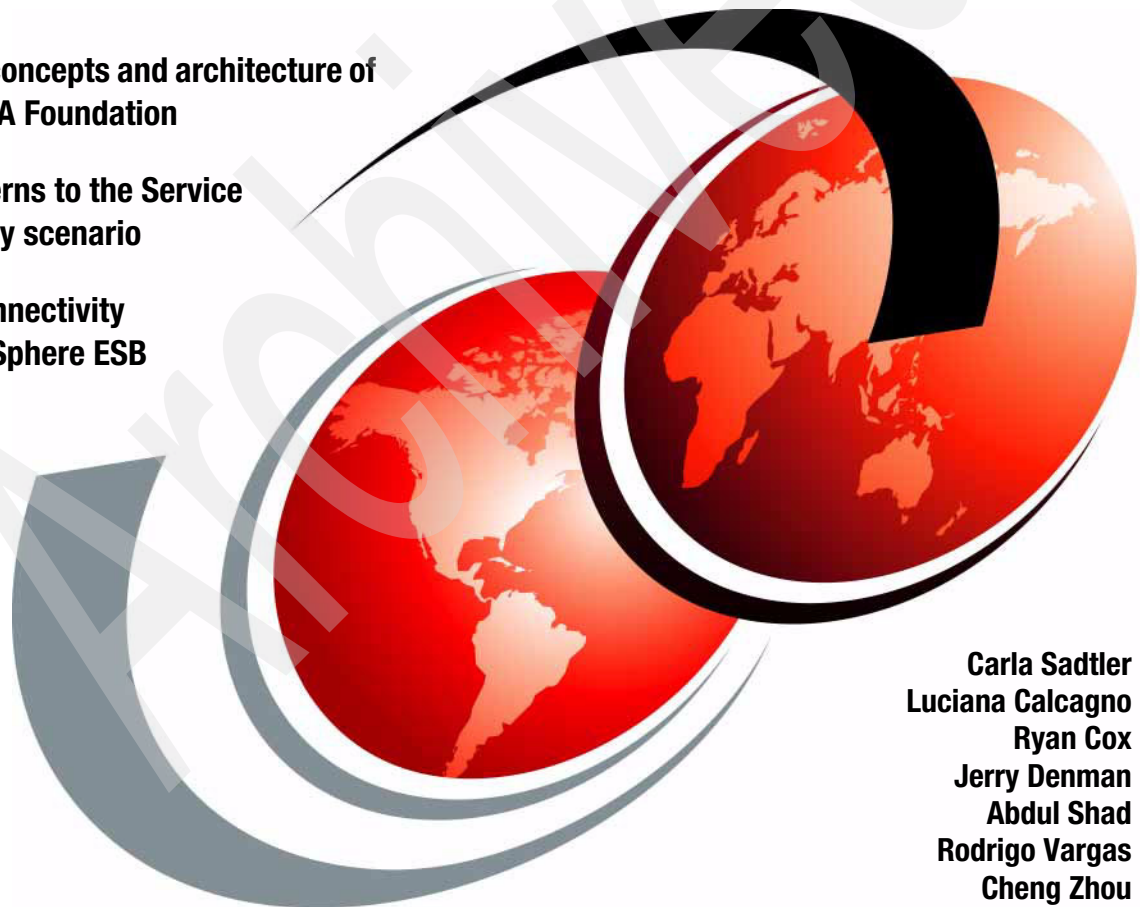


Patterns: SOA Foundation Service Connectivity Scenario

Learn key concepts and architecture of
the IBM SOA Foundation

Apply patterns to the Service
Connectivity scenario

Service Connectivity
using WebSphere ESB



Carla Sadtler
Luciana Calcagno
Ryan Cox
Jerry Denman
Abdul Shad
Rodrigo Vargas
Cheng Zhou



International Technical Support Organization

**Patterns: SOA Foundation Service Connectivity
Scenario**

August 2006

Archived

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

First Edition (August 2006)

This edition applies to WebSphere Enterprise Service Bus V6.0.1, WebSphere Integration Developer V6.0.1, Rational Software Architect V6.0.1, IBM Tivoli Monitoring 6.1.

© 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	xiii
Trademarks	xiv
Preface	xv
The team that wrote this redbook	xv
Become a published author	xviii
Comments welcome	xix
Chapter 1. IBM SOA Foundation	1
1.1 SOA Foundation life cycle	2
1.2 SOA Foundation Reference Architecture	5
1.3 SOA Foundation scenarios	9
1.4 Service Connectivity scenario	11
1.4.1 Secure connections to third parties and trading partners	13
1.4.2 Internal connectivity based on open standards	15
1.4.3 Adapt from third-party systems to Web services	17
1.4.4 Deliver an existing process through new business channels	19
Chapter 2. Process for applying SOA scenarios	21
2.1 Process for using SOA scenarios and patterns	22
2.1.1 SOA scenario selection table	24
2.2 Reuse patterns assets to accelerate the solution architecture	26
2.2.1 Introduction to the Patterns for e-business	27
2.2.2 Patterns for the SOA scenarios	30
Chapter 3. Patterns for e-business and Service Connectivity	31
3.1 The enterprise service bus	32
3.1.1 The role of an enterprise service bus	35
3.2 Business pattern and Application pattern selection	36
3.3 Runtime pattern selection	39
3.3.1 SOA profiles of the Runtime patterns	39
3.3.2 Runtime nodes	39
3.4 Self-Service business pattern	44
3.4.1 Directly Integrated Single Channel application pattern	44
3.4.2 Router application pattern	47
3.4.3 Decomposition application pattern	49
3.5 Application Integration pattern	52
3.5.1 Direct Connection application pattern	53
3.5.2 Broker application pattern	56

3.5.3 Router variation of the Broker application pattern	58
3.6 Extended Enterprise business pattern	60
3.6.1 Exposed Direct Connection application pattern	60
3.6.2 Exposed Broker application pattern	63
3.6.3 Exposed Router variation of Exposed Broker application pattern . . .	65
3.7 For more information	66
Chapter 4. Planning for connectivity	67
4.1 The ITSOMart scenario	68
4.1.1 ITSOMart overview	68
4.1.2 Business objectives	68
4.1.3 Customer registration business requirements	69
4.1.4 Business context diagram for ITSOMart	70
4.1.5 Functional requirements for ITSOMart	71
4.2 Considering SOA as a solution for ITSOMart	71
4.2.1 Incremental solution delivery	72
4.2.2 Integration cost reduction	73
4.2.3 Changing business/IT needs	73
4.2.4 Value delivery	73
4.2.5 Security	73
4.2.6 Management/monitoring	74
4.2.7 Readiness	74
4.3 Elements of an SOA solution	75
4.3.1 Using the SOA Foundation Reference Model	75
4.4 Selecting the SOA scenario and pattern	79
4.4.1 Fit gap analysis	79
4.4.2 Select the SOA scenario	80
4.4.3 Reuse patterns assets to accelerate solution architecture	80
4.5 Enterprise service bus product selection	82
4.5.1 WebSphere Enterprise Service Bus	84
4.5.2 WebSphere Message Broker	91
4.5.3 WebSphere DataPower SOA Appliance	96
4.5.4 Enterprise service bus implementations compared	99
4.6 ITSOMart product selection	101
4.6.1 Deployment	101
4.6.2 Modeling and design	103
4.6.3 Development and assembly	103
4.6.4 Monitoring and management	105
4.7 Installation considerations	105
4.7.1 Rational Software Architect, WebSphere Integration Developer . .	107
4.7.2 WebSphere ESB	109
4.7.3 WebSphere Application Server	112
4.7.4 ITCAM for SOA	113

4.7.5 Environments for testing and production	116
4.8 Security considerations	116
4.8.1 Securing communication using WebSphere ESB	116
4.8.2 Messaging security	117
4.8.3 Transport security using HTTPS	117
4.9 Scalability and performance considerations	118
4.10 System management and monitoring	119
4.10.1 IBM Tivoli Management Framework	119
4.10.2 IBM Tivoli Composite Application Manager for SOA	120
4.11 Where to find the implementation details	121
4.12 Summary	121
4.13 For more information	123
Chapter 5. Model with Rational Software Architect	125
5.1 Introduction to Rational Software Architect	126
5.1.1 Rational Unified Process guidance	126
5.1.2 Model-driven development	126
5.1.3 Modeling	128
5.1.4 Asset-based development	129
5.2 Modeling the ITSOMart sample	130
5.3 Tools used to model the application	131
5.3.1 Modeling perspective	132
5.3.2 UML projects	134
5.3.3 UML models	136
5.3.4 UML diagrams	137
5.4 Solution requirements	140
5.4.1 Use case diagram	141
5.4.2 Activity diagrams	142
5.5 Domain analysis	144
5.5.1 Sequence diagrams	144
5.5.2 Component diagram	148
5.6 Architectural design	149
5.6.1 Service components	151
5.6.2 Connecting services through the ESB	151
5.6.3 Mediations on the ESB	154
5.7 Modeling business objects: Transform UML to XSD	164
5.7.1 Create an XSD model	166
5.7.2 Create a package	169
5.7.3 Create a class	171
5.7.4 Create a class diagram	174
5.7.5 Run the UML to XSD transformation	175
5.7.6 Import the XSD into WebSphere Integration Developer	178
5.8 Modeling messaging resources: Transform UML to JACL	180

5.8.1	Import the WebSphere Platform Messaging Patterns asset	181
5.8.2	Model messaging resources	186
5.8.3	Run the UML-to-JACL transformation	204
5.8.4	Running the JACL script from a command line	208
5.9	Resources	209
Chapter 6. Assemble with WebSphere Integration Developer		211
6.1	Technology overview	212
6.1.1	Service Component Architecture	212
6.1.2	Service Data Objects	216
6.1.3	Service Message Objects	219
6.2	Introduction to WebSphere Integration Developer	220
6.2.1	Starting WebSphere Integration Developer	220
6.3	Development environment settings	224
6.3.1	Disable automatic build	224
6.3.2	Set the default target runtime	226
6.3.3	Configure Web services workspace preferences	226
6.3.4	Workspaces and test environment	229
6.4	Development process	230
6.4.1	Create a library	232
6.4.2	Create business objects	233
6.4.3	Define interfaces	236
6.4.4	Create a mediation module	240
6.4.5	Complete the module assembly	244
6.4.6	Implement the mediation flow component	251
6.4.7	Build the mediation flow	253
6.5	Testing mediations	256
6.5.1	Test servers	256
6.5.2	Test client	259
6.6	Packaging the mediation for deployment	260
Chapter 7. Building the Credit Rating and Credit Score mediations		263
7.1	Scenario overview	264
7.1.1	Business scenario	264
7.1.2	Get Credit Rating scenario stage 1	265
7.1.3	Get Credit Rating scenario stage 2	266
7.2	Preparing for the ITSOMart mediations	268
7.2.1	Create a library	268
7.2.2	Create the common business objects	269
7.3	Developing the Credit Rating mediation	275
7.3.1	Mediation development steps	275
7.3.2	Define the interface for the mediation	276
7.3.3	Define the interface to the Credit Rating Service Web service	278

7.3.4	Create the mediation module	281
7.3.5	Add the components to the module assembly	281
7.3.6	Build the mediation flow	285
7.3.7	Test the mediation.	303
7.4	Developing the Credit Score mediation.	309
7.4.1	Mediation development steps	310
7.4.2	Create the mediation module	310
7.4.3	Define the business objects	313
7.4.4	Define the interface to the Credit Score Service Web service	313
7.4.5	Add the components to the module assembly	315
7.4.6	Build the mediation flow	316
7.4.7	Test the mediation.	337
7.5	Calling the service from the application.	341
7.5.1	Import or copy the WSDL files	342
7.5.2	Generate the Web service client proxy	343
7.5.3	Update the application to call the service using the proxy	346
Chapter 8.	Building the CRM mediation	347
8.1	Scenario overview	348
8.1.1	Business scenario	348
8.1.2	CRM mediation	349
8.2	Developing a mediation to update a CRM system	351
8.2.1	Mediation development steps	351
8.2.2	Create the mediation module	352
8.2.3	Create an interface for each EIS system	352
8.2.4	Define the interface for the mediation	369
8.2.5	Add the components to the module assembly	372
8.2.6	Build the mediation flow	373
8.3	Calling the service from the application.	415
8.4	For more information.	416
Chapter 9.	Building the Register Shipping mediation	417
9.1	Scenario overview	418
9.1.1	Business scenario	418
9.1.2	Register Shipping mediation	419
9.2	Creating the Register Shipping Service emulator	421
9.2.1	Define the RegisterShippingService interface.	422
9.2.2	Create the Java component that implements the service	423
9.2.3	Implement the mediation flow for the service	426
9.2.4	Create the SOAP/JMS export binding.	427
9.3	Developing the Register Shipping mediation	429
9.3.1	Mediation development steps	429
9.3.2	Define the ShippingRegistration interface.	429

9.3.3 Create the mediation module	430
9.3.4 Implement the mediation flow	433
9.3.5 Export the module as a Web service	442
9.4 Testing the mediation	442
9.4.1 Testing the RegisterShippingService emulator	443
9.4.2 Test the Register Shipping mediation	444
9.5 Calling the service from the application	445
9.6 Considerations for handling arrays, decomposition	447
9.6.1 Handling an unknown number of input request elements	447
9.6.2 Handling multiple responses	448
Chapter 10. Building Log Registration mediation	449
10.1 Scenario overview	450
10.1.1 Business scenario	450
10.1.2 Log Registration mediation	451
10.2 Developing the mediation	452
10.2.1 Create the mediation module	454
10.2.2 Create the business object	454
10.2.3 Build the interface	456
10.2.4 Assemble the mediation components	457
10.2.5 Bind the imports to JMS	459
10.2.6 Build the mediation flow	462
10.3 Testing the mediation	467
10.3.1 Prepare the runtime	467
10.3.2 Test the mediation	468
10.4 Calling the service from the application	475
Chapter 11. Deploy with WebSphere ESB	479
11.1 Introduction to WebSphere ESB	480
11.1.1 Applications	481
11.1.2 Administration	481
11.1.3 Service integration bus	481
11.1.4 Web services support	482
11.1.5 Messaging support	484
11.1.6 Client support	485
11.1.7 Tivoli Access Manager	485
11.1.8 Common Event Infrastructure (CEI)	485
11.2 Working with profiles	486
11.2.1 Starting the profile creation wizard	486
11.3 Administrative console	487
11.4 Deploying mediation modules	490
11.5 Creating a service integration bus	490
11.6 Configuration for databases	491

11.6.1	Create a J2C authentication data entry for the database	491
11.6.2	Create a JDBC provider	492
11.6.3	Create a data source.	494
11.7	Configuration for adapter support	498
11.7.1	Create a J2C authentication data entry for Siebel	498
11.7.2	Create an output folder for the flat file.	501
11.8	Configuration for JMS bindings	501
11.8.1	Create a queue destination on the bus	502
11.8.2	Create a queue connection factory	503
11.8.3	Create a JMS queue	504
11.8.4	Creating a JMS activation specification	505
11.9	Connecting to WebSphere MQ	506
11.9.1	Configure WebSphere MQ	508
11.9.2	Configure the bus	514
11.9.3	Define a WebSphere MQ link	515
11.9.4	Create alias queues	519
11.9.5	Start the bus and WebSphere MQ connections	521
11.10	Deploying applications	523
11.10.1	Use the serviceDeploy command	523
11.10.2	Deploy an EAR file	524
11.10.3	Installing the ITSOMart applications	525
11.11	Testing ITSOMart	530
11.12	Network Deployment and clustering topologies	531
11.12.1	Workload management with a single cluster.	533
11.13	For more information.	553

Chapter 12. Service monitoring and management with IBM Tivoli Composite

	Application Manager SOA	555
12.1	Tivoli Composite Application Manager (ITCAM)	556
12.1.1	Composite applications.	558
12.2	IBM Tivoli Enterprise Monitoring framework	560
12.3	IBM Tivoli Composite Application Manager for SOA.	564
12.4	Tracking performance with ITCAM for SOA	566
12.4.1	Workspaces	566
12.4.2	Attributes.	571
12.4.3	Situations	573
12.4.4	Policies	574
12.4.5	Take Action commands.	575
12.4.6	Log files.	576
12.5	Monitoring ITSOMart.	577
12.5.1	Configure data collection.	578
12.5.2	Generate Web services traffic.	580
12.5.3	Enable Web service data logging	581

12.5.4 Content filtering	583
12.5.5 Using Web Services Navigator to analyze data	584
12.6 Summary	587
12.7 For more information	588
Appendix A. Sample application install summary	589
Overview	590
Prepare the development environment	591
Configure the workbench	591
Import the projects into the workbench	591
Prepare the runtime environment	593
Create a service integration bus	594
MessageLogApp application	595
Registration processor service	596
Runtime	596
Install the application	597
ITSOMart application	597
Credit check mediations	597
Create the database and configure the JDBC data source	597
Install the applications	600
CRM mediation	601
Register Shipping mediation	601
Registration Log Mediation	601
Create the bus destinations	601
Create the JMS queue connection factory	602
Create the JMS queues	602
Install the application	602
Common errors:	603
Appendix B. Tips and techniques	605
Creating a top-down SOAP/JMS Web service	606
Create the business object	606
Build the interface	607
Create an EJB project	608
Modify the WSDL file	610
Create JMS resources	619
Create the Web service	620
Implement the Web service	631
Create the Web service client	632
Test the Web service	638
Server errors in the test environment	639
Errors using XML Mapper without Internet connectivity	639
Creating a new server in the test environment	640

Create the WebSphere ESB Server profile	640
Changing WebSphere Integration Developer to use the new profile	651
Create a new server in the test environment to use the new profile	652
Installing WebSphere MQ Explorer as a plug-in	654
Appendix C. Installation details	657
Installing WebSphere Integration Developer	658
Using Rational Product Updater	662
Installing WebSphere Adapters	667
IBM WebSphere Adapter for Siebel Business Applications	668
IBM WebSphere Adapter for Flat Files	669
Installing Tivoli Composite Application Manager	671
IBM DB2 Universal Database installation	672
IBM Tivoli Monitoring installation	678
ITCAM for SOA Application Support installation	687
ITCAM for SOA Monitoring Agent installation and configuration	691
Web Services Navigator installation	698
Verify the installation	699
Appendix D. Additional material	701
Locating the Web material	701
Using the Web material	702
System requirements for downloading the Web material	702
How to use the Web material	703
Related publications	705
IBM Redbooks	705
Other publications	706
Online resources	706
How to get IBM Redbooks	709
Help from IBM	709
Index	711

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 about 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 or changes in the product(s) 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:

@server®
e-server®
Redbooks (logo) ™
developerWorks®
z/OS®
Ascendant®
AIX®
Candle®
Cloudscape™
CICS®

DataPower®
DB2 Universal Database™
DB2®
IBM®
IMS™
NetView®
OMEGAMON Monitoring Agent®
OMEGAMON®
Power PC®
Rational Unified Process®

Rational®
Redbooks™
RequisitePro®
RUP®
Tivoli Enterprise™
Tivoli Enterprise Console®
Tivoli®
WebSphere®

The following terms are trademarks of other companies:

EJB, Java, JavaBeans, JavaServer, JavaServer Pages, JDBC, JSP, JVM, J2EE, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Active Directory, Expression, Microsoft, Visio, 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.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

The IBM® SOA Foundation is an integrated, open standards based set of software, best practices, and patterns that is designed to provide you with what you need to get started with SOA. A set of SOA scenarios is being developed by IBM that describe key architectural scenarios for SOA solutions and bridge the gap between SOA and IBM products that can be used to implement these architectures.

This IBM Redbook focuses on the Service Connectivity scenario, which describes architectural solutions using an ESB. The focus of this scenario is the integration of service consumers and service providers across multiple channels.

You can find more information about the IBM SOA Foundation and the SOA scenarios in the following IBM Redbooks™:

- ▶ *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240
- ▶ *Patterns: SOA Foundation - Business Process Management Scenario*, SG24-7234

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.

Carla Sadtler is a certified IT Specialist at the ITSO, Raleigh Center. She writes extensively about the WebSphere® and Patterns for e-business areas. Before joining the ITSO in 1985, Carla worked in the Raleigh branch office as a Program Support Representative. She holds a degree in mathematics from the University of North Carolina at Greensboro.

Luciana Calcagno is an IT Specialist in Uruguay. She joined IBM in 2000. Luciana focuses on e-business application development, in particular Java™ and J2EE™ and WebSphere Application Server in projects with different lines of businesses. She became a specialist in WebSphere and in its related technologies. She holds a degree in Systems Engineering from Catholic University, Uruguay. Currently, she is also involved in pre-sales activities and in the development of projects based on SOA using WebSphere platform.

Ryan Cox is a software architect/technical specialist on the Cross-Brand Solution team in the IBM US sales organization. Ryan has provided pre-sales

technical support in various capacities for the IBM WebSphere brand for the past eight years. His expertise is in the design and implementation of enterprise solutions on the WebSphere platform, which over the years has included Component Broker, WebSphere Application Server, WebSphere Business Integration Server Foundation, WebSphere Portal Server, and more recently, WebSphere Process Server and WebSphere ESB. His current area of focus is around SOA technologies. He holds a BS in Computer Science from the University of North Texas, and currently lives in a small mountain town called Nederland, Colorado.

Jerry Denman is a Certified Executive IT Architect for IBM Global Business Services, based in Florida, USA. He has more than 20 years of IT experience, and his interests are in enterprise integration, service-oriented architect, event-driven architecture, business process management, integration patterns, and enterprise service bus architecture. His SOA and integration architecture experience includes projects for many of the world's largest enterprises. He is a member of the IBM SOA Center of Excellence and the Distribution Sector SOA Client Accelerator Team.

Abdul Shad is a Staff Software Engineer with the WebSphere Portal Server - Document Conversion Services team, India Software Lab, Bangalore. Prior to joining this team he was part of the WebSphere Adapters test team. Abdul joined IBM in 2003. He has six years experience in the IT industry. His areas of expertise include Java/J2EE, WebSphere Business Integration adapters, WebSphere Adapters, and WebSphere Application Server V6. He holds a Masters degree in Computer Applications.

Rodrigo Vargas is a Consulting IT Specialist working from San Diego for the IBM Software Services for WebSphere. His main area of expertise is Transaction Processing Systems. Rodrigo holds a degree in Computer and Systems Engineering from Universidad de los Andes in Bogota, Colombia, and a Master's degree in Computer Science from San Diego State University.

Cheng Zhou is the Business Integration Practice Lead at Ascendant® Technology. His areas of expertise include Java EE, SOA, and business process integration. While at Ascendant, he has led numerous engagements involving WebSphere Application Server, Portal, and various flavors of WebSphere Business Integration. A graduate of the University of Texas at Austin, he has more than 10 years of experience designing and building enterprise-class applications for Fortune 500 companies.

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

John Ganci
International Technical Support Organization, Raleigh Center

Martin Keen
International Technical Support Organization, Raleigh Center

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

Paula Díaz de Eusebio
AMS Integrated Delivery, IBM Spain

Adrian Spender
WebSphere ESB Development, IBM UK

Gabriel Telerman
WebSphere Software Services Consultant, IBM UK

Charlie Redlin
WebSphere Application Server WLM, IBM US

Erica Carmel
SOA User Experience, IBM US

Rachel Reinitz
SOA Senior Consultant, IBM US

Greg Flurry
IBM Enterprise Integration Solutions, IBM US

David Currie
EMEA Software Lab Services, IBM UK



Figure 1 Clockwise from top left: Ryan Cox, Carla Sadtler, Abdul Shad, Luciana Calcagno, Cheng Zhou, Rodrigo Vargas (Jerry Denman not pictured)

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 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. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Archived

IBM SOA Foundation

The IBM SOA Foundation is an integrated, open standards based set of IBM software, best practices, and patterns designed to provide what you need to get started with SOA from an architecture perspective. The key elements of the IBM SOA Foundation are the SOA life cycle (model, assemble, deploy, manage), reference architecture, and SOA scenarios.

The SOA Foundation scenarios (or simply SOA scenarios) are representative of common scenarios of use of IBM products and solutions for SOA engagements. The SOA scenarios communicate the business value, architecture, and IBM open standards-based software used within the SOA scenario. The SOA scenarios can be used as a reference architecture (starting point) to accelerate the SOA architecture and implementation of your customer scenario. The SOA scenarios can be implemented using an incremental SOA adoption approach, whereby a customer can incrementally add elements of other SOA scenarios to the environment to achieve their business objectives.

In this chapter we explore the following defining elements:

- ▶ SOA Foundation life cycle
- ▶ SOA Foundation Reference Architecture
- ▶ SOA Foundation scenarios
- ▶ Service Connectivity scenario

1.1 SOA Foundation life cycle

IBM customers have indicated that they think of SOA in terms of a life cycle. As seen in Figure 1-1, the IBM SOA Foundation includes the following life-cycle phases:

- ▶ Model
- ▶ Assemble
- ▶ Deploy
- ▶ Manage

There are a couple of key points to consider about the SOA life cycle. First, the SOA life cycle phases apply to all SOA projects. Second, the activities in any part of the SOA life cycle can vary in scale and the level of tooling used depending on the stage of adoption.

