service Skeleton window accept the defaults to create a new port and binding and click **Next**.
4. In the next window, change the package name under port location to `com.ibm.itso.Manufacturer`, accept other default entries as shown in Figure 14-8 on page 437 and click **Next**.

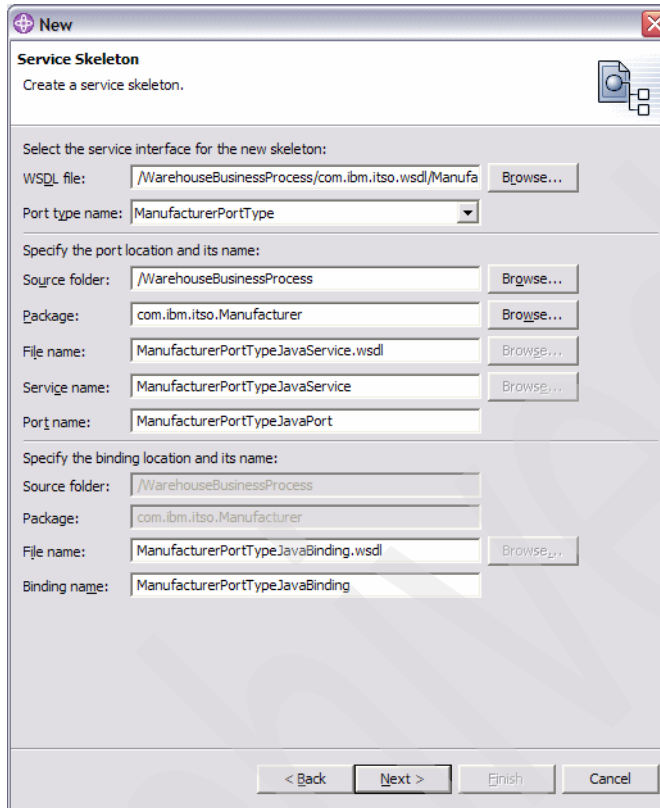


Figure 14-8 Service skeleton window

5. In the Java Skeleton window, confirm that the class name is **ManufacturerPortType.java**, accept the default entries and click **Finish**.
6. Review the Java class and WSDL files generated in the `com.ibm.itso.Manufacturer` package.
7. Repeat these steps to create a proxy for `LoggingFacility` as described in Table 14-4, for `ManufacturerB` as shown in Table 14-5 on page 438, and for `ManufacturerC` as shown in Table 14-6 on page 438.

Table 14-4 Settings for `LoggingFacility`

Field	Value
WSDL file	LoggingFacility.wsdl
Port Location - Package	com.ibm.itso.LoggingFacility

Field	Value
Port Location - File name	LoggingFacilityLogPortTypeJavaService.wsdl
Port Location - Service name	LoggingFacilityLogPortTypeJavaService
Port Location - Port name	LoggingFacilityLogPortTypeJavaPort
Binding Location - File name	LoggingFacilityLogPortTypeJavaBinding.wsdl
Binding Location - Binding name	LoggingFacilityLogPortTypeJavaBinding
Class name	LoggingFacilityLogPortType.java

Table 14-5 Settings for ManufacturerB

Field	Value
WSDL file	Manufacturer.wsdl
Port Location - Package	com.ibm.itso.ManufacturerB
Port Location - File name	ManufacturerBPortTypeJavaService.wsdl
Port Location - Service name	ManufacturerBPortTypeJavaService
Port Location - Port name	ManufacturerBPortTypeJavaPort
Binding Location - File name	ManufacturerBPortTypeJavaBinding.wsdl
Binding Location - Binding name	ManufacturerBPortTypeJavaBinding
Class name	ManufacturerBPortType.java

Table 14-6 Settings for ManufacturerC

Field	Value
WSDL file	Manufacturer.wsdl
Port Location - Package	com.ibm.itso.ManufacturerC
Port Location - File name	ManufacturerCPortTypeJavaService.wsdl
Port Location - Service name	ManufacturerCPortTypeJavaService
Port Location - Port name	ManufacturerCPortTypeJavaPort
Binding Location - File name	ManufacturerCPortTypeJavaBinding.wsdl
Binding Location - Binding name	ManufacturerCPortTypeJavaBinding

Field	Value
Class name	ManufacturerCPortType.java

8. To invoke the Web services clients, you have to change the build path for the WarehouseBusinessProcess project to include LoggingFacilityServiceClient and ManufacturerServiceClient.
 - a. Right-click **WarehouseBusinessProcess** project and select **Properties**.
 - b. In the Properties for WarehouseBusinessProcess window, select **Java Build Path** from the left hand pane. Click the **Projects** tab, check the **LoggingFacilityServiceClient** and **ManufacturerGWServiceClient** projects, and click **OK**.
9. Now we need to add code in these proxy classes to map data and invoke the Web services clients created in section “Generate Web services client” on page 432.
 - a. Open **LoggingFacilityLogPortType.java** in the Java editor and notice that the logEvent() method is a stub; it does not contain any code.
 - b. We have supplied a completed logEvent() method in the additional material supplied with this redbook. Open SerialGeneric\supportFiles\LoggingFacilityLogPortType.java and paste the contents of this file into the LoggingFacilityLogPortType.java file open in the Java editor. The logEvent () method is now populated.
 - c. Save the file.

Note: You can ignore any compilation errors regarding the ServiceException class not found. Ignore them for now. We fix these in the next section when a business process is created.

- d. Repeat this process to replace the contents of ManufacturerPortType.java with the content from SerialGeneric\supportFiles\ManufacturerPortType.java. Complete this for ManufacturerB and ManufacturerC as well. Again, ignore the errors.

14.3.6 Create a business process using Process Choreographer

In this section we create a new WS-BPEL business process to perform the Warehouse function. The Warehouse process contains the following characteristics:

- It is a short running process.

- ▶ It is initialized when a Web service request is received from the Retailer requesting the Web service operation shipGoodss
- ▶ The process has decision logic to create requests for different Manufacturers.
- ▶ After sending successful requests to the Manufacturers, the process sends a request to the Logging Facility.
- ▶ The process terminates after successfully replying back to the shipGoods request.

This section details a step-by-step procedure for creating a WS-BPEL process and links it with different partners. It assumes that you have knowledge of WS-BPEL concepts and development guidelines. For more information about these topics, see *WebSphere Business Integration Server Foundation V5.1 Handbook*, SG24-6318.

Create the Warehouse business process

The following steps describe how to build the Warehouse business process.

1. In the Business Integration perspective, right-click the **WarehouseBusinessProcess** project and select **New** → **Business Process**.
2. In the Business process window, enter the following information, then click **Next**:
 - Package name: com.ibm.itso.Warehouse
 - File name: WarehouseBP
3. Select **Sequence-based BPEL process** and click **Finish**. At this point, all the errors created in the previous section should disappear.
4. Import the Warehouse.wsdl and Warehouse.xsd files into the package com.ibm.itso.wsdl in the WarehouseBusinessProcess project. These files can be found in the additional materials supplied with this redbook in the SerialGeneric\supportFiles directory.
5. Open the **WarehouseBP.bpel** file in the BPEL editor.
6. Delete the Partner Link called **PartnerLink@** and Variable called **InputVariable**.
7. Click the **+** icon next to Partner Links to create a new partner link WarehouseLink.
8. Create a new Partner Link Type called WarehouseLink and select **Warehouse.wsdl** in the com.ibm.itso.wsdl package as the Port Type File. Set this partner link type as the **process role**.

9. Create four more Partner Links using the information in tables Table 14-7, Table 14-8, Table 14-9, and Table 14-10, then save the WarehouseBP.bpel file.

Table 14-7 Partner Link configuration for Manufacturer

Field	Value
Partner Link name	ManufacturerLink
Port Type File	Manufacturer.wsdl
Role	Partner role

Table 14-8 Partner Link configuration for ManufacturerB

Field	Value
Partner Link name	ManufacturerBLink
Port Type File	Manufacturer.wsdl
Role	Partner role

Table 14-9 Partner Link configuration for ManufacturerC

Field	Value
Partner Link name	ManufacturerCLink
Port Type File	Manufacturer.wsdl
Role	Partner role

Table 14-10 Partner Link configuration for LoggingFacilityr

Field	Value
Partner Link name	LoggingFacilityLink
Port Type File	LoggingFacility.wsdl
Role	Partner role

10. Link the **Receive** and **Reply** activities in the process to the **shipGoods** operation of **WarehouseLink** partner link as show in Figure 14-9 on page 442. Make sure to create new variables for shipGoods operation.

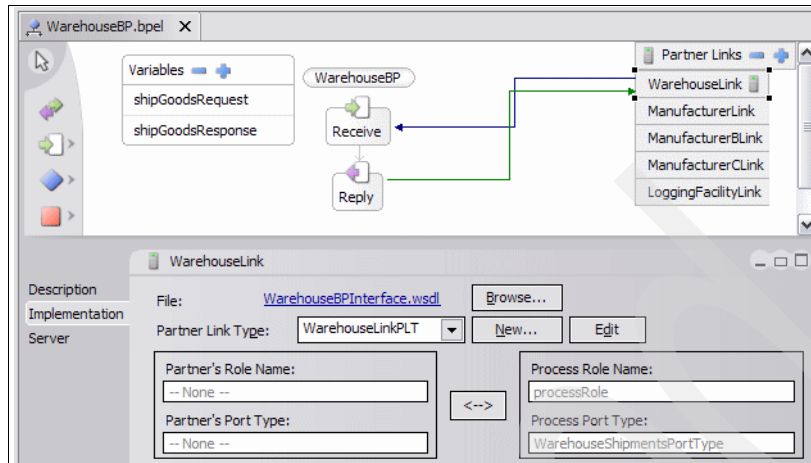


Figure 14-9 Receive-Reply activities linked to WarehouseLink

11. The Warehouse only invokes Manufacturers if the stock of an ordered item needs to be replenished. We use Switch activities and boolean variables in BPEL to make this decision.

Add three new variables called `manAOrderRequired`, `manBOrderRequired`, `manCOrderRequired` and link them to `booleanValue` message in `Manufacturer.wsdl` file as shown in Figure 14-10 .

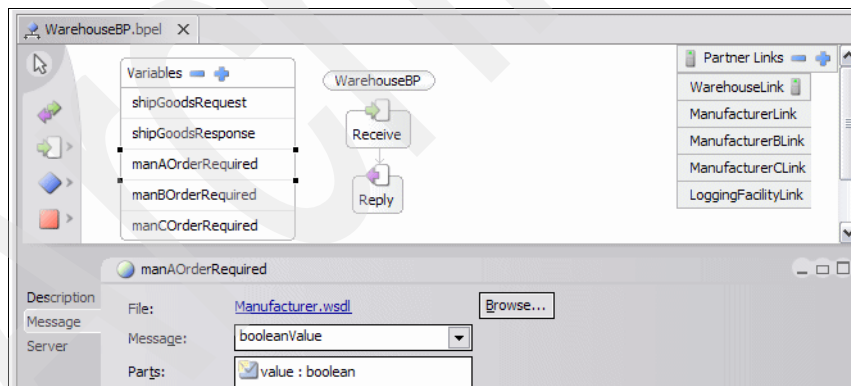


Figure 14-10 Variables in the BPEL editor

12. Add a new **Flow** activity between the Receive and Reply activity.

13. Add three sequences (Manufacturer, ManufacturerB, ManufacturerC) to the Flow activity.

14. Add a **Switch** activity, followed by a **Case** and **Invoke** activity in each of the three Manufacturer sequences as shown in Figure 14-11.

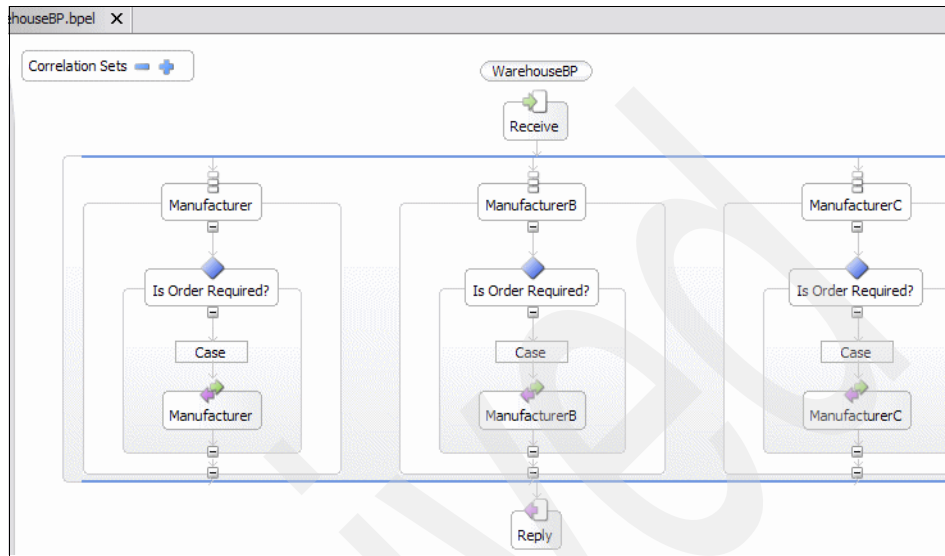


Figure 14-11 Flow activity in BPEL editor

15. Select **Case** activity under the Manufacturer sequence. In the Condition tab select Value **Visual Expression** and enter `manAOrderRequired.value == true` in the code panel.

Repeat this for ManufacturerB and ManufacturerC and entering respective visual expressions.

16. Link the Invoke activities with the respective Manufacturer Partner Links as shown in Figure 14-12 on page 444. Make sure to use the `submitPO` operation, and to create the request/response variables for each invoke activity.

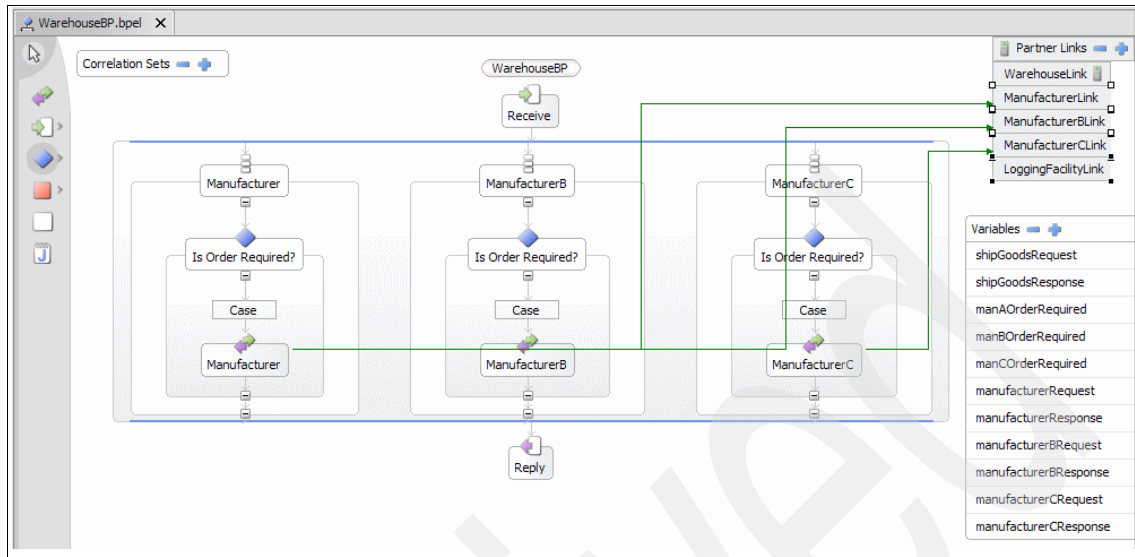


Figure 14-12 Invoke activities linked to Manufacturer partner links

17. Add a new **Java Snippet** between the Receive and Flow activity. Rename it to Prepare Manufacturer Requests.

Copy code from PrepareManufacturerRequestsSnippet.txt file into this Java snippet. You can find this file in the additional material supplied with this redbook in the SerialGeneric\supportFiles\ directory.

18. Add an **Invoke** activity called Log Msg, after the Flow activity. Link the Invoke activity with the LoggingFacility Partner Link as shown in Figure 14-13 on page 445 and select the **logEvent** operation. Make sure to create the request variable for the invoke activity called loggingRequest.

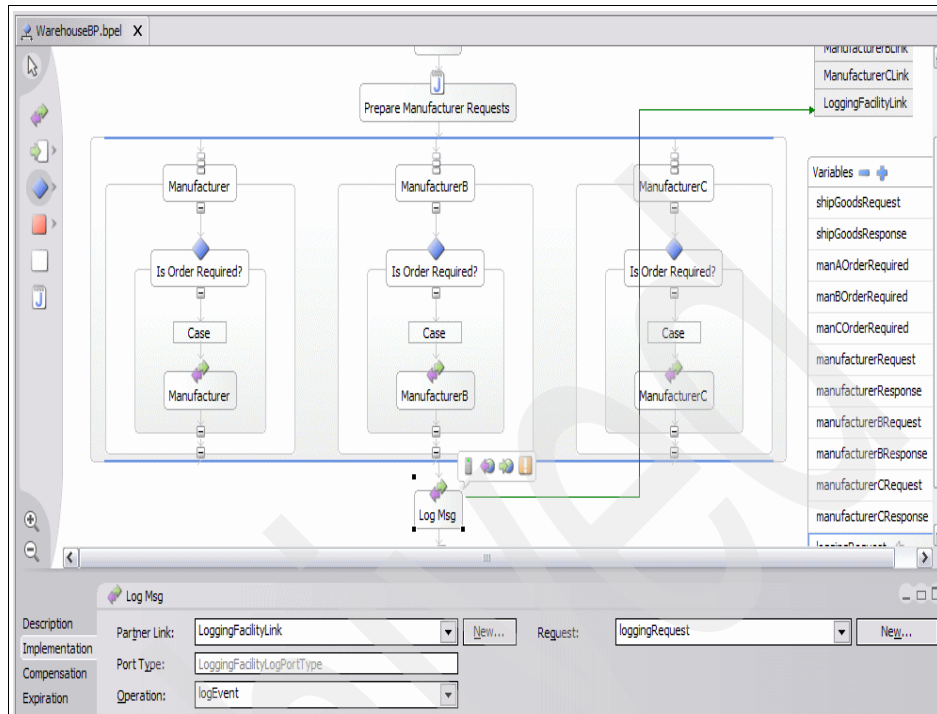


Figure 14-13 Log Msg invoke activity

19. Add a **Java snippet** between the Flow activity and the Log Msg activity. Rename it Prepare Log Request.

Copy code from PrepareLogRequestSnippet.txt file into this Java snippet. You can find this file in the additional material supplied with this redbook in the SerialGeneric\supportFiles\ directory.

20. Add a **Java snippet** between the Log Msg activity and Reply activity. Rename it Prepare Warehouse Response.

Copy code from PrepareWarehouseResponseSnippet.txt file into this Java snippet. You can find this file in the additional material supplied with this redbook in the SerialGeneric\supportFiles\ directory.

21. Save the Warehouse.bpel file. You should see no errors, and one warning.

You have now finished developing the WS-BPEL flow which communicates with Manufacturer (if required) and LoggingFacility services. Figure 14-14 on page 446 shows how your final Warehouse business process should look in a BPEL editor.

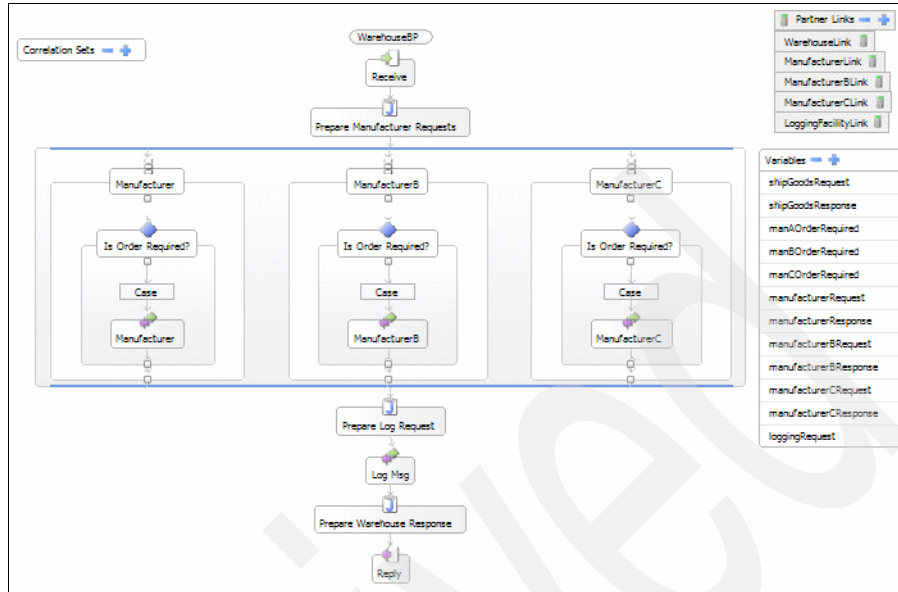


Figure 14-14 Warehouse BPEL flow

Generate deployment code for Warehouse business process

When generating deployment code for a WS-BPEL process, you need to select a binding for your process port type. By default all short running processes are deployed with an EJB binding and all long running as JMS. Other available options for short running process are JMS, SOAP/HTTP and SOAP/JMS. In our scenario, we deploy our process with an EJB binding.

You also need to provide a service WSDL file for each of the partners.

1. In WebSphere Studio Application Developer Integration Edition, under Business Integration perspective, right-click the **WarehouseBP.bpel** file and select **Enterprise Services** → **Generate Deploy code**.
2. In the Generate BPEL Deploy Code window, on the left side pane select **Interfaces for Partners** → **WarehouseShipmentPortType**. On the left-hand pane, make sure that nothing apart from the **EJB (default)** option is selected as shown in Figure 14-15 on page 447.

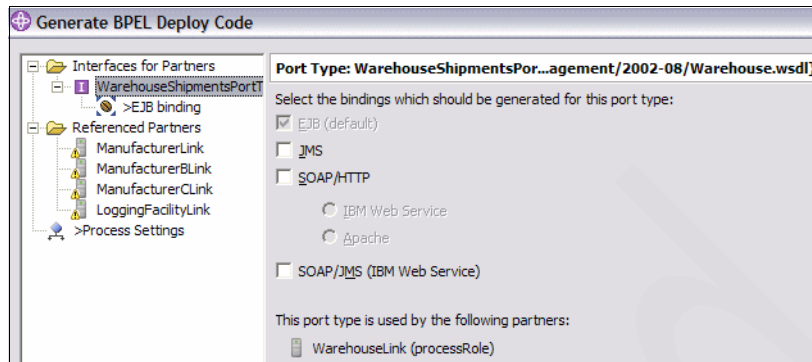


Figure 14-15 Selecting binding when generating BPEL deployment code

- On the left side pane, select **Referenced Partners** → **ManufacturerLink**. On the right -side pane click **Browse** and select **ManufacturerPortTypeJavaService.wsdl** file under **com/ibm/itso/Manufacturer** folder in **WarehouseBusinessProcess** project.