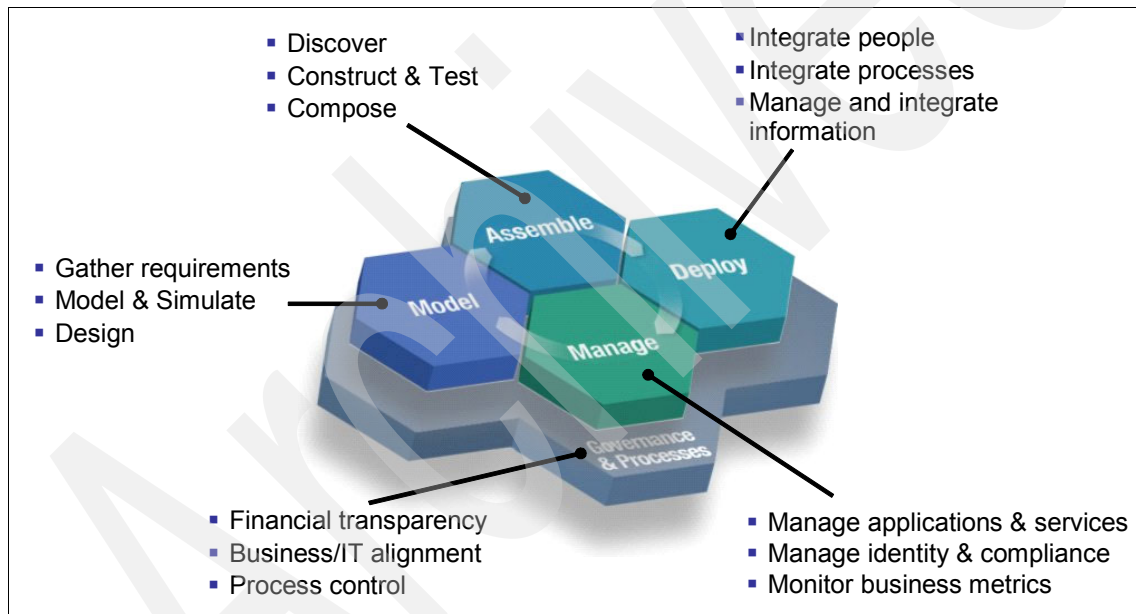


Figure 1-1 IBM SOA Foundation life cycle

Model

Modeling is the process of capturing the business design from an understanding of business requirements and objectives. The business requirements are translated into a specification of business processes, goals, and assumptions for creating a model of the business. Many businesses do not go through a formal modeling exercise. In some cases, businesses that do perform modeling use

primitive techniques such as drawing the design in Visio® or using text documents.

Capturing the business design using a sophisticated approach that includes the use of specialized tooling lets you perform *what-if* scenarios with various parameters the business may experience. The process can then be simulated using those parameters to predict the effect that process will have on the business and IT systems. If the achieved results do not match the business objectives, then the process definition can be refined.

The model will also capture key performance indicators, such as business metrics, that are important measurements of your business. For example, this could include a measure of the new accounts that you have opened in a given month. These key performance indicators are input to the assembly of the application. In addition, the indicators can be monitored in production to capture the critical data to measure whether the objectives are being met.

Assemble

The business design is used to communicate the business objectives to the IT organization that will assemble the information system artifacts that implement the design. The enterprise architect works closely with the business analyst to convert the business design into a set of business process definitions, as well as activities used to derive the required services from the activity definitions. The enterprise architect and business analyst work with the software architect to flesh out the design of the services.

During the process of resolving the design and implementation of the modeled business processes and services, a search should be performed of existing artifacts and applications in an effort to find components that meet the needs of the design. Some applications will fit perfectly, some will have to be re-factored, and some will have to be augmented to meet the requirements of the design.

These existing assets should be rendered as services for assembly into composite applications. Any new services required by the business design will need to be created. Software developers should use the SOA programming model to create these new services.

Lastly, the assemble phase includes applying a set of policies and conditions to control how your applications operate in the production runtime environment. For example, these policies and conditions include business and government regulations. In addition, the assemble phase includes critical operational characteristics such as packaging deployment artifacts, localization constraints, resource dependency, integrity control, and access protection.

Deploy

The deploy phase of the life cycle includes a combination of creating the hosting environment for the applications and the deployment tasks of those applications. This includes resolving the application's resource dependencies, operational conditions, capacity requirements, and integrity and access constraints.

A number of concerns are relevant to the construction of the hosting environment including the presence of the already existing hosting infrastructure supporting applications and pre-existing services. Beyond that, you must consider appropriate platform offerings for hosting the user interaction logic, business process flows, business services, Access Services, and information logic.

Manage

The manage phase includes the tasks, technology, and software used to manage and monitor the application assets such as services and business processes that are deployed to the production runtime environment.

Monitoring is a critical element of ensuring that the underlying IT systems and application are up and running to maintain the service availability requirements of the business. Monitoring also includes monitoring performance of service requests and timeliness of service responses. In addition, monitoring includes maintaining problem logs to detect failures in various services and system components, as well as localizing failures and restoring the operational state of the system.

Managing the system also involves performing routine maintenance; administering and securing applications, resources, and users; and predicting future capacity growth to ensure that resources are available when the demands of the business call for it. The security domain includes such topics as authentication, single sign-on, authorization, federated identity management, and user provisioning.

The manage phase also includes managing the business model, tuning the operational environment to meet the business objectives expressed in the business design, and measuring the success or failure of meeting those objectives. SOA is distinguished from other styles of enterprise architecture by its correlation between the business design and the software that implements that design, and its use of policy to express the operational requirements of the business services and processes that codify the business design. The manage phase of the life cycle is directly responsible for ensuring that those policies are being enforced, and for relating issues with that enforcement back to the business design.

Governance and processes

Governance and processes are critical to the success of any SOA project. Governance helps clients extend the planned SOA across the enterprise in a controlled manner. This new SOA governance capability helps customers set a baseline for measuring improvements, tracking SOA projects, building a pool of skilled resources, and establishing the structure for making decisions about SOA initiatives. Additionally, this services capability helps keep all SOA initiatives, architectures, and investments aligned to with business goals.

A key aspect of governance is that it defines rules, processes, metrics, and organizational constructs for support planning, decision making, steering, and control of IT and the SOA to achieve business objectives.

1.2 SOA Foundation Reference Architecture

This section describes the SOA Foundation Reference Architecture, which includes the components and middleware services used by applications in the runtime environment.

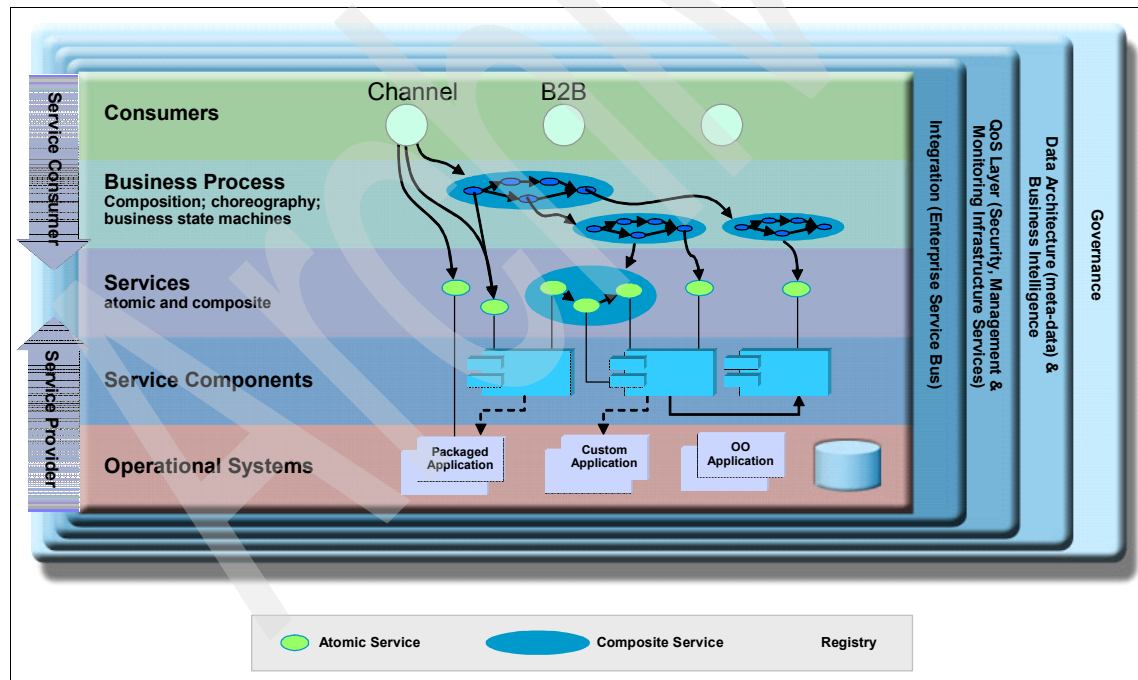


Figure 1-2 SOA Foundation Reference Architecture: solution view

Figure 1-2 on page 5 depicts the SOA Foundation Reference Architecture solution view used to decompose an SOA design. SOA puts a premium on the role of the *enterprise architect*, who is responsible for spanning between the business design and the information system that codifies that design.

When taking a top-down approach¹, the enterprise architect starts by identifying the business processes and business services used by the business users. The business users are consumers of the processes and services. Business processes should be treated as compositions of other business processes and services, and therefore should be decomposed into their subordinate sub-processes and services.

Services and business processes are then detailed into service components. Service components include a detailed set of definition metadata used to describe the service to the information system. Services can be aggregated into module assemblies. The module assemblies are used to establish related design concerns and begin the planning to determine what teams will collaborate to implement the related services to be deployed as a single unit.

The resulting set of business process definitions, services, and schemas make up the logical architecture of the application. The enterprise architect must then map that logical architecture to a physical architecture.

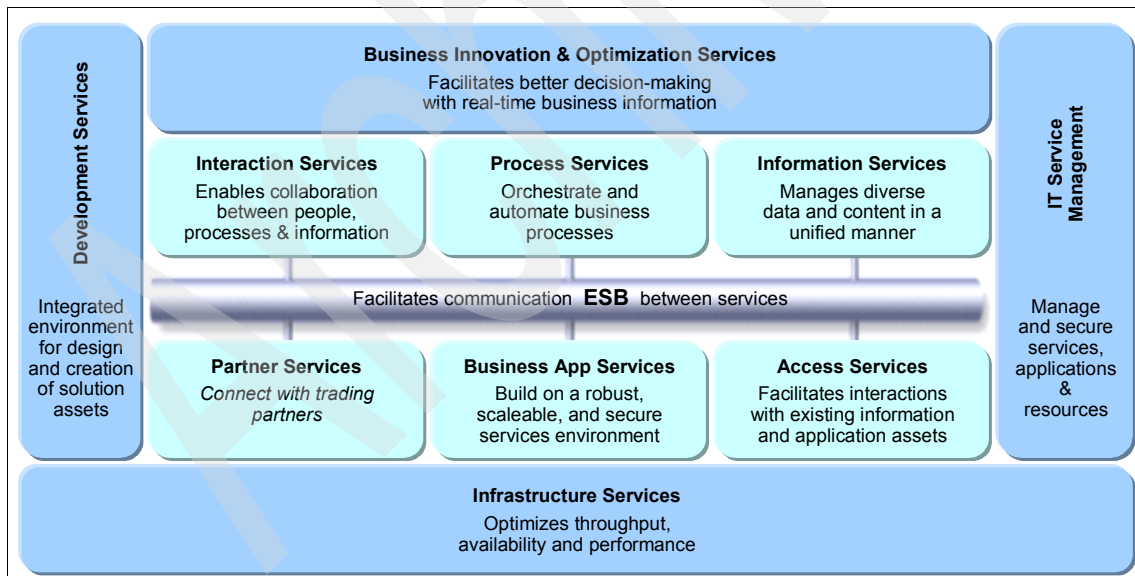


Figure 1-3 SOA Foundation Reference Architecture: middleware services view

¹ This flow describes a top-down approach. Other variations include a bottom-up approach and the more common meet-in-the-middle approach.

We include a summary description for each of the services found in the logical architecture displayed in Figure 1-3 on page 6. The services found in the center of Figure 1-3 on page 6 (Interaction, Process, Information, Partner, Business Application, Access) are the core set of services used by the application within the runtime environment when deployed. The other services displayed in the Figure 1-3 on page 6 (outer services) are used in support of the core services.

Core components of the logical architecture

This section includes a brief description of the following core components of the logical architecture:

- ▶ Interaction Services
- ▶ Process Services
- ▶ Business Application Services
- ▶ Information Services
- ▶ Access Services
- ▶ Partner Services

Interaction Services

Interaction Services provide the capabilities required to deliver IT functions and data to users, meeting their specific preferences.

Process Services

Process Services provide the control capabilities required to manage the flow and interactions of multiple services in ways that implement business processes.

Business Application Services

Business Application Services are called by service consumers. Service consumers include other components in the logical architecture such as portal or business processes.

Information Services

Information Services provide the capabilities necessary to federate, replicate, and transform disparate data sources.

Access Services

Access Services provide bridging capabilities between core applications, prepackaged applications, enterprise data stores, and the ESB to incorporate services that are delivered through existing applications into an SOA.

Partner Services

Partner Services provide the document, protocol, and partner management capabilities for business processes that involve interactions with outside partners and suppliers.

Supporting components of the logical architecture

This section includes a brief description of the supporting components of the SOA Foundation logical architecture used in support of the core components:

- ▶ Enterprise Service Bus
- ▶ Business Innovation and Optimization Services
- ▶ Development Services
- ▶ IT Service Management
- ▶ Infrastructure Services

Enterprise Service Bus

The Enterprise Service Bus (ESB), or simply bus, provides an infrastructure that removes the direct connection dependency between service consumers and providers. Consumers connect to the bus and not the provider that actually implements the service. This type of connection further decouples the consumer from the provider. A bus also implements further value-add capabilities such as security and delivery assurance. We prefer that you implement these capabilities centrally within the bus at an infrastructure level rather than within the application. The primary driver for an ESB, however, is that it increases decoupling between service consumers and providers.

Although building a direct link between a consumer and provider is relatively straightforward, these links can lead to an Interaction pattern that consists of building multiple point-to-point links that perform specific interactions. With a large number of interfaces this quickly leads to the buildup of a complex mass of links with multiple security and transaction models. When routing control is distributed throughout the infrastructure, there is typically no consistent approach to logging, monitoring, or systems management. This type of environment is difficult to manage or maintain and inhibits change.

Note: An ESB can be thought of as an architectural pattern, with an implementation to match the deployment needs. There are two IBM ESB products:

- ▶ IBM WebSphere Enterprise Service Bus
- ▶ IBM WebSphere Message Broker

In addition, there are a number of products that extend the capabilities of these ESBs, including DataPower® XML Security Gateway XS40.

Business Innovation and Optimization Services

Business innovation and optimization services are primarily used to represent the tools and the metadata structures for encoding the business design, including the business policies and objectives.

Business innovation and optimization services exist in the architecture to help capture, encode, analyze, and iteratively refine the business design. The services also include tools to help simulate the business design. The results are used to predict the effect of the design, including the changes the design will have on the business.

Development Services

Development services encompass the entire suite of architecture tools, development tools, visual composition tools, assembly tools, methodologies, debugging aids, instrumentation tools, asset repositories, discovery agents, and publishing mechanisms needed to construct an SOA-based application.

IT Service Management

Once the application has been deployed to the runtime environment it needs to be managed along with the IT infrastructure on which it is hosted. IT service management represents the set of management tools used to monitor your service flows, the health of the underlying system, the utilization of resources, the identification of outages and bottlenecks, the attainment of service goals, the enforcement of administrative policies, and recovery from failures.

Infrastructure Services

Infrastructure services form the core of the information technology runtime environment used for hosting SOA applications. These services provide the ability to optimize throughput, availability, performance, and management.

1.3 SOA Foundation scenarios

The SOA Foundation scenarios (or simply SOA scenarios) are representative of common scenarios of use of IBM products and solutions for SOA engagements. The SOA scenarios quickly communicate the business value, architecture, and IBM open standards-based software used within the SOA scenario. The SOA scenarios can be implemented as part of an incremental adoption of SOA growing from one scenario to using elements of multiple scenarios together. The concept of realizations are used to provide more specific solution patterns and IBM product mappings within the SOA scenarios.

The SOA scenarios can be used as a reference architecture implementation (starting point) to accelerate the SOA architecture and implementation of your customer scenario.

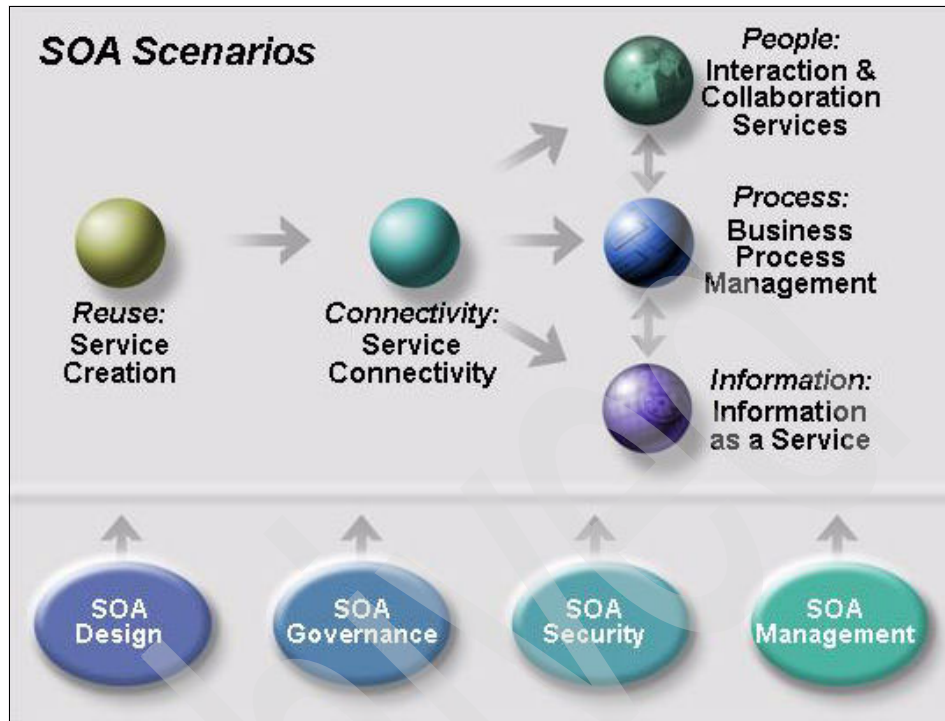


Figure 1-4 SOA scenarios and entry points

Figure 1-4 displays the SOA scenarios and the relationships between them:

- ▶ **Service Creation**

More details about this scenario can be seen in *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240.

- ▶ **Service Connectivity**

This book focuses on the Service Connectivity scenario.

- ▶ **Interaction and Collaboration Services**

- ▶ **Business Process Management**

More details about this scenario can be seen in *Patterns: Business Process Management with the SOA Foundation*, SG24-7234.

- ▶ **Information as a Service**

The scenarios can be used together and adopted incrementally. For example, the other scenarios commonly include service creation and often want connectivity. In addition, the scenarios can be used together, such as a portal

accessing a business process or a portal accessing an information service through an ESB from a service consumer.

SOA Design, Governance, Security, and Management can be used in each of the SOA scenarios based on customer requirements.

SOA Governance can be used to adopt SOA across the enterprise in a controlled manner with the objective of aligning the SOA initiative with the business objectives. Governance includes setting a baseline for measuring improvements, tracking SOA projects, building a pool of skilled resources, and establishing the structure for making decisions about SOA initiatives.

Companies that adopt an SOA need a solution for managing and monitoring services. In addition, they need of a security model that enables secure business transactions across enterprises and the Internet. The security domain includes topics such as authentication, single sign-on, authorization, federated identity management, and user provisioning.

1.4 Service Connectivity scenario

The Service Connectivity scenario is used to demonstrate the integration of service providers and consumers, allowing for the reuse of existing and new services across multiple channels. This scenario is appropriate for an enterprise that has a set of core services or systems that are to be made available as services to a variety of internal and external clients. Flexibility to make changes to service providers and make changes to service clients independent from each other is a requirement.

The focus of this scenario is on the underlying connectivity used to support business-centric SOA. An enterprise service bus provides decoupling between clients and providers, providing the flexibility to implement applications more quickly. In circumstances where services are provided to or consumed from a third party, an ESB gateway can be used in conjunction with the ESB to add security measures. An ESB gateway alone may be sufficient if all of your service interactions are with third parties and you have basic requirements to mediate between service consumers and providers.

Implementations of this scenario have the following features:

- ▶ Enables changes to the implementation of a service without affecting clients
- ▶ Registers services to a service registry
- ▶ Uses an enterprise service bus as the integration point between service providers and service consumers

- ▶ Enables clients to access a service with a different interface and protocol than what the service consumer supports
- ▶ Uses an ESB gateway to isolate and protect services
- ▶ Enables management and monitoring of services to insure service level agreements
- ▶ Provides security and credential mapping (where needed) to insure proper use of the services

Specific connectivity and integration requirements for an enterprise ultimately drive product selection of the ESB and supporting products. The choice of runtime products may include one or more of the following:

- ▶ WebSphere Message Broker
- ▶ WebSphere Enterprise Service Bus
- ▶ IBM DataPower SOA Appliances
- ▶ Web Services Gateway (WebSphere Application server Network Deployment V6)
- ▶ WebSphere Adapters
- ▶ WebSphere Service Registry and Repository (available 2H06)

The choice of SOA life cycle products depends largely on the runtime products selected. The following products can be used to support the runtime environment:

- ▶ WebSphere Message Broker Toolkit
- ▶ Rational® Application Developer
- ▶ WebSphere Integration Developer
- ▶ IBM Tivoli® Composite Application Manager for SOA (ITCAM for SOA)
- ▶ IBM Tivoli Composite Application Manager for WebSphere (ITCAM for WebSphere)
- ▶ IBM Tivoli Composite Application Manager for RTT V6.0 (TCAM for RTT)
- ▶ OMEGAMON® for Messaging
- ▶ Tivoli Access Manager
- ▶ Tivoli Federated Identity Manager

Realizations have been developed that will help you understand how the scenario can be used and how products are selected. A realization is an example business case that describes a customer situation and the solution. We have included a summary of the following common realizations of the Service Connectivity scenario:

- ▶ Secure connections to third parties and trading partners: For use when interactions with third parties are present and mediation requirements are basic
- ▶ Internal connectivity based on open standards: For use with standards-based interactions that require routing capabilities
- ▶ Adapt from third-party systems to Web services: For use when requiring access to EIS systems
- ▶ Deliver an existing process through new business channels: For use in a diverse, non-standards based environment

1.4.1 Secure connections to third parties and trading partners

An ESB gateway can be used alone or in conjunction with an ESB to provide controlled and secure service interaction between internal or external domain boundaries. In this realization, its primary function is to provide secure access to resources when interacting with third parties. An ESB gateway can also provide basic functionality such as protocol switching and message switching to enable interaction between service consumers and service providers.

This realization assumes that the customer has adopted standards-based technology, has an existing infrastructure, and has the following business requirements:

- ▶ Standards-based requestors/providers use SOAP/HTTP for transport.
- ▶ Dynamically add new providers and requestors at runtime.
- ▶ Support a defined, high response time with a moderate load.
- ▶ SOA security for interaction with requestors and providers. Security may need to be adapted between the requestors and providers.
- ▶ Requests and responses must be logged to a file.

The following technical requirements have been identified:

- ▶ Many services deployed require the same mediation flow. An ESB gateway minimizes administration and streamlines the process for making new services available.
- ▶ Services must be monitored for performance and usage.

- ▶ Monitoring for all components must be integrated into the existing management infrastructure.

The IBM products used for this realization are as follows:

- ▶ Deploy: IBM DataPower XML Security Gateway XS40
The customer chooses the DataPower XS40 model for the runtime. The XS40 is designed specifically to provide XML acceleration and SOA security and can provide the basic mediation functions required. Because the requirements for mediating the interaction between consumers and providers is met, the XS40 as an ESB gateway is sufficient and no ESB product is required.
- ▶ Assemble: DataPower Toolkit
- ▶ Manage: IBM Tivoli Composite Application Manager for SOA
ITCAM for SOA is used to monitor Web services flowing through the DataPower appliance.
- ▶ Manage: Tivoli Access Manager
The XS40 can be integrated with Tivoli Access Manager to secure applications.

Figure 1-5 shows an overview of the runtime topology.

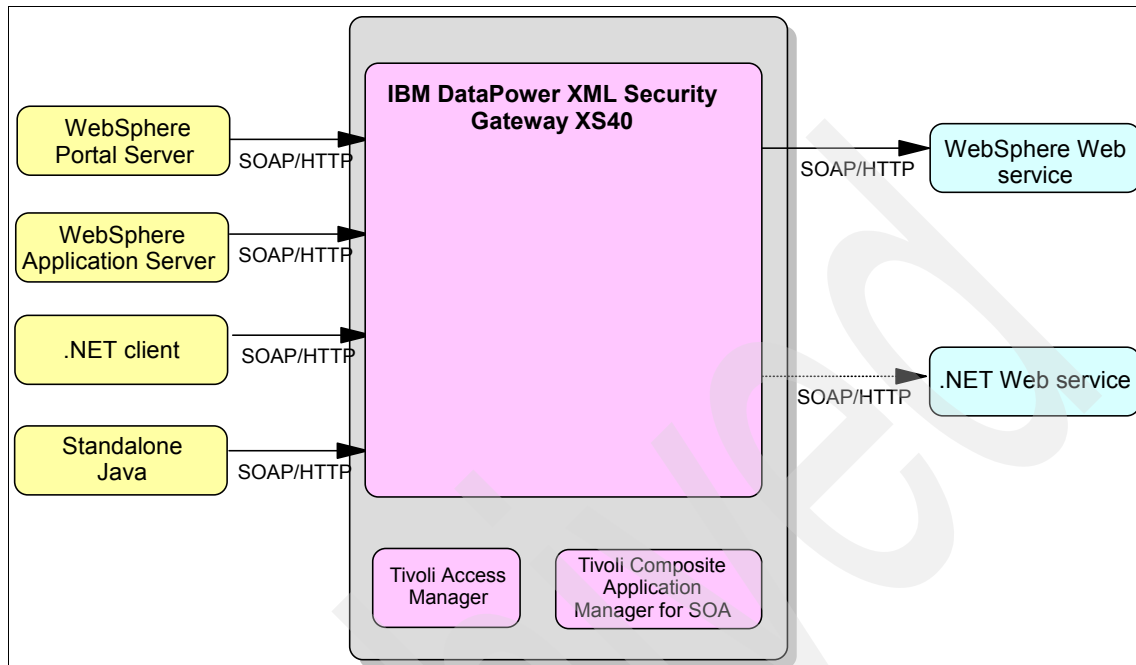


Figure 1-5 Gateway runtime topology

1.4.2 Internal connectivity based on open standards

This solution provides multi-channel access for clients to an existing service with a range of connectivity options for standards-based clients and services. With this type of connectivity, a client can request a secure service without knowledge of its location. Transparent to the client, requests can be routed to the service that can best handle the request. Also transparent to the client is the message format and transport protocol required to access the provider. The response could be immediate or delayed.

This realization assumes that the customer has adopted standards-based technology, has an existing WebSphere Application Server infrastructure, and has the following business requirements:

- ▶ Provide integration of multiple client channels to service providers.
- ▶ Provide routing of client requests to the appropriate service provider.
- ▶ Intranet environment that does not require WS-Security or other complex security considerations.
- ▶ Support moderate volume of requests.

The following technical requirements have been identified:

- ▶ Message data from clients must be examined in order to determine the service provider to route the request to.
- ▶ Clients and service providers use JMS, SOAP/JMS, or SOAP/HTTP.
- ▶ Data transformation is required. This should be done with XSLT.

The core IBM products used for this realization are as follows:

- ▶ Deploy: WebSphere Enterprise Service Bus
WebSphere ESB provides the transport flexibility to support the transports required by the customer. WebSphere ESB also has the mediation capabilities required to perform the message routing and transformation required.
- ▶ Assemble: WebSphere Integration Developer
WebSphere Integration Developer is the development and assembly tool for building WebSphere ESB mediations.
- ▶ Manage: IBM Tivoli Composite Application Manager for SOA
ITCAM for SOA will be used to monitor and manage message traffic between Web services.
- ▶ Governance: WebSphere Service Registry and Repository
WebSphere Service Registry and Repository is used to register business services for endpoint lookup.

Figure 1-6 shows an overview of the runtime topology.

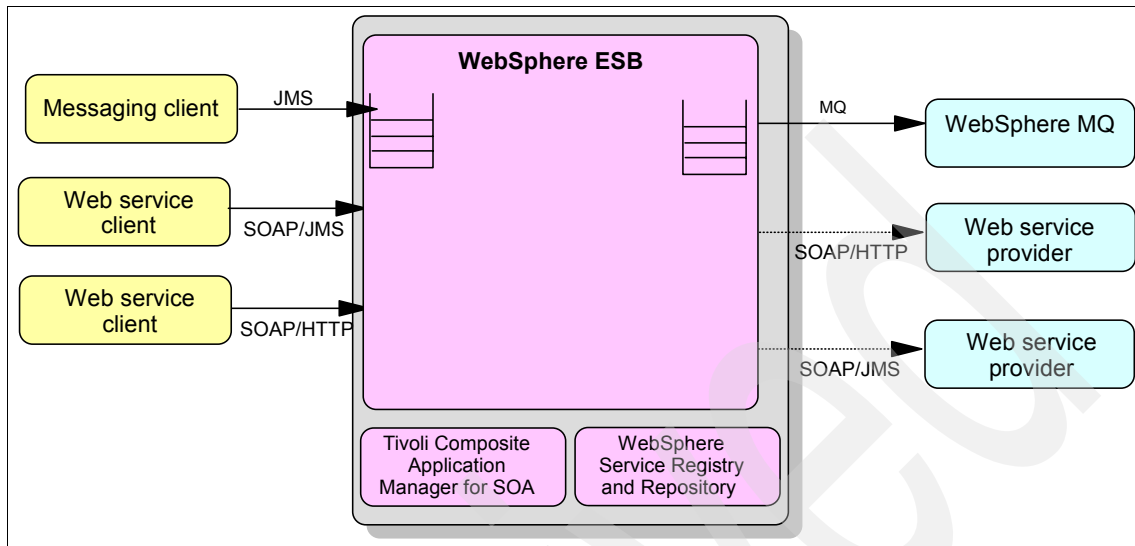


Figure 1-6 Local integration runtime

1.4.3 Adapt from third-party systems to Web services

An ESB can be used to provide access to EIS systems through the use of adapters. Mediations in the ESB are used to adapt the client request to a form understood by the adapter, and then to adapt the response to the client's format.

This realization assumes that the customer has adopted standards-based technology, has an existing WebSphere Application Server infrastructure, and has the following business requirements:

- ▶ Provide Web service access to functionality in an enterprise information system such as SAP R/3, PeopleSoft, or Oracle Financials.
- ▶ The intranet environment does not require WS-Security or other complex security considerations.
- ▶ The integration is based on message exchange/data replication scenarios — there is no business process or data synchronization between clients and EIS systems.
- ▶ It supports a moderate volume of requests.

The following technical requirements have been identified:

- ▶ The targeted integration is point-to-point, although multiple EISs can be exposed as Web services at the same time.

- ▶ Data transformation is required. This should be done with XSLT.
- ▶ Log the messages as they flow through the hub — we want to log asynchronously to a file.

The IBM products used for this realization are as follows:

- ▶ **Deploy: WebSphere Enterprise Service Bus and WebSphere Adapters**
WebSphere ESB supports the SOAP/HTTP transport required by the customer. WebSphere Adapters provide the EIS adapters required. WebSphere ESB also provides the mediation capability required to do XSLT transformation on the data and includes a logging function to log messages as they flow through the mediation.
- ▶ **Assemble: WebSphere Integration Developer**
WebSphere Integration Developer is the development and assembly tool for building WebSphere ESB mediations. It includes the enterprise discovery capabilities needed to incorporate the WebSphere Adapters into the mediation applications.
- ▶ **Manage: IBM Tivoli Composite Application Manager for SOA and IBM Tivoli Federated Identity Manager**
ITCAM for SOA will be used to monitor and manage message traffic between Web services. Tivoli Federated Identity Manager is used to manage identity and access to resources that span companies or security domains.

Figure 1-7 shows an overview of the runtime topology.

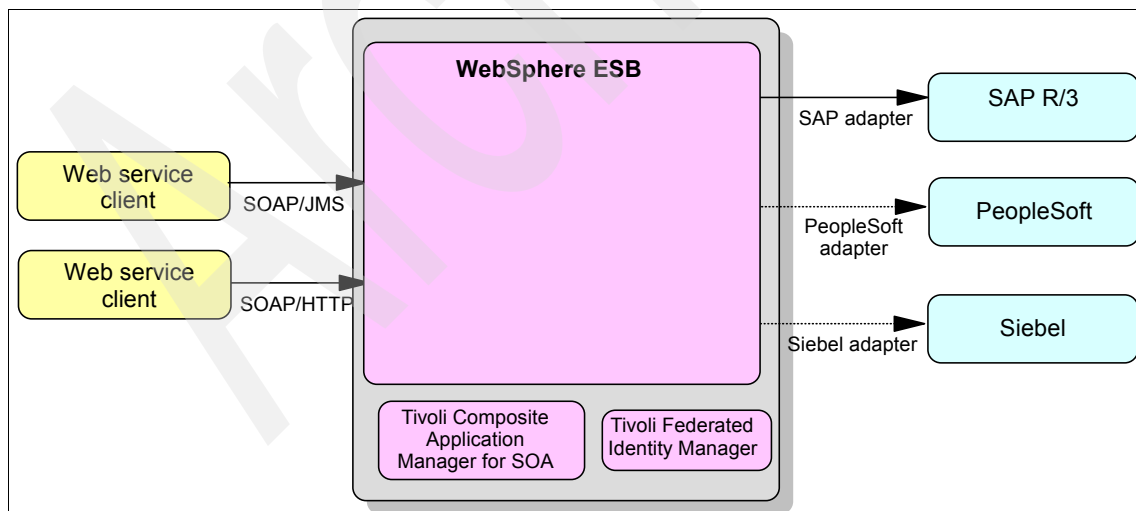


Figure 1-7 Adapt from third-party systems to Web services - runtime

1.4.4 Deliver an existing process through new business channels

An integration solution that includes a range of diverse business applications must provide connectivity for a wide range of service consumers and service providers as well as advanced options for message mediation, including message augmentation, message routing, and the ability to decompose messages into multiple requests and to recompose the responses.

This type of connectivity would provide the most advanced options for integrating dissimilar and wide-spread service consumers and service providers. Clients can request a secure service that may be provided by one or more service providers with the service composition occurring within the ESB. Services and clients also have a wide range of connectivity options. Connectivity to heritage applications as well as standards-based applications are managed by the ESB.

This realization assumes that the customer has extensive heritage systems as well as newer Web services based systems and has the following business requirements:

- ▶ Providers use a variety of heterogeneous protocols.
- ▶ Any provider must be accessible via basic Web services that will be used by a variety of clients.
- ▶ They must support moderate volume of requests.
- ▶ The intranet environment does not require SOA security or other complex security considerations.
- ▶ Global transactions across multiple heterogeneous transaction managers for some providers.

The following technical requirements have been identified:

- ▶ The ESB must support communication protocol conversion.
- ▶ The ESB must support flexible data model conversion, with acceptable performance and adequate tooling.
- ▶ There must be an enterprise class persistent messaging backbone.
- ▶ There must be global transactions management.
- ▶ The ESB must *adapt* the service definitions between the requestors and providers.

The IBM products used for this realization are as follows:

- ▶ Deploy: WebSphere Message Broker, WebSphere MQ, and WebSphere Adapters

WebSphere Message Broker is selected to provide the ESB capabilities, including mediation support. WebSphere MQ will be used to provide an

enterprise class persistent messaging backbone. This combination supports the wide variety of transport protocols and conversions required for the integration solution. WebSphere Adapters provide connectivity to traditional systems.

- ▶ **Assemble: Message Brokers Toolkit**

The Message Brokers Toolkit is the development tool for building mediation message flows in WebSphere Message Broker and provides the runtime configuration and management tools.

- ▶ **Manage: IBM Tivoli Federated Identity Manager, IBM Tivoli Access Manager**

Tivoli Access Manager provides a centralized, flexible, and scalable access control solution that enables you to control user access to protected information and resources. Tivoli Federated Identity Manager is used to manage identity and access resources that span companies or security domains.

- ▶ **Manage: IBM Tivoli OMEGAMON XE for Messaging**

Tivoli OMEGAMON XE for Messaging is used to monitor and manage the WebSphere MQ and WebSphere Message Broker environments.

Figure 1-8 shows an overview of the runtime topology for this realization.

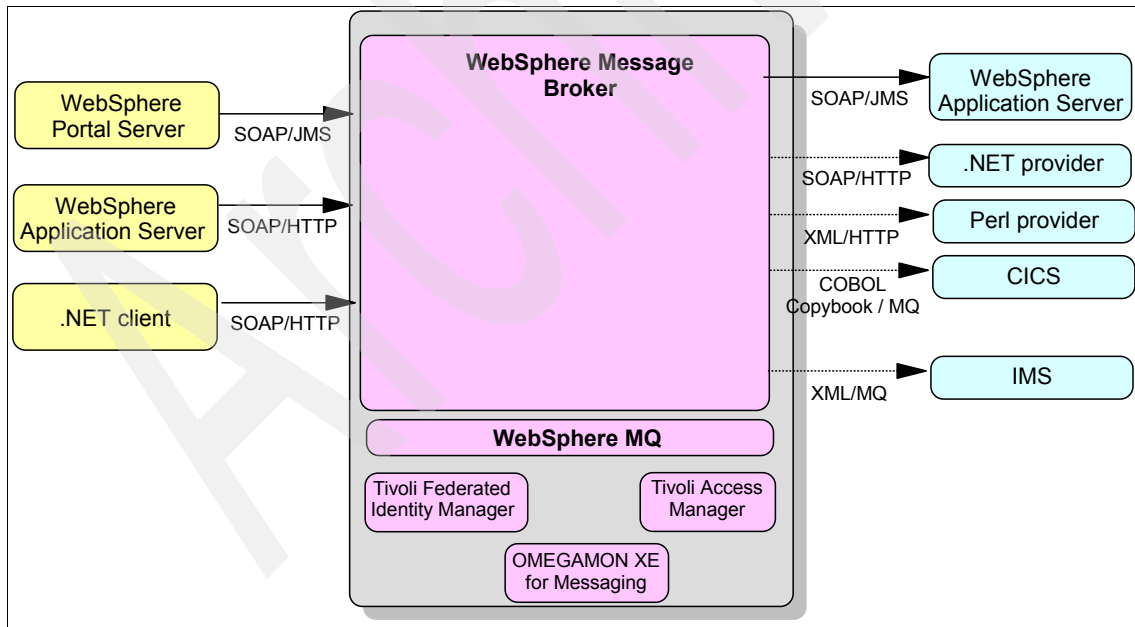


Figure 1-8 Expose existing systems to heterogeneous clients - runtime

Process for applying SOA scenarios

A process for selecting SOA scenarios and related reusable patterns assets has been designed and documented in *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240. This process can be use by specialists in pre-sales and post-sales roles, and by architects in a customer/consulting engagement.

Within this process, generic use cases have been designed to be representative of SOA capabilities possible with the software used in the SOA scenarios. The process includes a selection table that maps the generic use cases to the appropriate SOA scenario. Once the SOA scenario is identified, the user can then reference the related reusable patterns assets to help accelerate the solution design.

This chapter gives a brief overview of the SOA scenario selection process and discusses the generic use cases that lead to the Service Connectivity scenario. It then gives a brief introduction to Patterns for e-business.

This chapter contains the following topics:

- ▶ Process for using SOA scenarios and patterns
- ▶ Reuse patterns assets to accelerate the solution architecture

2.1 Process for using SOA scenarios and patterns

This section gives a brief overview of the process for using SOA scenarios and patterns. This process is discussed in detail in *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240.

Figure 2-1 depicts the high-level process for applying the SOA scenarios and related patterns.

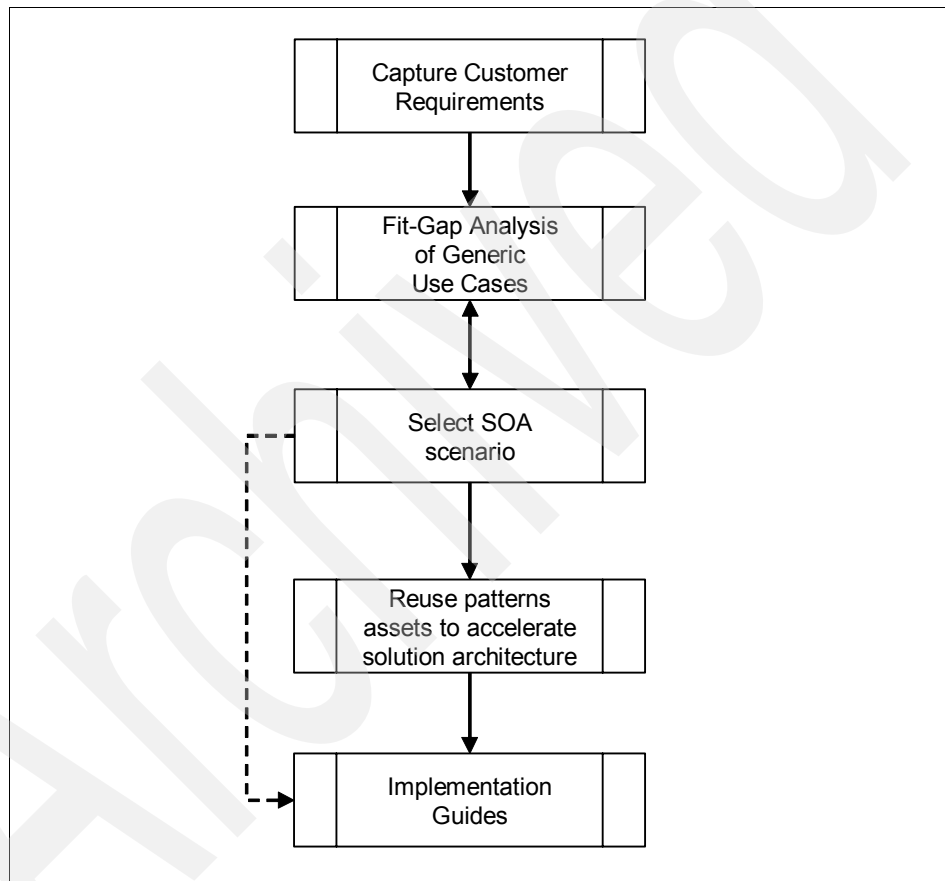


Figure 2-1 Process for using SOA scenarios and patterns

Below is a brief description of each phase of the high-level process depicted Figure 2-1:

1. Capture the Customer Requirements.

The collection of the customer requirements is a standard activity for development methodologies (for example, RUP® or IBM Method). This

includes an initial context, a vision of where the customer wants to be, and use cases representing the refined customer requirements.

2. Fit-gap analysis of generic use cases.

The business analyst or architect evaluates which of the generic use cases are representative of the defined customer requirements (use cases).

Note: The upward pointing arrow in Figure 2-1 on page 22 between *Select the SOA scenario* and *Fit-Gap Analysis* is a special case where the user already owns the software representative of an SOA scenario, and wants to know how to leverage the SOA capabilities of the software. As is the case in this book, the user knows the SOA scenario and will look at what generic use cases are possible in the SOA scenario selection table (see Table 2-1 on page 24).

3. Select the SOA scenario.

The business analyst or architect can use the SOA scenario selection table (see Table 2-1 on page 24), which lists the generic uses cases and the SOA scenario that enables the use case.

Once you have selected the SOA scenario, you have two options of where to go next:

- Reuse patterns assets to accelerate the solution architecture.

This option is appropriate if you are interested in reusable assets for the architecture and design of the solution. This option also leads to the implementation guides.

or

- Select the implementation guide (model, assemble, deploy, manage).

This option is appropriate if you are interested in a working example that demonstrates how to implement the scenario (model, assemble, deploy, manage).

4. Reuse patterns assets to accelerate the solution architecture.

We have mapped the reusable assets found in the Patterns for e-business for each of the SOA scenarios. The reusable assets are used to accelerate the solution architecture.

If you select this option, you will first learn about Patterns for e-business in 2.2, “Reuse patterns assets to accelerate the solution architecture” on page 26, and then proceed to Chapter 3, “Patterns for e-business and Service Connectivity” on page 31.

5. Select the implementation guide (model, assemble, deploy, manage).

Once you have identified the SOA scenario, and optionally the corresponding patterns, you can go to the corresponding implementation guide. The implementation guides provide detailed examples demonstrating how to implement selected use cases for the SOA offering. This includes implementation details for each phase of the life cycle using the solution offering software (model, assemble, deploy, manage).

This book is the implementation guide for the Service Connectivity scenario. Details about designing and implementing the ITSOMart solution can be seen starting with Chapter 4, “Planning for connectivity” on page 67.

Other works of interest are:

- *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212
- *Patterns: Implementing Self-Service in an SOA Environment*, SG24-6680
- *Enabling SOA Using WebSphere Messaging*, SG24-7163

2.1.1 SOA scenario selection table

A set of generic use cases has been developed to be used as selection criteria to determine the appropriate SOA scenario. Table 2-1 lists the generic use cases and corresponding SOA scenarios that fulfill the use case requirement.

Table 2-1 SOA scenario selection criteria

Generic use cases selection criteria	Service Creation	Service Connectivity	Interaction and Collaboration Services	Business Process Management	Information as a Service
U1: Reuse existing or create new application logic as a service within the enterprise.	X				
U2: Reuse existing or create new application logic as a service beyond the enterprise.	X				
U3: Point-to-point integration of enterprise applications using services.	X				
U4: Point-to-point integration of intra-enterprise applications using services.	X				
U5: Allow users to invoke services simply.	X				

Generic use cases selection criteria	Service Creation	Service Connectivity	Interaction and Collaboration Services	Business Process Management	Information as a Service
U6: Enable loose coupling of service consumers and providers using static routing.		X			
U7: Enable loose coupling of service consumers and providers using dynamic routing based on standards-based protocols.		X			
U8: Enable loose coupling of service consumers and providers using advanced dynamic routing and diverse protocols.		X			
U9: Improve an existing business process flow through business process and policy modeling and simulation.				X	
U10: Implement a new business process flow.				X	
U11: Analyze existing business process flow using monitoring.				X	
U12: Allow single-sign-on access to different services.			X		
U13: Personalize information based on user profile.			X		
U14: Allow users to create and manage content.			X		
U15: Allow users to Access Services through client devices.	X		X		
U16: Allow users to perform information inquiries.					X
U17: Populate information.					X
U18: Allow users seamless access to diverse data sources.					X

Because this book focuses on the Service Connectivity scenario, we next look specifically at the use cases that lead you to this scenario. If these use cases do not apply to your situation, consider going back to *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240, to explore the remaining use cases.

Generic use cases for the Service Connectivity scenario

The following generic use cases have been identified as leading to the Service Connectivity scenario:

- ▶ U6: Enable loose coupling of service consumers and providers using static routing.

A customer wishes to decouple the point-to-point connections between service consumers and service providers. A middle tier is added to provide routing between the consumer and provider. The middle tier provides static routing, where a request for a particular service from a consumer is always routed to the same service provider. The middle tier provides connection security and basic protocol and message transformation between consumers and providers.

- ▶ U7: Enable loose coupling of service consumers and providers using dynamic routing and standards-based protocols.

A customer wishes to decouple the point-to-point connections between service consumers and service providers. A middle tier is added to provide routing between the consumer and provider. Dynamic routing determines the service provider to use based on routing rules applied to the request received from the service consumer. The middle tier provides connection security. It also provides basic protocol and message transformation between standards-based consumers and providers.

- ▶ U8: Enable loose coupling of service consumers and providers using advanced dynamic routing and diverse protocols.

A customer wishes to extend the loose coupling between service consumers and service providers to include conversion of the message format or protocol used by the service consumer to a suitable message format and protocol understood by the service provider. This requires a middle tier between the service consumer and provider to perform the conversion. The middle tier provides an advanced degree of routing logic, allowing the invocation of multiple services providers from a single service consumer request, in which case the middle tier must disaggregate and aggregate requests and responses from the service providers. The middle tier provides connection security. It also provides advanced protocol and message transformation between a diverse set of consumers and providers.

2.2 Reuse patterns assets to accelerate the solution architecture

The target audience of this section is IT architects interested in leveraging the knowledge captured in the Patterns for e-business, with the objective of accelerating the solution architecture.

2.2.1 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 about the problem, developing an outline of 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. Reusing proven assets saves time, money, and effort and helps ensure delivery of a solid, properly architected solution.

IBM Patterns for e-business (P4eb) 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.

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:

- ▶ *Customer requirements*, which ultimately define the selected solution. We have developed a series of generic use cases that are representative of common SOA scenarios. The customer requirements are analyzed and when appropriate mapped to the generic use cases. The generic use cases are used as selection criteria to identify the appropriate SOA scenario and reusable patterns assets.
- ▶ *Business patterns*, which identify the interaction between users, businesses, and data.
- ▶ *Integration patterns*, which tie multiple Business patterns together when a solution cannot be provided based on a single Business pattern.
- ▶ *Composite patterns*, which represent commonly occurring combinations of Business patterns and Integration patterns.

- ▶ *Application patterns*, which provide a conceptual layout that describes how the application components and data within a Business pattern or Integration pattern interact.
- ▶ *Runtime patterns*, which define the logical middleware structure that supports and Application pattern. Runtime patterns depict the major middleware nodes, their roles, and the interfaces between these nodes.
- ▶ *Product mappings*, which identify proven and tested software implementations for each Runtime pattern.
- ▶ *Best-practice guidelines*, which discuss design, development, deployment, and management of e-business applications.

Figure 2-2 shows these assets and their relationships to each other.