The service and port type fields are defaulted from the service WSDL file, as shown in Figure 14-16,

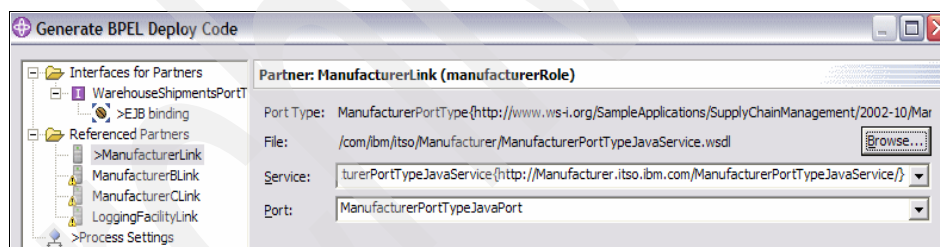


Figure 14-16 Selecting Partner service wsdl files when generating deployment code

- Repeat the previous step using the settings in Table 14-11 to select the service wsdl file for other Partners

Table 14-11 Referenced partners

Partner	WSDL File
ManufacturerB	ManufacturerBPortTypeJavaService.wsdl
ManufacturerC	ManufacturerCPortTypeJavaService.wsdl
LoggingFacility	LoggingFacilityLogPortTypeJavaService.wsdl

- Click **OK**. The code generation might take a few minutes.

When finished you will see that a WarehouseBusinessProcessEJB and WarehouseBusinessProcessWeb project has been created. The EJB module contains runtime WS-BPEL code. The Web module is currently empty but we will later create our HTTP/SOAP Warehouse service in this Web module.

Also note that a WSDL binding file called WarehouseBP_WarehouseShipmentPortType_EJB.wsdl has been created in the com.ibm.itso.Warehouse package.

6. We create a WSIF proxy class using the WSDL binding file created the previous step. This WSIF class is used by clients to invoke the BPEL process. Right-click **Warehouse_WarehouseShipmentPortType_EJB.wsdl** in the WarehouseBusinessProcess project and select **Enterprise Services** → **Generate Service Proxy**.
7. On the Proxy Selection window, accept the default proxy type **Web Services Invocation Framework (WSIF)** and click **Next**.
8. Review and accept the default entries on the Service Proxy window, as shown in Figure 14-17, and click **Next**.

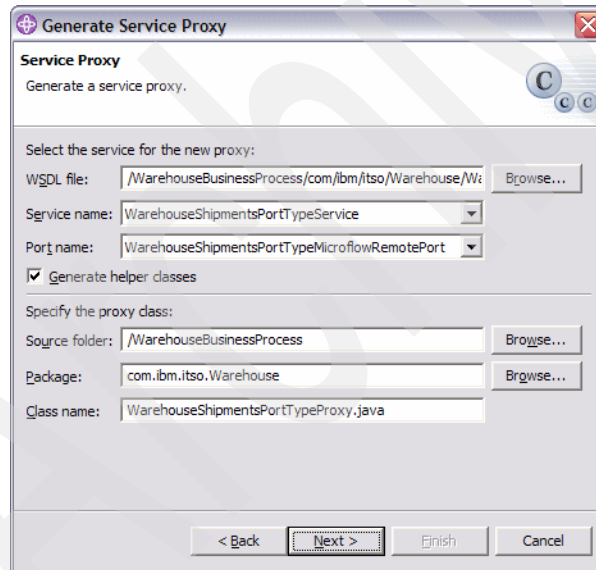


Figure 14-17 Generating WSIF proxy

9. On the next window, accept the default proxy style **Client Stub**, select **shipGoods** operation and click **Finish**.

Confirm that WarehouseShipmentsPortTypeProxy.java has been generated in the com.ibm.itso.Warehouse package.

14.3.7 Create the Warehouse service

In this section we use the Warehouse.wsdl file provided by the sample application to generate a Java bean. When the Retailer invokes the Warehouse service, the Web services engine accepts the service request and invokes this Java bean. This Java bean is responsible for initializing the BPEL process using the WSIF proxy created in the previous section.

We create the Warehouse service in a Web module. In our scenario, we use the WarehouseBusinessProcessWeb project for this purpose. You can choose to use a different Web project.

Create a new server and server configuration

The wizard that generates the Java bean skeleton from WSDL files requires you to select a server in your workspace. Therefore, we create a new server in this section.

1. In WebSphere Studio Application Developer Integration Edition, switch to Business Integration perspective. Select **Window** → **Show View** → **Server Configuration**.
2. Right-click in the Service Configuration window and select **New** → **Server and Server Configuration**.
3. Enter WarehouseServer as Server name.
4. In the Server type section, expand **WebSphere version 5.1** and select **Integration Test Environment**.
5. Click **Finish**. Look for WarehouseServer in the Server Configuration window.

Importing WSDL files

To import the WSDL files, follow these steps:

1. In the WarehouseBusinessProcessWeb project, create a package, `com.ibm.itso.wsdl`, under JavaSource.
2. Import the following WSDL and XSD files hosted on the HTTP Server into package `com.ibm.itso.wsdl`:
 - Configuration.wsdl
 - Configuration.xsd
 - envelope.xsd
 - Warehouse.xsd
 - Warehouse.wsdl
 - Warehouse_Impl.wsdl

Tip: `<HTTP_Server_home>\htdocs\en_US\wsdl` is the location of directory where these WSDL files are hosted on the `itsogood.itso.ral.ibm.com` machine.

3. Notice some compilation errors reported in some of the WSDL files. This is because the import statements in some of these files refer to the HTTP server which cannot be accessed from your WebSphere Studio Application Developer Integration Edition workspace at development time. Change these WSDL files to import the locally stored WSDL files.

To fix these compilation errors open each of the WSDL file reporting errors and change import path for WSDL. Change the text:

```
location = "http://appsrv1a.itso.ral.ibm.com/wsdl/<wsdl file>"
```

Replace it with this line:

```
location="<wsdl file>"
```

This should fix all the compilation errors in the workspace.

4. Right-click the **Warehouse_Impl.wsdl** file and select **Web Services** → **Generate Java Bean Skeleton**.
5. In the Service Deployment Configuration window click **Edit** to select **Integration Test Environment v5.1**. Accept all other defaults as shown in Figure 14-18 on page 451 and click **Next**.

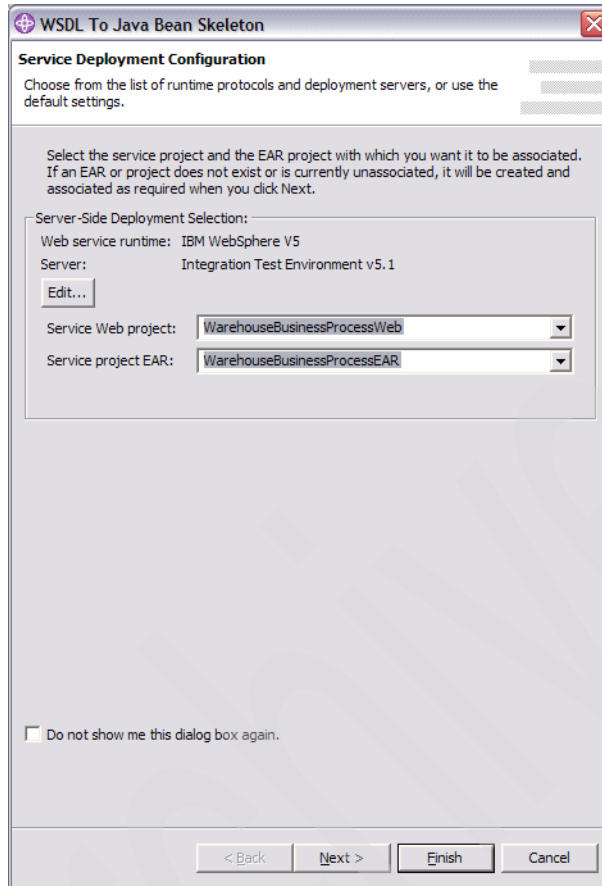


Figure 14-18 Generating web services client

6. Confirm that the Web services URI entered is
/WarehouseBusinessProcessWeb/JavaSource/com/ibm/itso/wsd1/Warehouse_Impl.wsdl and click **Next**.
7. Accept all defaults on the next window and click **Finish**.
8. Navigate to **WarehouseBusinessProcess** → **JavaSource** → **org.ws_i.www** and open the **WarehouseSoapBindingImpl.java** file.
9. Replace code in the WarehouseSoapBindingImpl.java file with code in the additional material supplied with this redbook from SerialGeneric\supportFiles\WarehouseSoapBindingImpl.java.

14.3.8 Exporting the Enterprise Application files

The WarehouseBusinessProcess enterprise application must be exported from the WebSphere Studio Application Developer Integration Edition workspace to an EAR file, so that it can be deployed to WebSphere Business Integration Server Foundation.

To export the WarehouseBusinessProcess enterprise application, follow these steps:

1. Click **File** → **Export**.
2. In the Export wizard, highlight **EAR file** and click **Next**.
3. In the EAR project pull-down, select **WarehouseBusinessProcessEAR**. Click **Browse** and locate a directory to where you want to store the enterprise application. Click **Save**. The enterprise application is called WarehouseBusinessProcess.ear by default.
4. Click **Finish** to generate WarehouseBusinessProcessEAR.ear.

14.4 Runtime guidelines

This section describes how to test and deploy the Warehouse business process built in 14.3, “Development guidelines” on page 425.

It contains the following sections:

- ▶ Testing with Web Services Explorer
- ▶ Testing the business process with ITSO Good
- ▶ Deploying the business process

As shown in Figure 14-19 on page 453, the ITSO Good application has been split into two applications:

- ▶ One with the Warehouse running as a WS-BPEL business process in WebSphere Business Integration Server Foundation V5.1
- ▶ The other with Retailer, SCMSampleUI, and LoggingFacility running in WebSphere Application Server Network Deployment V6.

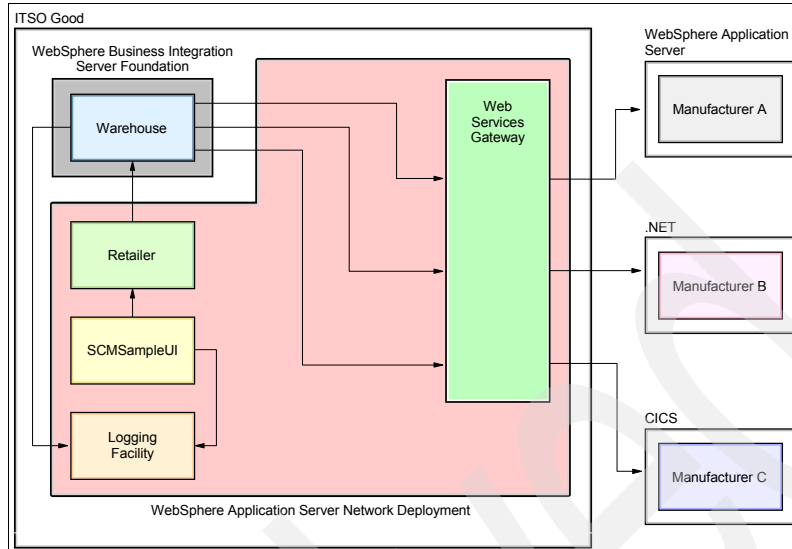


Figure 14-19 Runtime deployment topology

14.4.1 Testing with Web Services Explorer

Before testing the business process with the ITSO Good sample application, you can give the process a trial run using the Web Services Explorer. Perform the following:

1. In WebSphere Studio Application Developer Integration Edition, click the **Servers** tab, then right-click **WarehouseServer** and select **Add and remove projects**.
2. In the Add and Remove Projects window, select **WarehouseBusinessProcessEAR** and click **Add** to add it to the list of configured projects. Click **Finish**.
3. Start the server by right-clicking **WarehouseServer** and clicking **Start**.
4. When the server has started, you can invoke the Warehouse business process using the Web service interface you have defined for it. In the Package Explorer view locate WarehouseBusinessProcessWeb\WebContent\wsdl\Warehouse_Impl.wsdl. This WSDL file is used to invoke the business process. Right-click **Warehouse_Impl.wsdl** and click **Web Services** → **Test with Web Services Explorer**.
5. The Web Services Explorer starts. In the Navigator section expand **WarehouseSoapBinding** and click **ShipGoods**.

6. The Actions section allows you to invoke the shipGoods operation. Enter the following information (Figure 14-20).
 - a. Set ProductNumber to 605001
 - b. Set Quantity to 1
 - c. Set Customer to A12345-9876543-xyz

Actions

[Source](#)

Invoke a WSDL Operation

Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints
http://localhost:9080/WarehouseBusinessProcessWeb/services/Warehouse

ItemList

Item Add Remove

Item	Content
ProductNumber	nonNegativeInteger 605001
Quantity	unsignedShort 1

Customer normalizedString
A12345-9876543-xyz

Go Reset

Figure 14-20 Web Services Explorer input

7. Click **Go** to invoke the business process. When the call is complete, the status bar should show the message response. Click **Source** to see the SOAP request and response message. Figure 14-21 on page 455 shows a successful invocation, where the response message is set to true.

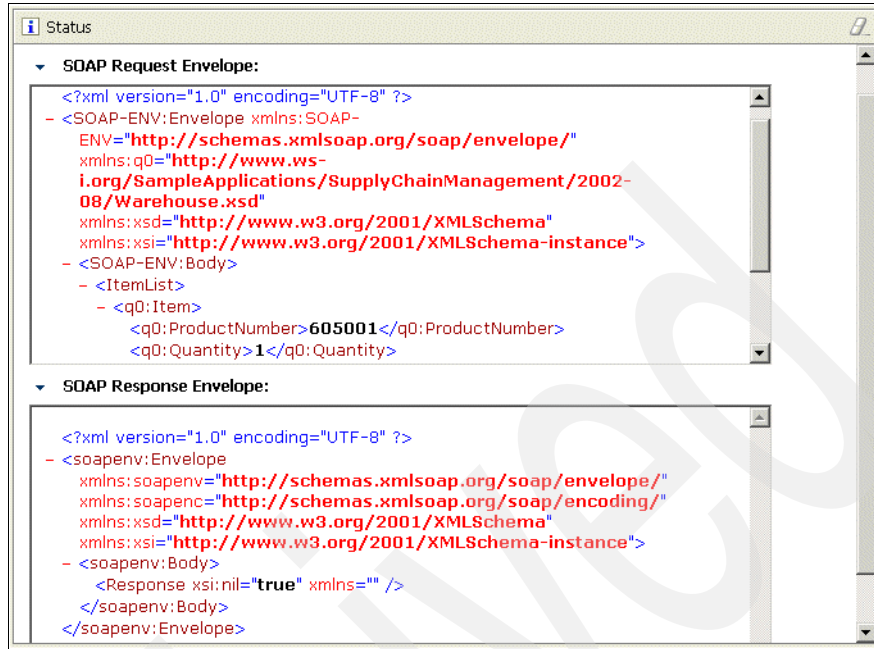


Figure 14-21 SOAP request and response message

14.4.2 Testing the business process with ITSO Good

We can test that the Warehouse business process works with the ITSO Good sample application while the business process is running in the WebSphere Studio Application Developer Integration Edition test environment. For detailed instructions about using the ITSO Good sample application see 8.2, “ITSO Good sample business scenario” on page 138.

1. In WebSphere Studio Application Developer Integration Edition, ensure the Warehouse business process is added to a test server, and that test server is running, as described in 14.4.1, “Testing with Web Services Explorer” on page 453.
2. Start the ITSOGoodProfile and ManufacturerProfile WebSphere Application Server instances on the itsogood.itso.ral.ibm.com machine. Also start ManufacturerB and ManufacturerC if you have configured them.
3. You need to change the Web services client binding for the Retailer to point to the new Warehouse business process. Perform the following tasks:
 - a. Log in to the ITSOGoodProfile machine where the ITSOGood enterprise application is hosted.

- b. Click **Applications** → **Enterprise Applications** → **ITSOGood** → **Web modules** → **RetailerWeb.war** → **Web services client bindings**.
 - c. Click **Edit** for WarehouseService in the Port Information column.
 - d. In the Overridden Endpoint URL field, enter the URL of the Warehouse business process Web service. You need to include the IP address of the machine where the Warehouse Web service is running in the test environment of WebSphere Studio Application Developer Integration Edition. For example if the test server were using IP address 1.2.3.4, you would enter:
`http://1.2.3.4:9080/WarehouseBusinessProcessWeb/services/Warehouse`
 - e. Click **OK** then save your changes.
 - f. Restart the ITSOGood enterprise application.
4. You are now ready to test the ITSO Good sample application. To start, test the Warehouse business process without invoking any Manufacturers:
- a. Launch the ITSO Good sample application on the machine hosting the ITSOGood enterprise application:
`http://itsogood.itso.ral.ibm.com:9080/SCMSampleUI`
 - b. Click **Place New Order**.
 - c. Order 1 item of product 605001 and click **Submit Order**. This makes a Web service call from the Retailer to the Warehouse Web service running in WebSphere Studio Application Developer Integration Edition.
 - d. To check if the call was successful, click **Track Order**. You should see an entry stating BPEL Warehouse is able to ship the goods, as shown in Figure 14-22 on page 457.



Supply Chain Management Sample Application

Track Order

Review details of the order. Then Configure another order or review Order Status.

Service ID	Event ID	Description
Retailer.submitOrder	UC1-5	Order placed by A12345-9876543-xyz for 605001
WarehouseA.ShipGoods	UC2-2-1	BPEL Warehouse is able to ship the goods
Retailer.submitOrder	UC1-9	Processing of the order from A12345-9876543-xyz has finished normally

Order Status
Configure

Figure 14-22 Successful invocation of the Warehouse

- e. You should also see an entry in the Console view of WebSphere Studio Application Developer Integration Edition to indicate the LoggingFacility Web service was invoked:


```
SystemOut 0 logged
```
5. Now test the ITSO Good sample application again, this time placing an order that requires the Warehouse business process to invoke a Manufacturer:
 - a. Click **Configure** to return to the front screen of the ITSO Good sample application.
 - b. Click **Place New Order** then order 6 items of product 605001. Click **Submit Order**. This requires the Warehouse business process to invoke the Manufacturer Web service to replenish the stock for this product.
 - c. Click **Track Order** to see if the invocation was successful. You should see the same entries as shown in Figure 14-22 again.
 - d. There are two ways to confirm that the Manufacturer was invoked. Look in the Console view of WebSphere Studio Application Developer Integration Edition for the following entry:


```
SystemOut 0 Manufacturer_A has received and processed a request.
```
 - e. You should also see the following entry in the SystemOut.log file of the ManufacturerProfile application server:


```
SystemOut 0 Manufacturer A: Processing Purchase Order.
```
6. Try to order 6 items of product 605002 to test ManufacturerB, or 6 items of product 605003 to test ManufacturerC.

14.4.3 Deploying the business process

The Warehouse business process can be deployed to a WebSphere Business Integration Server Foundation server instead of using the test environment of WebSphere Studio Application Developer Integration Edition.

For detailed instructions on how to install and configure WebSphere Business Integration Server Foundation for WS-BPEL business processes, consult *WebSphere Business Integration Server Foundation V5.1 Handbook*, SG24-6318.

At a high level, perform the following:

1. Install WebSphere Business Integration Server Foundation V5.1.1.
2. Install the BPE container.

Note: The WS-BPEL process developed in our scenario is a short running process. Short running processes do not need any database, messaging, or staff configurations.

3. Install and start the WarehouseBusinessProcess.ear file exported from 14.3.8, “Exporting the Enterprise Application files” on page 452.

Exposed Serial Process runtime pattern: SOA profile

This chapter extends Chapter 14, “Exposed Serial Process runtime pattern: generic profile” on page 419 by describing how to build an Exposed Serial Process using an SOA profile.

In this scenario, the Warehouse WS-BPEL business process is modified to use an ESB and Exposed ESB Gateway to make calls to the Manufacturers. The business process is further modified remove the processing logic which determines which Manufacturer is used to replenish which item of stock. This logic has been moved to the SOA infrastructure of the ESB and Exposed ESB Gateway, where it has been implemented as a mediation.

The Exposed Serial Process runs in WebSphere Business Integration Server Foundation, the ESB is implemented using the service integration bus of WebSphere Application Server Network Deployment, and the Exposed ESB Gateway is implemented using the Web services gateway component of WebSphere Application Server Network Deployment.

15.1 Business scenario

The business scenario that is implemented in this chapter is exactly the same as discussed in Chapter 14, “Exposed Serial Process runtime pattern: generic profile” on page 419.

But in addition to addressing the business requirement to make the ITSO Good business process flexible and responsive, ITSO Good as part of their strategic SOA transformation of their IT services would like to build an infrastructure that can support these business-aligned services in an enterprise scale.

The decision of which Manufacturer to use to replenish stock is removed from the Warehouse business process, and instead placed as a mediation in part of the SOA infrastructure.

15.2 Design guidelines

In this section, we analyze the business requirements and apply Patterns for e-business to determine the appropriate runtime pattern for the solution. We then discuss the various design options available to us in implementing the solution and also look at the product mappings.

15.2.1 Analyze the business requirement

The given business scenario requires the need to externalize the process execution logic from the individual application services. We are then able to use a process manager to automate the coordination of business process flow between the Warehouse and the Manufacturing partners of ITSO Good.

Because the business process in the given scenario goes across organization boundaries, the following additional system requirements also need to be addressed:

- ▶ Interoperability standards should be used where possible to minimize future changes required to the source and target applications.
- ▶ Security is a primary concern when opening business processes to external organizations. As a result, the solution should include robust security mechanisms to protect enterprise resources.

15.2.2 Selecting a pattern

We use the Patterns for e-business to determine the appropriate Runtime pattern to apply to this scenario.

Described below is a step-by-step approach used to navigate the Patterns for e-business asset catalog:

1. Business pattern

We select the Extended Enterprise business pattern because the given scenario requires interactions between the business processes in the Warehouse and Manufacturer systems that reside in separate enterprises.

2. Application pattern

Because the source application (Warehouse) initiates an interaction that is to be distributed to multiple target partner applications in a serial manner, we choose the Exposed Serial Process application pattern.

3. Runtime pattern

The selection of the application pattern provides us with the possible runtime patterns for the proposed solution. Since the business requirement mandates an SOA infrastructure, we select the *SOA profile* of the Exposed Serial Process application pattern.

Figure 15-1 on page 462 shows the level 0 decomposition of the SOA profile of the Exposed Serial Process runtime pattern, mapped on to the Exposed Serial Process application pattern.

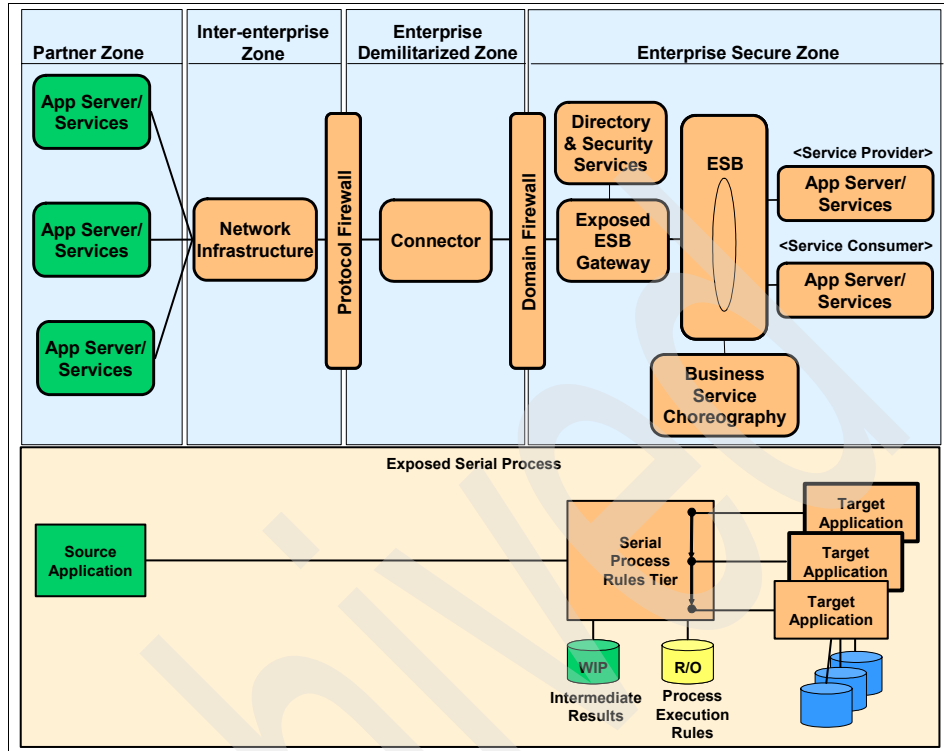


Figure 15-1 Exposed Serial Process runtime pattern: SOA profile

15.2.3 Analyze design options

The design decisions made in the previous chapters also apply to this scenario, specifically:

- ▶ “Implementing an Exposed Serial Process” on page 423
- ▶ “WS-BPEL” on page 423

In addition, this scenario implements the Exposed Process Manager in the generic profile scenario, as the Business Service Choreography node in this scenario. The Business Service Choreography node uses an ESB to communicate to service providers, and the ESB is also used when service consumers initiate a process. When service consumers and providers are in an extended enterprise, the Exposed ESB Gateway is responsible for handling these requests on behalf of the ESB.

15.2.4 Products

In this section we look at the products available to implement the various components in the Exposed Broker runtime pattern.

Product implementation options

Product choices for this scenario are based on:

- ▶ Design decisions that we made in 15.2.3, “Analyze design options” on page 462
- ▶ Extended Enterprise capabilities of the products
- ▶ Products that are currently available

Exposed Process Manager component

We can use the following currently available products to implement the Exposed Process Manager component in the given scenario:

- ▶ WebSphere MQ Workflow
- ▶ WebSphere Business Integration Server Foundation
- ▶ WebSphere Process Server

For this scenario, WebSphere Business Integration Server Foundation V5.1 meets all of the requirements and, as a result, is the product of choice.

ESB component

We can use the following currently available products to implement the ESB component in the given scenario:

- ▶ WebSphere Application Server Network Deployment V6.0.2
- ▶ WebSphere Message Integration Message Broker

For this scenario, the service integration bus in WebSphere Application Server Network Deployment V6.0.2 meets all of the requirements and, as a result, is the product of choice.

Exposed ESB Gateway component

We can use the following currently available products to implement the ESB component in the given scenario:

- ▶ Web services gateway component in WebSphere Application Server Network Deployment V6.0.2
- ▶ WebSphere Partner Gateway

In the given scenario, there is no requirement for advanced functions such as Partner Management and provisioning that are provided by the WebSphere Partner Gateway. Therefore, the Web services gateway component in

WebSphere Application Server Network Deployment V6.0.2 is used to implement the Exposed ESB Gateway.

The complete product mapping for this scenario is shown in Figure 15-2.

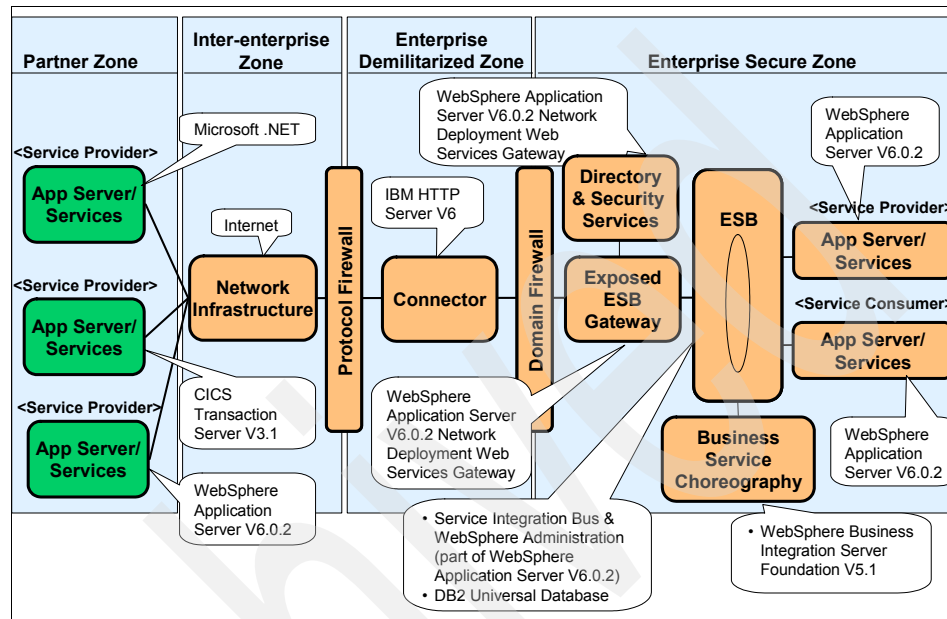


Figure 15-2 Exposed Serial Process: SOA profile product mapping

This Product mapping uses WebSphere Application Server V6.0.2 to host the Application Services in the Enterprise Secure Zone.

The ESB is run as a service integration bus within WebSphere Application Server Network Deployment V6.0.2, providing service location transparency between Service Consumers and Service Providers within the enterprise. With the Network Deployment offering, you can implement a scalable clustering of multiple WebSphere Application Server servers.

An application service uses the ESB to invoke the automated process instance implemented by the Business Service Choreography node using the Web Services Invocation Framework (WSIF). The Business Service Choreography node is implemented using WebSphere Business Integration Server Foundation V5.1.

A local DB2 Universal Database database is used to store the SDO repository.

The Web services gateway provided with WebSphere Application Server Network Deployment V6.0.2 is the Exposed ESB Gateway in our Product

mapping. It is used to provide a standard, consistent interface for the internal processes to access external processes. The use of an Exposed ESB Gateway minimizes the disruption caused by changes in the external partner infrastructure.

In the Directory and Security services node, the service integration bus within WebSphere Application Server Network Deployment V6.0.2 is configured secure all transactions to the external Partner Zone to use WS-Security integrity and confidentiality.

The IBM HTTP Server V6 acts as an Adapter Connector by providing protocol transformation to SOAP/HTTPS thus delivering transport level security between the enterprise and the partner organizations.

15.3 Development guidelines

This section describes how to implement the Exposed Serial Process and integrate the process with an ESB.

Note: This section requires WebSphere Studio Application Developer Integration Edition V5.1.1 plus Cumulative Fix 010 or higher installed.

15.3.1 Scenario implementation: Serial process interaction

The WS-I sample scenario implemented in Chapter 15, “Exposed Serial Process runtime pattern: SOA profile” on page 459 is the starting point for this chapter.

The Warehouse business process is modified to invoke an inbound service on the ESB implemented as the service integration bus of WebSphere Application Server. The ESB forwards the request to the Web services gateway. We create a mediation service on the gateway which is responsible for creating and sending requests to different manufacturers (if required). When the manufacturers reply, the mediation service collates these responses and forwards a single response to the Warehouse business process through the ESB.

We use the Process Choreographer tool in WebSphere Studio Application Developer Integration Edition to create and generate deployment code for a business process.

15.3.2 Creating the basic infrastructure

This scenario assumes that you have completed the scenario described in Chapter 14, “Exposed Serial Process runtime pattern: generic profile” on page 419. We build from that infrastructure in this chapter.

If you did not complete building the business process in the previous chapter, you can add it by importing the project interchange file `CompletedProcess.zip` into a workspace in WebSphere Studio Application Developer Integration Edition. You can find this file in the `SerialGeneric` directory in the additional material supplied with this redbook.

Before you can build the WS-BPEL business process for this scenario, you need to make the following changes to the base infrastructure:

- ▶ Create a service integration bus.
- ▶ Create an HTTP endpoint listener for the bus.
- ▶ Define a Manufacturer inbound service on the service integration bus.
- ▶ Export the Manufacturer inbound service WSDL.

Creating a service integration bus

Create a new service integration bus on the ITSOGoodProfile server running on the `itsogood.itso.ra1.ibm.com` machine as follows:

1. Log in to the ITSOGoodProfile server administration console.
2. Create a new service integration bus called **ESBBus**:
 - a. Expand **Service integration**, click **Buses**, and click **New** to create a new bus.
 - b. Enter a name of **ESBBus** and click **Apply**.
3. Click **Bus members** and click **Add**. Click **Next**, then **Finish** to create the bus member.
4. Save your changes.

Creating an HTTP endpoint listener

To enable clients to communicate to ESBBus using SOAP/HTTP, create an HTTP endpoint listener as follows:

1. Click **Servers** → **Application Servers** → **server1** → **Endpoint listeners**.
2. Create a new Endpoint listener:
 - a. Click **New**.
 - b. Enter a Name of `SOAPHTTPChannel1`, a URL root of `http://itsogood.itso.ra1.ibm.com:9080/wsgwsoaphttp1` and a WSDL serving HTTP URL root of `http://appsrv1a.itso.ra1.ibm.com/wsd1`

- c. Click **Apply**.
3. Assign the endpoint listener to ESBBus:
 - a. Click **Connection Properties**.
 - b. Click **New**.
 - c. Set the Bus name to ESBBus and click **OK**.
4. Save your changes.

Defining a Manufacturer inbound service

Add a Manufacturer inbound service to the ESBBus service integration bus as follows.

1. Click **Service integration** → **Buses** → **ESBBus** → **Inbound Services**.
2. Define an inbound service called ManufacturerService:
 - a. Click **New**.
 - b. Leave the Service destination name to default. We will change it later.
 - c. In the Template WSDL location field enter `http://apprsv1a.itso.ral.ibm.com/wsdl/Manufacturer_Impl.wsdl`
 - d. Click **Next**, and **Next**.
 - e. Set the Inbound service name to ManufacturerInboundService, then click **Next**.
 - f. Click **Finish**.
3. Change the inbound port name to Manufacturer:
 - a. In the ManufacturerInboundService properties click **Inbound Ports**.
 - b. Click on the existing inbound port name.
 - c. Set the Name field Manufacturer
 - d. Click **OK**.
4. Save your changes.

Exporting the Manufacturer inbound service WSDL

We need to export the WSDL describing how to invoke the Manufacturer inbound service so it is available to the business process we build in WebSphere Studio Application Developer Integration Edition. Perform the following steps:

1. Click **Service integration** → **Buses** → **ESBBus** → **Inbound Services** → **ManufacturerInboundService**.
2. Export the Manufacturer inbound service WSDL files to the local file system:
 - a. Click **Publish WSDL files to ZIP file**.

- b. Click **ManufacturerInboundService.zip** and then click **Open** in the resulting dialog. The ZIP file opens in the application your operating system has associated with .zip files
 - c. Save the following files to a temporary directory on your local file system:
 - ESBBus.ManufacturerInboundServiceBindings.wsdl
 - ESBBus.ManufacturerInboundServicePortTypes.wsdl
 - ESBBus.ManufacturerInboundServiceService.wsdl
3. Copy each of the three exported files to the machine where WebSphere Studio Application Developer Integration Edition is running, so you will have access to them when building the Warehouse business process.

15.3.3 Creating a Manufacturer Web service client

In this scenario, the Manufacturers are accessed through a service integration bus and Web services gateway. The Warehouse business process invokes each Manufacturer by placing a single call to the Manufacturer inbound service running in the service integration bus.

The Manufacturer inbound service uses a mediation to determine which manufacturers need to be called, and places calls to the relevant Web services gateway services, that, in turn, contact the manufacturers.

This section describes how to create a Web services client that points to the Manufacturer inbound service. For detailed steps on creating a Web services client see 14.3.4, “Creating Manufacturer and LoggingFacility Web services clients” on page 429.

1. In the WebSphere Studio Application Developer Integration Edition workspace used for the previous chapter, create a new Java project called `ManufacturerESBServiceClient`.
2. Import the following WSDL files into the folder `ManufacturerESBServiceClient\wsdl`. Find the ESBBus* files in the temporary directory you used in the previous step, and you can find others in the HTTP server on the `itsogood.itso.ral.ibm.com` server:
 - Configuration.wsdl
 - Configuration.xsd
 - envelope.xsd
 - ESBBus.ManufacturerServiceBindings.wsdl
 - ESBBus.ManufacturerServicePortTypes.wsdl
 - ESBBus.ManufacturerServiceService.wsdl
 - ManufacturerPO.xsd
 - ManufacturerSN.xsd

3. The ESBBus* files import with errors. Fix these by removing the references to `http://appsrv1a.itso.ral.ibm.com/wsd1`.
4. Create a Web services client for the Manufacturer inbound service:
 - a. Right-click **ESBBus.ManufacturerServiceService.wsdl** and select **Web Services** → **Generate Client**.
 - b. Click **Next**.
 - c. Select **Choose server first**, then select **Integration Server v5.1**. Click **Finish**.
5. Delete **ManufacturerPortTypeProxy.java** in the `org.ws_i.www` package of `ManufacturerESBServiceClient`. There should be no errors, only warnings, in the Tasks view.

15.3.4 Modify the Manufacturer proxy class

The Manufacturer proxy class created in the previous chapter invokes the Manufacturer Web services gateway service. We change this proxy to call the Manufacturer inbound service client created in the previous section.

1. Change the build path of **WarehouseBusinessProcess** to include `ManufacturerESBServiceClient` and exclude the `ManufacturerGWServiceClient` project. You can see some compilation errors in the Tasks view. These are fixed in the following steps.
2. Open **ManufacturerPortType.java** file under `com.ibm.itso.Manufacturer` in the `WarehouseBusinessProcess` project. Replace the code for the entire class with the code in `ManufacturerPortType.java`, which you can find in the `SerialSOA\supportFiles` directory of the additional material supplied with this redbook.
3. Delete the Java packages **com.ibm.itso.ManufacturerB** and **com.ibm.itso.ManufacturerC** in the `WarehouseBusinessProcess` project.
4. All of the compilation errors should now be fixed.

15.3.5 Modify the Warehouse business process

In this section we modify the Warehouse business process to call the Manufacturer inbound service on the ESB.

1. Open the **WarehouseBP.bpel** file in the BPEL editor. You can find this file in the `com.ibm.itso.Warehouse` package of the `WarehouseBusinessProcess` project.
2. Delete the Partner Links called **ManufacturerBLink** and **ManufacturerCLink**.

3. Delete the following variables:
 - manBOrderRequired
 - manCOrderRequired
 - manufacturerBRequest
 - manufacturerBResponse
 - manufacturerCRequest
 - manufacturerBResponse
4. Rename variable manAOrderRequired to manOrderRequired.
5. Delete the sequences **ManufacturerB** and **ManufacturerC**.
6. Your BPEL process should look like the process shown in Figure 15-3.

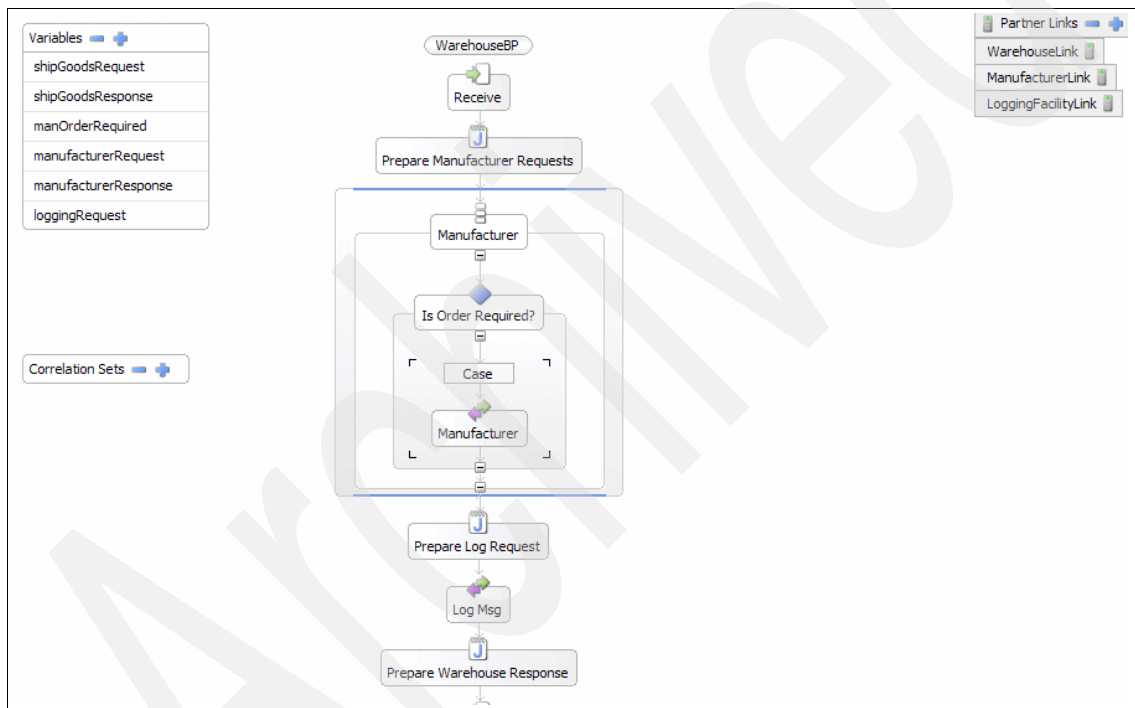


Figure 15-3 Warehouse Business Process

7. Replace the code in the **Prepare Manufacturer Requests** Java snippet with PrepareManufacturerRequestsSnippet.txt which you can find in the SerialSOA\supportFiles directory of the additional material supplied with this redbook.
8. Modify the Case visual expression to be:


```
manOrderRequired.value == true
```

9. Save the WarehouseBP.bpel file. There should be no errors in the WarehouseBusinessProcess project (although there will be errors in the WarehouseBusinessProcessEJB project which we will fix next).

15.3.6 Generate deployment code and export the process

To generate deployment code for the new business process, perform the following:

1. Right click **WarehouseBP.bpel** and select **Enterprise Services → Generate Deploy Code**.
2. In the Generate BPEL Deploy Code window, click **OK** to generate the deployment code.
3. There should be no errors in the Tasks view.

You can optionally export the deployed business process to an EAR file if you intend to deploy it to WebSphere Business Integration Server Foundation. To export the WarehouseBusinessProcess enterprise application:

1. Click **File → Export**.
2. In the Export wizard, highlight **EAR file** and click **Next**.
3. In the EAR project pull-down menu, select **WarehouseBusinessProcessEAR**. Click **Browse** and locate a directory to where you want to store the enterprise application. Click **Save**. The enterprise application is called WarehouseBusinessProcess.ear by default.
4. Click **Finish** to generate WarehouseBusinessProcessEAR.ear.

15.4 Runtime guidelines

This section describes how to test and deploy the Warehouse business process built in 15.3, “Development guidelines” on page 465.

It contains the following sections:

- ▶ Configuring the ESB
- ▶ Configuring the Exposed ESB Gateway
- ▶ Testing the business process with ITSO Good

As shown in Figure 15-4 on page 472, the ITSO Good application remains split into two applications: one with the Warehouse running as a WS-BPEL business process in WebSphere Business Integration Server Foundation V5.1, and the other with Retailer, SCMSampleUI, and LoggingFacility running in WebSphere Application Server Network Deployment V6.

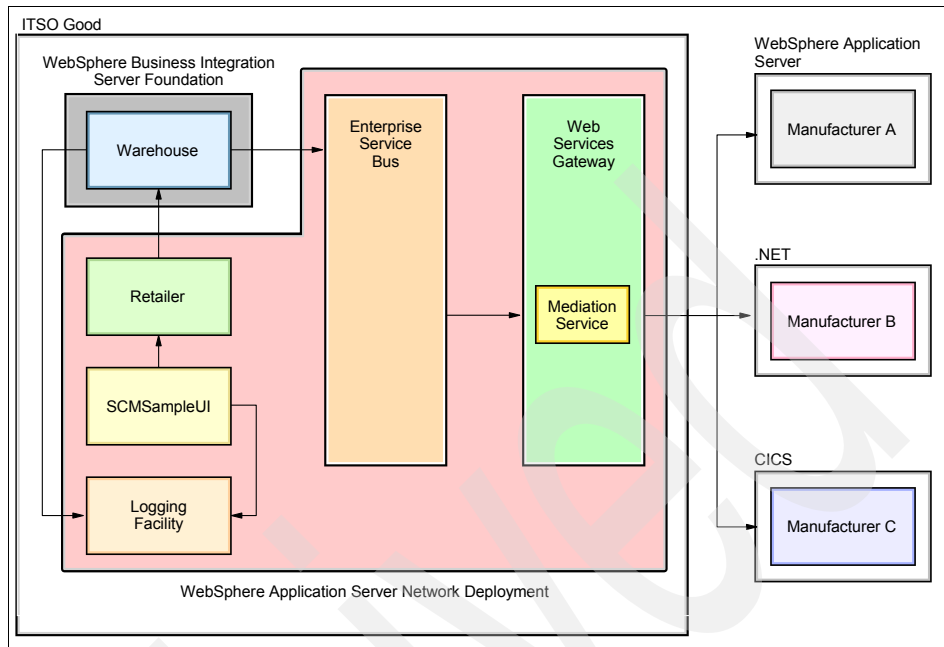


Figure 15-4 Runtime deployment topology

The Warehouse application deployed in WebSphere Business Integration Server Foundation V5.1 is invoked by the Retailer application running in WebSphere Application Server Network Deployment V6.0.2. The Warehouse application then invoke the Manufacturer or Manufacturers if required with the ESB and the Web services gateway. The Web services gateway includes a mediation service which checks the incoming message from the Warehouse and identifies the relevant Manufacturers to be called. It also then aggregates the response messages from the various Manufacturers and creates a single message which is returned to the Warehouse for logging.

15.4.1 Configuring the ESB

This section describes how to configure the ESB implemented by the ESBBus service integration bus.

You need to perform the following tasks:

- ▶ Define an outbound service to point to the Manufacturer Gateway service.
- ▶ Reconfigure the Manufacturer inbound service to point to the Manufacturer outbound service.

Defining a Manufacturer outbound service

Perform the following in the ITSOGoodProfile server administrative console, where the ITSOGood enterprise application is installed:

1. Click **Service integration** → **Buses** → **ESBBus** → **Outbound Services**.
2. Define a new outbound service called `ManufacturerGatewayOutboundService` as follows:
 - a. Click **New**.
 - b. In the WSDL location field enter the location of the Manufacturer Gateway service WSDL file, which should be `http://appsrv1a.itso.ral.ibm.com/wsdl/ExposedESBGatewayBus.ManufacturerGatewayServiceService.wsdl`
 - c. Click **Next**, **Next**, and **Next**.
 - d. Set the Outbound service name to `ManufacturerGatewayOutboundService`, the Service destination name to `ManufacturerGatewayOutboundServiceDestination`, and the Port destination name to `ManufacturerGatewayOutboundPort`.
 - e. Click **Next**, then **Finish**.
3. Save your changes.

Reconfiguring the Manufacturer inbound service

Reconfigure the Manufacturer inbound service so that it points to the destination used by the Manufacturer outbound service. Perform the following:

1. Click **Service integration** → **Buses** → **ESBBus** → **Inbound Services**.
2. Click **ManufacturerInboundService**.
3. Set the Service destination name to **ManufacturerGatewayOutboundServiceDestination** and click **OK**.
4. Save your changes.

15.4.2 Configuring the Exposed ESB Gateway

This section describes how to configure the Exposed ESB Gateway which is implemented by the ExposedESBGateway Web services gateway.