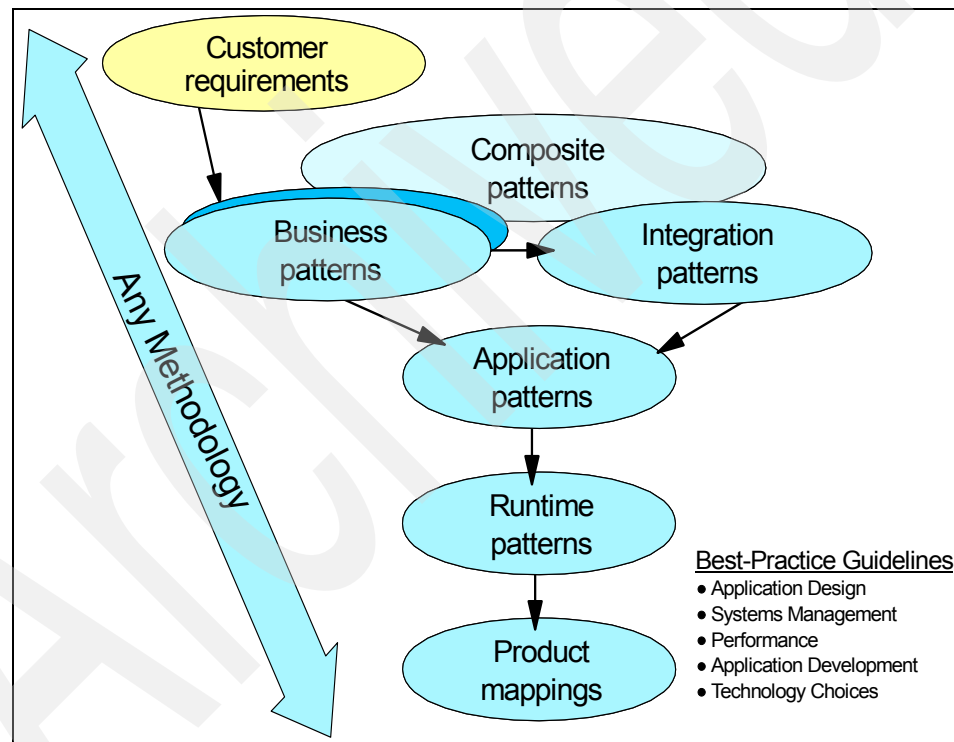


Figure 2-2 The Patterns for e-business layered asset model

Summary description of Business patterns

Table 2-2 provides a brief description of the Business, Integration, and Composite patterns.

Table 2-2 Summary description for the Business, Integration, and Composite patterns

Business, Integration, Composite Pattern	Description
Self-Service	The Self-Service business pattern captures the essence of direct interactions between interested parties and a business. Interested parties include customers, business partners, stakeholders, employees, and all other individuals with whom the business intends to interact.
Collaboration	The Collaboration business pattern enables interaction and collaboration between users. This pattern can be observed in solutions that support small or extended teams that need to work together in order to achieve a joint goal.
Extended Enterprise	The Extended Enterprise business pattern covers interactions between applications from different enterprises. This pattern can be observed in solutions that implement programmatic interfaces to connect inter-enterprise applications.
Information Aggregation	The Information Aggregation business pattern exists in e-business solutions that allow users to access and manipulate data that is aggregated from multiple sources. This Business pattern captures the process of taking large volumes of data, text, images, video, and so on, and using various user-controlled tools to extract useful information from them.
Application Integration	The Application Integration pattern serves to integrate multiple Business patterns or to integrate applications and data within an individual Business pattern.
Access Integration	The Access Integration pattern gives users a single, consistent, and seamless access mechanism to various applications that would otherwise require the use of several different access mechanisms.
Portal Composite	The Portal Composite pattern leverages various mechanisms (for example, content management, user interface formatting and display, data aggregation) to bring together the appropriate information and existing systems to serve the goals of the business.

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 and 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/>

2.2.2 Patterns for the SOA scenarios

Once we know which SOA scenario meets our customer requirements, the next step is to identify reusable assets found in the Patterns for e-business that we can leverage to accelerate the creation of our solution architecture.

This book is the primary reference for the Service Connectivity scenario.

For a description of patterns that apply to the Service Connectivity scenario, see Chapter 3, “Patterns for e-business and Service Connectivity” on page 31.

Other works of interest are:

- ▶ Self-Service business pattern
 - The Self-Service business pattern found at:
<http://www.ibm.com/developerworks/patterns/u2b/index.html>
 - *Patterns: Implementing Self-Service in an SOA Environment*, SG24-6680
- ▶ Application Integration pattern
 - The Application Integration pattern found at:
<http://www.ibm.com/developerworks/patterns/application/index.html>
 - *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6*, SG24-6494
- ▶ Extended Enterprise business pattern
 - The Extended Enterprise business pattern found at:
<http://www.ibm.com/developerworks/patterns/b2bi/index.html>
 - *Patterns: Extended Enterprise SOA and Web Services*, SG24-7135

Patterns for e-business and Service Connectivity

A process for selecting SOA scenarios and related reusable patterns assets was outlined in Chapter 2, “Process for applying SOA scenarios” on page 21. Within this process, you were given the option to reuse patterns assets to accelerate solution architecture. That option leads you to this chapter, which gives a brief overview of the patterns relevant to the Service Connectivity scenario.

In Chapter 4, “Planning for connectivity” on page 67, you will see an example of using the process to select a SOA scenario and the appropriate Business and Application pattern.

This chapter contains the following topics:

- ▶ The role of an enterprise service bus
- ▶ Business pattern and Application pattern selection
- ▶ Runtime pattern selection
- ▶ Self-Service business pattern
- ▶ Application Integration pattern
- ▶ Extended Enterprise business pattern

3.1 The enterprise service bus

One of the first questions we often encounter is “What is an enterprise service bus?” The favorite answer is “It depends.” This is due to the overloading of the term. You must understand the context of the question before the answer is apparent. The enterprise service bus is a physical component of the IBM SOA Foundation and a design pattern that is widely accepted throughout the industry.

An ESB runtime pattern has been identified and detailed in *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6*, SG24-6494.

Figure 3-1 illustrates the IBM Patterns for e-business ESB Pattern.

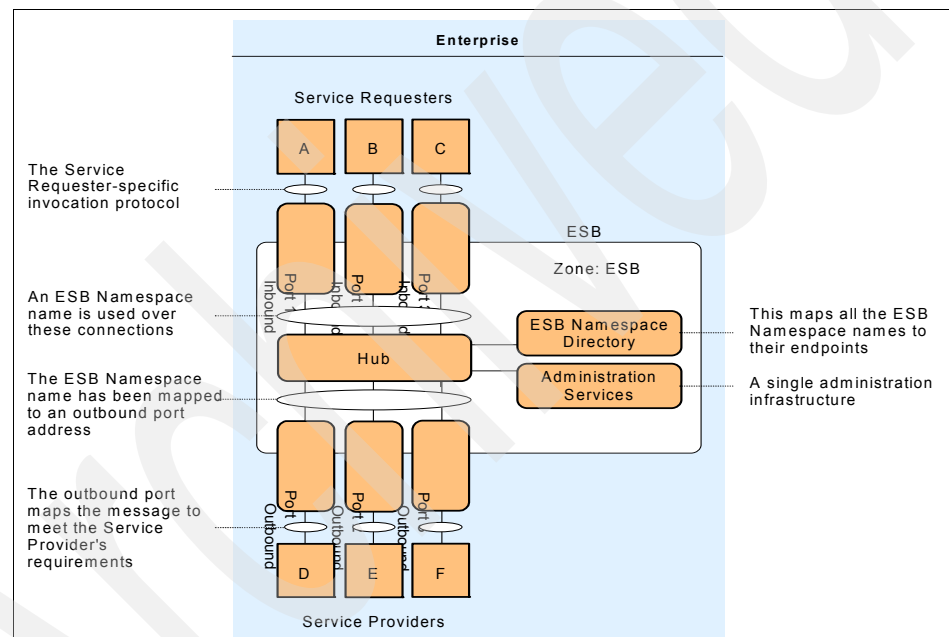


Figure 3-1 ESB Pattern

There are a few key differences between this design pattern and previous design patterns (for example, hub-and-spoke). The ESB design pattern has specific components that are not a part of previous integration design patterns. For example, most EAI design patterns did not show a namespace directory. The concept of a namespace comes from XML and allows two elements to have the same name so long as their name spaces are different. It is also important to note that an ESB is under the control of a single administrative services infrastructure.

Tip: For more information about IBM Patterns for e-business go to:
<http://www.ibm.com/developerworks/patterns>

In Figure 3-2 the ESB is depicted as a logical component in a service-oriented architecture. It acts as the mediator between the service consumers and service providers. The service providers and service consumers never interact directly. They always use the ESB as a mediator. The ESB provides services to resolve differences in protocol and format, and decouples the service consumer from the service provider.

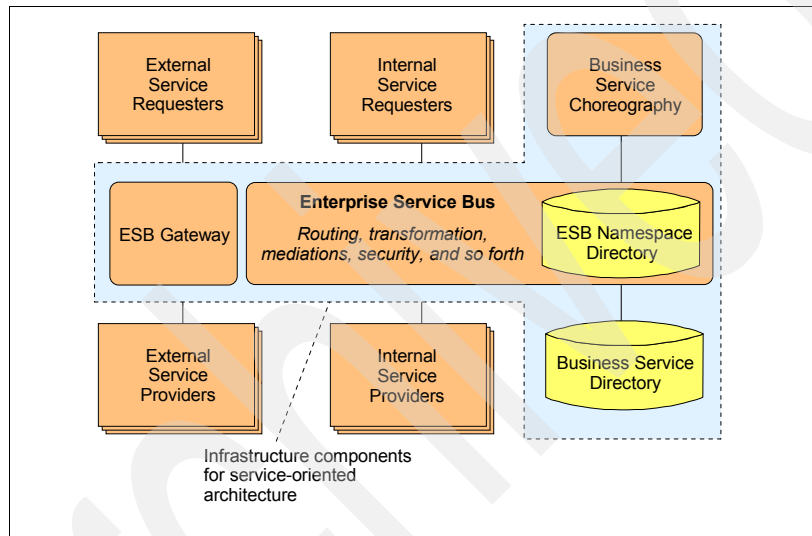


Figure 3-2 ESB and SOA

The ESB is a software package. IBM currently offers two ESB products that serve two different markets. WebSphere ESB is built on proven messaging and Web services technologies, and it provides standards-based Web services connectivity and XML data transformation. WebSphere Message Broker is an advanced ESB product that provides universal connectivity (including Web services) and any-to-any data transformation. See Figure 3-3.

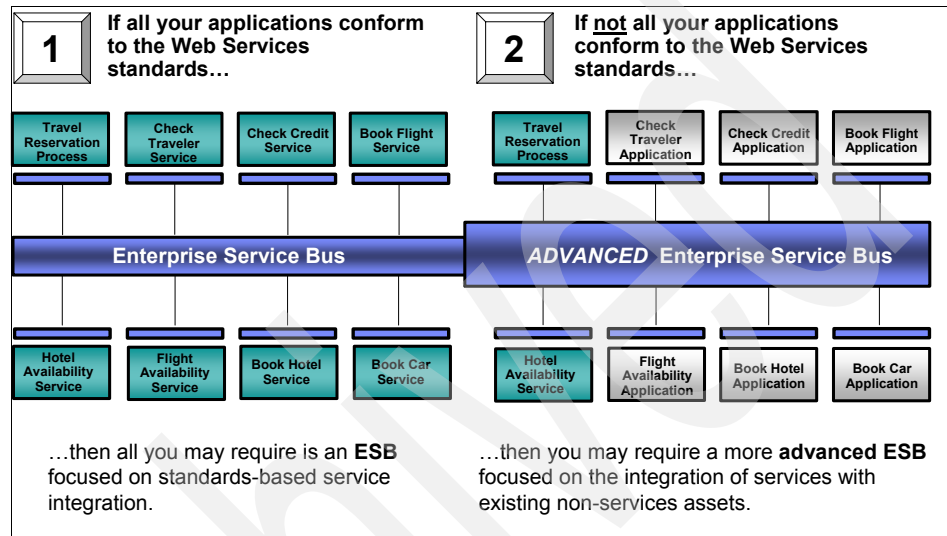


Figure 3-3 IBM ESB products

Tip: More information about IBM ESB can be found here:

<http://www-306.ibm.com/software/info/middleware/applications/index.jsp>

As you can see, the ESB is a powerful addition to an enterprise integration architecture. It enables faster, simpler, and more flexible integration, which allows your integration to respond at the speed of the business. It also shrinks the number of interfaces and improves the reusability of interface components to cut cycle time from design to deployment.

Tip: Additional reading material on ESB patterns, products, and how they fit into a service-oriented architecture can be found at:

<http://www-128.ibm.com/developerworks/architecture/application.html>

3.1.1 The role of an enterprise service bus

The use cases that lead you to the Service Connectivity scenario consistently refer to a middle tier that provides the functionality that allows the decoupling of the service consumer and the service provider. This middle tier can be a simple gateway that provides basic security and static routing features. More likely, it will be an enterprise service bus that provides advanced features to facilitate intelligent message handling and advanced connectivity options.

An ESB introduces features that can improve responsiveness, customer service, transaction time, and partner interactions. An ESB provides capabilities that enhance both direct connection between applications and routing requests among applications.

An ESB supports the concepts of SOA implementation by:

- ▶ Decoupling the consumer's view of a service from the implementation of a service
- ▶ Decoupling technical aspects of service interactions
- ▶ Integrating and managing services in the enterprise

Decoupling the consumer's view of a service from the actual implementation greatly increases the flexibility of the architecture. It allows the substitution of one service provider for another (for example, because another provider offers the same services for lower cost or with higher standards) without the consumer being aware of the change or without the need to alter the architecture to support the substitution.

This decoupling is better achieved by having the consumers and providers interact through an intermediary. Intermediaries publish services to consumers. The consumer binds to the intermediary to access the service, with no direct coupling to the actual provider of the service. The intermediary maps the request to the location of the real service implementation.

In an SOA, services are described as being loosely coupled. However, at implementation time, there is no way to loosely couple a service or any other interaction between systems. The systems must have some common understanding to conduct an interaction. Instead, to achieve the benefits of loose coupling, consideration should be given to how to couple or decouple various aspects of service interactions, such as the platform and language in which services are implemented, the communication protocols used to invoke services, and the data formats used to exchange input and output data between service consumers and providers.

Further decoupling can be achieved by handling some of the technical aspects of transactions outside of applications. This could apply aspects of interactions such as:

- ▶ How service interactions are secured
- ▶ How the integrity of business transactions and data are maintained (for example, through reliable messaging, the use of transaction monitors, or compensation techniques)
- ▶ How the invocation of alternative service providers is handled in the event that the default provider is unavailable

The role of the ESB is to fulfill these needs by providing functions such as:

- ▶ Map service requests from one protocol and address to another.
- ▶ Transform data formats.
- ▶ Support a variety of security and transactional models between service consumers and service providers and recognize that consumers and providers might support or require different models.
- ▶ Aggregate or disaggregate service requests and responses.
- ▶ Support communication protocols between multiple platforms with appropriate qualities of service.
- ▶ Provide messaging capabilities such as message correlation and publish/subscribe to support different messaging models such as events and asynchronous request/response.

3.2 Business pattern and Application pattern selection

The benefits of an ESB can be seen in the following client-to-application and application-to-application interaction types:

- ▶ Direct connectivity

Direct connectivity implies a one-to-one topology between the client or source application and the target application. A direct connection is appropriate for applications with a small number of interfaces that are not likely to be reused by other applications. Communication can be a one-way request or follow a request/reply pattern.

Although communication is direct, the connection can benefit from the decoupling of the source and target and from advanced capabilities such as protocol or data transformation, message augmentation, or security that an ESB can provide.

► Router connectivity

Router connectivity is based on one-to-one connections where an interaction is between a client or source application and, at most, one of multiple possible target applications. Between the source and the target is a Router tier that intercepts the request and makes a determination as to which target to send the request to. The Router tier should provide the same capabilities that you find in a direct connection and must be able to inspect the incoming message for information that helps determine to which target to send the request.

Routing features are common to ESB implementation products.

► Broker connectivity

Broker connectivity is based on one-to-N connections. An interaction from a single client or source application is distributed to multiple target applications concurrently. Between the source and target is a Broker tier that intercepts the request and determines which targets to send the request to.

The Broker tier should provide the same capabilities that you find in the Router tier. In addition, the capability of decomposing a request into multiple requests for distribution to target applications and then recomposing the responses for the reply to the client or source application is required. This requirement is more advanced and may be a differentiator in the ESB product selected.

These connectivity types can be seen in the Application patterns found in Table 3-1. This table shows Business and Integration patterns with their relevant Application patterns that fulfill the generic use cases for the Service Connectivity scenario.

Table 3-1 Summary of patterns for the Service Connectivity scenario

Business and Integration pattern	Application pattern	Generic use cases		
		U6: Enable loose coupling of service consumers and providers using static routing	U7: Enable loose coupling of service consumers and providers using dynamic routing and standards-based protocols	U8: Enable loose coupling of service consumers and providers using advanced dynamic routing and diverse protocols
Self-Service	Directly Integrated Single Channel	X		
	Router		X	
	Decomposition			X
Application Integration	Direct Connection	X		
	Router variation of the Broker		X	
	Broker			X
Extended Enterprise	Exposed Direct Connect	X		
	Exposed Router variation of the Broker		X	
	Exposed Broker			X

Each of these Application patterns has one or more corresponding Runtime patterns that have a defined SOA profile. These Runtime patterns are particularly well-suited for implementation using the Service Connectivity scenario.

3.3 Runtime pattern selection

Runtime patterns are used to define the logical middleware structure supporting the Application patterns. In other words, Runtime patterns describe the logical architecture required to implement an Application pattern. Runtime patterns depict the major middleware nodes, their roles, and the interfaces between these nodes.

When a Runtime pattern is depicted in a diagram, the corresponding Application pattern is overlaid, illustrating the logical placement of the Application pattern tiers within the Runtime pattern.

3.3.1 SOA profiles of the Runtime patterns

Each Runtime pattern discussed in this book has both a generic and an SOA profile. The generic profile describes the basic implementation of each Runtime pattern in the context of the Business pattern. The generic profile specifies an infrastructure that can be used by all applications, including services in an SOA.

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.

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. In this book we specifically focus on the use of an ESB and only look at the SOA profiles of the Runtime patterns.

3.3.2 Runtime nodes

A Runtime pattern consists of 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 patterns you must review the following node definitions.

User node

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

Domain Name Server (DNS) node

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

Public Key Infrastructure (PKI)

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

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

Web server redirector node

The *Web server redirector node* includes the function of an HTTP server (also known as a Web server) and a redirector that can forward, or redirect, requests to an application server. The Web server serves HTTP pages and the redirector forwards servlet and JSP™ requests to the application servers. The advantage of using a redirector is that you can move the application server behind the domain firewall into the secure network, where it is more protected than within the DMZ.

Application server node

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

Directory and security services node

The *directory and security services* node supplies information about the location, capabilities, and attributes (including user ID and password pairs and certificates) of resources and users known to this Web application system. This node can supply information for various security services (authentication and

authorization) and can also perform the actual security processing (for example, verifying certificates). The authentication in most current designs validates the access to the Web application server part of the Web server, but this node also authenticates for access to the database server.

Firewall nodes

A *firewall* is a hardware and software or just 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 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 the 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), typically implemented as an IP Router.
- ▶ Application gateways (the *domain* firewall), typically implemented as a dedicated server node

A pair of firewall nodes provides increasing levels of protection at the expense of increasing computing resource requirements.

Existing applications and data node

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

Business service directory

The role of the *business service directory* is to provide details of services that are available to perform business functions identified within a taxonomy. The business service directory can be implemented as an open-standard UDDI registry. Catalogs, such as a UDDI registry, can achieve one of the primary goals of a business service directory: to publish the availability of services and encourage their reuse across the development activity of an enterprise. The vision of Web services defines an open-standard UDDI registry that enables the

dynamic discovery and invocation of business services. However, although technologies mature toward that vision, more basic solutions are likely to be implemented in the near future.

ESB node

The *ESB* is a key enabler for a SOA, as it provides the capability to route and transport service requests from the service requester to the correct service provider. 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. Furthermore, the ESB needs to be centrally managed and administered and have the ability to be physically distributed.

App server/services node

These nodes represent applications that request a service from the ESB or provide a service to the ESB. These applications can be implemented in any technology as long as they are able to interact using one of the protocols and messaging models that is supported by the ESB.

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

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.

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) that 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 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 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).

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.

The Exposed ESB Gateway 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 provides namespace mapping.

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.

3.4 Self-Service business pattern

The Self-Service business pattern captures the essence of direct interactions between interested parties and a business. Interested parties include customers, business partners, stakeholders, employees, and all other individuals with whom the business intends to interact. For simplicity, these interested parties are referred to as users. In this definition business represents various types of organizations including large enterprises, small and medium businesses, government agencies, and so on.

The Self-Service business pattern covers a wide range of uses. Applications of this pattern can range from the very simple function of allowing users to view data built explicitly for one purpose, to taking requests from users, decomposing them into multiple requests to be sent to multiple, disparate data sources, personalizing the information, and recomposing it into a response for the user.

For this reason, there are currently seven defined Application patterns that fit this range of functions. Three of these have been identified in Table 3-1 on page 38 as candidates for the Service Connectivity scenario. They are:

- ▶ Directly Integrated Single Channel application pattern
- ▶ Router application pattern
- ▶ Decomposition application pattern

Product mappings: Once a Runtime pattern has been selected, the next step is to select the appropriate products for implementation. Product mappings show possible product selection combinations for a specific Runtime pattern. This book will use an example to illustrate the design and implementation of a Service Connectivity scenario.

The example is based on the Self-Service business pattern and we will extend our discussion of the Runtime patterns to include a sample product mapping.

3.4.1 Directly Integrated Single Channel application pattern

The Directly Integrated Single Channel application pattern provides point-to-point connectivity between the user and the existing back-end applications. It assumes that one delivery channel and the user interface is handled by the presentation tier. The business logic can reside in the Web application tier and in the back-end application.

The Web application tier has access to local data that exists primarily as a result of this application, for example, customer profile information or cached data. It is also responsible for accessing one or more back-end applications. The back-end applications contain business logic and are responsible for accessing the

existing back-end data. The communication between the presentation tier and Web application tier is synchronous. The communication between the Web application tier and the back-end can be either synchronous or asynchronous, depending on the characteristics and capabilities of the back-end application.

Runtime pattern: SOA profile

The Runtime pattern shown in Figure 3-4 represents a solution for the Directly Integrated Single Channel application pattern that takes advantage of the functionality provided by an ESB. Note that the Directly Integrated Single Channel application pattern is shown in the box at the bottom.

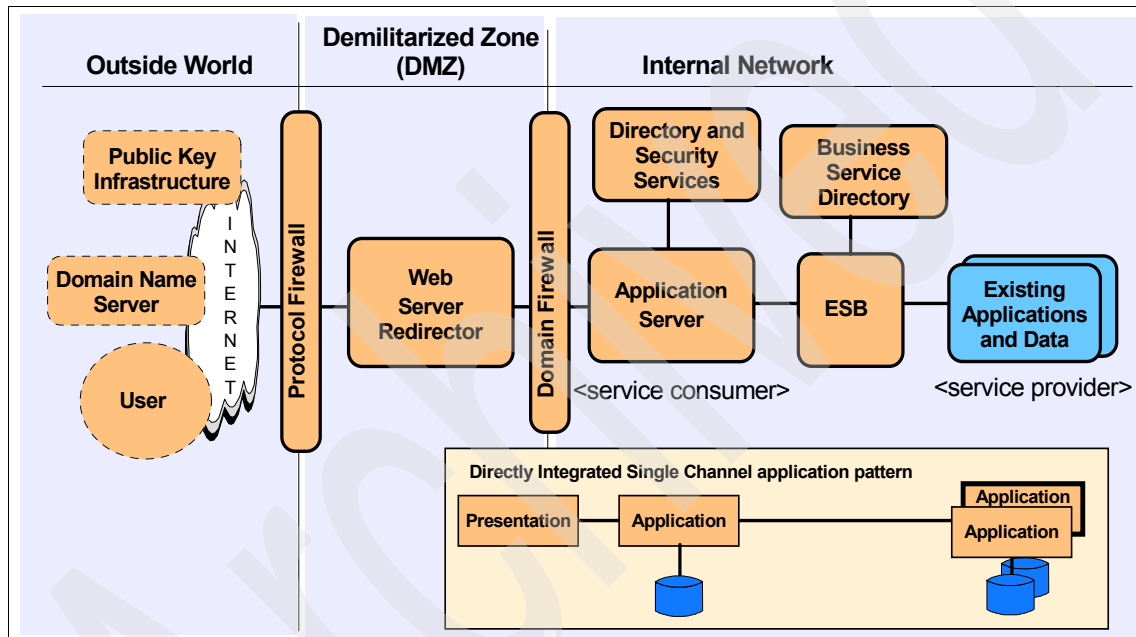


Figure 3-4 Directly Integrated Single Channel application pattern::Runtime pattern: SOA profile

This Runtime pattern uses a Web server redirector node in the DMZ to serve static HTML pages to the client. Requests for dynamic data are forwarded to the application server in the internal network. Together, these two nodes provide the presentation tier, capable of handling multiple, diverse, presentation styles. Using a redirector allows you to place the bulk of the business logic behind the protection of both the protocol and the domain firewalls.