You need to perform the following tasks:

- ▶ Install the mediation and define the request and response mediations.
- ▶ Assign the mediations to the Web services gateway service.

Installing and defining the mediations

The first step is to install the mediation enterprise application, then define the request and response mediations to the ExposedESBGatewayBus service integration bus. Perform the following steps in the ITSOGoodProfile server administrative console, where the ITSOGood enterprise application is installed:

1. Install the ManufacturerBroker.ear mediation as follows:
 - a. Click **Applications** → **Install New Application**.
 - b. In the Specify path field locate ManufacturerBroker.ear. This EAR file is in the additional material, provided with this book, in the \BrokerGeneric\ears directory.
 - c. Click Next then click through the install wizard to install the enterprise application.
 - d. Save your changes.
2. Start the ManufacturerBroker enterprise application.
3. Define the request and response mediations to the ExposedESBGatewayBus service integration bus as follows:
 - a. Click **Service integration** → **Buses** → **ExposedESBGatewayBus** → **Mediations**.
 - b. Click **New**.
 - c. Define a new mediation with the following properties, then click **OK**.
 - Mediation name: RequestMediator
 - Handler list name: RequestMediator
 - Select **Global transaction**
 - d. Define a second mediation with the following properties, then click **OK**.
 - Mediation name: ResponseMediator
 - Handler list name: ResponseMediator
 - Select **Global transaction**
4. This scenario needs context properties for both the mediations. Define these as follows:
 - a. Click **Service integration** → **Buses** → **ExposedESBGatewayBus** → **Mediations** → **RequestMediator** → **Context Properties**.
 - b. Click **New** and create a context property with the following values, then click **OK**:
 - Name: logQueueName
 - Context type: String
 - Context value: logQ

- c. Create a second context property for RequestMediator with the following values:
 - Name: tmpStorageQueueName
 - Context type: String
 - Context value: tmpStorageQ
- d. Define these two context properties (logQueueName and tmpStorageQueueName) for ResponseMediator.
5. You need to create the logQ and tmpStorageQ destinations as follows:
 - a. Click **Service integration** → **Buses** → **ExposedESBGatewayBus** → **Destinations**.
 - b. Click **New**, and define a new queue destination called logQ.
 - c. Define a second queue destination called tmpStorageQ.
6. Save your changes.

Assigning the mediations to the Gateway service

When a request for a Manufacturer Web service is received by the Manufacturer inbound service on ESBBus, it is forwarded to the ManufacturerGatewayService on the ExposedESBGatewayBus. We need to assign the request and response mediations to the ManufacturerGatewayService as follows:

1. Click **Service integration** → **Buses** → **ExposedESBGatewayBus** → **Web service gateway instances**.
2. Click the **ExposedESBGateway** Web service gateway instance.
3. Click **Gateway services**, then click the **ManufacturerGatewayService** Gateway service.
4. Set the following fields, as shown in Figure 15-5, then click **OK**.
 - Request mediation: **RequestMediator**
 - Request mediation bus member: **ITSOGoodNode:server1**
 - Response mediation: **ResponseMediator**
 - Response mediation bus member: **ITSOGoodNode:server1**

Buses > ExposedESBGatewayBus > Web service gateway instances > ExposedESBGateway > Gateway services > ManufacturerGatewayService

A gateway service provides the configuration of the web service enablement of a service destination, along with the information that maps to one or more target destinations.

Configuration

General Properties	Additional Properties
<p>Gateway service name ManufacturerGatewayService</p> <p>Description <input type="text"/></p> <p>Gateway request destination name ManufacturerGatewayRequestDestination</p> <p>Request mediation RequestMediator</p> <p>Request mediation bus member ITSOGoodNode:server1</p> <p>Gateway response destination name ManufacturerGatewayResponseDestination</p> <p>Response mediation ResponseMediator</p> <p>Response mediation bus member ITSOGoodNode:server1</p>	<ul style="list-style-type: none"> Inbound web service enablement Target services

Figure 15-5 Mediations assigned to the Gateway service

5. Save your changes.
6. For all of the changes to take effect, restart the ITSOGoodProfile application server.

15.4.3 Testing the business process with ITSO Good

We can test that the Warehouse business process works with the ITSO Good sample application while the business process is running in the WebSphere Studio Application Developer Integration Edition test environment. For detailed instructions about using the ITSO Good sample application, see 8.2, “ITSO Good sample business scenario” on page 138.

1. In WebSphere Studio Application Developer Integration Edition, ensure the Warehouse business process is added to a test server, and that this test server is running.
2. Start the ITSOGoodProfile and ManufacturerProfile WebSphere Application Server instances on the itsogood.itso.ral.ibm.com machine. Also start ManufacturerB and ManufacturerC, if you have configured them.

3. You are now ready to test the ITSO Good sample application. To start, test the Warehouse business process by placing an order that invokes only ManufacturerA:
 - a. Launch the ITSO Good sample application on the machine hosting the ITSOGood enterprise application:

`http://itsogood.itso.ra1.ibm.com:9080/SCMSampleUI`
 - b. Click **Place New Order**.
 - c. Order 6 items of product 605001 and click **Submit Order**. This makes a Web service call from the Retailer to the Warehouse Web service running in WebSphere Studio Application Developer Integration Edition, and eventually invoke ManufacturerA using the mediation.
 - d. To check that the call is successful, click **Track Order**. You should see an entry stating BPEL Warehouse is able to ship the goods, as shown in Figure 15-6.



WS-I
WEB SERVICES
INTEROPERABILITY
ORGANIZATION

Supply Chain Management Sample Application

Track Order

Review details of the order. Then Configure another order or review Order Status.

Service ID	Event ID	Description
Retailer.submitOrder	UC1-5	Order placed by A12345-9876543-xyz for 605001
WarehouseA.ShipGoods	UC2-2-1	BPEL Warehouse is able to ship the goods
Retailer.submitOrder	UC1-9	Processing of the order from A12345-9876543-xyz has finished normally

 Order Status
 Configure

Figure 15-6 Successful invocation of the Warehouse

- e. To confirm the mediation is invoked, check the SystemOut.log file of the machine running the ITSOGoodProfile application server for the messages shown in Example 15-1.

Example 15-1 Expected output showing the ManufacturerBroker mediation was invoked

```

SystemOut    0 ManufacturerBroker REQUEST Mediator: Starting ...
SystemOut    0 ManufacturerBroker REQUEST Mediator: Processing product 605001.
SystemOut    0 ManufacturerBroker REQUEST Mediator: Finished.
SystemOut    0 ManufacturerBroker RESPONSE Mediator: Started.
SystemOut    0 ManufacturerBroker RESPONSE Mediator: Message from Manufacturer: Manufacturer_A
has received and processed a request.
  
```

- f. There are two ways to confirm the Manufacturer is invoked. Look in the Console view of WebSphere Studio Application Developer Integration Edition for the following entry:

SystemOut 0 Manufacturer_A has received and processed a request.

- g. You should also see the following entry in the SystemOut.log file of the ManufacturerProfile application server:

SystemOut 0 Manufacturer A: Processing Purchase Order.

- 4. If you have ManufacturerB running, perform the following to test the configuration further:

- a. Click **Configure** to return to the front screen of the ITSO Good sample application.
- b. Click **Place New Order** then order 6 items of product 605001 and 6 items of product 605002. Click **Submit Order**. This requires the Warehouse business process to invoke the Manufacturer Web service to replenish the stock for this product.
- c. Click **Track Order** to see if the invocation is successful. You should see the same entries as shown in Figure 15-6 on page 477 again.
- d. Check the Console view of WebSphere Studio Application Developer Integration Edition for the following entries:

SystemOut 0 Manufacturer_A has received and processed a request.

SystemOut 0 Manufacturer_B has received and processed a request.



Part 4

Appendixes

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/SG247135>

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, SG247135.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
SG247135.zip	Zipped code samples

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	10MB for source material
Operating System:	Windows operating system
Memory:	1GB to run source material

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.

Microsoft .NET Web services

This appendix provides a brief overview of Web services on the Microsoft .NET platform and how they play a role in our development and runtime scenarios. In addition, it describes how to:

- ▶ Implement a Web service using Microsoft Visual Studio® .NET 2003 and Internet Information Services V5.0. This includes designing, building, deploying, and testing the Web service.
- ▶ Implement a J2EE Web service consumer client using IBM Rational Application Developer V6.0. This also includes designing, building, deploying, and testing the J2EE client.
- ▶ Enable Microsoft Internet Information Services V5.0 to require transport-layer security using SSL for all incoming requests.

The appendix is divided into the following sections:

- ▶ B.1, "Overview and context of .NET Web services"
- ▶ B.2, "Implementing a Microsoft .NET Web service"
- ▶ B.2 "Implementing a Microsoft .NET Web service" on page 486
- ▶ B.3 "Enabling transport-level security with SSL" on page 500

B.1 Overview and context of .NET Web services

In today's world, Web services provide the necessary building blocks in enabling organizations to achieve their business process refinement goals. On a high-level, Microsoft's .NET platform provides tools similar to IBM in creating and deploying Web services. Table B-1 shows this comparison.

Table B-1 Table comparing development and deployment technologies

DEVELOPMENT TECHNOLOGIES	DEPLOYMENT TECHNOLOGIES
IBM Rational Application Developer V6.0	IBM WebSphere Application Server
Microsoft Visual Studio .NET 2003	Microsoft Internet Information Services

B.1.1 How Microsoft .NET is used in the Redbook scenarios

In this Redbook, we developed several sample scenarios based on SOA Extended Enterprise patterns. We decided to implement .NET technologies in one of the organizations throughout these scenarios in order to simulate real-world issues where two unique technologies must share a common language in order to share information.

Throughout this Redbook, we use the ITSO Good supply chain management scenario. Figure B-1 shows that the Microsoft .NET Web service plays the role of a Web service target in the supply chain sample application. Specifically, it holds a role as a manufacturer of supplies (or Manufacturer B, as it is called in the scenario).

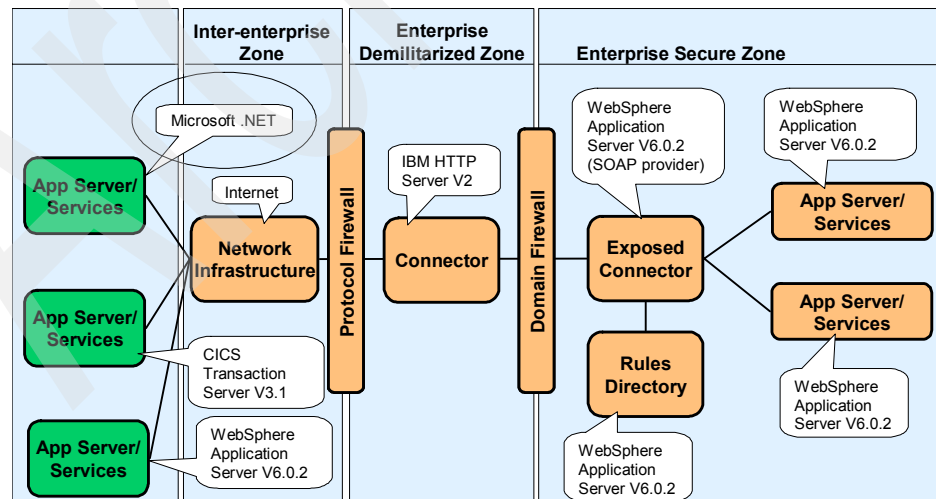


Figure B-1 How the Microsoft .NET Web service fits into our scenarios

Each manufacturer (Manufacturers A, B, and C) uses a unique technology to implement Web services. When completing the scenarios that employ three unique manufacturers, we are showing how organizations can bridge their technology gaps through the use of a common language, Web services using SOAP over HTTP.

B.1.2 Microsoft .NET Web service development overview

The implementation of the Microsoft .NET Web service we developed for our scenarios is written in the C# programming language. There are two design options developers must choose from when creating their Web service:

- ▶ Implementation First development
- ▶ WSDL First development

Implementation First development

In this design choice, a C# developer writes the business logic of an application. This application is then used to provide the implementation for a Web service. Visual Studio is used to generate a WSDL description of the C# code. This generated WSDL interface is used by Web service clients to connect to the Web service implemented in C#.

This option should be used when you want to expose an existing application as a Web service. See Figure B-1 for an illustration of how Implementation First development works.

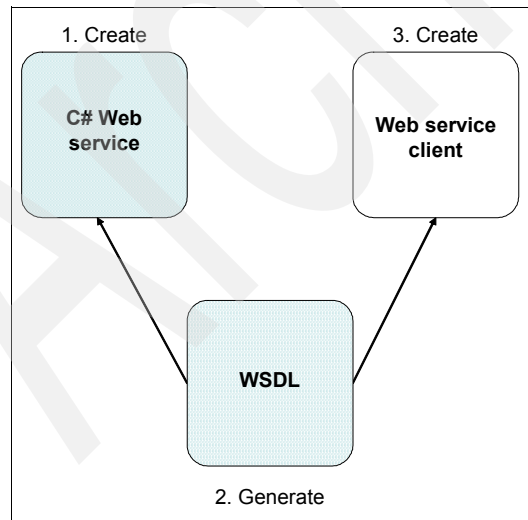


Figure B-2 Implementation First development

WSDL First development

In this design choice, a Web service is generated from a pre-existing WSDL interface rather than from a pre-existing application. The WSDL interface is used to generate a skeleton C# application. The names of the classes, methods, and parameters in the skeleton C# application are derived from the equivalent names in the WSDL file. A C# developer must then add business logic to the C# skeleton application.

This development technique is preferred in environments where unique systems must communicate data with each other through a common medium. By using primitive XML types in the WSDL interface and then mapping to platform-specific types, the likelihood of platform-interoperability increases dramatically. See Figure B-3 on page 486 for an illustration of how WSDL First development works.

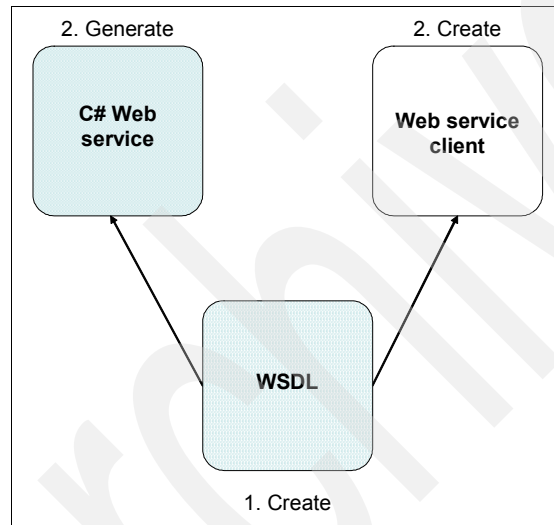


Figure B-3 WSDL First development

B.2 Implementing a Microsoft .NET Web service

This section provides the user with a detailed explanation on implementing a Web service using Microsoft technologies. It has the following sections:

- ▶ B.2.1, “Create a new Web service project”
- ▶ B.2.2, “Generating a C# file using a WSDL file and wsdl.exe”
- ▶ B.2.3, “Modifying the C# file”
- ▶ B.2.4, “Finalizing and deploying the Web service”
- ▶ B.2.5, “Testing the .NET Web service”

Important: This implementation was completed using the following software:

- ▶ Microsoft Windows 2000 Server Edition
- ▶ Microsoft Visual Studio .NET 2003
- ▶ Microsoft .NET Framework V1.1
- ▶ Microsoft Internet Information Services V5.0
- ▶ Microsoft Certificate Services for Windows 2000 Server

B.2.1 Create a new Web service project

Microsoft Visual Studio .NET 2003 provides a project creator that generates several important aspects of the Web service being created. For our purposes, we will be using this software to create a new ASP .NET Web service project.

1. Start Microsoft Visual Studio .NET 2003 by selecting **Start** → **Programs** → **Microsoft Visual Studio .NET 2003** → **Microsoft Visual Studio .NET 2003**.
2. Select **File** → **New** → **Project**.
3. Under Project Types, select the **Visual C#® Projects** folder. Under Templates, select **ASP.NET Web service**.
4. In the **Location** text box, remove the existing text and replace it with `http://localhost/ManufacturerB`, and then click **OK**:

Tip: Ensure that you have started the **IIS Admin Service** and **World Wide Publishing Service** services. The project will not load properly if the Web server hosting the Web service is in a stopped or suspended state.

5. Now the Web service is open but we must select **Click here to switch to code view** in order to see the service's C# code. This can also be done by right-clicking **Service1.aspx** in the **Solutions Explorer** and selecting **View code**.
6. Right-click **Service1.aspx** in the **Solutions Explorer** frame and **Rename** the Web service from `Service1.aspx` from to `ManufacturerB.aspx`.
7. Close Microsoft Visual Studio .NET 2003.

B.2.2 Generating a C# file using a WSDL file and wsdl.exe

The next step in creating a Microsoft .NET Web service is to generate a template Web service source code file based on the XML schemas and operations described in a WSDL file. For our scenario, we use the `Manufacturer.wsdl` file found with the additional materials supplied with this redbook. For instructions on

how to obtain this additional material, see Appendix A. "Additional material" on page 481.

Microsoft Visual Studio .NET 2003 is packaged with an application called `wsdl.exe`. It parses a WSDL file and other files referenced in the WSDL, and generates a C# skeleton based on the WSDL interface.

1. Copy the contents of the `\dotNET\resources` directory from the additional material supplied with this Redbook to a temporary directory on your local machine.
2. Click **Start** → **Programs** → **Microsoft Visual Studio .NET 2003** → **Visual Studio .NET Tools** → **Visual Studio .NET 2003 Command Prompt**.
3. In the resulting command prompt, navigate to the location of the temporary directory where you extracted the additional material.
4. Enter the following command:

```
wsdl.exe /server Manufacturer.wsdl Configuration.xsd ManufacturerP0.xsd  
ManufacturerSN.xsd envelope.xsd
```

Note: For more details on the `wsdl.exe` application and how it functions, including a list of all usable attributes, please visit:

<http://msdn.microsoft.com/library/en-us/cptools/html/cpgrfWebServicesDescriptionLanguageToolWsdl.exe.asp>

The output should look similar to Example B-1:

Example: B-1 wsdl.exe command output

```
Microsoft (R) Web Services Description Language Utility  
[Microsoft (R) .NET Framework, Version 1.1.4322.573]  
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.  
  
Writing file 'C:\Temp\output.cs'.
```

We now have a template C# source code file to use as the foundation for our Web service that is called `output.cs`.

B.2.3 Modifying the C# file

The code in `output.cs` contains an abstract class for the Web service because we chose to use the `/server` option when running the `wsdl.exe` application. Therefore, we must edit the C# code directly in order to turn the abstract class into a concrete one. We must also make additional modifications to the code so that Web service consumers will have information returned to them when they call it using SOAP/HTTP.

1. Open the output.cs file in a text editor.
2. Make the first necessary modification to the code by removing the two **abstract** statements, which are seen bolded in Example B-2:

Example: B-2 The output.cs abstract class modifications

```
[System.Web.Services.WebServiceBindingAttribute(Name="ManufacturerSoapBinding",
Namespace="http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10
/Manufacturer.wsdl")]
public abstract class ManufacturerSoapBinding : System.Web.Services.WebService
{

    public ConfigurationType Configuration;

    public StartHeaderType StartHeader;

    /// <remarks/>
    [System.Web.Services.Protocols.SoapHeaderAttribute("Configuration")]
    [System.Web.Services.Protocols.SoapHeaderAttribute("StartHeader")]
    [System.Web.Services.WebMethodAttribute()]
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute("",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Bare)]
    [return: System.Xml.Serialization.XmlElementAttribute("ackPO",
Namespace="http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10
/ManufacturerPO.xsd")]
    public abstract string
submitPO([System.Xml.Serialization.XmlElementAttribute(Namespace="http://www.ws
-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer" +
"PO.xsd")] PurchOrdType PurchaseOrder);
}
```

3. In addition, remove the semicolon at the end of the ManufacturerSoapBinding class (its located in the second to last line shown in Example B-2) and add the following C# code:

```
{
    return "Manufacturer_B has received and processed a request.";
}
```

This simple return statement is what the Web service will return to the consumer.

4. Finally, we must comment some of the generated source code in order to avoid errors when the Web service is targeted by the J2EE consumer. Comment out these two lines by adding comment marks (//) in front of them:

```
[System.Xml.Serialization.XmlAttributeAttribute(Namespace="http://schemas.x
mlsoap.org/soap/envelope/")]

public bool mustUnderstand;
```

Important: The first and second lines of the code displayed here appear on one line in the actual source code.

5. Now, with these modifications made, the two main sections of changed code should be identical to Example B-3, except where the text returned to the Web service consumer is customized to the user's preference.

Example: B-3 Final code for the output.cs C# source code file

```
[System.Web.Services.WebServiceBindingAttribute(Name="ManufacturerSoapBinding",
Namespace="http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10
/Manufacturer" +
".wsdl")]
public class ManufacturerSoapBinding : System.Web.Services.WebService {

    public ConfigurationType Configuration;

    public StartHeaderType StartHeader;

    /// <remarks/>
    [System.Web.Services.Protocols.SoapHeaderAttribute("Configuration")]
    [System.Web.Services.Protocols.SoapHeaderAttribute("StartHeader")]
    [System.Web.Services.WebMethodAttribute()]
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute("",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Bare)]
    [return: System.Xml.Serialization.XmlElementAttribute("ackPO",
Namespace="http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10
/Manufacturer" +
"PO.xsd")]
    public string
    submitPO([System.Xml.Serialization.XmlElementAttribute(Namespace="http://www.ws
-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer" +
"PO.xsd")] PurchOrdType PurchaseOrder)
    {
        return "Manufacturer_B has received and processed a request.";
    }
}
...

//[System.Xml.Serialization.XmlAttributeAttribute(Namespace="http://schemas.xml
soap.org/soap/envelope/")]
//public bool mustUnderstand;

...
```

6. Save your changes to this text file.

B.2.4 Finalizing and deploying the Web service

Now we must take our C# file, which was generated with `wsdl.exe` and hand-edited to meet our needs, and import it into the existing Web service we created with Microsoft Visual Studio .NET 2003. This means that we must replace our file with the one generated by Visual Studio when we first created our Web service project.