In addition to presentation logic, primarily in the form of JavaServer™ Pages™ (JSPs), the application server contains some business logic. This is primarily in the form of the controlling servlets required to access the back-end applications.

The application server builds a request based on user input and invokes services that can fulfill the request.

The application server node becomes the service consumer with the back-end applications acting as service providers. The connection between service consumer and service provider is implemented with a simple enterprise service bus.

Direct connections can benefit from the majority of the functions provided by the ESB node (see “ESB node” on page 42). The obvious exception to this for direct connections is routing capability.

The ESB approach:

- ▶ Minimizes the number of adapters required to link service consumers to service providers
- ▶ Improves reuse
- ▶ Addresses any technical and information model discrepancies amongst services
- ▶ Provides a single configuration point for distributed deployment
- ▶ Decouples service requesters from providers
- ▶ Provides a common access point for service requesters
- ▶ Provides centralized security for services

Note: Implementing the SOA profile with an ESB adds extra capabilities to the runtime pattern, for example, routing and decomposition capability. Because of this, the SOA profile for the Directly Integrated Single Channel runtime pattern can be applicable to multiple Self-Service application patterns. This highlights the fact that using SOA facilitates the future expansion of solution functionality without requiring major changes to the middleware structure.

Product mapping

Figure 3-5 shows a Product mapping for the SOA profile of the Runtime pattern. This product mapping shows a Web services provider, but any type of service provider supported by the ESB could be substituted.

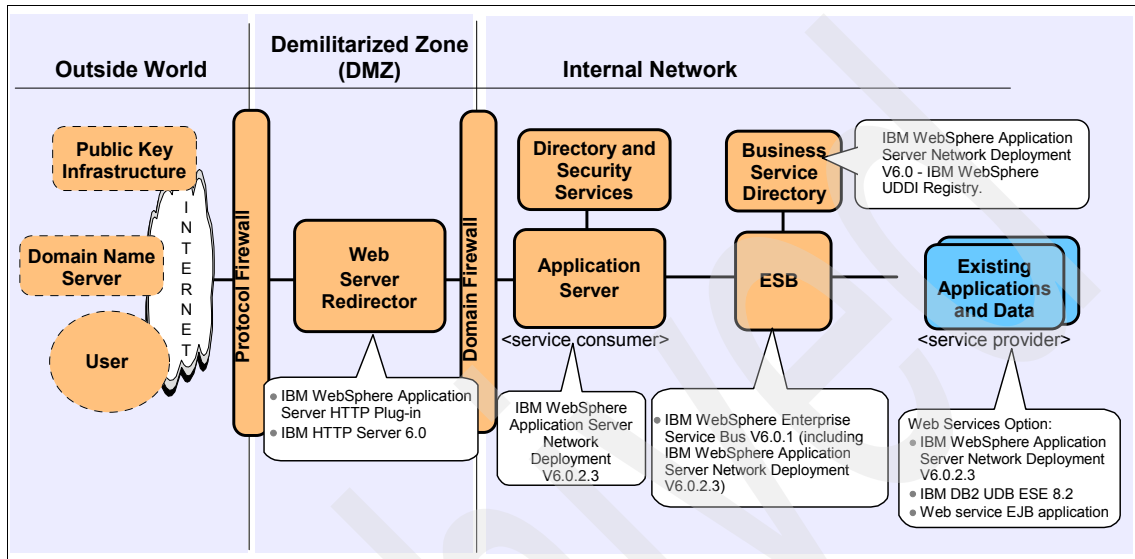


Figure 3-5 Directly Integrated Single Channel application pattern:: Product mapping

3.4.2 Router application pattern

The Router application pattern provides intelligent routing from multiple channels to multiple back-end applications using a hub-and-spoke architecture. The interaction between the user and the back-end application is a one-to-one relation, meaning the user interacts with applications one at a time. The Router maintains the connections to the back-end applications and pools connections when appropriate, but there is no true integration of the applications themselves. The Router can use a read-only database, most probably to look up routing information. The primary business logic still resides in the back-end application tier.

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

Runtime pattern: SOA profile for Router

The Runtime pattern shown in Figure 3-6 represents a solution for the Router application pattern that takes advantage of the functionality provided by an ESB. You can see the Router application pattern overlaid on the Runtime pattern at the bottom of the figure.

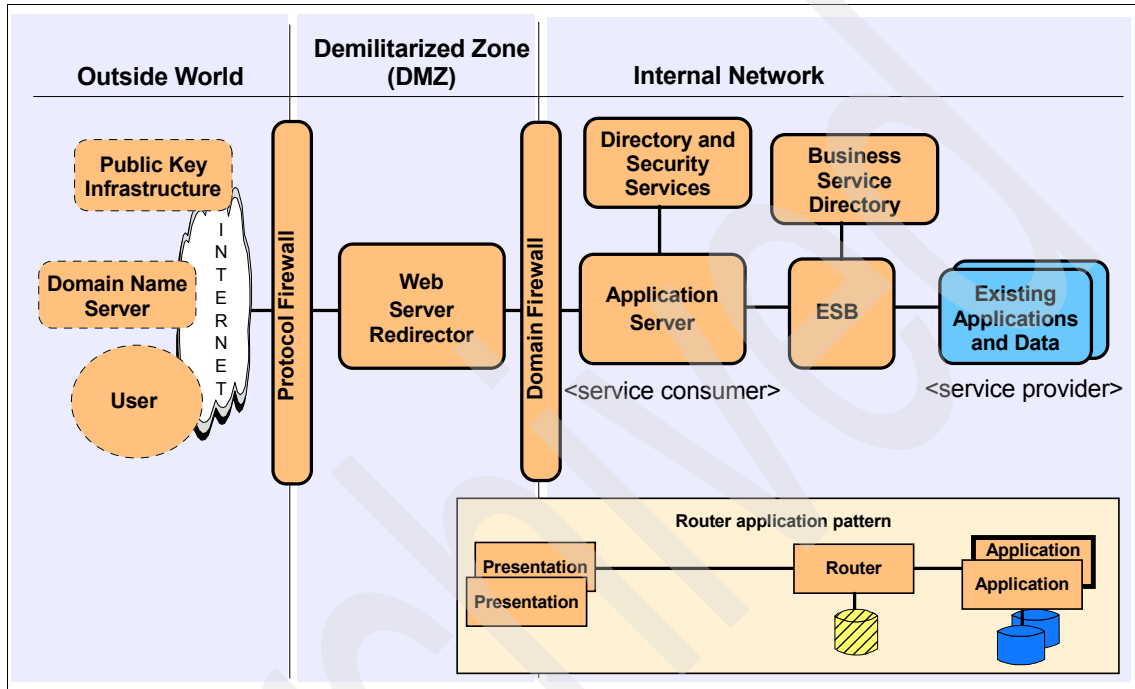


Figure 3-6 Router application pattern::Runtime pattern: SOA profile

This Runtime pattern is very similar to the SOA profile for the Directly Integrated Single Channel. The difference lies in the interaction pattern between the service consumer and service provider and the ESB capabilities required to provide the connection.

Because a single request from the service consumer can be destined for any one of multiple service providers, routing capability must be present in the ESB. When a request from a service consumer is sent to the ESB node, the ESB must examine the request, determine the appropriate destination, and forward it to the chosen back-end application where the primary business logic resides. This may involve activities such as message transformation, protocol conversion, security management, and session concentration.

Product mapping

Figure 3-7 shows a Product mapping for the SOA profile of the Runtime pattern.

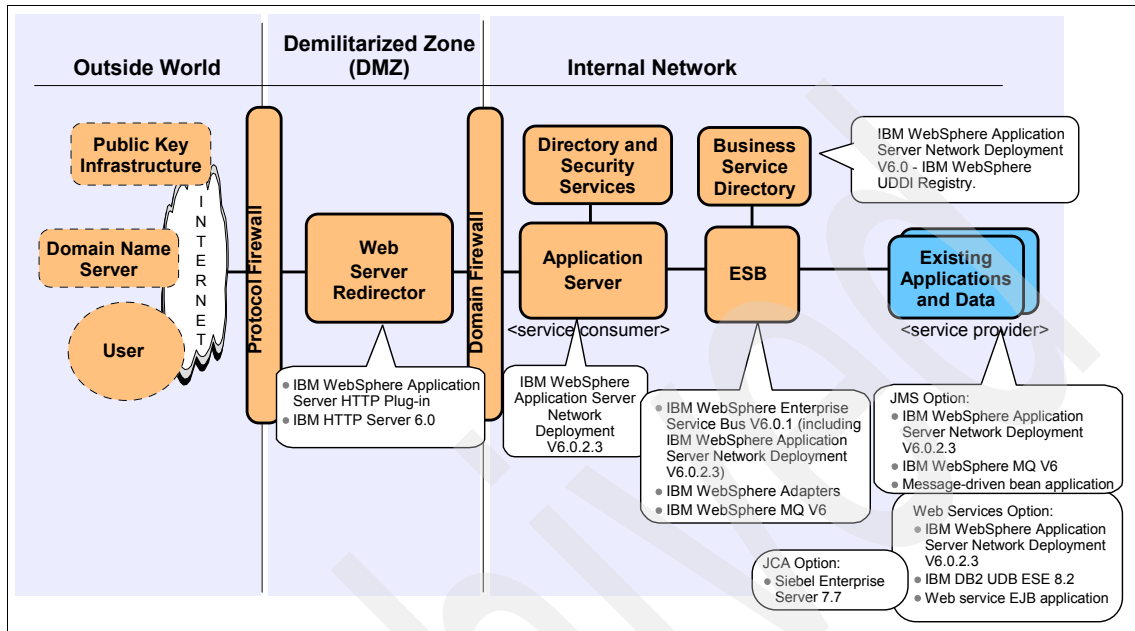


Figure 3-7 Router application pattern::Product mapping

WebSphere ESB provides the ESB functionality in the mapping. It provides basic content-based routing and other services for standards-based consumers and providers. In addition to standard Web services support, this mapping shows additional connectivity options.

WebSphere Adapters provide EIS connectivity from the ESB. WebSphere Adapters are compliant with J2EE Connector Architecture (JCA 1.5).

JMS connectivity can be managed through the WebSphere ESB service integration bus or in conjunction with a WebSphere MQ network.

3.4.3 Decomposition application pattern

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

the user. This moves some of the business logic into the decomposition tier, but the primary business logic still resides in the back-end application tier.

From a service-oriented architecture (SOA) perspective, the decomposition tier of this application pattern facilitates the invocation of business services hosted by a number of back-end applications. In doing so, the Decomposition application pattern fully leverages the integration capabilities described by the Application Integration::Broker pattern.

If an interaction initiated by a user requires the execution of an end-to-end business process or workflow where process and workflow rules are better externalized, the Decomposition application pattern would leverage the integration capabilities of more advanced process integration alternatives such as the Application Integration::Serial Process or Serial Workflow and the Application Integration::Parallel Process or Parallel Workflow. Since the end result is the decomposition/recomposition capability discussed above, these variations are not documented as Decomposition application pattern variations, but rather may be captured as different runtime patterns where applicable.

Runtime pattern: SOA profile for Decomposition

The Runtime pattern shown in Figure 3-8 represents a solution for the Decomposition application pattern that takes advantage of the functionality provided by an ESB.

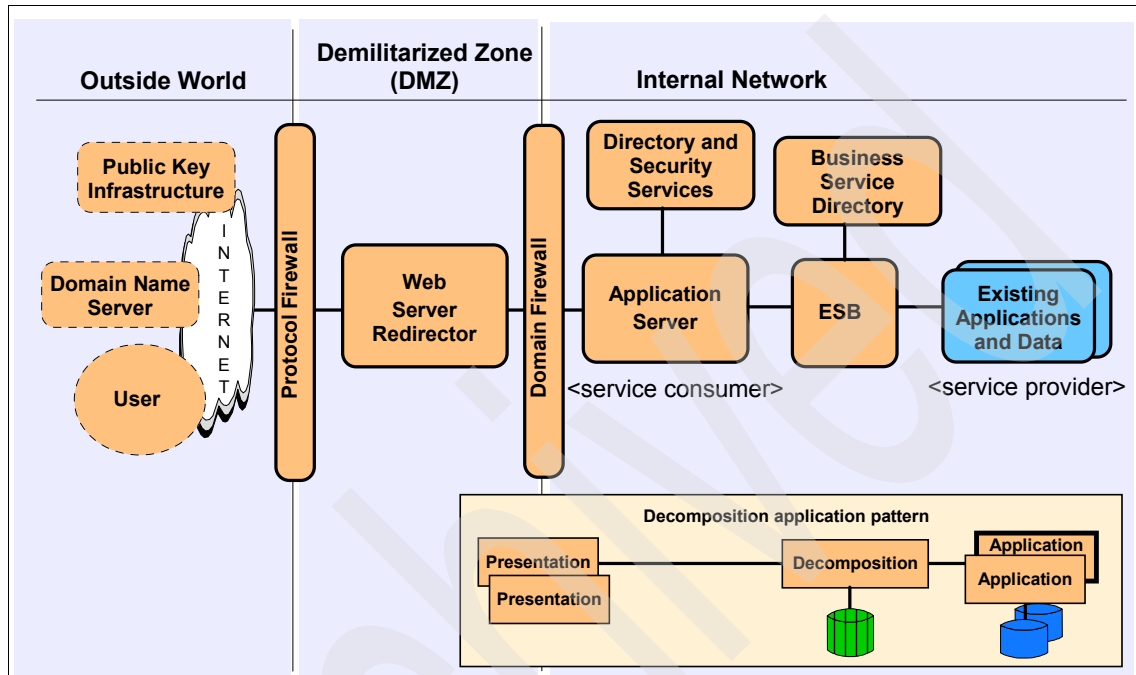


Figure 3-8 Decomposition application pattern::Runtime pattern: SOA profile

Similar to the SOA profile for the Router, this Runtime pattern provides the next level of complexity. Where the interaction in the Router is between one service consumer and any one of a number of service providers, the Decomposition pattern takes a single request from a consumer and decomposes it into multiple requests to be sent to multiple service providers.

The ESB node can take a single complex message from a service consumer, decompose it into multiple messages, and route those messages to the appropriate service providers. It is also capable of managing these messages such that it can wait for responses and recompose them into a single response to be sent back to the service consumer. This effectively takes multiple, diverse back-end applications and unifies them into one interface for the user.

Product mapping

The requirement for decomposition leads us to WebSphere Message Broker as the ESB of choice. See Figure 3-9.

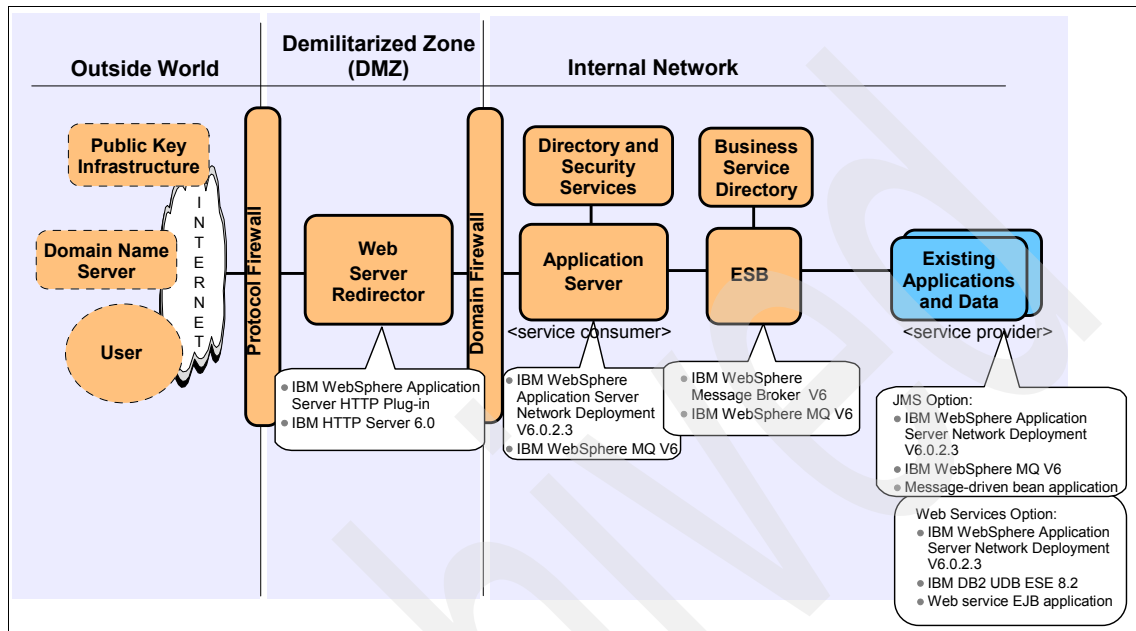


Figure 3-9 Decomposition application pattern:: Product mapping

WebSphere Message Broker provides a wide range of connectivity options to non standards-based consumers and providers. This product mapping shows Web services and JMS connectivity.

3.5 Application Integration pattern

The Application Integration (also known as Enterprise Application Integration or EAI) pattern serves to integrate multiple Business patterns or to integrate applications and data within an individual Business pattern. The requirements that gave rise to this pattern call for the seamless execution of multiple applications and access to their respective data in order to automate a complex, new business function. Reliable integration of applications — be they traditional

stovepipe applications, packaged software applications, or custom applications — requires the use of proven replicable patterns. At its highest level, application integration can be divided into two essentially different approaches:

- ▶ Process Integration - The integration of the functional flow of processing between the applications
- ▶ Data Integration - The integration of the information used by applications

Neither approach is necessarily better than the other. Rather, specific integration requirements dictate which approach best solves a given business problem. For example, the integration of an e-commerce application with an Enterprise Resource Planning (ERP) system for a newly created sales order would most definitely be a Process integration activity. However, in the same solution, the master data synchronization of the product catalog between the ERP system and the e-commerce system would be a data integration activity.

The following Process Integration Application patterns were identified in Table 3-1 on page 38 as potential solutions for the Service Connectivity scenario:

- ▶ Direct Connection application pattern
 - Message Connection variation
 - Call Connection variation
- ▶ Broker application pattern
 - Router variation

Note: This section gives you a brief look at these Application patterns and the SOA profile of at least one Runtime pattern for each. Much work has been done in this area. If you are planning on using the Application Integration pattern, please consult the Application Integration section of the IBM Patterns for e-business Web site at:

<http://www-128.ibm.com/developerworks/patterns/application/index.html>

3.5.1 Direct Connection application pattern

The Direct Connection application pattern represents the simplest interaction type and is based on a 1-to-1 topology. It allows a pair of applications within the organization to directly communicate with each other. 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 will have modeled connection rules such as business rules associated with them, as shown above. Connection rules

are generally used to 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

The Direct Connection application pattern has two variations:

- ▶ Message Connection variation

This variation applies to solutions where the business process does not require a response from the target application within the scope of the interaction.

- ▶ Call Connection variation

The Call Connection variation applies to solutions where the business process depends on the target application to process a request and return a response within the scope of the interaction.

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

Both variations may 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 while the Message Connection variation favors asynchronous protocols.

Runtime pattern: SOA profile

The variations of the Direct Connection application pattern share the same Runtime pattern. Figure 3-10 shows the SOA profile of a Runtime pattern for the Direct Connection application pattern.

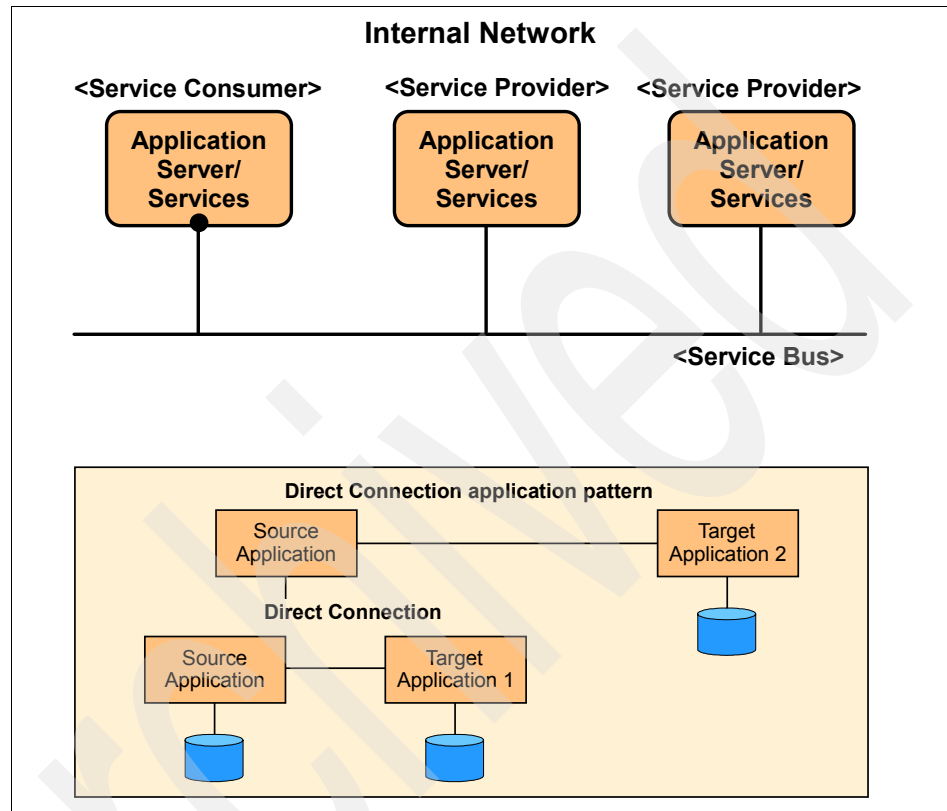


Figure 3-10 Direct Connection runtime pattern: SOA profile

The source and target applications both rely on services provided by their respective hosting servers. These are modeled using the App Server/Services node.

Figure 3-10 shows a service consumer connected to two other service providers via a simple service bus. The application pattern overlays in the figure above show that multiple Direct Connection application patterns can be deployed using the service bus. The service consumer (or Source Application) can use the service bus to initiate direct connections to two service providers — one to target application 1 and the other to target application 2.

The service bus approach:

- ▶ Minimizes the number of adapters required for each point-to-point connection to link service consumers to service providers
- ▶ Improves reuse in multiple point-to-point scenarios
- ▶ Addresses any technical and information model discrepancies among services

3.5.2 Broker application pattern

The Broker application pattern is based on a 1-to-N topology that separates distribution rules from the applications. It allows a single interaction from the source application to be distributed to multiple target applications concurrently. This application pattern reduces the proliferation of point-to-point connections.

The Broker application pattern applies to solutions where the source application starts an interaction that is distributed to multiple target applications that are within the organization. It separates the application logic from the distribution logic based on Broker rules. The decomposition/recomposition of the interaction is managed by the Broker rules tier.

The Broker pattern reuses the Direct Connection pattern to provide connectivity between the tiers. The Broker rules may support Message variation or Call variation (or both variations) of the Direct Connection pattern.

Runtime pattern: SOA profile

The Broker tier in the Application pattern is implemented in this Runtime pattern with an ESB. The ESB is responsible for the routing and distribution of incoming messages to the target applications. It has the ability to decompose and recompose messages.

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

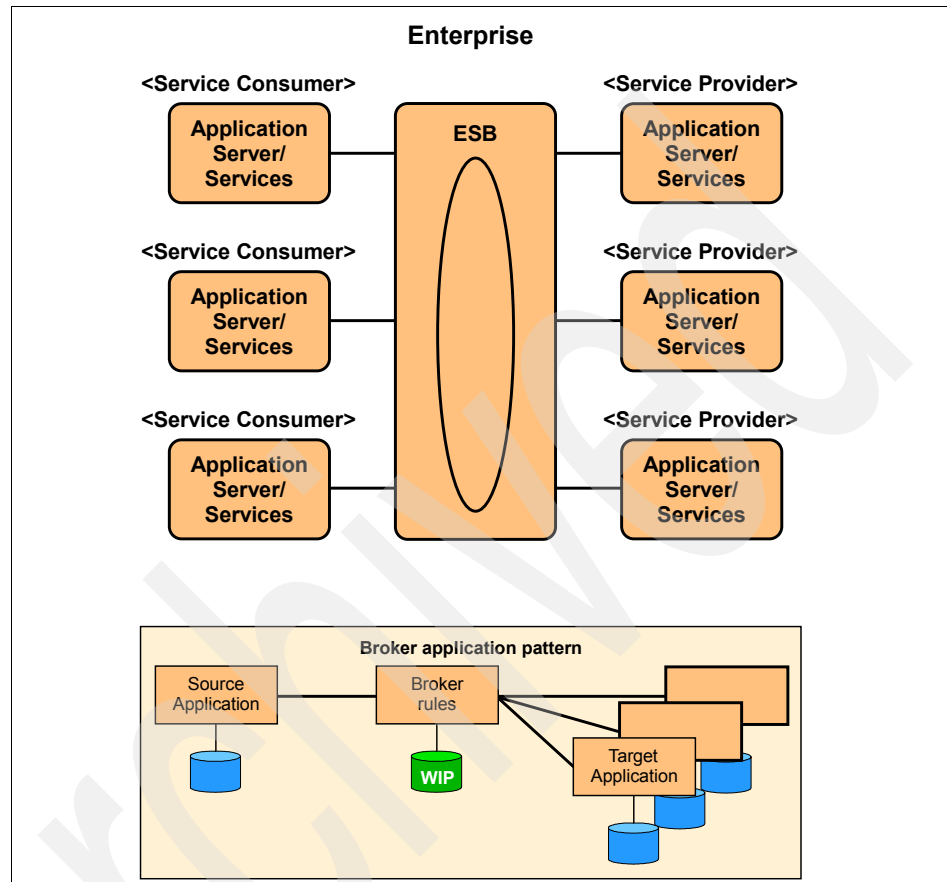


Figure 3-11 Broker runtime pattern: SOA profile

The use of an ESB offers the following benefits:

- ▶ The ESB is a bus with a single configuration and distributed deployment. Managing communications through the bus provides many advantages, including decoupling of service requesters and providers, and centralized control of a service namespace.
- ▶ Protocol conversion occurs inside the ESB (for example, SOAP/HTTP to SOAP/JMS). Requesters using one protocol can invoke services that are exposed using a different protocol.
- ▶ The ESB can provide logging and transformation of service requests and service responses.