1. Navigate to the temporary directory where you have saved `output.cs` using Windows Explorer, and rename the file from `output.cs` to `ManufacturerB.asmx.cs`, as shown in Figure B-4:

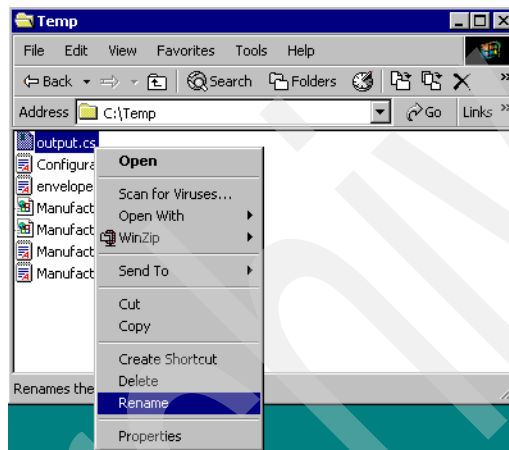


Figure B-4 Renaming `output.cs` to `ManufacturerB.asmx.cs`

2. Navigate to the location where Microsoft Visual Studio .NET 2003 has saved our Web service using another instance of Windows Explorer, (in our example, `C:\Inetpub\wwwroot\ManufacturerB`), then copy the `ManufacturerB.asmx.cs` file from the temporary directory to here as shown in Figure B-5 on page 492:

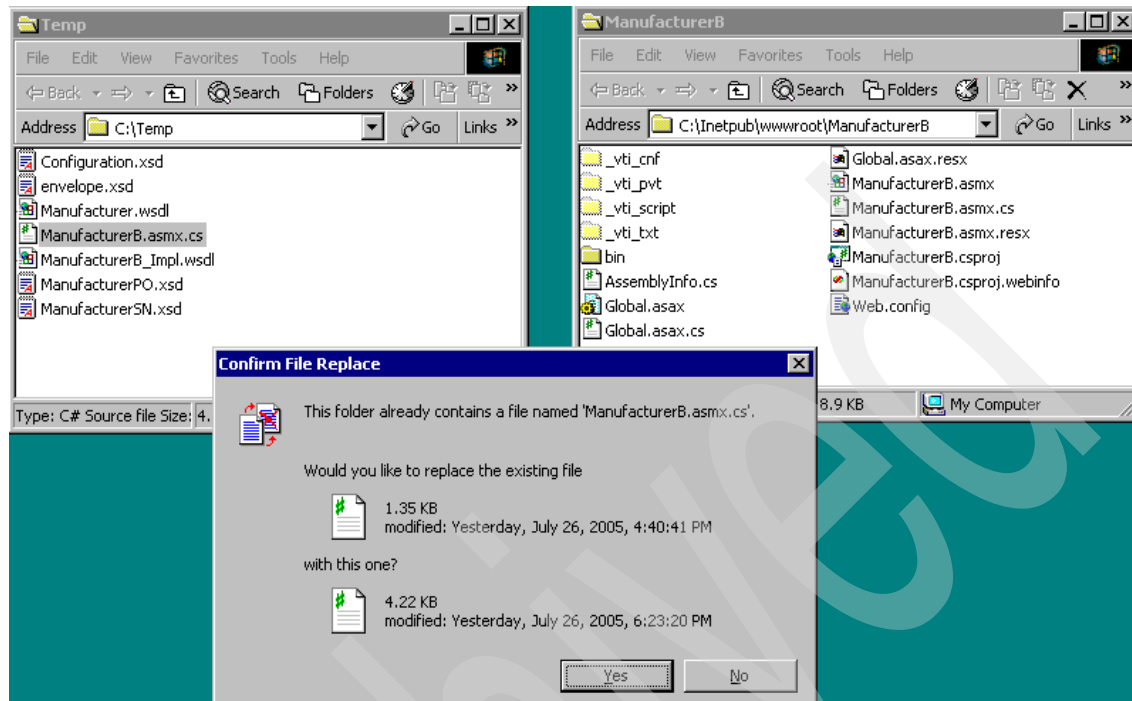


Figure B-5 Replacing Microsoft Visual Studio .NET 2003's generated C# file with the edited C# file

Important: Microsoft prompts you with a warning about overwriting the existing file. Accept the overwrite.

3. Now, start Microsoft Visual Studio .NET 2003 by selecting **Start** → **Programs** → **Microsoft Visual Studio .NET 2003** → **Microsoft Visual Studio .NET 2003**, and open the **ManufacturerB** project.
4. Now, with the Web service open, select **Click here to switch to code view** in order to see the service's C# code. Make sure that the code now appearing in this file is identical to the code we just generated and edited.

We have now created and deployed a .NET Web service using Microsoft Visual Studio .NET 2003. The next steps explain how to test for operability from within Visual Studio. Do not close the ManufacturerB.asmx Web service; we use it for testing purposes in the next section.

B.2.5 Testing the .NET Web service

Now we have a complete Web service. As the previous instructions suggested, return to the opened Web service at this time in order to test for basic operability.

1. With the `ManufacturerB.asmx` source code still open, select **Build** → **Build ManufacturerB** from the menu bar. After it has completed its build, a message in the output frame (located at the bottom of the interface) should state:

Build: 1 succeeded, 0 failed, 0 skipped

2. Now, with the Web service successfully built, we can test for operability by pressing F5, which loads a Web service test page based on our `ManufacturerB.asmx` Web service. The Web browser window that appears should look similar to the window shown in Figure B-6.

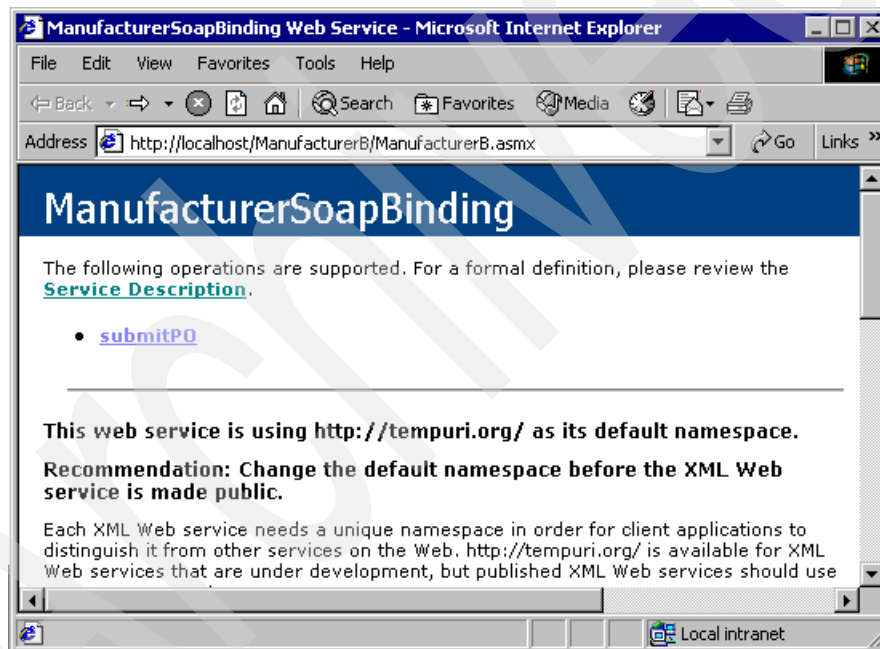


Figure B-6 Testing the Web service

3. By clicking on the **submitPO** link, it loads another browser instance with the following message, followed by a sample SOAP message request and response:

The test form is only available for methods with primitive types or arrays of primitive types as parameters.

This response is normal because our Web service does not use primitive types as parameter. The test is only to prove operability at a basic level.

We have now implemented a Web service successfully using Microsoft Visual Studio .NET 2003. The next section describes how to develop a test J2EE client using IBM Rational Application Developer that will consume our .NET Web service.

Implementing a test J2EE Client

This section describes the implementation of a test J2EE client using IBM Rational Application Developer V6. We use it to prove interoperability between the ManufacturerB .NET Web service (a target) and a Java-based client (consumer). This section has the following sub-headings:

- ▶ B.2.6 "Creating a new Rational Application Developer project" on page 494
- ▶ B.2.7 "Importing the necessary WSDLs and XSDs" on page 495
- ▶ B.2.8 "Deploying and testing the J2EEClient" on page 497

B.2.6 Creating a new Rational Application Developer project

To create a new Web project in Rational Application Developer, perform the following steps:

1. Start Rational Application Developer V6.
2. From the main application window, select **File** → **New** → **Project**.
3. From the New project window, accept the default selection of **Dynamic Web Project** and click **Next**.
4. The application now prompts you to enter a project name. Enter J2EEClient, as shown in Figure B-7, and click **Finish**. The application then begins building the dynamic Web project through the use of a creation wizard.

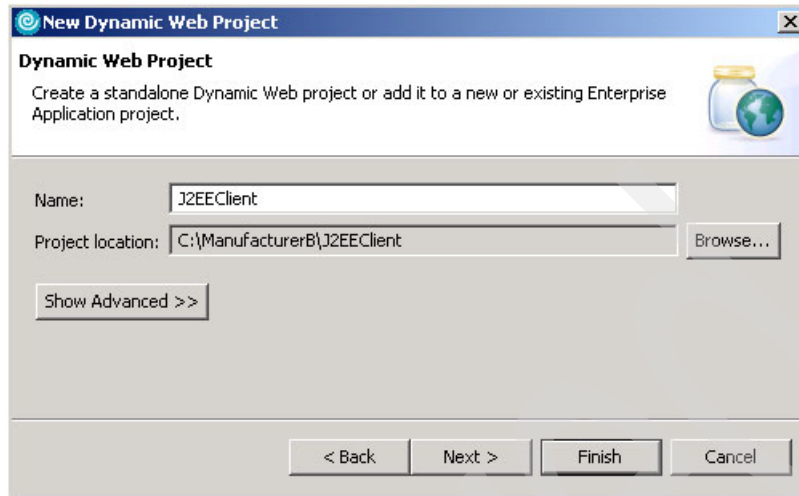


Figure B-7 Naming the Web project

5. When prompted with the Confirm Perspective Switch window, click **Yes**.

We have now created the template for a J2EE client that will consume a .NET Web service. The next step is to examine import ManufacturerB_Impl.wsdl, and its related files, into this project.

B.2.7 Importing the necessary WSDLs and XSDs

To this point, we have created a dynamic project template in IBM Rational Application Developer V6.0 that will serve as the J2EE test client consuming a .NET Web service. Now the ManufacturerB_Impl.wsdl file, and the other files that it calls in its import statements, must be added to the dynamic Web project template in order to complete a sample Web service request and response scenario.

1. We need to import the following files into the J2EEClient Rational Application Developer project. These files can be found in the additional material supplied with this redbook in the dotNet\resources directory:
 - Manufacturer.wsdl
 - ManufacturerB_Impl.wsdl
 - Configuration.xsd
 - envelope.xsd
 - ManufacturerPO.xsd
 - ManufacturerSN.xsd
2. In Rational Application Developer, expand the **Dynamic Web Projects** folder, right-click the **J2EEClient** project and select **Import** → **Import**.

3. Inside of the Import window, select **File system** and click **Next**.
4. In the **From directory** field enter the location of the additional material stored in the dotNet\resources directory. Then check each of the following files (Figure B-8):
 - Manufacturer.wsdl
 - ManufacturerB_Impl.wsdl
 - Configuration.xsd
 - envelope.xsd
 - ManufacturerPO.xsd
 - ManufacturerSN.xsd

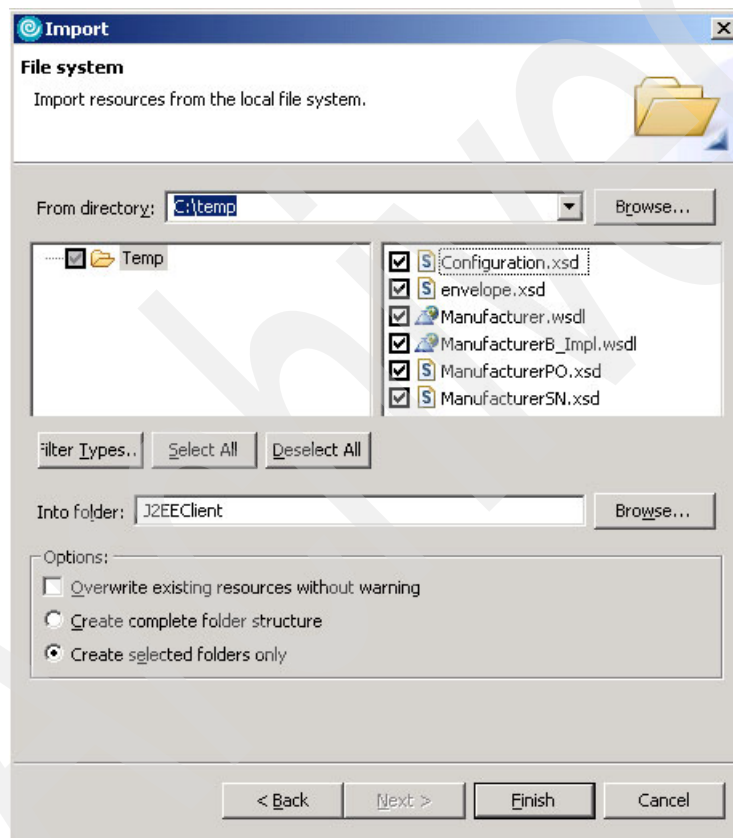


Figure B-8 Importing the six files essential to the functioning of J2EEClient.

5. Click **Finish**. The imported WSDL files contain a number of warnings. These can be ignored.
6. Open the newly imported **ManufacturerB_Impl.wsdl** in a text editor in Rational Application Developer. Notice the following line:

```
<wsdlsoap:address  
location="http://itsogood.itso.ra1.ibm.com/ManufacturerB/ManufacturerB.asmx  
"/>
```

Change this line to point to the address where the .NET Web service is running, for example:

```
<wsdlsoap:address  
location="http://1.2.3.4/ManufacturerB/ManufacturerB.asmx"/>
```

7. Save changes to this file.

B.2.8 Deploying and testing the J2EEClient

Before we begin to deploy and test the client, we must first modify preferences in Rational Application Developer that allow us to utilize the Web Services Explorer, a powerful tool used by the application to test Web service calls. Perform the following, tasks:

1. From the menu bar, select **Window** → **Preferences**.
2. In the Preferences window, expand the **Workbench** listing and click **Capabilities**.
3. Inside the **Capabilities** frame, expand the **Web Developer (advanced)** listing and place a check in the box next to **Web Services Development**, as seen in Figure B-9 on page 498.

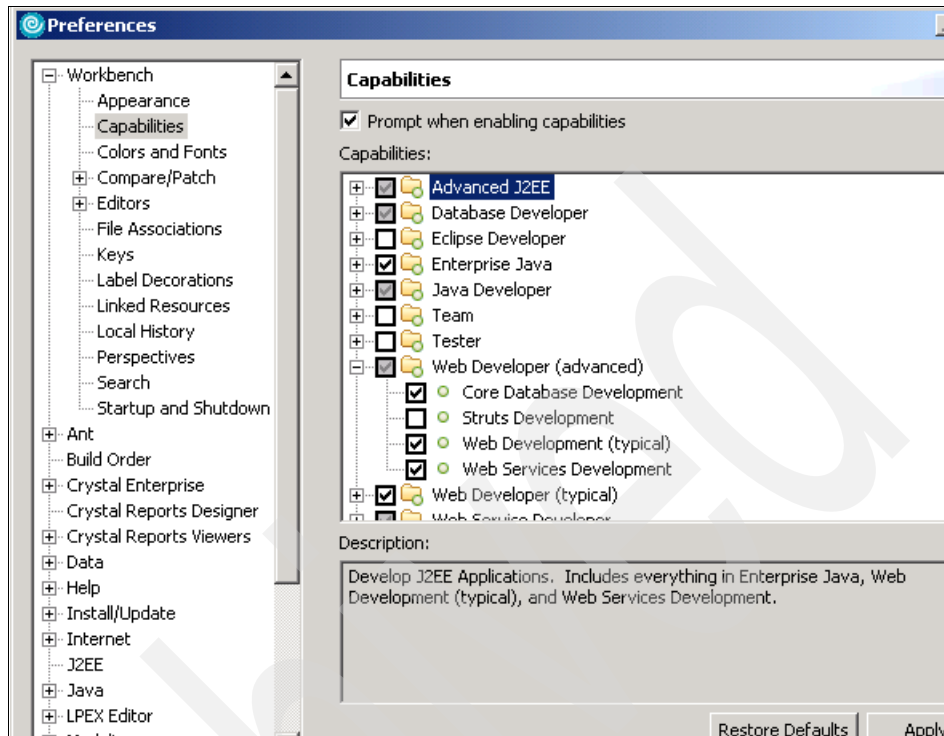


Figure B-9 Enabling Web Services development tools.

4. Click **OK**. Now we can begin to deploy and test the J2EE client properly.
5. In the Project Explorer view, right-click **ManufacturerB_Impl.wsdl** and select **Web Services** → **Test with Web Services Explorer**.
6. After the environment is loaded, you can see a Web browser tab in Rational Application Developer. In the action frame, the submitPO operation and a properly defined endpoint are shown, as seen in Figure B-10 on page 499.

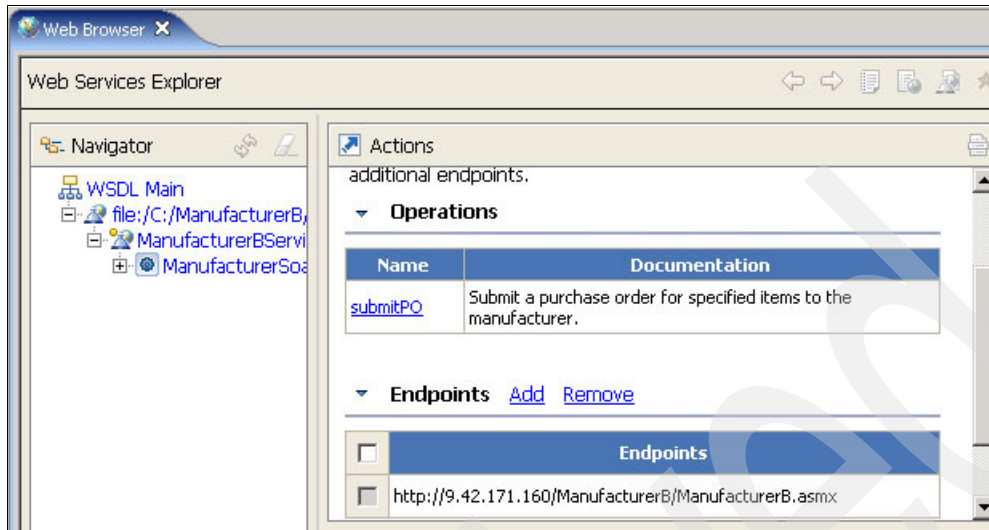


Figure B-10 Testing with the Web Services Explorer

7. Click **submitPO** inside of the Actions frame and provide test input for each of the following items listed on the screen. This is the information that the .NET Web service will receive in a SOAP message.

Important: Obey the input guidelines for each object type that requires information (for example, do not write text inside of inputs that require numeric values, such as *float* or *nonNegativeInteger*).

Also, the customerRef object is of a *normalized string* type, meaning that it has special restrictions that must be considered when choosing a value. We recommend using the following value for this object: A12345-9876543-xyz. It should be inputted exactly as its written here, paying special attention to the two dashes and the upper and lower-case letter specifics contained within it. This is illustrated in Figure B-11 on page 500.

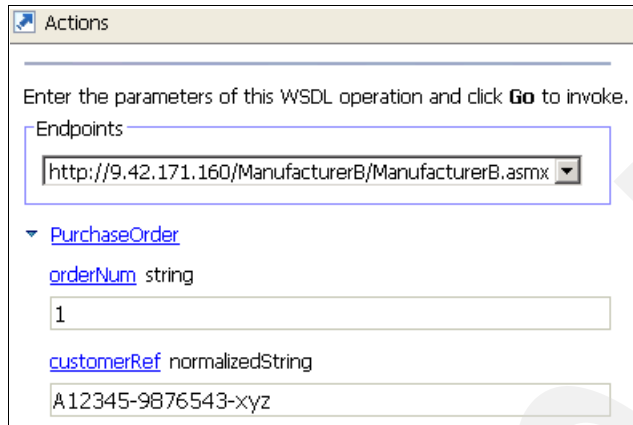


Figure B-11 An example of a special needs object type, *customerRef*.

8. After inputting all of the values, click **Go** at the bottom of the **Actions** frame.
9. If everything runs properly, the message that is shown in the Status frame of the Web Services Explorer should be identical to the message coded into the .NET Web service. This response is shown in Figure B-12.

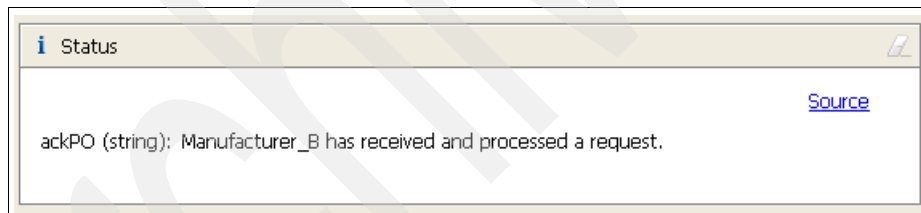


Figure B-12 Successful response from the .NET Web service

10. Click the Source link to see the SOAP request and response messages used during the Web services interaction.

Congratulations! You have implemented J2EE client consumer that has successfully sent to and received messages from a .NET Web service. We now explain how to enable Microsoft Internet Information Server to require transport-level security using SSL for all incoming requests.

B.3 Enabling transport-level security with SSL

In remaining consistent with the Extended Enterprise context, it is common security practice to introduce a layer of transport-level security between a Web service provider and client. This security takes the form of SSL/HTTP in order to

encrypt the entire message being sent. We now implement message-level security in this scenario for Internet Information Server V5.

B.3.1 Configuring the .NET Web service to require SSL

The .NET Web service depends on Microsoft Internet Information Services (IIS) in order to provide SSL support. Therefore, we must begin by visiting the IIS console and the .NET directory previously created there. From here, we request, then install, a server certificate for the usage of SSL between the .NET Web service and J2EE client.

Tip: In our sample scenario, we chose not to request the necessary server certificate from an outside Certificate Authority (CA); we chose to use the Microsoft Certificate Services to generate our server certificate. Microsoft Certificate Services is a Windows 2000 Server component that can be added to the operating system from the Windows 2000 Server CD, if it has not been installed previously.

Requesting a server certificate

Perform the following where the .NET Framework and IIS are installed:

1. Click **Start** → **Programs** → **Administrative Tools** → **Internet Services Manager** to load the IIS console.
2. Navigate to the Web page where the ManufacturerB virtual directory exists. If you followed prior instructions, this would be located in the **Default Web Site** virtual root.
3. Right-click **Default Web Site** and select **Properties**.
4. In the properties window, navigate to the **Directory Security** tab and click the **Server Certificate** button. This loads the Web Server Certificate Wizard.
5. Click **Next** to bypass the welcome dialog box of the wizard.
6. Select the **Create a new certificate** radio button and click **Next**.
7. In the next window, select **Prepare the request now, but send it later** radio button and click **Next**.
8. Here, input Manufacturer B Web Server in the Name field and select **1024** in the bit length drop-down menu (Figure B-13 on page 502) and click **Next**.

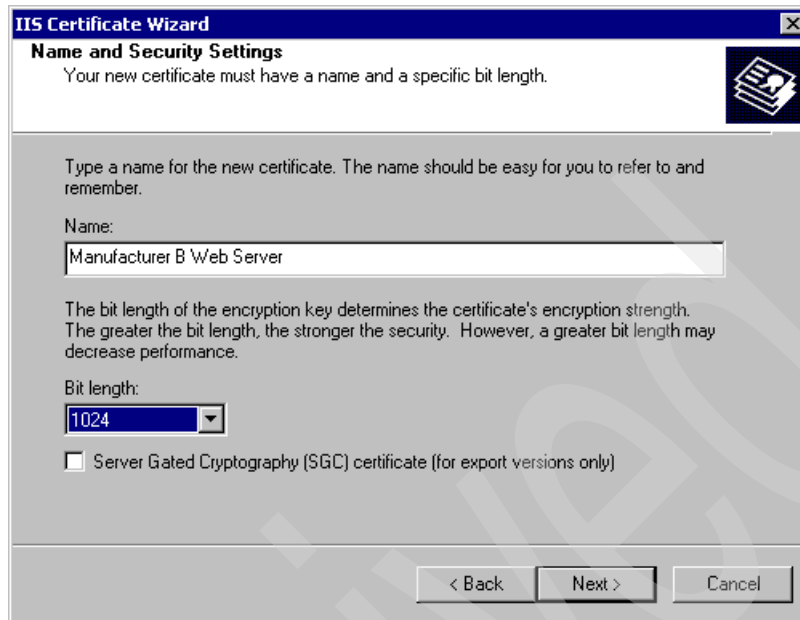


Figure B-13 Input a common name and bit length for the server certificate

9. Under Organization, input **ManufacturerB** and under **Organizational Unit**, input **Manufacturing**. Click **Next**.
10. For the **Common name** field, input the IP address or fully qualified host name of the machine running the .NET Web service. We used our web server's IP address. Click **Next**.
11. Input pertinent geographical information in the following dialog box and click **Next**.
12. The certificate request will be saved in a .txt file for future usage. Here, name the .txt file to something easily remembered and save it in a specific directory, or leave it as the default name and location as we did (C:\certreq.txt). Click **Next**.
13. The final dialog box is just a summary of all previously-entered information. Ensure that it is all correct and click **Next**.
14. Click **Finish** to exit the server certificate request wizard.

We have now made a request for a server certificate that will enable the .NET Web server to communicate with SSL.

Processing the Web server certificate request

In this section, we use the Microsoft Certificate Services in order to process the Web server certificate request previously generated.

1. Open a Web browser and navigate to:

`http://localhost/CertSrv`

Where `localhost` should be replaced with the domain name or IP address of the server running the Microsoft Certificate Services.

The Microsoft Certificate Services Web page should be displayed as shown in Figure B-14.

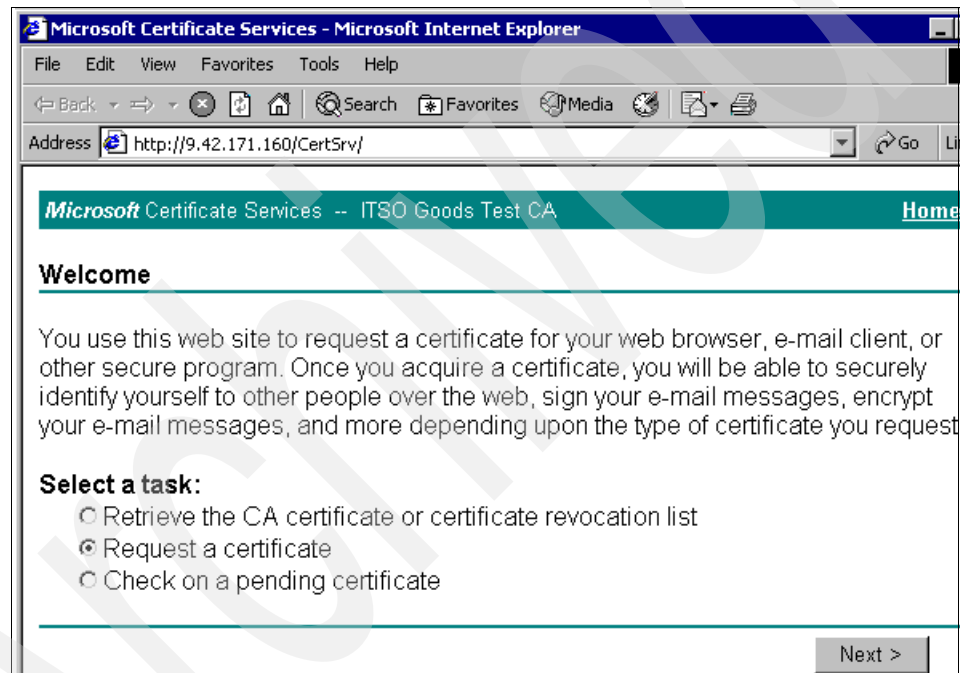


Figure B-14 The Microsoft Certificate Services application

2. Select **Request a certificate** and click **Next**.
3. On the next page, select **Advanced request** and click **Next**.
4. Select the **Submit a certificate request using a base64 encoded PKCS #10 file or a renewal request using a base64 encoded PKCS #7 file** option and click **Next**.
5. The following window contains a text input box that allows a user to input a base64 encoded PKCS #10 certificate request.

Use a text editor to open the certificate request .txt file that was created using the Web server certificate request wizard in the previous section (in our case C:\certreq.txt). When the file has been found, copy and paste its entire contents into the **Saved Request** input box as shown in Figure B-15. Then, click **Submit**.

6. The following window is a confirmation window informing the user that the certificate request has been received and is waiting to be processed. Exit this window.

We now have a certificate request that is waiting to be issued.

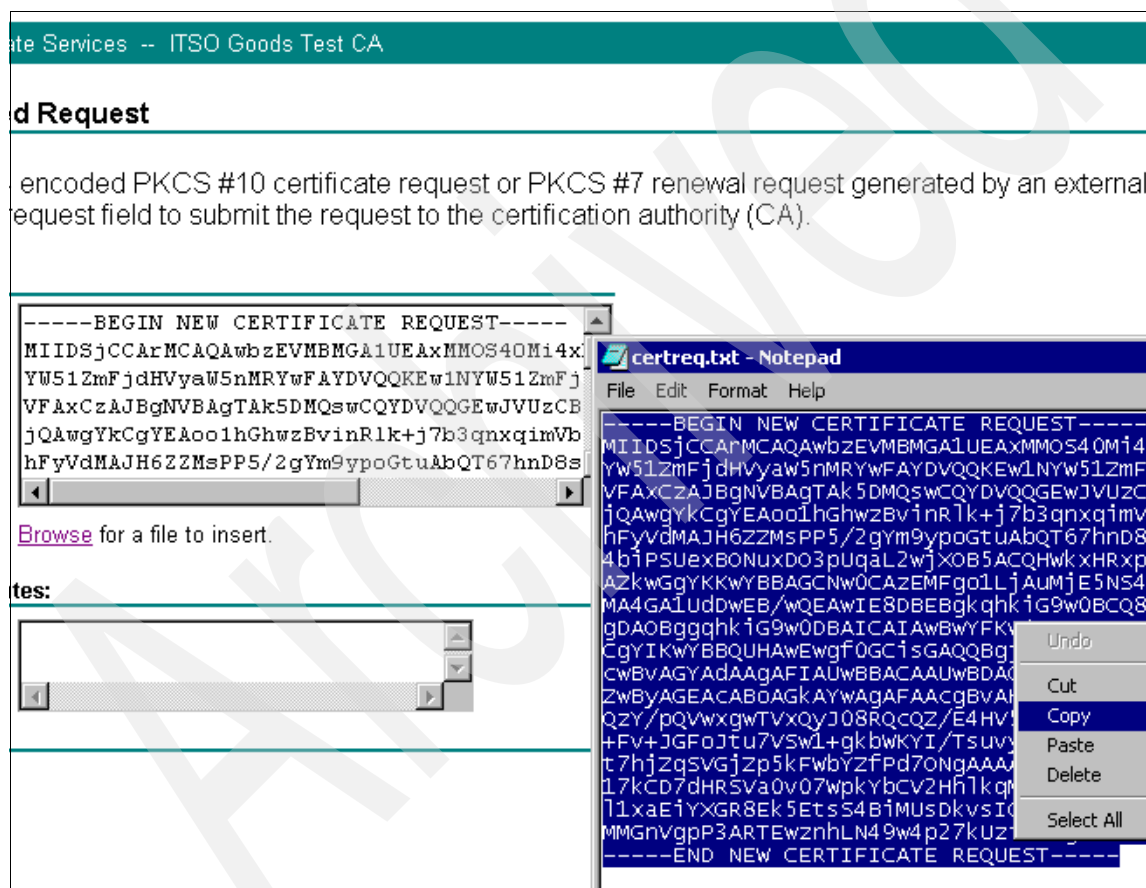


Figure B-15 Copying the certificate request to the Saved Request dialog box.

7. Click **Submit**. The resulting window is a confirmation window informing the user that the certificate request has been received and is waiting to be processed. Exit this window.

We now have a certificate request that is waiting to be issued.

Issuing the Web server certificate

We now have a Web server certificate request waiting to be issued. To issue the Web server certificate, complete the following steps.

1. Click **Start** → **Programs** → **Administrative Tools** → **Certification Authority**.
2. Expand the name of the CA and open the **Pending requests** folder, as seen in Figure B-16.

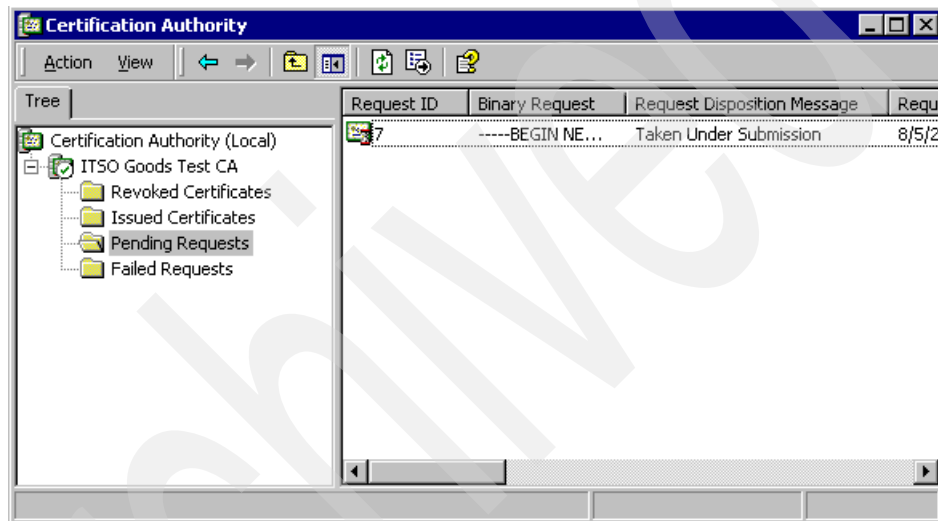


Figure B-16 Viewing the Pending Requests in the Certification Authority console

3. Right click the request we made in the previous section and select **All Tasks** → **Issue**. If there is more than one request listed, sort them on the **Request ID** data field and select the highest numbered request.
4. Now, with the pending request issued, select the **Issued Certificates** folder. Inside here, our issued Web server certificate is listed. If there is more than one listed, sort them on the **Request ID** field and locate the highest numbered request.
5. Double-click the certificate. Then, click the **Details** tab and select **Copy to file** to load the Certificate Export Wizard.
6. Bypass the welcome dialog box by selecting **Next**.
7. In the Expected File Format dialog box, select **Base-64 encoded X.509 (.CER)** and click **Next**.

8. In the File to Export dialog box, enter a filename and directory where you want the certificate to be saved (we entered C:\Man_B_server.cer). Click **Next**.
9. In the summary dialog box, double-check all of your settings, then click **Finish**. A prompt stating the export was successful will appear: click **OK** to close it.

Now, we are ready to install the certificate onto the Web server.

Installing the Web server certificate

1. Click **Start** → **Programs** → **Administrative Tools** → **Internet Services Manager** in order to load the IIS console.
2. Navigate to the Web page where the ManufacturerB virtual directory exists. If you followed prior instructions, this would be located in the Default Web Site virtual root.
3. Right-click **Default Web Site** and select **Properties**.
4. Navigate to the **Directory Security** tab and click the **Server Certificate** button to load the Web Server Certificate Wizard.
5. Click **Next** to bypass the welcome dialog box of the wizard.
6. Select **Process the pending request and install the certificate** and click **Next**.
7. In the following dialog box, enter the directory name and filename of the certificate that was just exported (we entered C:\Man_B_server.cer) and click **Next**.
8. The following dialog box shows a summary of the certificate's information: double-check it and then click **Next**.
9. Now, with the certificate installed, simply close the concluding dialog box by clicking **Finish**. Do not close IIS, however. We use it again in the next set of instructions.

A security certificate has now been installed for the Web server. Next, we must edit the security settings for the server in order to require SSL connections to the .NET Web service.

Configuring the .NET Web service to require SSL connections

Attention: Though, in theory, we could enable the Web server to require SSL connections for the entire Web page in which the .NET client virtual directory is located, for the sake of this exercise, we configure the server to require SSL connections for **only** the .NET client, and not other resources located inside of this Web page.

In proper business scenarios, however, best practice would most likely mandate securing all resources on the Web server. This follows the exact instructions described here, with the exception that you select the entire Web page in the first step, instead of just one resource inside of it.

1. From the IIS console, expand the **Default Web Site** (or the name of the Web page where your .NET Web service virtual directory is located), right-click **ManufacturerB** and select **Properties**.
2. Click the **Directory Security** tab and select **Edit** in the Secure communications area.
3. Inside the Secure communications dialog box, put a check-mark in the box labeled **Require secure channel (SSL)**. Accept all other defaults and click **OK**.
4. Finally, click **OK** to close the Properties window.

We have completed the steps necessary to require SSL communication between the Web server running the .NET Web service and any incoming requestors. The final steps in completing SSL configuration involve importing the Web server certificate into a key database that a client can reference when attempting connections to the secure source.

B.3.2 Importing the SSL certificate into a key database

We must now export the certificate installed onto the Microsoft Web server and then import it into a key database that can be utilized by a client when sending messages to that Web server.

Attention: Provided with the sample code for this Redbook is a key database file that the runtime scenarios reference when sending SSL-secured messages to Manufacturer A, B, and C. Therefore, the database (saved as a .kdb file) contains the certificates required to connect to ManufacturerA, ManufacturerB, and ManufacturerC

If you have implemented the runtime scenarios in this book and followed the instructions for creating a .NET Web service in this Appendix, you can use that key database to complete SSL-enablement, effectively nullifying your need to complete the following steps.

However, if you are implementing this .NET Web service independent of the runtime scenarios in this book, then these instructions must be followed to allow clients to access the Web service.

Export the Microsoft .NET Web server certificate to a file

Tip: If your Microsoft .NET Web service is received requests directly from a Web browser, then these steps are not necessary because the Web browser will automatically prompt you to accept the certificate directly from the Web server. However, if this Microsoft .NET Web service is being sent requests from a Web service client, then the certificates must be imported into a key database that the client can use because the user will never be prompted to accept the certificates for the client's sake.

First, we must export the certificate from the Web server's certificate store so that it can then be imported into a key database.

1. Click **Start** → **Programs** → **Administrative Tools** → **Internet Services Manager** in order to load the **IIS console**.
2. Navigate to the Web page where the ManufacturerB virtual directory exists. If you followed prior instructions, this would be in the Default Web Site virtual root.
3. Right-click **Default Web Site** and select **Properties**.
4. Click the **Directory Security** tab and click **View Certificate**.
5. Click the **Details** tab and click **Copy to file**.
6. Click **Next** to bypass the welcome dialog box to the Certificate Export Wizard.
7. Choose **Yes, export the private key** and click **Next**.

Tip: Because we chose to export the private key, we have a certificate for the Manufacturer B Web Server certificate, as well as one for ITSO Good, which is the root certificate of the CA that authorized the creation of the ManufacturerB Web Server certificate.

8. In the Export File Format dialog box, put a check in the box next to **Include all certificates in the certification path, if possible** and **Enable strong protection (requires IE 5.0, NT 4.0 SP4 or above)** and click **Next**, as shown in Figure B-17 on page 509.

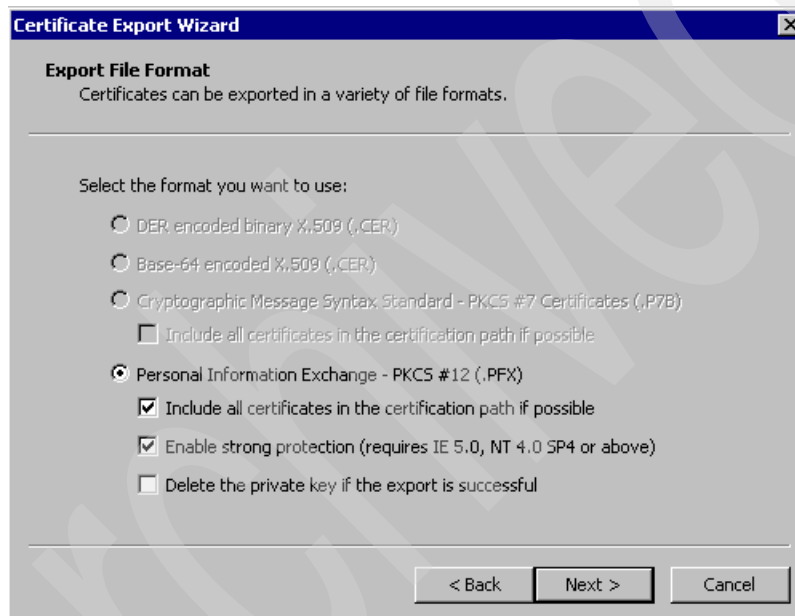


Figure B-17 Choosing the correct options for Export File Format

9. In the following window, enter a password into the fields and click **Next**.
10. In the next window, enter a path and filename where you would like the certificate to be exported (we entered C:\Temp\ManBCert.pfx) and click **Next**.
11. The final dialog box is just a summary of all entered information. Double-check your selections and click **Finish**. Click **OK** to the message that says the export was completed successfully."

Importing the certificate into the Microsoft Certificate Store

We now have the Microsoft .NET Web service certificate exported from its Web server. Next, we must import it into the Microsoft Certificate Store. Then, we

conclude this exercise by taking the certificate from the Microsoft Certificate Store and importing it into a key database file.

1. Using Windows Explorer, navigate to the location of the exported certificate (C:\Temp\ManBCert.pfx, in our case).
2. Double-click it to open the Certificate Import Wizard. Click **Next** to bypass the opening dialog box.
3. In the next dialog box, the path and filename of the certificate being imported should already be entered into the File name field. If not, enter the path and filename of ManBCert.pfx and click **Next**.
4. Enter the password used when exporting the certificate in the next dialog box, put a check in the box labeled **Mark the private key as exportable**, and click **Next**.
5. In the following dialog box, select the **Place all certificates in the following store** radio button, then click **Browse** and select **Trusted Root Certification Authorities** from the list of stores. Then click **OK** and select **Next**.
6. The final dialog box is just a summary of selections. Double-check them and click **Finish**. Click **Yes** to any warnings about installing certificates then click **OK**.

Now, with the certificate stored in the Microsoft Certificate Store, we use iKeyman to create a new key database and place the newly stored certificate into the database.

Extracting and importing into a new key database file

To complete this step you must have the iKeyman utility installed on the same machine as the Microsoft Certificate Store. The iKeyman utility is packaged with IBM HTTP Server V6.

1. Open the **iKeyman** utility by clicking **Start → Programs → IBM HTTP Server 6.0 → Start Key Management Utility**.
2. Next, select **Key Database File → Open**. Then, select **Microsoft Certificate Store** from the Key database type dropdown menu and click **OK**.
3. In the Key database content frame, select **Signer Certificates** from the drop-down menu and locate the **{ITSOGood}** certificate from the list. Single-click it, then click the **Extract Certificate** button.

Note: We select {ITSOGood} because it was the name we gave to the Microsoft Certificate Authority when we installed the Microsoft Certificate Services for Windows 2000 Server add-on component to the Web server. If you gave the Microsoft CA a different name, choose your name for it, instead.

4. In the Extract Certificate to a File window, type in a certificate file name and location (we entered ManBCA.arm and C:\keys). Leave the Data type field as its default. Click **OK**.

Now, we must take the extracted certificate and import it into a new key database file for use by a Web service client (like the one utilized in the runtime scenarios of this Redbook).

Creating a new key database file

To create a new key database file, follow these steps:

1. From the **Key Database File** menu, select **New**.
2. In the Key database type drop-down menu, select **CMS**. In the File Name field, enter a name for the key database (we entered ihs.kdb), and in the Location field, enter a directory (we entered c:\keys). Click **OK**.
3. In the Password Prompt message box, enter a password and confirm it, then set an expiration time for the password. We entered 1000 for the expiration time. Finally, put a check mark in the box next to **Stash the password to a file** label and click **OK**.
4. Now, in the main window, select **Add**. This begins the process of adding the root CA certificate and the Microsoft .NET Web server certificate into the newly created key database.
5. In the Add CA's Certificate from a File window, click **Browse**. In the **File name** field, enter the location of the recently extracted root CA certificate (for example, C:\keys\ManBCA.arm). Click **OK**.
6. Enter a label for the certificate (we entered Root CA Certificate) and click **OK**.

We have now created a key database file that a J2EE client can utilize in making Web service requests to the Microsoft .NET Web service. You can copy this key database file to the IBM HTTP Server that will be used to redirect Web services calls to the .NET Web service.

Archived

CICS Transaction Server Web services

This chapter provides an overview of the CICS Transaction Server V3.1 support for Web services and provides high level steps for creating a ManufacturerC Web service for use in the scenario chapters in Part 3 of this redbook.

It contains the following sections:

- ▶ CICS Transaction Server V3.1 Web services support
- ▶ Creating Web services for CICS
- ▶ Creating and hosting a ManufacturerC Web service

For more detailed information about configuring CICS support for Web services, consult the CICS Transaction Server V3.1 InfoCenter at:

<http://publib.boulder.ibm.com/infocenter/cicsts31/index.jsp>

C.1 CICS Transaction Server V3.1 Web services support

CICS Transaction Server V3.1 introduces formal support for Web services. It provides the following features:

- ▶ A CICS application can participate in a heterogeneous Web services environment as a service requester, as a service provider, or both.
- ▶ It provides support for HTTP and MQ.
- ▶ It includes the CICS Web services assistant, a set of utility programs that help you map WSDL service descriptions into high-level programming language data structures, and vice versa. The utility programs support the following programming languages:
 - COBOL
 - PL/I
 - C
 - C++
- ▶ The CICS support for Web services conforms to open standards including:
 - SOAP V1.1 and V1.2
 - HTTP V1.1
 - WSDL V1.1
- ▶ CICS support for Web services ensures maximum interoperability with other Web services implementations by conforming with the Web Services Interoperability Organization (WS-I) Basic Profile V1.0.

C.2 Creating Web services for CICS

There are two basic approaches for building Web services to run in a CICS environment:

- ▶ Bottom-up

An existing piece of business logic is wrapped as a Web service. WSDL is created to describe the Web service interface to this Web service.
- ▶ Top-down

An existing WSDL interface document is used to create a skeleton application. This skeleton represents the operations and data structures defined in the WSDL. The skeleton application is then populated with business logic.

Both bottom-up and top-down Web service implementations can be developed using the CICS Web services assistant.

C.2.1 CICS Web services assistant

The CICS Web services assistant is a set of batch utilities which can help you to transform existing CICS applications into Web services and to enable CICS applications to use Web services provided by external providers. When you use the Web services assistant for CICS, you do not have to write your own code for parsing inbound messages and for constructing outbound messages. CICS maps data between the body of a SOAP message and the application program's data structure.

The CICS Web services assistant comprises two utility programs:

- ▶ **DFHLS2WS**

This program generates a Web service binding file from a language structure. This utility also generates a Web service description. This is used for bottom-up development.

- ▶ **DFHWS2LS**

This program generates a Web service binding file from a Web service description. This utility also generates a language structure that you can use in your application programs. This is used for top-down development.

The JCL procedures to run both programs are in the `<CICS_HIGH_LEVEL_QUALIFIER>.XDFHINST` library.