- ▶ The ESB can provide centralized security for Web services invocations. It can, for example, authenticate all service requesters centrally.
- ▶ The ESB provides a common access point for service requesters that need access to services providers. The ESB intercepts and routes requests to the relevant service provider. A change in the location of the service provider only affects the ESB routing. The service provider location remains transparent to the service requester.

3.5.3 Router variation of the Broker application pattern

The Router variation of the Broker application pattern applies to solutions where the source application initiates an interaction that is forwarded to at most one of multiple target applications.

Where the Broker application pattern enables 1:N connectivity, the Router application pattern enables 1:1 connectivity, where the Router Rules tier selects the target.

Runtime pattern: SOA profile

The Router tier in the Application pattern is implemented in this Runtime pattern with an ESB. The ESB is responsible for the routing and distribution of incoming messages to the target applications.

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

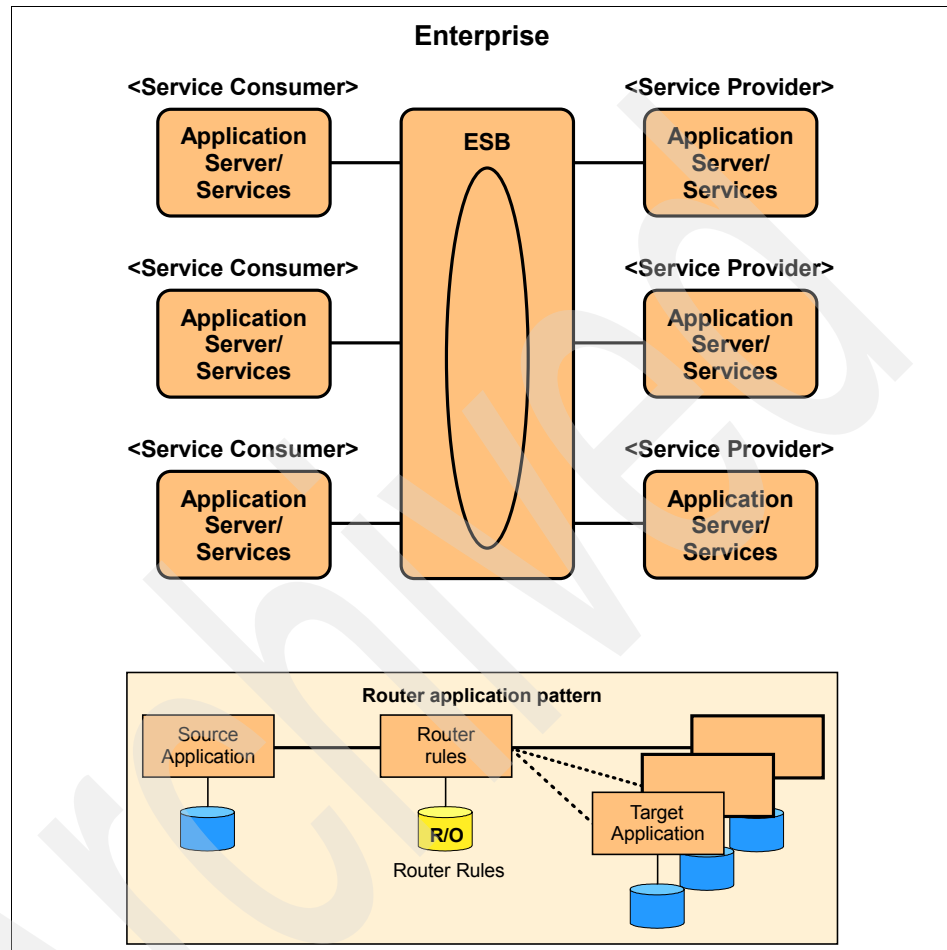


Figure 3-12 Router variation runtime pattern: SOA profile

The use of an ESB offers the same benefits seen in the Broker. Specifically, we use the ESB in the Router variation to provide:

- ▶ Service routing of requests from service requesters to the relevant service provider based on a routing table
- ▶ Protocol transformation to allow the decoupling of the protocol that is used between the service requesters and service providers

3.6 Extended Enterprise business pattern

The Extended Enterprise business 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 inter-enterprise applications. In other words, it does not cover applications that are directly invoked using a user interface by business partners across organizational boundaries.

For this reason, there are currently six defined Application patterns that fit this range of functions. Three of these are shown Table 3-1 on page 38 as candidates for the Service Connectivity scenario. They are:

- ▶ Exposed Direct Connection application pattern
- ▶ Exposed Broker application pattern
- ▶ Exposed Router variation of Exposed Broker application pattern

Note: This section gives you a brief look at these Application patterns and the SOA profile of at least one Runtime pattern for each. Much work has been done in this area. If you are planning on using the Application Integration pattern, please consult the Extended Enterprise section of the IBM Patterns for e-business Web site at:

<http://www-128.ibm.com/developerworks/patterns/b2bi/index.html>

3.6.1 Exposed Direct Connection application pattern

The Exposed Direct Connection application pattern represents the simplest interaction type based on a 1-to-1 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 will have modeled connection rules such as business rules associated with them. Connection rules are generally used to 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

The Exposed Direct Connection application pattern has two variations:

- ▶ Message Connection variation

The Message Connection variation applies to solutions where the business process does not require a response from the exposed target application within the scope of the interaction.

- ▶ Call Connection variation

The Call Connection variation applies to solutions where the business process depends on the exposed target application to process a request and return an response within the scope of the interaction.

All applications of the Direct Connection application pattern will be one variation or the other. The variation required depends on whether the initiating source application needs an immediate response from the target application in order to proceed with execution. Both variations may 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, while the Message Connection variation favors asynchronous protocols.

Runtime pattern: SOA profile for Exposed Direct Connection

The Runtime pattern for the Exposed Direct Connection application pattern, shown in Figure 3-13, allows two different organizations to talk to each other with a mutually agreed-upon message format and protocol. Each partner can use its own internal messaging format, then use a connector adapter to convert from the internal format to the external format.

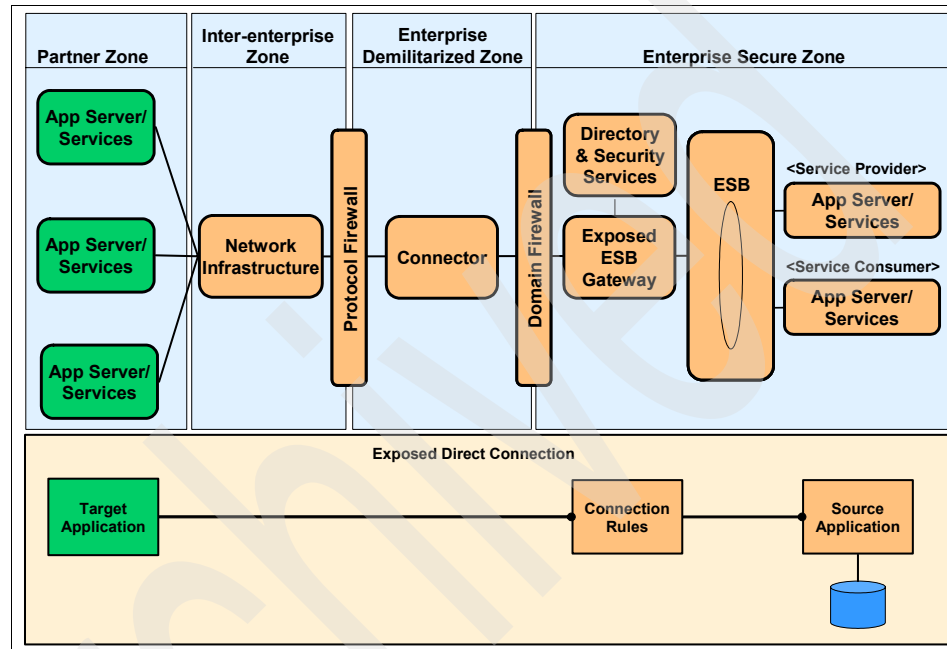


Figure 3-13 Exposed Direct Connection runtime pattern: SOA profile

The ESB node provides the connectivity between two partner enterprises. It 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.

If needed, the ESB node uses a persistent data technology to hold the read-only execution rules that support these processes. These rules control the sequencing of activities, and therefore support flow control within the context of an end-to-end process flow.

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.

3.6.2 Exposed Broker application pattern

The Exposed Broker application pattern is based on a 1-to-N topology that separates distribution rules from the applications. It allows a single interaction from an enterprise's source application to be distributed to multiple target partner applications concurrently.

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/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 may support Message variation or Call variation (or both variations) of the Exposed Direct Connection pattern.

Runtime pattern: SOA profile for Exposed Broker

The Runtime pattern shown in Figure 3-14 represents a solution for the Exposed Broker application pattern that takes advantage of the functionality provided by an ESB.

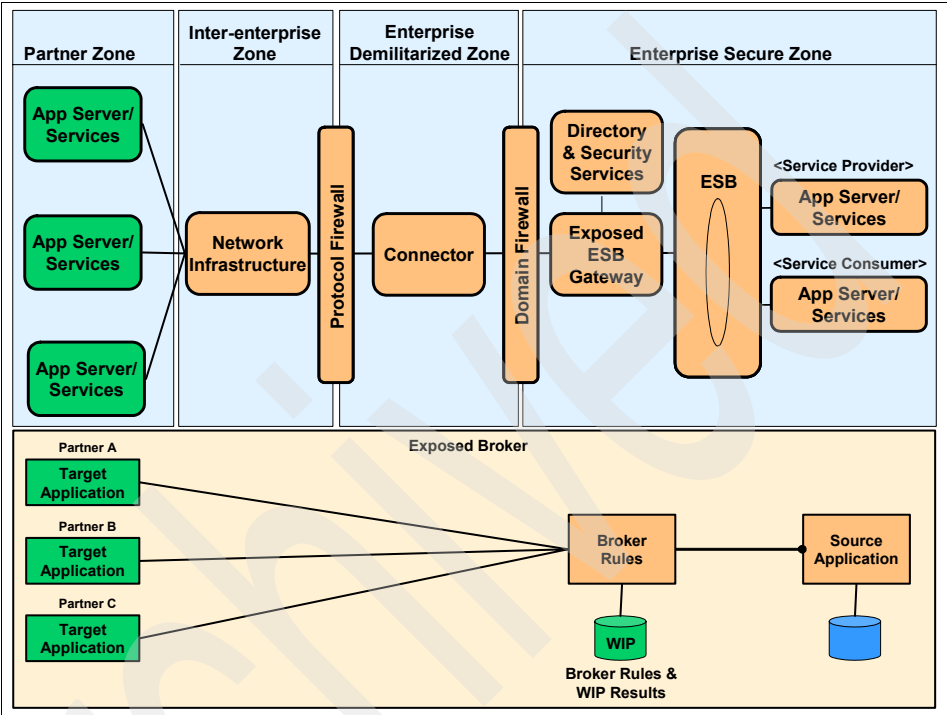


Figure 3-14 Exposed Broker runtime pattern: SOA profile

This Runtime pattern is similar to the SOA profile for the Exposed Direct Connection. The difference lies in the ESB functions required for the connections.

In this pattern, the ESB node fulfills the requirements of a Broker, allowing 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 ESB separates the application logic from the distribution logic based on Broker rules and manages the decomposition and recomposition of the interaction using these rules.

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

The ESB node also provides the Rules Directory node functionality. It 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.

3.6.3 Exposed Router variation of Exposed Broker application pattern

The Exposed Router variation of the Exposed Broker application pattern applies to solutions where the enterprise'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.

Runtime pattern: SOA profile for Exposed Router

The Runtime pattern shown in Figure 3-15 represents a solution for the Exposed Router application pattern that takes advantage of the functionality provided by an ESB.

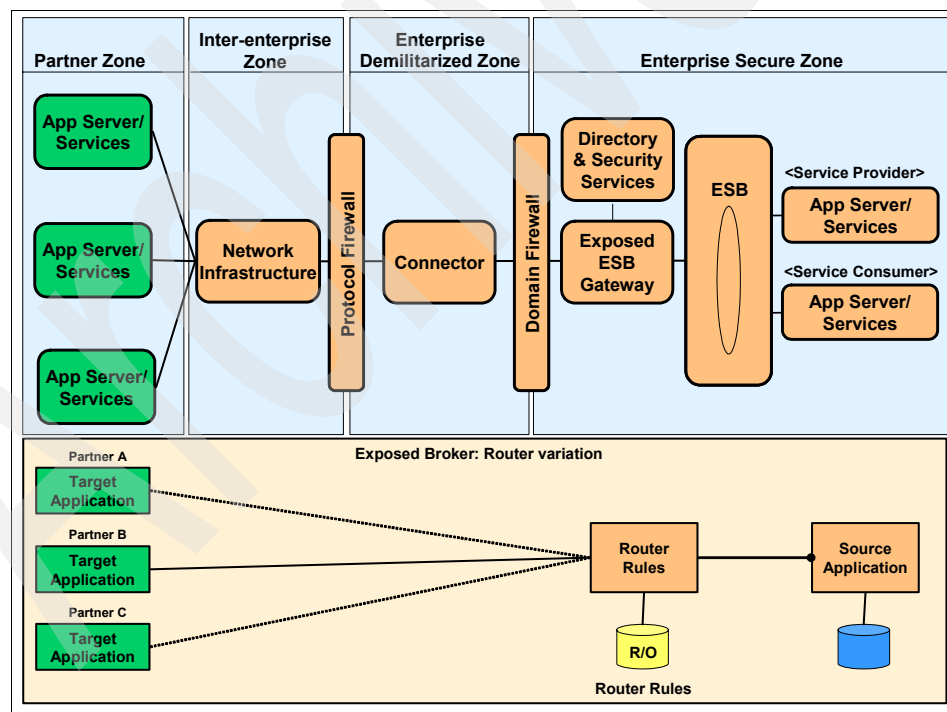


Figure 3-15 Exposed Router runtime pattern: SOA profile

In this pattern, the ESB node fulfills the requirements of a Router, allowing 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. In addition, the ESB node provides the functionality of a Rules Directory node.

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

3.7 For more information

For more information see the following:

- ▶ *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240
- ▶ IBM Patterns for e-business
<http://www-128.ibm.com/developerworks/patterns/>
- ▶ *Patterns: Extended Enterprise SOA and Web Services*, SG24-7135

Planning for connectivity

This chapter is an introduction to connectivity solution planning. It discusses planning for integration in general and product selection and describes how a fictional company, ITSOMart, applied these principles to plan for a customer registration solution.

This chapter includes the following topics:

- ▶ The ITSOMart scenario
- ▶ Considering SOA as a solution for ITSOMart
- ▶ Elements of an SOA solution
- ▶ Selecting the SOA scenario and pattern
- ▶ Enterprise service bus product selection
- ▶ ITSOMart product selection
- ▶ Installation considerations
- ▶ Security considerations
- ▶ Scalability and performance considerations
- ▶ System management and monitoring
- ▶ Where to find the implementation details

4.1 The ITSOMart scenario

ITSOMart is a fictional business scenario designed to illustrate some of the key concepts of the Service Connectivity scenario. The primary business objective of ITSOMart is to provide customers with the capability of placing online orders and then having those orders delivered to their home or place of business. As the project will almost certainly change in scope over time and as each phase is brought into production, the application must be flexible enough to quickly respond to new business opportunities as they arise.

4.1.1 ITSOMart overview

ITSOMart is a well-established grocery chain that has been operating for the past 40 years. The target customers are in the high income group. ITSOMart focuses on higher margin, luxury, and speciality products. It has 1,000 stores nation-wide. The business is currently geared toward business and residential customers. In the future, ITSOMart would like to have additional lines of business, starting with institutional service.

Product types, quantities, marketing strategy, and delivery services differ depending on the customer type. Although there is one common warehouse, the company has created two divisions to handle these customer types. Each division has its own account database with information specific to the customer.

4.1.2 Business objectives

Market research has shown that there is a growing demand in the high-income group for full-service, online home shopping. ITSOMart wants to capitalize on this demand by taking their store services and delivery online. ITSOMart wants to put the customer management and order systems online and make them accessible over the Web. In the process, the company wants to use, rather than replace, their significant investment in the existing customer relationship management (CRM) application. Also, ITSOMart would prefer to use a third-party Credit Rating Service rather than implementing one from scratch. The solution delivery is expected over several phases, beginning with online customer self-registration.

4.1.3 Customer registration business requirements

The basic features of customer self-registration include:

- ▶ Automated customer credit checking
- ▶ Additional processing steps if a customer has insufficient credit
- ▶ CRM application update
- ▶ Shipping address update
- ▶ Auditing

Credit checking

The credit checking function is a third-party credit checking application that determines a customer's credit rating. The credit rating system accepts the customer's first and last name as well as a billing address. From this information the credit rater returns a numerical credit rating score for the customer. ITSOMart plans to segment customers based on credit in order to reduce risk as well as offer premium services to high-value customers in the future. One other requirement of this credit checking function is that the underlying credit check must easily be transferable to a different third-party for credit checking. ITSOMart wishes to maintain the ability to leverage lower-priced vendors for similar services should the current vendor raise the price charged for the credit checking.

Insufficient credit processing

While the third-party credit checking returns a numerical score, the company wishes to segment customers into one of three separate customers based on their credit rating: gold, silver, or bronze. The company plans to offer premium services and target specific marketing campaigns at gold customers, while silver customers will receive standard service. Customers with a gold credit rating are registered immediately. Customers with a silver credit rating are considered potential credit risks and the business has decided that such customers must submit to additional credit validation per current business rules before the company is willing to accept their orders. Customers with a bronze status are denied the opportunity to register.

CRM application update

In order to more effectively gather information about customers, every customer that self-registers will automatically be added to the CRM system. ITSOMart has a significant investment in the current CRM implementation and does not wish to adopt a new system.

Shipping address update

ITSOMart will allow up to two shipping addresses to be entered. This allows a customer to enter a home and a business shipping address. As ITSOMart expands it is possible that multiple shipping providers will be used.

Audit

ITSOMart wants to keep track of all incoming customer registration attempts, whether successful or not. The company invested in an audit system some time ago. However, the old system is slow and has many problems. To resolve these problems, the IT department has decided to implement a new auditing system. That system is expected to launch in approximately 18 months. ITSOMart would like to make the customer self-registration functionality available sooner and would like to continue to use the system while the replacement system is being developed. However, once the new auditing system has been launched, the company wishes to avoid additional development costs in order to integrate with the new system.

4.1.4 Business context diagram for ITSOMart

Figure 4-1 illustrates the proposed high-level business context for the ITSOMart example scenario.

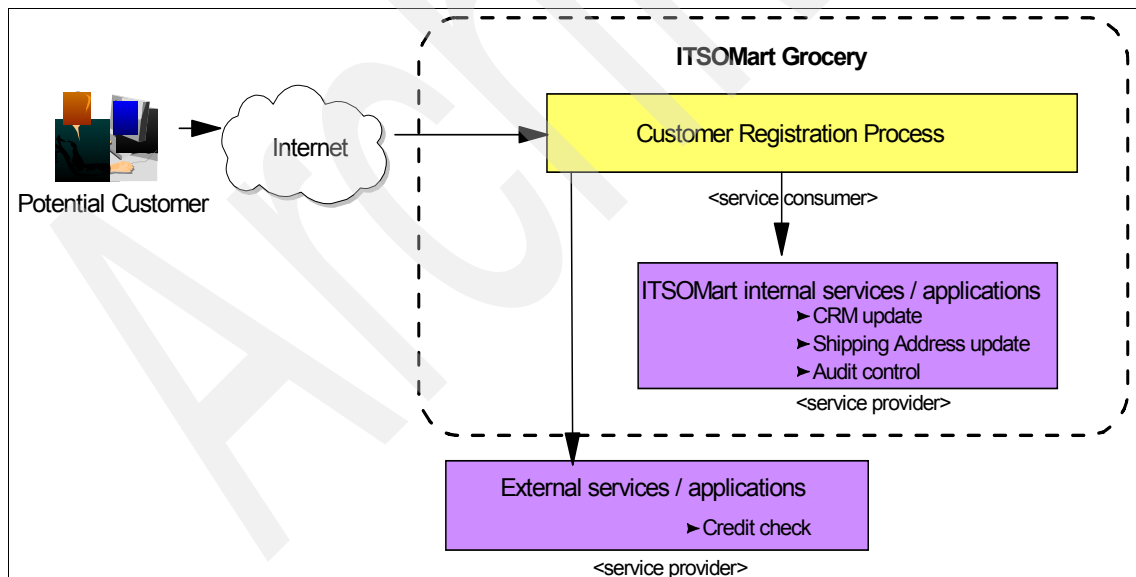


Figure 4-1 High-level business context for the ITSOMart example scenario

4.1.5 Functional requirements for ITSOMart

This section highlights the functional requirements for the ITSOMart working example. We have outlined the key functional requirements:

- ▶ FR1: Expose a new business function to customers.
Expose a new business function, the ITSOMart customer registration, to external customers. The user interface is to be a Web application that will invoke the process as a Web service.
- ▶ FR2: Create a Web services client application to consume services.
Create an application that performs the tasks required to register a customer.
- ▶ FR3: Expose internal applications as Web services.
Expose internal applications as services that can be called during the customer registration process.
- ▶ FR4: Integrate applications so that they can function together to perform a business task.
Allow the service consumer (customer registration process) to call the required services with little to no knowledge of their location, message format, or transport requirements.
- ▶ FR5: Incorporate simple routing logic to allow dynamic decisions based on customer input and responses from invoked services.
Use routing logic to determine the appropriate flow through the process, in particular, to determine how a customer request is processed based on the results of a credit check.
- ▶ FR6: Incorporate flexibility to easily change or add services.
Allow service providers to be changed without affecting the service consumers.

4.2 Considering SOA as a solution for ITSOMart

Service-oriented architecture (SOA) is an approach to transforming the way business and IT operate. An organization's SOA transformation is a gradual process of continuous improvement, made possible by the business and IT agility gained through service orientation. There are many different stages to the adoption of SOA. These include:

- ▶ *Services-based implementation* in which new applications are built around services (perhaps Web Services) and inherited applications are wrapped by services so that they may participate in a services-based environment.

- ▶ *Service Connectivity* where applications are composed of many loosely coupled services. As business needs change, IT can quickly create new applications through a combination of building new services and combining them with existing services.
- ▶ *Enterprise Level IT Transformation* in which SOA is the unifying theme in all IT applications. Creation of new applications primarily involves assembly of existing services rather than development of new functionality. Applications are rarely rewritten due to technology constraints.
- ▶ *On Demand Business Transformation* is a fundamental shift in the way business and IT collaborate. SOA pervades the entire line of business (LOB) application development process. Businesses are able to rapidly deploy new business processes and transform existing processes to accommodate market needs.

ITSOMart has a number of challenges in satisfying the business needs of customer self-registration. The application must be able to deal with the introduction of new applications as the online shopping system evolves, as well as the ever-changing needs of both IT and business.

In order to evaluate service connectivity as a solution for the ITSOMart implementation, the enterprise architect must consider the following:

- ▶ Is delivery of the solution going to be incremental or all at once?
- ▶ Are there ongoing costs of integration that could be reduced by adopting a more flexible integration architecture?
- ▶ Are business/IT needs expected to change?
- ▶ How soon does the effort need to demonstrably have a return on investment?
- ▶ Will the solution meet corporate security guidelines?
- ▶ How difficult will solution management be over so many disparate systems?
- ▶ Is the IT organization ready for service connectivity?

4.2.1 Incremental solution delivery

ITSOMart does not expect to produce a complete online shopping solution all at once. Initially, customers should be able to self-register and communicate information such as delivery location. As new solution components are deployed, the solution must integrate with these components with minimal cost of change. A solution based on SOA would allow new components to easily be added to the current environment without rewriting existing functionality to support the new system.

4.2.2 Integration cost reduction

Point-to-point application integration solutions are often expensive to maintain because all of the integration logic lies within the application. A solution based on SOA has distinct advantages due to reduced maintenance costs. All integration logic resides in an enterprise service bus. By using an assembly technique to build integration solutions, such logic may easily be changed without high development costs. While ITSOMart does not have an existing integration framework, the lower cost of maintaining an SOA solution makes it a more attractive solution approach.

4.2.3 Changing business/IT needs

ITSOMart is a dynamic application whose needs are expected to change over time. IT wishes to replace older systems with updated ones, and the business may decide to switch to different service vendors. An SOA solution gives both business and IT the ability to make these changes quickly and efficiently without the costly and error-prone process of application re-engineering. The loose-coupling and encapsulation properties of SOA allow components to change independently. Centralized deployment leads to lower cost of integration. Finally, the Service Connectivity scenario is a great step towards a more enterprise-wide transformation towards a full SOA environment.

4.2.4 Value delivery

The business stakeholders of the ITSOMart solution need to realize a return on their investment as soon as possible. Building the solution around SOA allows immediate return on investment from the reduced costs of maintenance as well as the indirect gains from increased business/IT agility. One of the keys to lowering effort comes from the loosely coupled way in which developers build SOA solutions. By encapsulating integration logic in a single layer of the architecture, the integration logic can be easily understood and widely known, making changes to business processes less cumbersome. The increased visibility of integration logic and the ease of modifying that logic combined with increased business flexibility helps an SOA deliver value as soon as the first project moves into production. For ITSOMart, this is an important factor because the company is unwilling to invest a large amount of development time and money into any projects that cannot quickly show business value.

4.2.5 Security

The prevalence of heterogeneous environments within the enterprise IT environment makes implementing and enforcing security increasingly difficult. ITSOMart requires all applications to comply with relevant regulatory rules and

internal corporate standards. An ESB helps reduce some of the complexity involved in enforcing security through enforcing the use of a centralized security policy. Security is enforced at the bus, making enforcement less complex.

4.2.6 Management/monitoring

As IT applications become increasingly complex, management and monitoring of those applications becomes more difficult. Applications consisting of many tiers on disparate hardware must be monitored as an end-to-end application rather than a collection of components. Solutions built using a distributed architecture, like an SOA, are more difficult to monitor than monolithic applications. However, the componentized approach of SOA also means that, unlike monolithic applications, individual parts of the system can be easily monitored. Also, service levels may be centralized in the enterprise service bus, which reduces the management overhead.

4.2.7 Readiness

ITSOMart does not wish to engage in an enterprise-wide business and IT transformation. However, the company still wishes to leverage some of the advantages of using service orientation when creating applications. Service connectivity gives the company the ability to adopt SOA in phases. ITSOMart has already begun experimenting with Web services, although there are some traditional queue-based applications using Java Messaging Service (JMS) as well. Service connectivity allows all such systems to function as parts of the new composite application without rewriting heritage applications. Existing heritage applications can participate in an SOA environment using service wrappers. Service creation uses many of the same tools that already exist within most IT development organizations. Finally, since SOA is protocol and technology neutral, it allows IT to focus on solving business problems using familiar tools and technologies rather than learning a new set of technologies. These factors make service connectivity easy to introduce into most IT organizations and can provide a great entry into SOA.

4.3 Elements of an SOA solution

A service-oriented architecture solution includes a number of characteristics of SOA. These include:

- ▶ An architectural style consisting of service providers, requestors, and service descriptors
- ▶ A set of design principles including loose-coupling, encapsulation, and ease of composition
- ▶ A set of standards-based tools and technology to support the SOA programming model

The ITSOMart self-registration business scenario provides opportunities to embody each of these characteristics in the final solution. During the solution planning stage, enterprise architects should carefully consider the consequences that architectural decisions have on design, implementation, deployment, and governance. The following sections highlight those decisions and provide a suggested approach to planning an SOA solution.

4.3.1 Using the SOA Foundation Reference Model

The SOA Foundation Reference Model defines the comprehensive IT services required to support your SOA at each stage in the SOA life cycle. The Reference Model includes development environment, services management, application integration, and runtime Process Services. The capabilities of the architecture can be implemented on a build-as-you-go basis as new requirements are addressed over time.

The reference architecture provides all of the architecture elements needed to support the ITSOMart solution. According to the reference architecture, the SOA system is divided into a number of service types, each with different concerns and functionality. The ITSOMart solution focuses primarily on the integration aspects of these services and leaves some elements as heritage components. However, because the solution design segments these heritage components into individual service areas, the solution may evolve and replace heritage components with those built using SOA principles.

As ITSOMart began the process of solution design, they considered each element of the architecture and how, and if, it applied to their situation. Figure 4-2 shows the elements of the SOA Foundation Reference Model that apply to the ITSOMart solution (in bold).

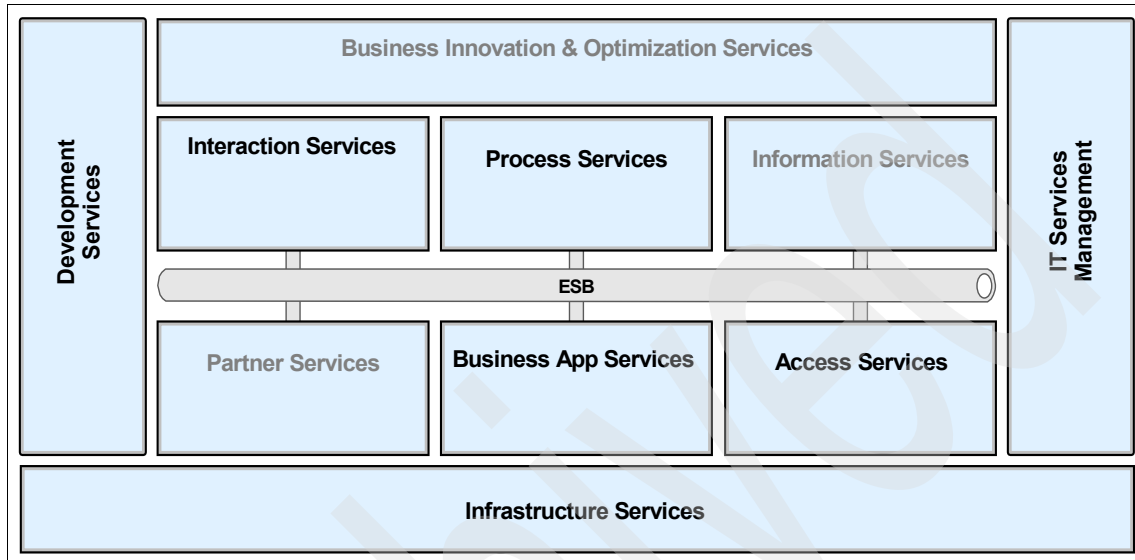


Figure 4-2 Elements of ITSOMart within the SOA Foundation Reference Architecture

Development Services provide IT architects, service developers, and integration developers with the tools and methodologies needed to design and create an open standards-based solution. These services support a variety of activities that occur during the course of the SOA life cycle.

The ITSOMart solution illustrates the use of each of these types of services:

- ▶ *Design* includes creating both high and detailed designs of the overall system, as well as interactions between the assembled components. The solution design also includes the mediations necessary to support the interactions between service providers and requestors.
- ▶ *Service Implementation* involves the creation of new services or wrapping existing applications with service facades in order for those applications to participate in the solution.
- ▶ *Integration Assembly* consists of assembling services together in order to achieve the desired business functionality. Integration developers may frequently recompose applications as the IT and business requirements evolve.

Runtime Services provide the deployment, monitoring, and governance functions necessary to implement the solution.

- ▶ *Interaction Services* are the points of contact between end users and the system. Ideally, application developers should also be able to create composite applications using various Interaction Services. A portal is an example of the Interaction Services layer that supports composition. The ITSOMart solution uses a traditional Web application in place of a portal. The solution design allows this layer to easily be replaced when the need arises.
- ▶ *Process Services* provide the composition logic that binds various services. Typically, Process Services refer to either business process flow execution or a business state machine (a finite state machine that orchestrates the service interaction).

Because the primary purpose of the ITSOMart solution is to demonstrate integration patterns, the solution currently uses a single fixed service to provide business process flow control. As business needs evolve over time, this process flow component could be replaced by a business process flow controlled by BPEL or a business state machine.

- ▶ *Business Application Services* represent fundamental business logic and represent the building blocks for composition. Typically, a business process orchestration service combines multiple Business Application Services to produce a flexible, coarse-grained service that can then be consumed by other service requestors.

ITSOMart contains a number of such Business Application Services. These are:

- Credit check

In reality, a credit check process would most likely itself be a composite service orchestrated by a business process flow. However, because this service component is entirely opaque to the ITSOMart solution, the design treats this service as an atomic building block that cannot be further decomposed.

- CRM update
- Shipping address registration
- Audit log

- ▶ *Access Services* augment heritage applications, often with a simple service wrapper so that such applications may participate in the SOA ecosystem without modification. Previous architectures treat Access Services as adapters. In the reference architecture, these adapters are explicitly treated as services so that they may be used in the same way as other services.

ITSOMart will require adapters to access the CRM system and functions that require flat file access.

- ▶ *Infrastructure Services* refer to services provided by the integration runtime. The primary purpose of these services is to abstract the underlying execution platform so that service implementors and integration developers are concerned only with their specific responsibilities. In addition, infrastructure services may consist of boundary services whose purpose allows service creators to focus on core functionality rather than meeting the needs of an entire enterprise. For example, a service that is only available via SOAP over HTTP should not be restricted from participating in environments requiring SOAP over JMS or SOAP over HTTPS — these are not the concerns of the service creator. Rather, a boundary service can seamlessly perform protocol transformation to extend the reach of the service beyond the original intended scope.
- ▶ *ESB* provides the connectivity for intercommunication between various services. For example, a traditional point-to-point integration style would use a mesh-style connectivity model. A messaging hub style integration, on the other hand, would use a hub and spoke connectivity model. In an SOA architecture, the preferred connectivity model is an enterprise service bus. ITSOMart currently has no ESB implementation, but plans to add this capability as part of the new solution.
- ▶ *IT Service Management* involves monitoring and managing the operational aspects of services. This includes service uptime, performance, and resource utilization. Management also includes preventing invalid or erroneous requests from reaching the service and filtering these earlier in the sequence of service processing steps. Service management is essential to maintaining a healthy SOA environment so that problems may be dealt with proactively rather than in a reactive fashion. ITSOMart will require that the systems be monitored for use and performance.

The SOA Foundation Reference Architecture provides a complete enterprise architecture for delivering SOA solutions. By considering the ITSOMart solution as an entire integration ecosystem rather than a single application, the solution architecture may easily be broken down into a core set of services that can be delivered by developers focused on service implementation and assembled by integration developers. The flexibility to compose and recompose applications allows for the re-use of existing services as well as greater IT agility to meet the evolving needs of business.

4.4 Selecting the SOA scenario and pattern

This section demonstrates how to use the process defined in Chapter 2, “Process for applying SOA scenarios” on page 21, to accelerate the design of the solution architecture for the ITSOMart working example scenario.

4.4.1 Fit gap analysis

In 4.1.5, “Functional requirements for ITSOMart” on page 71, we identified the functional requirements for the ITSOMart solution. As part of the defined process, we now perform a fit-gap analysis of the ITSOMart functional requirements with the generic use cases.

Table 4-1 provides a summary of the ITSOMart functional requirements that map to the generic use cases. There may be functional requirements that are not mapped to a generic use case specific to your business scenario. There may also be functional requirements that map to generic use cases for multiple SOA scenarios.

Table 4-1 ITSOMart - fit gap analysis

ITSO car rental example functional requirements	Generic use cases
<ul style="list-style-type: none">▶ FR1: Expose a new business function to customers.▶ FR3: Expose internal applications as Web services.	<ul style="list-style-type: none">▶ U1: Reuse existing or create new application logic as a service within the enterprise.
<ul style="list-style-type: none">▶ FR2: Create a Web services client application to consume services.	<ul style="list-style-type: none">▶ U3: Point-to-point integration of enterprise applications.▶ U5: Allow users to invoke services simply.
<ul style="list-style-type: none">▶ FR4: Integrate internal applications so they can function together to perform a business task.▶ FR5: Incorporate simple routing logic to allow dynamic decisions based on customer input and responses from invoked services.▶ FR6: Incorporate flexibility to easily change or add services.	<ul style="list-style-type: none">▶ U7: Enable loose coupling of service consumers and providers using dynamic routing based on standards-based protocols.

4.4.2 Select the SOA scenario

At this stage in the process we have identified the generic use cases that apply to the ITSOMart example. Now we use the generic use cases as criteria to select the appropriate SOA scenario containing software capable of fulfilling the requirements. This task is accomplished by using the SOA scenario selection table.

Table 4-2 shows a subset of the generic use cases found in Table 2-1 on page 24 that fulfill the functional requirements of the ITSOMart example scenario (Service Connectivity).

Table 4-2 SOA scenario selection criteria identifying the Service Connectivity scenario

Generic use cases selection criteria	SOA scenarios				
	Service Creation	Service Connectivity	Interaction and Collaboration Services	Business Process Management	Information as a Service
U1: Reuse existing or create new application logic as a service within the enterprise.	X				
U3: Point-to-point integration of enterprise applications.	X				
U5: Allow users to invoke services simply.	X				
U7: Enable loose coupling of service consumers and providers using dynamic routing based on standards-based protocols.		X			

Because the solution requires routing capability, the Service Connectivity scenario is the most appropriate. The Service Connectivity scenario, with the appropriate product selection, can also provide the function required by the use cases identified for Service Creation.

4.4.3 Reuse patterns assets to accelerate solution architecture

This section demonstrates how to apply the reusable assets found in the Patterns for e-business to accelerate the design of the solution architecture.

Select the Business pattern

Looking at the overall business context diagram, we can identify three potential Business or Integration patterns that might come into play (Figure 4-3).

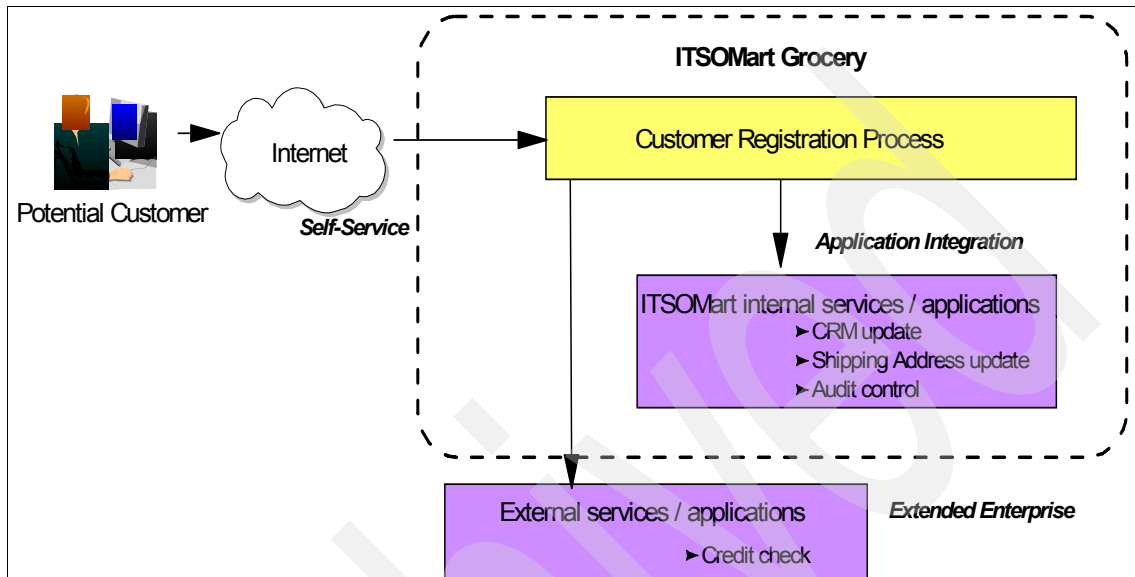


Figure 4-3 Potential Business pattern choices

The patterns are:

- ▶ Self-Service business pattern

Because the intent of the ITSOMart solution allows consumer access to internal resources, the Self-Service business pattern is clearly the pattern that defines the overall functionality of the solution. This is the pattern that the ITSOMart solution designers will use as their guide.

For details refer to 3.4, “Self-Service business pattern” on page 44.

- ▶ Application Integration pattern

The interaction between the Customer Registration Process and the ITSOMart internal services and applications is indicative of an Application Integration pattern. As the ITSOMart solution is designed, guidelines defined for this pattern will be consulted for best practices information.

For details refer to 3.5, “Application Integration pattern” on page 52.

- ▶ Extended Enterprise business pattern

The invocation of external services brings the Extended Enterprise business pattern into play. The concerns addressed by this pattern focus on the unique requirements exposed by the integration of applications in separate

enterprises, for example, security concerns. Enterprises must address these concerns up front when designing a solution, although, for simplicity, the ITSOMart example will not address these in this book.

For details refer to 3.6, “Extended Enterprise business pattern” on page 60.

Select the Application pattern

Now that we know the focus is on the Self-Service business pattern, we can select an Application pattern. Application patterns are used to provide a conceptual layout of how the application components and data within a Business pattern or Integration pattern interact.

Table 3-1 on page 38 provides a summary of the Business and Integration patterns and corresponding Application patterns that map to the generic use cases representative of the Service Connectivity scenario. As you can see from this table, there is one Application pattern for Self-Service that is most suitable to fulfill the use cases that describe the ITSOMart requirements. In particular, the routing requirement of use case U7 drives this selection, the Router application pattern. For details refer to 3.4.2, “Router application pattern” on page 47.

Runtime patterns

In the previous section we identified the Application patterns that apply to the Service Connectivity scenario. Each of the Application patterns has a corresponding Runtime pattern, which is used to define the logical middleware structure and interaction between nodes to support the Application pattern.

The Runtime pattern selected is shown in “Runtime pattern: SOA profile for Router” on page 48. The Directory and Security Services, and Business Service Directory nodes are optional for the scenario. We did not implement the functionality of these nodes for the ITSOMart example. In a production environment, they typically would be implemented.

Product mapping

The Product mapping used for the ITSOMart example is shown in “Product mapping” on page 49. The product mapping was a result of the selection of the ESB implementation and the services required for the customer registration process. We discuss the options for ESB implementation and the choices made by ITSOMart in the next sections.

4.5 Enterprise service bus product selection

What makes the ESB so important in a Service Connectivity scenario is that it allows a higher level of abstraction for service integration. The ESB hides the

details of service locations and transport so that service implementations only need to deal with invoking the service via the ESB.

ESBs also exist to resolve differences between service requestors and service providers, a common problem in loosely coupled systems. For example, the target service may not implement the same service interface or message structure as the client. The bus must provide a mechanism to resolve the differences between the service requestor and provider so that these details are hidden from the actual service implementation.

Another common issue is that many services may provide the same fundamental business function but with subtle differences (for example, performance). The enterprise service bus performs service request routing based on predefined rules that stipulate which of the available service providers receives the request. For example, in a credit-checking scenario, one provider may return faster results but charge more per transaction versus a slower but cheaper provider. The service bus could route credit check requests to the more optimal provider based on factors such as the urgency of the request.

All of the functions of the service bus (routing, logging, and message transformation) are hosted within the bus in the components called *mediations*. The mediation logic may often consist of a set of predefined basic logic functions such as message logging, message augmentation, or transformation. A mediation development environment such as WebSphere Integration Developer could leverage these patterns to allow consistent mediation development. However, there are instances when the demands on the mediation exceed the basic mediation logic, in which case the ESB should be capable of hosting more complex, custom mediation logic.

When considering which ESB implementation is suitable for your solution, you should consider the product capabilities with regard to the following:

- ▶ Transport protocol support and conversion capability
- ▶ Interactions supported (one-way, synchronous request/response, asynchronous request/response, pub/sub, aggregation, and so on)
- ▶ Routing capabilities
- ▶ Dynamic configuration capabilities
- ▶ Mediation capabilities
- ▶ Ease of development and administration
- ▶ Quality of Service capabilities
- ▶ Existing conditions in the enterprise and customer skill set

The following sections will describe two products from IBM that may be used to implement an enterprise service bus, WebSphere Message Broker, and WebSphere Enterprise Service Bus. In addition, we give a brief description of the IBM DataPower SOA appliances that can fulfill the requirement of an ESB gateway. The products are from different backgrounds and have varying levels of capabilities, but each can be used separately, or together, to achieve the functionality of an enterprise service bus.

The ESB product selected will drive the choice of products for the other life-cycle phases. Table 4-3 shows suggested combinations of products.

Table 4-3 Possible product selections for implementing SOA connectivity

Deploy	Model/assemble	Manage
DataPower XS40	DataPower Tooling (included with DP)	Tivoli Access Manager
WebSphere ESB 6.0.1	WebSphere Integration Developer 6.0.1	Tivoli Composite Application Monitor for SOA Tivoli Composite Application Monitor for RTT
WebSphere Message Broker V6	WebSphere Message Brokers Toolkit Rational Application Developer	OMEGAMON

4.5.1 WebSphere Enterprise Service Bus

WebSphere ESB is an ESB runtime that allows rapid implementation of the ESB pattern.

Connectivity

WebSphere ESB is built on top of WebSphere Application Server and provides all of the same connectivity capabilities. The service bus can seamlessly perform message transport and protocol transformation for any of the supported protocols, which include:

- ▶ Java Messaging Service (JMS)
- ▶ SOAP over HTTP
- ▶ SOAP over JMS

WebSphere ESB supports all of the common messaging models including:

- ▶ Request/response
- ▶ Publish/subscribe
- ▶ One-way message distribution

Adapters

WebSphere ESB supports connectivity for WebSphere Adapters, both JCA-based WebSphere Adapters and the WebSphere Business Integration (WBI) Adapters.

The WBI Adapters provide a wide range of connectivity that support interaction with suppliers, trading partners, and customers. WBI Adapters attach packaged, traditional, custom, mainframe, and Web applications to the ESB.

WebSphere Adapters are compliant with J2EE Connector Architecture (JCA 1.5). JCA is the J2EE standard for EIS connectivity. EIS Import and EIS Export provide SCA components with the uniform view of the services external to the module. This allows components to communicate with the variety of external EIS systems using the consistent SCA programming model. WebSphere Adapters are assembled in WebSphere Integration Developer from imported RAR files and then exported within an Enterprise Application Archive (EAR) file and deployed on WebSphere ESB.

WebSphere Adapters include the following:

- ▶ IBM WebSphere Adapter For Flat Files, Version 6.0
- ▶ IBM WebSphere Adapter for JDBC™, Version 6.0
- ▶ IBM WebSphere Adapter for PeopleSoft Enterprise, Version 6.0
- ▶ IBM WebSphere Adapter for Siebel Business Applications, Version 6.0
- ▶ IBM WebSphere Adapter for SAP Applications, Version 6.0

For information about adapters see:

<http://www-306.ibm.com/software/integration/wbiadapters/>

Mediation modules

At its core, WebSphere ESB is a runtime environment capable of hosting message mediations. Mediations are deployed to the runtime as a type of SCA module called a mediation module. Mediation modules contain imports, exports, and SCA components that typically include a mediation flow component and optionally, Java SCA components. Mediation flow components contain mediation flows that operate on the message requests and responses. Figure 4-4 illustrates a mediation module.

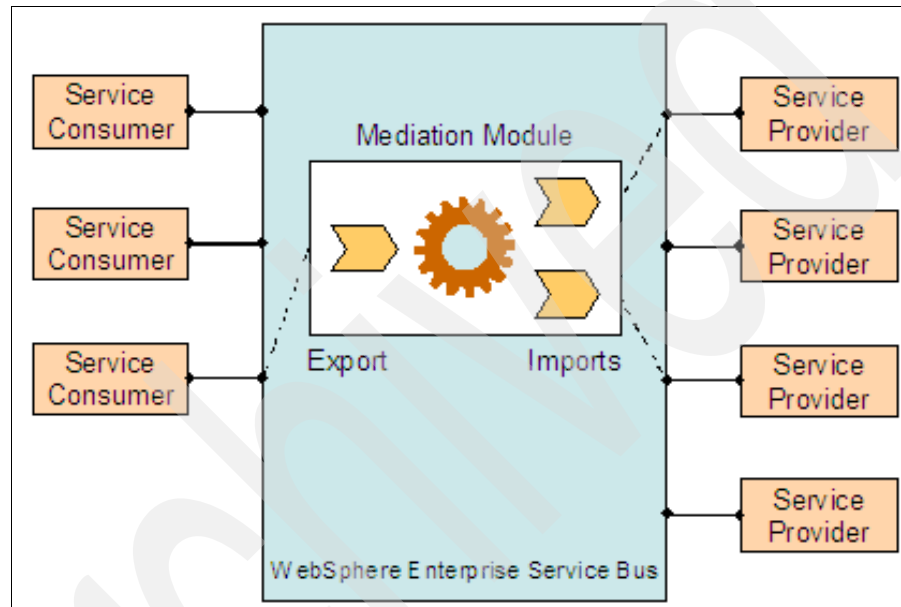


Figure 4-4 Mediation module

Mediation modules may use other mediation modules to perform a complete message mediation. This allows more flexible service assembly, and developers may reuse mediation modules across several mediations. Figure 4-5 illustrates how multiple modules may be used together to produce a complete message mediation.