C.2.2 CICS resources for Web services

The following CICS resources support Web services:

- ▶ **PIPELINE** provides information about the message handler programs that act on a service request and on the response. Typically, a single PIPELINE definition defines an infrastructure that can be used by many applications.
- ▶ **WEBSERVICE** is required only when the mapping between application data structures and SOAP messages generated using the CICS Web services assistant. It defines aspects of the runtime environment for a CICS application program deployed in a Web services setting.
- ▶ **URIMAP** is required only for a service provider, and contains information that maps the URI of an inbound Web service request to the other resources (such as the PIPELINE) that will service the request.
- ▶ **TCPIPSERVICE** is required for a service provider that uses the HTTP transport, and contains information about the port on which inbound requests are received.

C.3 Creating and hosting a ManufacturerC Web service

This section provides a high-level overview of how to implement a Web service in CICS Transaction Server using the top-down approach. This Web service implementation can be used as the ManufacturerC Web service in the scenario implementations described in this redbook.

For more detailed information about how to perform the following steps, consult the CICS Transaction Server V3.1 InfoCenter at:

<http://publib.boulder.ibm.com/infocenter/cicsts31/index.jsp>

Perform the following tasks

1. Upload Manufacturer.wsdl into Unix System Services. You will also need to upload all of the WSDL and XSD files that Manufacturer.wsdl imports. You can find all of these files in the additional material supplied with this redbook in the \DirectConnection\wsdl directory.
2. Open Manufacturer.wsdl in a text editor and modify the following line (it is the first line in the WSDL file):

```
<?xml version="1.0" encoding="UTF-8"?>
```

Remove the encoding attribute, as this attribute is incorrect for a z/OS environment. The line should now read:

```
<?xml version="1.0">
```

Repeat this for each XSD and WSDL file you upload to z/OS.

3. Run the DFHWS2LS utility, specifying Manufacturer.wsdl as the WSDL interface to use. This generate a language structure and Web service binding file.
4. Create a CICS application (in a programming language of your choice) that uses the language structure. The only requirement of this application is that it populates the ackPO Web service response message with the following text string:

```
Manufacturer_C has received and processed a request.
```
5. Configure a CICS region for Web service support, and add deploy the new ManufacturerC application to it.
6. Update the ManufacturerC_Impl.wsdl file to point to the CICS Web service you have created.

WSAdmin Automation Platform

WSAdmin Automation Platform (WAP) is a set of utility functions that sit on the top of wsadmin to provide you with an easier interface to manage your WebSphere Application Server configuration. WAP provides functional APIs and allows an individual to write his/her own automation scripts without worrying about complex wsadmin (**\$AdminConfig**, **\$AdminControl**, **\$AdminApp**) syntax. With WAP, you get functions to manage the most common WebSphere Application Server object types. WSAdmin Automation Platform was developed by David Ghazaleh.

D.1 Employing WSAdmin Automation Platform

This section provides an overview of WSAdmin Automation Platform, describes where you can download it, and provides some examples of use.

D.1.1 Overview of WSAdmin Automation Platform

WebSphere Application Server includes a scripting solution called `wsadmin`, allowing an administrator to configure and control a WebSphere Application Server installation. The *wsadmin* scripting program is a powerful, non-graphical command interpreter environment enabling you to execute administrative operations interactively in a scripting language.

This tool is intended for production environments and unattended operations. The `wsadmin` tool supports a full range of product administrative activities.

The `wsadmin` scripting tool has three modes of operation:

1. *Interactive* mode lets you enter commands and view the response on a command line prompt.
2. *Batch* mode lets you supply a set of script commands in a file that the tool executes as a program.
3. *Command* mode lets you enter a single command from the regular operating system command window and executes this one command, returning control to the operating system command shell.

The `wsadmin` tool is most often executed as a client attached to a running server. You can also run it in a local execution mode where a running server is not required; however, the function is limited to only configuration changes since a server runtime is not available to receive operational requests.

WSAdmin Automation Platform is a set of utility functions that sit on the top of `wsadmin` to provide you with an easier interface to manage your WebSphere Application Server configuration. WSAdmin Automation Platform provides functional APIs and allows you to write your own automation scripts without worrying about complex `wsadmin` syntax such as `$AdminConfig`, `$AdminControl`, and `$AdminApp`.

You will find functions to manage the most common WebSphere Application Server object types, including: Application, Application Server, Cell, DataSource, JDBCProvider, JMSProvider, JMS Server, WebSphere JMS Connection Factory, Generic JMS Connection Factory, MQSeries® JMS Connection Factory, WebSphere JMS Destination, Generic JMS Destination, JAASAuthData, MQSeries JMS Destination, NameServer, Node, Server, URLProvider, URL, Variable, VirtualHost and much more.

WSAdmin Automation Platform can be executed from any host that has WebSphere Application Server installed. WSAdmin Automation Platform can connect to WebSphere Application Server running on the local or remote host. WSAdmin Automation Platform behaves just like wsadmin, in that it accepts input from the user and then send the commands to be executed by wsadmin. You can also submit scripting language programs for execution.

D.1.2 Downloading WSAdmin Automation Platform

WSAdmin Automation Platform can be downloaded from the IBM Techdocs Web site.

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS981>

WSAdmin Automation Platform is provided in a zip file called *wap.zip*. Download and unzip *wap.zip* file into your local directory. To configure the WSAdmin Automation Platform runtime environment, follow these steps:

1. Set an environment variable, called WAP_SOURCE. This variable must point to the directory where you unzipped WAP. Example D-1 shows how to set the environment variable on Unix and Example D-2 on Windows.

Example: D-1 Setting WAP_SOURCE variable on Unix

```
export WAP_SOURCE=/opt/wap
```

Example: D-2 Setting WAP_SOURCE variable on Windows

```
set WAP_SOURCE=c:/wap
```

Note: Do not use \ (back slash); use / (forward slash) instead.

2. Add jWAP.jar file to the CLASSPATH environment variable. See Example D-3 and Example D-4.

Example: D-3 Setting CLASSPATH variable on Unix

```
export CLASSPATH=$CLASSPATH:$WAP_SOURCE/jWAP.jar
```

Example: D-4 Setting CLASSPATH variable on Windows

```
set CLASSPATH=%CLASSPATH%;%WAP_SOURCE%\jWAP.jar
```

3. Insert WAP_SOURCE directory into the PATH environment variable. See Example D-5 on page 519 or Example D-6 on page 520.

Example: D-5 Setting PATH variable on Unix

```
export PATH=$PATH:$WAP_SOURCE
```

Example: D-6 Setting PATH variable on Windows

```
set PATH=%PATH%;%WAP_SOURCE%
```

4. On the Unix platform, make sure all shell script (*.sh) files have execution permission. To change permission, type the following in Example D-7.

Example: D-7 Changing shell script permission

```
chmod 755 $WAP_SOURCE/*.sh
```

5. You are now ready to run WSAdmin Automation Platform.

D.1.3 Running WSAdmin Automation Platform

In this section, you can find information about how to run WSAdmin Automation Platform. There are three different ways to start WSAdmin Automation Platform:

- ▶ **jwap** : starts a Java command line interface.
- ▶ **jwap -gui** : starts a Java graphical user interface.
- ▶ **wapStart.sh** or **wapStart.bat** : starts a Unix or Windows command line prompt interface.

jWAP syntax

To start WSAdmin Automation Platform from a Java command line interface, use the script in Example D-8:

Example: D-8 Starting from a Java command interface

```
jwap.sh | jwap.bat [ ? | -(h)elp ]  
    [-gui]  
    [-wap_path <wap_source_directory>]  
    [-debug]  
    [-noverbose]  
    [-noexec]  
    [ -c <command> ]  
    [ -wsadmin_classpath classpath ]  
    [ script parameters ]  
    [ <WSADMIN options> ]
```

wapStart syntax

To start from a Unix or Windows command interface, using the script in Example D-9 on page 521.

```
wapStart.sh | wapStart.bat [ -? | -(h)elp ]
               [-wap_path <wap_source_directory>]
               [-debug]
               [-noverbose]
               [-noexec]
               [ -c <command> ]
               [ -wsadmin_classpath classpath]
               [ script parameters ]
               [ <WSADMIN options> ]
```

Descriptions of the options

The following options are available:

-? or **-help** shows the help message.

-gui starts WSAAdmin Automation Platform in a graphical interface. This option is only available on jWAP syntax.

-noverbose runs WSAAdmin Automation Platform in silent mode.

-debug prints debugging information. This options is used to trace the execution.

-noexec runs the commands in the JACL script but does not execute WSAAdmin Automation Platform commands. This is useful for doing a *dry run*, or practice run, of scripts.

-wap_path is the path where WSAAdmin Automation Platform is installed. If this option is omitted, the value in WAP_SOURCE environment variable will be used instead.

-c is the command to be passed to the script processor. If you need to execute a command that contains white spaces, you must surround the command with a \" (backslash double quote). For instance:

```
wapStart.sh -c \"createJDBCProvider j1 -dbtype db2xa\"
```

-wsadmin_classpath is an additional Java class path to be appended to built-in Java class path.

script parameters is any other additional parameter that you want to pass to your script. These are passed to the script in the variable. The number of parameters is also available in the *argv* variable. For instance:

```
set numberOfParameter [length $argv]
set parameter0 [lindex $argv 0]
```

WSADMIN options is any supported wsadmin options. For more details about wsadmin options use `-?` or `-h`.

An interpreter shell is created for interactive use. To leave an interactive scripting session, use the `exit` command.

Note: For the complete WSAAdmin Automation Platform utility functions syntax, refer to WSAAdmin Automation Platform documentation. The documentation can be found at `<WAP_SOURCE>/docs/wap.html`.

WSAdmin Automation Platform examples can be found under `<WAP_SOURCE>/samples` directory.

D.1.4 WSAAdmin Automation Platform examples

Start WSAAdmin Automation Platform by entering `jwap` or `wapStart` commands in a command line window or shell prompt. Example D-10 shows how to start WSAAdmin Automation Platform using `jwap -profileName AppServer1 -user ADNAG -password TheQueen` on Windows. The `-profileName` is a wsadmin option that indicates which WebSphere Application Server profile you will use. When WSAAdmin Automation Platform is started you receive the `wap>` prompt.

Example: D-10 Starting WAP from a command line

```
C:\wap> jwap -profileName AppServer1 -user ADNAG -password TheQueen
```

```
WAP: Starting WAP for WebSphere Application Server.
WASX7209I: Connected to process "dmgr" on node DAGCellManager01 using SOAP
connector; The type of process is: DeploymentManager
WASX7026W: String "WebSphere:type=Server,*" corresponds to 3 different MBeans;
returning first one.
WASX7026W: String "WebSphere:type=Server,*" corresponds to 3 different MBeans;
returning first one.
```

```
Default scopes values :
```

```
-----
Cell   is set to : DAGCell01
Node   is set to : DAGCellManager01
Server is set to : dmgr
```

```
Warning :
```

```
-----
Use the commands "setCell" , "setNode" and "setServer"
to set the name of the default scope that will be used for
all subsequent WAP commands. You may place a call to these
functions at the beginning of every block of commands that
are for a specific scope.
```

Need help? Type help.

```
WASX7411W: Ignoring the following provided option: [-wap_path, c:/wap,
-verbose]
WAPisReadyToGo
wap>
```

Setting default scope values

You can specify the default scopes values for the WSAdmin Automation Platform commands by using the command **setCell**, **setNode** and **setServer**. In the example above the cell scope is set to DAGCell101, the node is set to DAGCellManager01 and the server is set to dmgr.

To change the default node scope type **setNode <nodeName>**. For example:

```
setNode ITS0GoodNode
```

To change the default server scope, type **setServer <serverName>** For example:

```
setServer server1
```

To get the default scopes values use **getCell**, **getNode** and **getServer** commands.

Creating a JDBC provider

To create a DB2 JDBC provider type the following command:

```
createJDBCProvider DB2Jdbc_1 -dbtype db2xa -cell [getCell]
```

To get help on how to use JDBCProvider utility functions type **help JDBCProvider**. Example D-11 shows the output of the **help** command.

Example: D-11 Help JDBCProvider output

```
wap> help JDBCProvider
JBCPROVIDER FUNCTIONS:
createJDBCProvider [-node <nodename> | -cell <cellname> | -server
<servername>]
                    -dbtype <dbvendor>
                    [-classpath <classpath>]
                    [-implclass <implementationclass>]
                    [-attr {{attr1 value} {attr2 value}} ...]
                    [-<YouNameIt> <value>]
                    <jdbcprovider_name>
modifyJDBCProvider -dbtype <dbvendor>
                  [-node <nodename> | -cell <cellname> | -server <servername>]
                  [-classpath <classpath>]
```

```

[-impclass <implementationclass>]
[-attr {{attr1 value} {attr2 value}} ...]
[-<YouNameIt> <value>]
<jdbcprovider_name>
removeJDBCProvider [-node <nodename> | -cell <cellname> | -server <servername>]
<jdbcprovider_name>

```

Creating a data source

To create a data source for the JDBCProvider DB2Jdbc_1 type the following command:

```

createDataSource DB2DS -jdbcprovider DB2Jdbc_1 -scope cell -dbtype db2
-jndi jdbc/DB2DS -cmp on -componentAuthAlias compAA -componentUser
SrIbrahim -componentPassword DonaMarly -containerAuthAlias contAA
-containerUser MaiteMarianeDawudDahrel -containerPassword goodkids
-maxConnections 150 -databaseName DoItDB

```

This command creates the data source JAAS authentication data alias, CMP connector and custom properties automatically.

Saving or resetting the configuration

To save the configuration changes, type **save**. To reset the configuration, type **reset**.

Testing a data source connection

To test a data source connection, type the **testDataSource** command:

```

testDataSource -scope cell DB2DS -jdbcprovider DB2Jdbc_1 -user andrea
-password goodgirl

```

More examples

Example D-12, Example D-13, Example D-14, and Example D-15 on page 525 show more uses of the WSAAdmin Automation Platform.

Example: D-12 Modifying application server transaction timeout

```

modifytransactionsservice -pot ApplicationServer -totalTranLifetimeTimeout 300

```

Example: D-13 Setting WebSphere variables

```

setVar -name DB2_JDBC_DRIVER_PATH -value "/DB2Driver" -scope node

```

Example: D-14 Modifying log file size

```

set serverList [$AdminConfig list StreamRedirect]

```

```
foreach sl "$serverList" {
    modifyconfigobject -cot StreamRedirect -con $sl -maxNumberOfBackupFiles 10
-rolloverSize 100
}
```

Example: D-15 Installing and starting an application

```
installApplication -appname MarillionMarbles -earfile
c:/tmp/MarillionMarbles.ear -node [getNode] -server [getServer] -deployejb
-deployejb.dbtype DB2UDB_V81 -BackendIdSelection "{{GreenCardEJB
GreenCardEJB.jar,META-INF/ejb-jar.xml DB2UDBNT_V8_1}}" -verbose
```

WSAdmin Automation Platform graphical user interface

To start WSAdmin Automation Platform in graphical mode, type **jwap -gui**. The graphical mode is only available on **jwap** command. Figure D-1 shows the graphical interface.

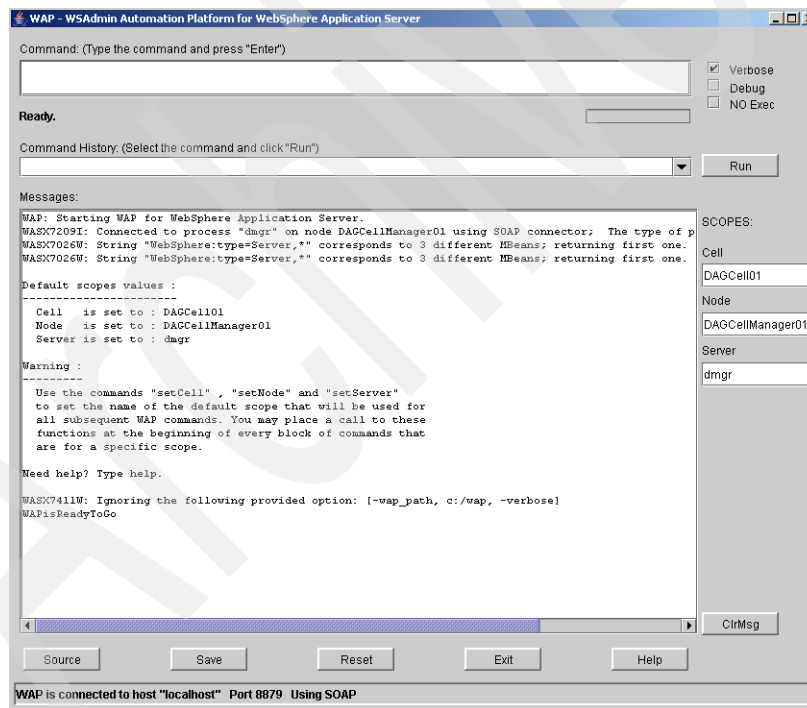


Figure D-1 WAP Graphical User Interface

D.1.5 The You Name It option

In most cases, each WSAAdmin Automation Platform function's option is related to one config object attribute. For instance, the option **-classpath** in **createJDBCProvider** command is related to the JDBCProvider's classpath attribute. The *You Name It* option is used when you cannot find a predefined option that corresponds to a config object attribute. Suppose you want to create a JDBCProvider. The create JDBCProvider function does not provide the option **-description**, and *description* is a JDBCProvider attribute. To set a JDBCProvider description you use the You Name It capability by typing **-description <description value>**. WSAAdmin Automation Platform will try to match the option with a config object attribute. If the attribute is not found an error is returned.

To list all attributes of a specific config object, type **listAttr <ConfigObject>**. For instance, **listAttr JDBCProvider** returns the output in Example D-16.

Example: D-16 Config object attribute list

```
List of attributes for config object JDBCProvider
-----
1 - classpath String*
2 - description String
3 - implementationClassName String
4 - name String
5 - nativepath String*
6 - propertySet J2EEResourcePropertySet
7 - providerType String
8 - xa boolean
```

Abbreviations and acronyms

ACID	Atomicity, Consistency, Isolation, Durability	JNDI	Java Naming and Directory Interface™
API	Application Programming Interface	JRE	Java Runtime Environment
BLOB	Binary Large Data Object	JSP	Java Server Page
CICS	Customer Information Control System	JVM	Java Virtual Machine
CORBA	Common Business-Oriented Language	OASIS	Organization for the Advancement of Structured Information Standards
COTS	Commercial Off-The-Shelf	PKI	Public Key Infrastructure
DBMS	Database Management System	RMI	Remote Messaging Interface
DMZ	Demilitarized Zone	RPC	Remote Procedure Call
DTD	Document Type Definition	SCM	Supply Chain Management
DVD	Digital Video Disc	SDK	Software Development Kit
EAI	Enterprise Application Integration	SDO	Service Data Object
EAR	Enterprise Archive	SOA	Service Oriented Architecture
EJB	Enterprise JavaBean	SSL	Secure Socket Layer
ESB	Enterprise Service Bus	TCP/IP	Transmission Control Protocol / Internet Protocol
GUI	Graphical User Interface	TLS	Transport Layer Security
HTML	Hypertext Markup Language	UDDI	Universal Description, Discovery, and Integration
HTTP	Hypertext Transfer Protocol	UML	Unified Modeling Language
HTTPS	Hypertext Transfer Protocol over Secure Sockets Layer	URI	Universal Resource Identifier
IBM	International Business Machines	URL	Universal Resource Locator
IDE	Integrated Development Environment	WAN	Wide Area Network
IIS	Internet Information Services	WS-BPEL	Web Services Business Process Execution Language
ITSO	International Technical Support Organization	WS-I	Web Services Interoperability
JAAS	Java Authentication and Authorization Service	WSDL	Web Services Description Language
JAR	Java Archive	XML	Extensible Markup Language
JAX-RPC	Java API for XML-based Remote Procedure Call	XSD	XML Schema Definition
JDBC	Java Database Connectivity		

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 531. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303
- ▶ *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346
- ▶ *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6*, SG24-6494
- ▶ *Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture*, SG24-6773
- ▶ *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451
- ▶ *WebSphere Application Server V6: Security Handbook*, SG24-6316
- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- ▶ *Patterns: Serial Process Flows for Intra- and Inter-enterprise*, SG24-6305
- ▶ *WebSphere Business Integration Server Foundation V5.1 Handbook*, SG24-6318

Other publications

These publications are also relevant as further information sources:

- ▶ *Patterns for e-business: A Strategy for Reuse*, by Jonathan Adams, Srinivas Koushik, Guru Vasudeva, and George Galambos, ISBN 1931182027

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ Patterns for e-business Web site
<http://www.ibm.com/developerWorks/patterns/>
- ▶ Open Grid Services Architecture
<http://www-106.ibm.com/developerworks/grid/library/gr-visual/>
- ▶ The role of private UDDI nodes in Web services, Part 1: Six species of UDDI
<http://www.ibm.com/developerworks/webservices/library/ws-rpu1.html>
- ▶ The role of private UDDI nodes, Part 2: Private nodes and operator nodes
<http://www.ibm.com/developerworks/webservices/library/ws-rpu2.html>
- ▶ Security in a Web Services World: a Proposed Architecture and Roadmap
<http://www.ibm.com/developerworks/library/ws-secmap/>
- ▶ Web Services Security: Moving up the stack
<http://www.ibm.com/developerworks/webservices/library/ws-secroad/>
- ▶ Updated: Web Services Reliable Messaging: A new protocol for reliable delivery between distributed applications
<http://www.ibm.com/developerworks/webservices/library/ws-rm/>
- ▶ Implementation Strategies for WS-ReliableMessaging: How WS-ReliableMessaging can interact with other middleware communication systems
<http://www.ibm.com/developerworks/webservices/library/ws-rmimp/>
- ▶ WS-BPEL specification
<http://www.ibm.com/developerworks/library/ws-bpel/>
- ▶ Business Process with WS-BPEL, a series of introductory articles and references
<http://www.ibm.com/developerworks/webservices/library/ws-bpelcoll/>
- ▶ WS-BPEL support in WebSphere Business Integration Server Foundation
<http://www.ibm.com/software/integration/wbisf/features/>
- ▶ WS-BPEL support in WebSphere Studio Application Developer Integration Edition
<http://www.ibm.com/software/integration/wsadie/features/>
- ▶ WS-AtomicTransaction specification
<http://www.ibm.com/developerworks/library/ws-atomtran/>

- ▶ WS-BusinessActivity specification
<http://www.ibm.com/developerworks/webservices/library/ws-busact/>
- ▶ Transactions in the world of Web Services, part 1 and part 2
<http://www.ibm.com/developerworks/webservices/library/ws-wstx1/>
<http://www.ibm.com/developerworks/webservices/library/ws-wstx2/>
- ▶ WS-Coordination specification
<http://www.ibm.com/developerworks/library/ws-coor/>
- ▶ WS-Policy framework specification
<http://www.ibm.com/developerworks/library/ws-polfram/>
- ▶ Web Services Policy Framework: New specifications improve WS-Security
<http://www.ibm.com/developerworks/webservices/library/ws-polfram/summary.html>
- ▶ WS-ResourceFramework overview
<http://www.ibm.com/developerworks/webservices/library/ws-resource/ws-wsrfpaper.html>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Numerics

80/20 situation 11

A

Adapter connector 104
Advanced and future Web services standards
 Business Process Execution Language for Web Services 55
 Web services security 54
 WS-Policy 54
 WS-Privacy 54
 WS-Security 54
 WS-Trust 54
 Web services transactions 56
 Web Services Policy Framework 57
 WS-Policy 57
 Web Services Resource Framework 57
 WS-ResourceFramework 57
 WS-AtomicTransaction 56
 WS-BusinessActivity 56
 WS-Coordination 56
 WS-ReliableMessaging 55
AMI 93
App server/services 102
Application Access paradigm 28
Application gateways 103
Application patterns 12, 19
 Exposed Broker 76
 Exposed Direct Connection 71
 Exposed Serial Process 80
 Extended Enterprise 68
Application Server Toolkit 180
AS1 96
AS2 96
Asymmetric algorithms 168
Atomic business functions 38
Automation 33
Autonomic managers 39

B

B2B *See* Business-to-Business pattern
Best practices 12, 24

BinarySecureToken 167
Binding 52
BLOB domain 94
Broker Rules tier 77
Broker scenario
 Runtime
 Defining mediations 377
BSC runtime pattern
 Persistence manager 106
 Process manager
 Branching 108
 Correlation 108
 Monitoring 107
 Non-functional requirements 108
 Process abstractions 108
 Process definition standards 107
 Rules directory 106
Business and IT drivers
 Call Connection variation 75
 Exposed Broker Application pattern 76
 Exposed Direct Connection Application pattern 72
 Exposed Router variation 79
 Extended Enterprise Application pattern 69
 Extended Enterprise Business pattern 65
 Message Connection variation 75
 Serial Process Application pattern 81
 Serial Workflow variation 83
Business Function Services 46
Business outsourcing 28
Business patterns 12, 15, 63
Business Process Choreographer 94
Business process choreography 39
Business Process Services 46
Business Service Choreography 108
Business Transaction Services 46
Business-to-Business pattern 63

C

Call Connection variation 72, 75
 Business and IT drivers 75
Canonicalization method algorithm 194
CICS Transaction Server 102, 513

- Web services 513
 - CICS resources 515
 - PIPELINE 515
 - TCIPSERVICE 515
 - URIMAP 515
 - WEBSERVICE 515
 - CICS Web services assistant 515
 - DFHLS2WS 515
 - DFHWS2LS 515
 - Creating 514
 - Bottom-up 514
 - Top-down 514
- CICS Transaction Server V3.1 126
- CICS Web services assistant 515
- Collaboration 33
- Commercial Off-The-Shelf 43
- Composite patterns 12, 17
- confidentiality
 - dialog 205
 - encryption information 213
 - key locator 212
 - required confidentiality 209
 - token consumer 210
- connection 73
- connection rules 71
- Connection Rules tier 73
- Connector 103
- CORBA 48
- Coupling business processes 35
- Cross-industry communication 50
- Cryptographic protocols 170
- Customized technologies 58

D

- DB2 Connect 92
- DB2 Universal Database Enterprise Server Edition V8.2 91
 - Autonomic computing solutions 92
 - Data warehouse server 92
 - Performance 92
 - Satellite administration 92
 - Scalability 92
- Decoupling technology 35
- Denial of service 169
- Directory and Security Services 106
- Distributed transactions 57
- Document Manager 96
- Domain firewall 103

- Dynamic workload distribution 93

E

- Early warning missile system 59
- e-business on demand
 - Automation 33
 - Self-configuring 34
 - Self-healing 34
 - Self-optimization 34
 - Self-protecting 34
 - Business drivers 29
 - Focused 29
 - Resilient 30
 - Responsive 30
 - Variable 30, 49
 - Integration 31
 - Applications 32
 - Data 32
 - People 31
 - Process 32
 - Systems 32
 - Key technological attributes 30
 - Automation 30
 - Integration 30
 - Open standards 30
 - Virtualization 30
- On Demand Operating Environment
 - Enterprise Service Bus 36
 - Infrastructure services 39
 - Resource virtualization services 39
 - Service level automation and orchestration 39
 - Utility business services 39
 - Integration services 37
 - Business function services 38
 - Business process choreography services 38
 - Common services 38
 - Information management services 38
 - User access services 37
 - User interaction services 38
- Open Grid Services Architecture 33
- Open standards 34
- Service-oriented architecture
 - Automation 50
 - Integration 50
 - Open standards 50
 - Virtualization 50

- Virtualization 32
 - Distributed systems 33
 - Servers 33
 - Storage 33
- Enterprise Application Integration 43–44
- Enterprise Service Bus 36
- ERP based systems 60
- ESB Gateway
 - Gateway endpoint 106
- ESB runtime pattern
 - App server / services 102
 - Hub node
 - Addressing 105
 - Infrastructure intelligence 106
 - Integration 105
 - Message processing 105
 - Messaging styles 105
 - Modelling 106
 - Quality of service 105
 - Routing 104
 - Service interface definition 105
 - Service level 106
 - Service messaging model 105
 - Transport protocols 105
- Exotic protocols 43
- Explicit implementation independent interfaces 47
- Exposed Broker 106
- Exposed Broker Application pattern 76
 - Business and IT drivers 76
 - Router variation 78
 - Business and IT drivers 79
- Exposed Broker product mapping 129
 - Generic profile 129
 - DB2 Universal Database 130
 - IBM HTTP Server 130
 - WebSphere Application Server 129
 - WebSphere Application Server Network Deployment 130
 - WS-Security 130
- Exposed Broker runtime pattern 112, 339
 - Business scenario 340
 - Design guidelines 341
 - Designing the broker component 344
 - Generic profile 112
 - Application Server/Services 113
 - Broker Rules tier 112
 - Directory and Security Services 113
 - SOA profile 113
 - ESB 113
 - Exposed ESB Gateway 113
 - Service Consumers 113
 - Service Providers 113
- Exposed Direct Connection Application pattern 71
 - Business and IT drivers 72
 - Call Connection variation 72, 75
 - Message Connection variation 72, 74
- Exposed Direct Connection product mapping 126
 - Generic profile 127
 - IBM HTTP Server V6 127
 - WebSphere Application Server 127
 - SOA profile 127
 - DB2 Universal Database 129
 - IBM HTTP Server 129
 - Web services gateway 128
 - WebSphere Application Server 128
 - WebSphere Application Server Network Deployment 128
 - WS-Security confidentiality 128
 - WS-Security integrity 128
- Exposed Direct Connection runtime pattern 109, 157, 237
 - Business scenario 158, 238
 - Design guidelines 159, 239
 - Communication 240
 - Distributed 240
 - Flexibility 240
 - Implementing an ESB 244
 - WebSphere Application Server 246
 - WebSphere Business Integration Message Broker 245
 - Integration 240
 - Integration options 163
 - COTS 164
 - Custom technology specific adapters 164
 - Securing Web services 165
 - Standard based integration 164
 - Interoperability 240
 - Maintaining an audit trail 246
 - Management 240
 - Mediation services 240
 - Quality of service 240
 - Securing the Web service interaction 246
 - Wide Area Networks 159
- Generic profile 109
 - Connector 110
 - Directory 109
 - Directory and Security Services 109

- Path Connectors 110
- SOA profile 111
 - ESB 111
 - Exposed ESB Gateway 111
 - Service Consumers 111
 - Service Providers 111
- Exposed ESB Gateway 104
- Exposed ESB Gateway runtime pattern
 - ESB Gateway 104
- Exposed ESB Gateway scenario
 - ESB capabilities
 - Addressing 243
 - Communications 243
 - Integration 243
 - Restricting service access 243
 - Security 243
 - Service interaction 243
- Exposed Process Manager 107
- Exposed Router 107
- Exposed Router product mapping
 - SOA profile 130
 - DB2 Universal Database 131, 395
 - IBM HTTP Server 131, 396
 - WebSphere Application Server Network Deployment 131, 395
 - WebSphere Partner Gateway 131, 395
- Exposed Router variation 115
 - Generic profile 115
 - Application Server/Services 116
 - Directory and Security Services 116
 - Exposed Router node 115
 - SOA profile 116
 - ESB 116
 - Exposed ESB Gateway 116
 - Service Consumers 116
 - Service Providers 116
- Exposed Serial Process Application pattern 80
 - Workflow variation 83
- Exposed Serial Process product mapping
 - Generic profile 132
 - IBM HTTP Server 133, 425
 - Web Services Invocation Framework 132, 425
 - WebSphere Application Server Network Deployment 133, 425
 - WebSphere Business Integration Server Foundation 132
 - SOA profile 133
 - DB2 Universal Database 134, 464
 - IBM HTTP Server 134, 465
 - Web services gateway 134, 464
 - Web Services Invocation Framework 134, 464
 - WebSphere Application Server 133, 464
 - WebSphere Business Integration Server Foundation 133
- Exposed Serial Process runtime pattern 117
 - Generic profile 117
 - Application Server/Services 118
 - Directory and Security Services 118
 - Exposed Process Manager 117
 - SOA profile 119
 - Business Service Choreography 119
 - ESB 119
 - Exposed ESB Gateway 119
 - Service Consumers 119
 - Service Providers 119
- Exposed Serial Workflow variation 120
 - Generic profile 120
 - Directory and Security Services 121
 - Exposed Process Manager 120
 - Staff Worklist Adapter 121
 - SOA profile 121
 - Business Service Choreography 122
 - ESB 122
 - Exposed ESB Gateway 122
 - Service Consumers 122
 - Service Providers 122
- Extended Enterprise 100
 - Node types 102
 - App server/services 102
 - Business Service Choreography 108
 - Connector 103
 - Directory and Security Services 106
 - Domain firewall 103
 - ESB 104
 - Exposed Broker 106
 - Exposed ESB Gateway 104
 - Exposed Process Manager 107
 - Exposed Router 107
 - Network infrastructure 102
 - Protocol firewall 102
 - Rules Directory 106
 - Staff Worklist Adapter 108
- Extended Enterprise Application pattern 68
 - Business and IT drivers 69
- Extended Enterprise Business pattern 63
 - Business and IT drivers 65

F

File Transfer Protocol 170
Fine-grained services 46
Firewalls 43

G

Global deployment 44
Global Security 223
Globalization 40
Grid computing 33
Guidelines 12, 24

H

Harvard Business Review 28
Heterogeneity 40
Horizontal business processes 45
HTTP 152
HTTP/S 153
Hypervisors 33

I

IBM Cloudscape 92
 Administration 92
 Java Virtual Machine 92
 JDBC 93
 Locking 93
 Migration path 93
 Network server 93
 Online transaction processing 92
 Open source 92
 Performance 93
 Rapid application development 92
 Resource management 92
 SQL-92E 93
 Standard Distributed Relational Database Architecture 93
 Stored procedures 93
 Triggers 93
IBM Eclipse SDK 3.0 97
IBM Emerging Technologies Toolkit 54
IBM HTTP Server 224
 Create a keystore 225
 SSL pass-through 224
Implementation independent interfaces 58
IMS 94
IMS Transaction Manager 102
Incremental adoption 43

Infrastructure services 39
Integration 31
Integration approaches 59
 Traditional 59
 High dependency on real-time communication 59
 Low number of static integration points 59
 Technology centric integrations 60
Integration patterns 12, 16
Integration services 37
integrity
 key information 194, 200
 key locator 193, 200
 part reference 195
 required integrity 197
 signing information 194, 201
 token consumer 198
 transform 196
Inter-enterprise network infrastructure 102
ITSO Good sample business scenario 138
 Applications 139
 Logging Facility 140
 Manufacturer 140
 Retailer 139
 SCMSampleUI 139
 Warehouse 140
 Business context 139
 Example usage 140

J

J2EE 150
jaas.config 209
Java Application Programming Interface 150
Java Community Process 151
Java keytool 186
 Export a client RSA certificate 186
 Export a server RSA certificate 187
 Generate a client RSA key 186
 Generate a server RSA key 186
 Import a client RSA certificate 187
 Verify 187
Java Virtual Machine 150
JAX-RPC 152
JMS 151

K

Kerberos tickets 167
Key Information 193

Key Locators 192
KeyLocator 193

L

Large-grained services 46
Lightweight Directory Access Protocol 170
Location transparency 48, 105
Logging Facility 140
Lotus Domino 94

M

Manufacturer 140
MAPE loop 39
Mediations 348
 Assigning mediation handlers 364
 Configuring context properties for a mediation 380
 Defining mediations 377
 Developing a mediation handler 352
 Request mediation 356
 Response mediation 361
 Install 376
 Mediating a destination 379
 Mediation API 349
 MediationHandler 349
 MessageContext 350
 SIMediationSession 350
 SIMessage 350
 Routing paths 351
 SDO DataGraphs 351
Message Connection variation 72, 74
 Business and IT drivers 75
Message flows 94
Message Repository Manager 94
Metering 39
Microsoft .NET 126
Microsoft .NET Web services 483
 Development 485
 Create a new Web service project 487
 Deploying 491
 Generating a C# file 487
 Implementation First 485
 Modifying a C# file 488
 WSDL First 486
 Testing 493
 Transport-level security 500
 Installing a Web server certificate 506
 Issuing a Web server certificate 505

Processing a Web server certificate request 503

 Requesting a server certificate 501

Microsoft Certificate Services 503
Microsoft Exchange 94
MQI 93
Multi-modality 38
Mutual authentication 167

N

Network bandwidth 43
Network infrastructure 102
Non-repudiation repository 96

O

On Demand Business 28, 49
On Demand Operating Environment 33, 35
 Enterprise Service Bus 36
 Infrastructure services 39
 Integration services 37
on demand Operating Environment ??–39
Open Grid Services Architecture 33
Open standards 34
Open-standard technologies 58

P

Part References 195
Path connector 104
Patterns for e-business
 Application patterns 12, 19
 Best practices 12, 24
 Business patterns 12, 15
 Composite patterns 12, 17
 Guidelines 12, 24
 Integration patterns 12, 16
 Product mappings 12, 23
 Runtime patterns 12, 21
 Web site 13
Peer-to-peer 38
Personalization 38
Point-to-point 93
Policy declarations 39
Port 52
Portal development 96
Portal user interaction 40
PortType 52
Product mappings 12, 23

Proprietary technologies 58
Protocol firewall 102
Public key cryptography 96
Publish/subscribe 93

Q

Quality of Service
 availability 60
 Extended Enterprise pattern 71
 federation 60
 performance 61
 security 61
 standards compliance 61

R

Rational Application Developer V6 96
 Automated deployment 96
 Automated test 96
 Code analysis 96
 Portal development 96
 Team tools 96
 UML editing 96
 Version control 96
Rational Software Development Platform 97
Rational Web Developer 97
Redbooks Web site 531
 Contact us xvii
Reliable messaging 43
Resilient routing 53
Resource tuning 34
Retailer 139
Reusable function 46
Router Rules tier 79
Router variation 78
RSA-V1.5 168
Rules Directory 106
Runtime patterns 12, 21

S

SAP/R3 94
SCMSampleUI 139
Screening routers 103
Secure Sockets Layer 170
Self-configuring 34
Self-healing 34
Self-optimization 34
Self-protecting 34

Serial Process Application pattern
 Business and IT drivers 81
 Workflow variation
 Business and IT drivers 83
Serial Process Rules tier 81
Serial Workflow Rules tier 84
Service granularity 46
 Granularity of service operations 47
 Granularity of service parameters 47
 Level of abstraction of services 47
Service integration bus 257, 298
 Adding a bus member 259
 Configuring the SDO repository 260
 Creating a bus 258
 Creating a foreign bus 299
 Creating inbound services 273
 Creating outbound services 270
 Creating the endpoint listener 268
 Creating the service integration bus link 300
 Installing the SDO repository 260
 Installing the Web services applications 267
 Overriding Web services client bindings 278
 Service integration bus link 298
Service substitution 105
Service-oriented architecture
 Component Based Design 44
 Customized 43
 Drivers
 Business processes 42
 Business systems 42
 Flexible pricing 41
 Increasing speed 41
 Reducing costs 41
 Return on investment 41
 Simplifying integration 41
Object Oriented development 44
Proprietary 43
Service
 Deployment time 47
 Implementation-independent 46
 Loosely bound 46
 Reusable 46
 Runtime 47
 Substitution 47
Settlement 39
Signature method algorithm 194
Signing Information 194
Simple Mail Transfer Protocol 170
Simple Object XML 148

- SMTP 48
- SOAP 148
- Source Application tier 73, 77, 79, 81, 84
- Speech-based interactions 38
- SQL-92E 93
- Staff Worklist Adapter 108
- Standard Distributed Relational Database Architecture 93
- Sun Microsystems 150
- Symmetric algorithms 168

T

- Target Application tier 73, 77, 80, 82, 84
- TCP/IP Monitor 233
 - Act as Listener 233
 - Launch 233
 - PATH environment variable 233
 - Viewing a SOAP 236
- Technical Function Services 46
- Token Consumer 197
- Token Generator 191
- Transaction management 33
- Transforms 196
- Transport Layer Security 170
- Transport protocols 152
- Triple DES 168
- Trustworthy network 61

U

- U.S. government taxonomy of businesses 53
- UDDI 53, 149
 - private directory 53
 - public directory 53
- UDDI Service Directory 51
- UML editing 96
- UsernameToken 167

V

- Virtual Ethernet 33
- Virtualization 32

W

- Warehouse 140
- WC_defaulthost_secure 225
- Web service client bindings 227
- Web service gateway 281
 - Applying WS-Security 333

- Configuring an inbound port 289
- Configuring an outbound service and destination 287
- Configuring request and response destinations 285
- Create a gateway instance 283
- Create a gateway service 284
- Exporting gateway WSDL 292
- Renaming a inbound port 290
- Selecting an outbound port and port destination 288
- Web services 50, 146
 - Basic callback usage scenario 146
 - One-way usage scenario 146
 - SOAP 148
 - Synchronous request/response usage scenario 146
 - WSDL 148
 - XSD 148
- Web services architecture
 - Namespaces 53
 - Service Directories 51
 - Service Providers 51
 - Service Requesters 51
 - SOAP 52
 - Universal Description, Discovery, Integration 53
 - Web Services Description Language 52
- Web services for J2EE 151
- Web Services Interoperability Organization 138
- Web services security 54
- WebSphere Application Server 91, 223
 - Global Security 223
- WebSphere Application Server - Express 90
- WebSphere Application Server Network Deployment 91
- WebSphere Application Server Network Deployment V6 91
- WebSphere Application Server V6 88
 - Highlights and benefits 89
 - Packaging for distributed platforms
 - WebSphere Application Server - Express V6
 - WebSphere Application Server - Express V6 90
 - WebSphere Application Server Network Deployment V6 91
 - WebSphere Application Server V6 91
- WebSphere Business Integration Message Broker V5 94

- BLOB domain 94
- Message flows 94
 - Message domains 94
 - Processing message content 94
 - Routing 94
 - Transformation 94
- Message Repository Manager 94
- Message routing 94
- Publish/subscribe 94
- Transformation 94
- WebSphere MQ 94
- XML domain 94
- WebSphere Business Integration Server Foundation V5.1
 - Business Process Choreographer 94
 - Directed graph 95
 - Event driven 95
 - Human resources 95
 - Interruptible process 95
 - IT resources 95
 - Long-running process 95
 - Short-running process 95
 - Single transaction 95
 - Web service 95
 - WS-BPEL 94
 - WSDL 95
- WebSphere MQ 49
- WebSphere MQ V5.3 93
 - AMI 93
 - Assured delivery 93
 - Availability 93
 - CICS Transaction Server 94
 - Connectors 94
 - Dynamic workload distribution 93
 - Gateways 94
 - Hot standby capabilities 93
 - IMS 94
 - JMS 93
 - Lotus Domino 94
 - Microsoft Exchange 94
 - MQI 93
 - Point-to-point 93
 - Publish/subscribe 93
 - Queue managers 93
 - SAP/R3 94
 - Transport layer 93
- WebSphere Partner Gateway V6
 - AS1 96
 - AS2 96
 - Authentication 96
 - Certificate validation 96
 - Data sharing 95
 - Digital signature verification 96
 - Document engine 95
 - Document Manager 96
 - Non-repudiation repository 96
 - Partner profile management 96
 - Process integration 95
 - Public key cryptography 96
 - Security management 96
 - Small-and-medium business 95
 - SSL 96
 - Transformation 95
 - User exits 96
 - Validation 95
 - WebSphere Partner Gateway Advanced 95
 - WebSphere Partner Gateway Enterprise 96
 - WebSphere Partner Gateway Express 95
- WebSphere Portal 97
- WebSphere Studio Application Developer 97
- WebSphere Studio Application Developer Integration Edition V5.1 97
 - Visual process editor 98
- Workflow variation 83
- Workload management 33
- World Wide Web Consortium 52
- WS Binding 183
- WS Extension 183
- WSAdmin Automation Platform 517
- WS-AtomicTransaction 56
- WS-Authorization 150
- WS-BPEL 150
- WS-BusinessActivity 56
- WS-Coordination 56
- WSDL 148
- WS-Federation 150
- WS-I Basic Security Profile 169
- WS-I sample application 138
- WS-I sample business scenario 138
- WS-I Supply Chain Management Technical Architecture 138
- WS-I Supply Chain Management Use Cases 138
- WS-I Usage Scenarios 138
- WS-Policy 54, 57, 106, 150
- WS-Privacy 54, 150
- WS-ReliableMessaging 55
- WS-SecureConversation 150
- WS-Security 54, 150, 179, 304

- Configuring 180
- Configuring WS-Security confidentiality
 - Rational Application Developer
 - Confidentiality 205
 - Encryption Information 213
 - Encryption information 207
 - Key Information 207, 212
 - Key Locators 206, 211
 - Required Confidentiality 208
 - Service integration bus
 - Encryption information 332
 - Key information 330
 - Key locator 328
 - Keys 329
 - Message parts 327
- Configuring WS-Security integrity
 - Rational Application Developer 188
 - Configuring the client integrity 189
 - Key Information 193
 - Key Locators 192
 - Part References 195
 - Signing Information 194
 - Token Consumer 197
 - Token Generator 190–191
 - Transforms 196
 - Service integration bus
 - Callback handler 317
 - Key information 321
 - Key locator 319
 - Keys 318
 - Part references 325
 - Security token reference 322
 - Signing information 323
 - Token generator 314
 - Transform 326
- Generating sample key stores 185
 - Algorithm 185
 - Alias 186
 - Certificate file 186
 - Distinguished Name 186
 - Key size 186
 - Key store file 186
 - Keypass 186
 - Storepass 186
 - Storetype 186
- WS Binding 183
- WS Extension 183
- WS-Authorization 150
- WS-Federation 150
- WS-Policy 150
- WS-Privacy 150
- WS-SecureConversation 150
- WS-Trust 150
- WS-Trust 54, 150

X

- X.509 Certificates 167
- X509TokenConsumer 198
- XML Digital Signatures 54
- XML domain 94
- XML Encryption 54
- XML Transformations 148
- XSD 148

Z

- z/OS 516



Redbooks

Patterns: Extended Enterprise SOA and Web Services

(1.0" spine)
0.875" <-> 1.498"
460 <-> 788 pages



Redbooks

Patterns: Extended Enterprise SOA and Web Services

Design secure business-to-business solutions using WebSphere

Use the Extended Enterprise Patterns for e-business

Learn by example with practical scenarios

Service-oriented architecture (SOA) promotes the ability to communicate with external enterprises. This IBM Redbook addresses issues for Web services implementations of SOA, using the Patterns for e-business.

The Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying On Demand Business applications. This IBM Redbook focuses on building Extended Enterprise SOA solutions using WebSphere Application Server V6, WebSphere Partner Gateway V6, the Web services gateway component of WebSphere Application Server Network Deployment V6, and WebSphere Business Integration Server Foundation V5.1.

Part 1 introduces the Patterns for e-business, and describes the patterns and product mappings for building Extended Enterprise solutions.

Part 2 describes the business scenario used throughout this book, and the technologies for implementing an SOA solution.

Part 3 provides a set of Extended Enterprise scenarios, that include simple as well as more complex SOA solutions that use an Enterprise Service Bus.

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