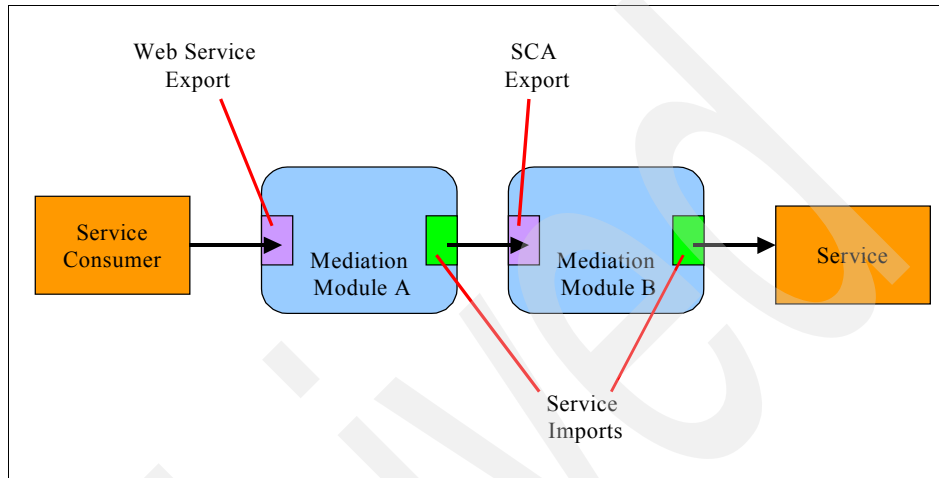


Figure 4-5 Multiple mediation modules

In the example, the service consumer sends a request to *module A*, which uses an SCA import from *module B* to invoke the mediation flow within *module B*, which eventually calls the service.

Mediation flow

A mediation module can contain one mediation flow component that contains mediation flows. Mediation flows within WebSphere ESB support a number of basic mediation patterns:

- ▶ Message protocol transformation
- ▶ Content-based message routing
- ▶ Message format transformation
- ▶ Message augmentation

Message protocol transformation

Message protocol transformation allows service requestors to use one particular protocol while the service provider uses a different protocol. This functionality is useful because otherwise service providers would need to support every potential service request protocol. Figure 4-6 shows an example of protocol transformation.

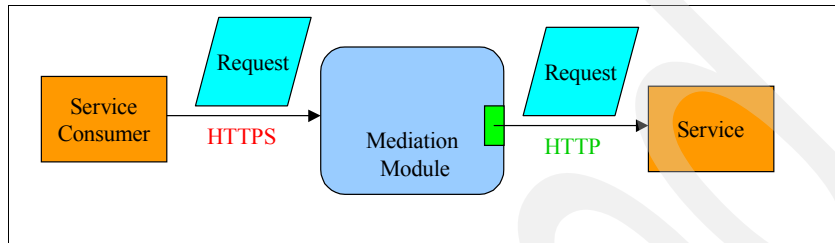


Figure 4-6 Protocol translation

In this example the service request travels over a secure SOAP-over-HTTPS channel. The mediation module makes a request to the service that operates using SOAP over HTTP. However, this service could have been SOAP over JMS, or even JMS over an MQ queue.

Content-based message routing

Sometimes a service request may be fulfilled by more than one service provider. During service invocation, the service requestor may not and should not know about the details of the service provider that is ultimately servicing the request. Content-based request routing allows WebSphere ESB to dynamically decide how to route messages using filtering rules.

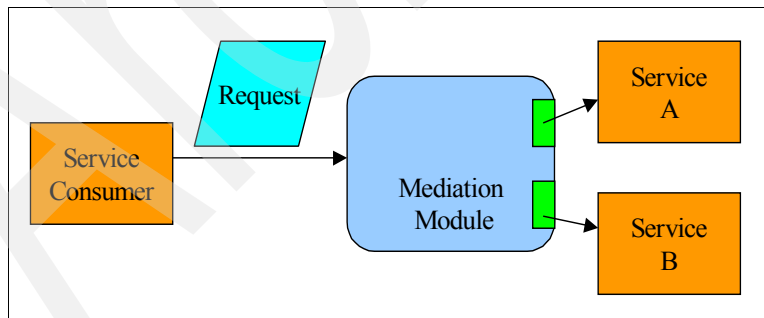


Figure 4-7 Content-based routing

Figure 4-7 shows a service request that then can be handled either by *service A* or *service B*. The mediation module decides based on the message attributes (content, sender, and so on) which of the services will be used to handle the

request. A typical use case for this type of functionality is quality of service. For example, with a stock quote service, premium clients can receive real-time quotes while regular clients receive delayed quotes.

Message format transformation

Service requestors and service providers may use different syntax for requests. WebSphere ESB is able to rewrite requests and responses as they flow through the bus so that the service request and response are in the correct syntax for the requestor and provider.

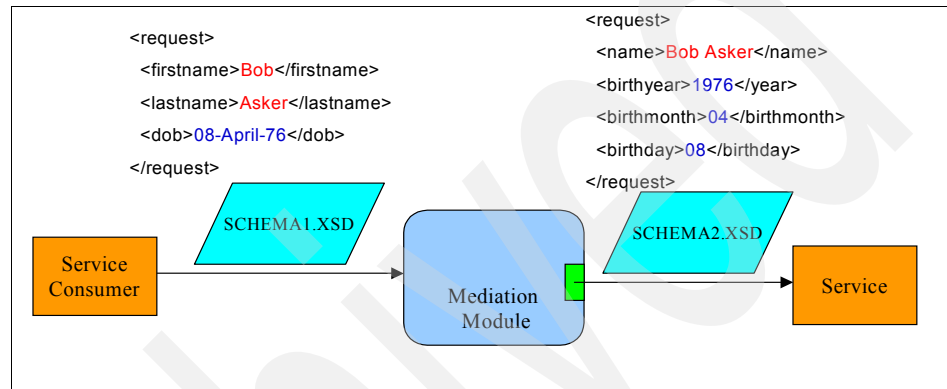


Figure 4-8 Message format transformation

Figure 4-8 shows a service whose specification uses schema2.xsd. However, the service consumer creates requests according to schema1.xsd. In this case, the mediation module can translate requests using the consumer schema into requests using the provider schema, as well as translate the response messages. This allows the consumer and provider to communicate even though they use different syntax.

Message augmentation

In addition to message format transformation, WebSphere ESB is also able to perform additional message augmentation. For example, suppose that the service requester provides a first and last name but no account number. The mediation can perform a database lookup to find the account number for the user and add it to the message before sending it on to the service provider.

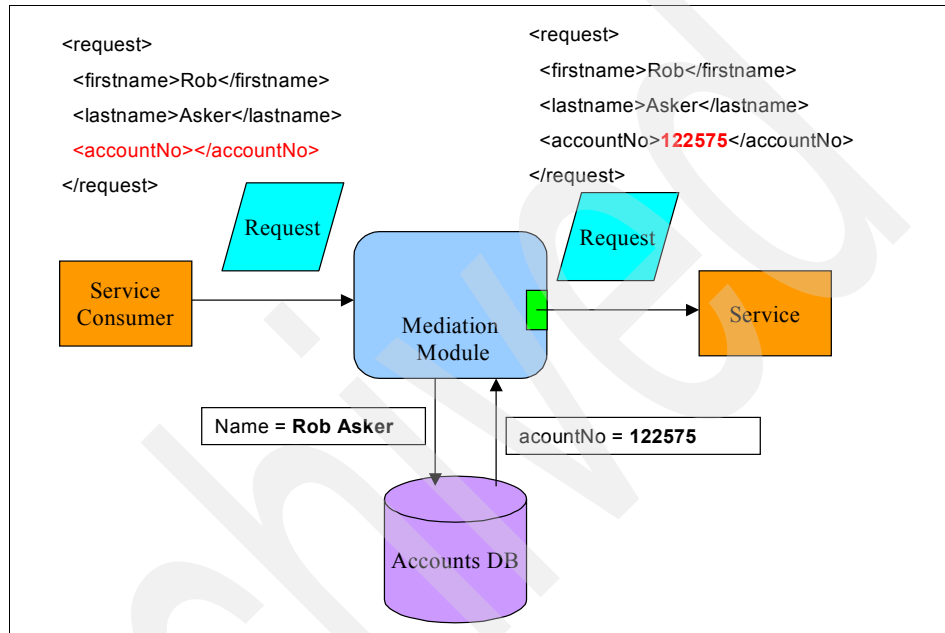


Figure 4-9 Database lookup message augmentation

Figure 4-9 shows a request message augmentation that uses a database to look up user account numbers. The service consumer provides a first name and a last name. The mediation module uses an accounts database to look up the user's account number and sends the request to the service using this account number.

Messaging clients

WebSphere ESB provides Message Service Clients that extend the connectivity of the enterprise service bus.

- Message Service Clients for C/C++ and .NET enable non-Java applications to connect to WebSphere ESB.

Message Service Clients for C/C++ and .NET provide an API called XMS that provides a set of interfaces similar to the Java Message Service (JMS) API. Message Service Client for C/C++ contains two implementations of XMS, one for use by C applications and another for use by C++ applications. Message

Service Client for .NET contains a fully managed implementation of XMS, which can be used by any .NET compliant language.

- ▶ Web Services Client for C++ provides a set of libraries and Java tools that enable you to build ANSI C++ Web service client applications from existing Web Service Description Language (WSDL) files.

The ANSI C++ Web service client applications that you build from existing WSDL files, using the Web Services Client for C++ libraries and Java tools, are able to communicate with applications over HTTP using TCP/IP with SOAP protocols.

You can also install and use the Application Client for WebSphere Application Server to provide J2EE client support. This includes the Web services Client, EJB™ Client, and JMS Client.

4.5.2 WebSphere Message Broker

WebSphere Message Broker may also be used for implementing an ESB.

Unlike WebSphere ESB, WebSphere Message Broker is not based on WebSphere Application Server but is built around a traditional message-based integration approach, leveraging WebSphere MQ, and provides a robust and flexible messaging hub for mediation. However, the product can be implemented and deployed as an enterprise service bus pattern. Other features include the ability to extend the reach of the enterprise service bus to non-SOAP messages such as AL3, HL7, Swift, HIPAA, EDI, and so on. The product also includes support for creating complex message transformations without the need for complex programming.

Like WebSphere ESB, WebSphere Message Broker provides the following capabilities:

- ▶ Message protocol transformation
- ▶ Content-based message routing
- ▶ Message format transformation
- ▶ Message augmentation

In WebSphere Message Broker terminology, the mediation hosted by an ESB implemented using WebSphere Message Broker consists of two key components: *message models* and *message flows*.

Message models

WebSphere Message Broker includes parsers capable of parsing and emitting a broad range of message formats. Some formats such as XML are self-describing, that is, these formats include the message metadata as well as

message data, enabling generic parsers to understand the message. Other formats such as COBOL and C binary messages contain no such metadata. Thus, WebSphere Message Broker requires message models to describe the format of messages so that the parser can correctly interpret the incoming message. Message models also improve message flow development, as the development tool is capable of importing the message model to document the message format. WebSphere Message Broker includes a number of parsers that operate on different types of message formats. The use of a particular parser is defined by the message domain. The following message domains are available:

- ▶ MRM handles a wide variety of message types including binary, formatted text, as well as XML.
- ▶ XML handles XML message formats.
- ▶ JMS handles messages produced by JMS providers, such as WebSphere MQ.
- ▶ IDOC handles messages produced in the SAP IDoc format.
- ▶ MIME handles multi-part MIME messages such as SOAP attachments.
- ▶ Custom for a user-defined parser.

Although XML messages do not need a message model, implementing one is helpful for WebSphere Message Broker to properly interpret attribute data types (otherwise everything is treated as a text string), as well as aiding the message flow development process.

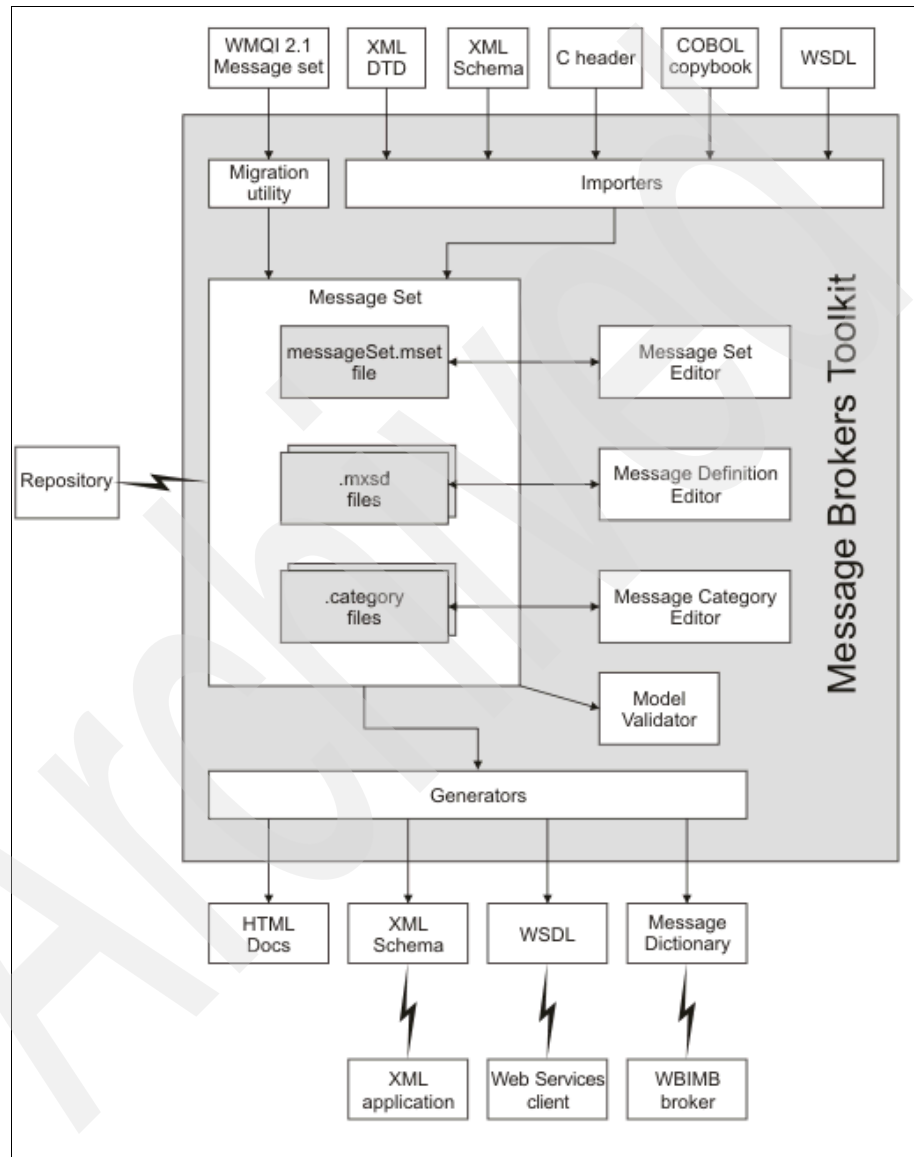


Figure 4-10 WebSphere Message Broker message model

Message flows

Like WebSphere ESB, message flows within WebSphere Message Broker define the set of processing steps to execute when a message is flowing through the system. Message flows consist of a set of *nodes* that define a particular action to take against the message. All message flows must include an input node that specifies the message source. In addition, the following nodes are available for building message flows:

- ▶ Built-in nodes are nodes that are shipped with WebSphere Message Broker (includes input, output, decision, and error handling nodes).
- ▶ User-defined nodes are extensions to standard WebSphere Message Broker built-in nodes. User-defined nodes must use the standard C or Java APIs provided by WebSphere Message Broker.
- ▶ Subflow nodes consist of additional message flows that are part of a larger message flow. Subflows allow sharing of commonly used message flows.

Applying the graph paradigm to message flows, with nodes being vertices in the graph, *connections* are the edges. Connections link the output terminal of one message node to the input terminal of a subsequent message node. Connections represent the flow of data through the message flow.

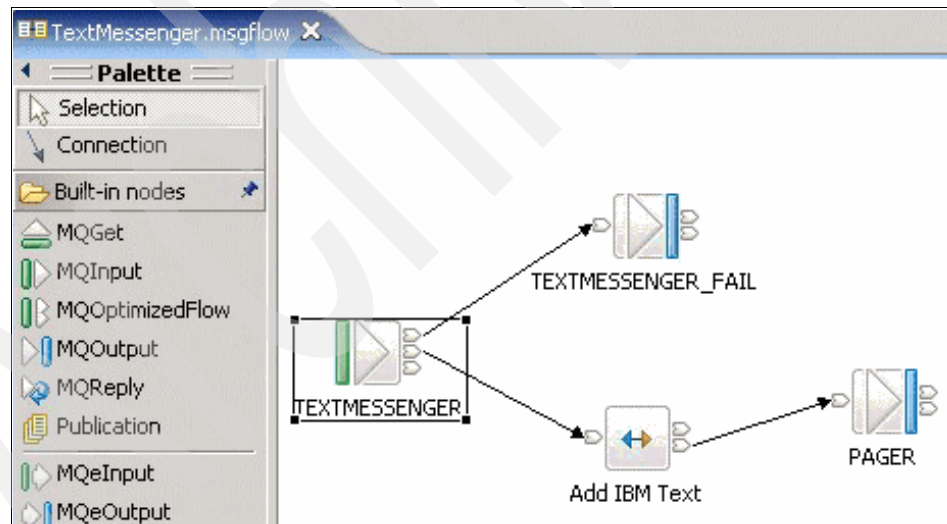


Figure 4-11 WebSphere Message Broker message flow

ESQL

WebSphere Message Broker includes a built-in message manipulation language called Extended Structured Query Language (ESQL). Based on Structured Query Language (SQL), ESQL allows message flow nodes to perform

computations based on the message content as well as database content. For example, an ESQL compute node might augment the message content using data from a relational database. Example 4-1 shows a sample ESQL that performs different actions based on the type of credit the user is submitting.

Example 4-1 Sample ESQL

```
IF InputBody.Order.Payment.CreditCardType='VISA' THEN
    DO;
        -- SOME ACTION
    END IF;
IF InputBody.Order.Payment.CreditCardType='MASTERCARD' THEN
    DO;
        -- SOME OTHER ACTION
    END IF;
```

Developing for WebSphere Message Broker

Integration developers create message models and message flows using a development tool called the *Message Brokers Toolkit*. Like the WebSphere Integration Developer, the Message Brokers Toolkit is an Eclipse framework-based application. The primary component of the Message Brokers Toolkit is called the workbench. The workbench consists of a collection of Eclipse perspectives and views that provide the development functions. The workbench also includes a flow debugger that can aid in testing message flows. The flow debugger can attach to running flow engines and observe and manipulate in-flight messages as they pass through the nodes in the message flow. Finally, the workbench is capable of deploying message flow applications to the runtime environment, both for testing and production.

Runtime

The WebSphere Message Broker runtime consists of a various runtime components, including *Broker domains*, *configuration managers*, *Brokers*, and *execution groups*. Broker domains are a group of Brokers along with a shared configuration manager that controls all of the Brokers. The configuration manager is responsible for communicating between the workbench and the Brokers. The configuration manager controls the Broker domains' runtime configuration, manages deployment of message flows, and communicates with the other components of the Broker runtime. A Broker hosts execution groups that are a set of message flows grouped together. The Broker isolates the message flows within its execution groups from each other and allows separate

runtime configuration (each execution group receives its own thread pool). Figure 4-12 illustrates the components of the WebSphere Message Broker runtime.

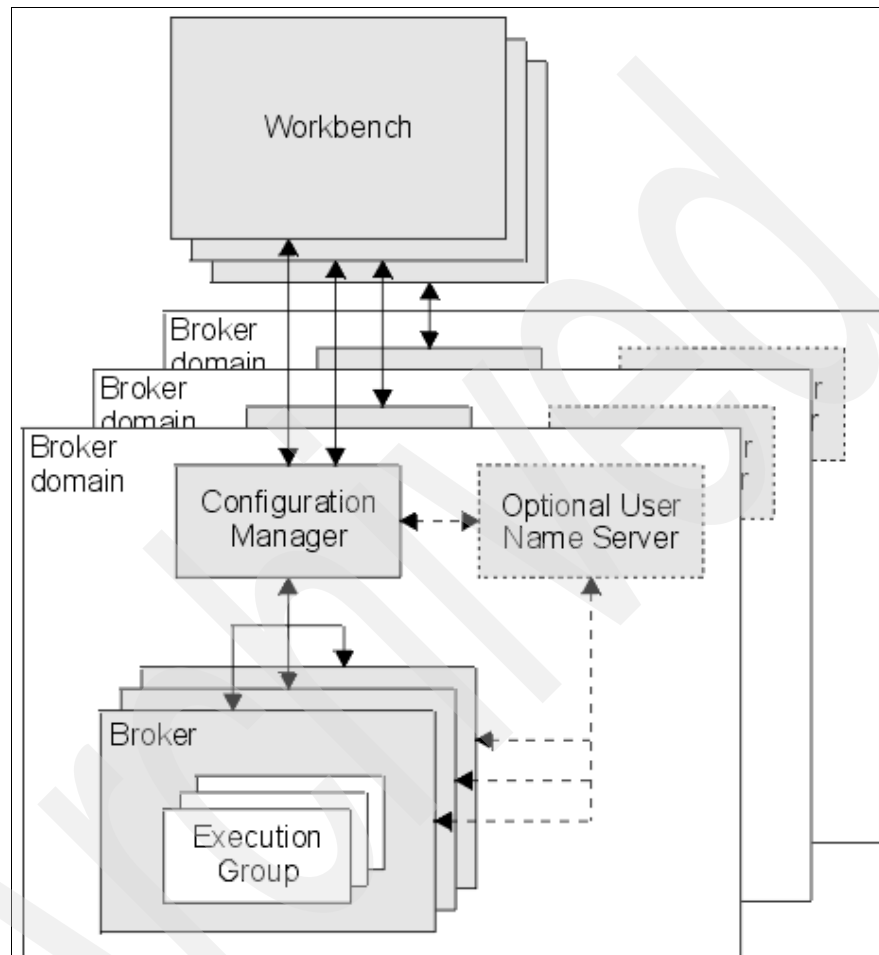


Figure 4-12 WebSphere Message Broker runtime

4.5.3 WebSphere DataPower SOA Appliance

Extensible Markup Language (XML) has proven to be a great force in the software industry. XML's flexible, self-describing, language-independent format makes decoupling partner systems much easier. However, the heavy reliance on XML for data transfer between services also presents some problems. For example, XML can result in lengthy message payloads and large amounts of overhead for schema validation and parsing. The processing overhead of dealing

with XML can tax application servers and middleware infrastructure, drastically decreasing performance.

The evolution of network infrastructure has seen an increasing trend toward replacing general purpose software systems with dedicated hardware for increased performance. As shown in Figure 4-13, there has been a move from network Routers based on UNIX® towards dedicated hardware Routers, from software Web servers towards load-balancing infrastructures, content switches, and SSL accelerators.

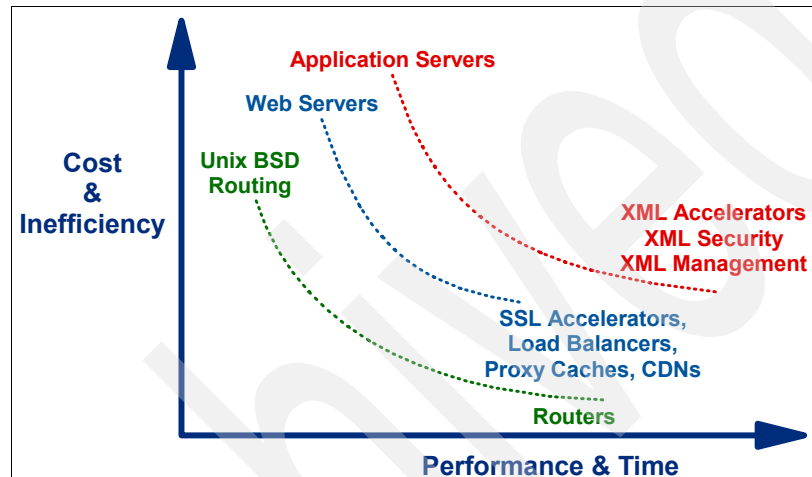


Figure 4-13 Evolution towards hardware-based network devices

In this same way, there is an evolution towards using dedicated hardware for performing repetitive XML tasks such as parsing, schema validation, and XML Stylesheet Language (XSL) translation.

Service protocols based on XML also lack any inherent built-in security mechanisms. SOAP over HTTP passes potentially sensitive data in plaintext over the network. While there have been emerging standards such as WS-Security to help deal with security concerns, implementing these standards further drains computing resources on critical servers.

The IBM line of DataPower SOA appliances helps address the performance and security needs of enterprise-level SOA architectures by off-loading the XML processing onto dedicated hardware, freeing the CPU resources of application servers and middleware platforms to provide higher service throughput. DataPower SOA appliances provide a range of features including:

- ▶ XML/SOAP firewall, filtering based on message content, headers, or other network variables

- ▶ Incoming/outgoing data validation
- ▶ Schema validation
- ▶ XML security, access control, authentication, and authorization
- ▶ Integration with various security and monitoring products such as IBM Tivoli Enterprise™ Monitoring, Tivoli Access Manager, Netegrity SiteMinder, and so on

The performance advantage of DataPower appliances are often close to seventy times higher than when using general purpose systems alone. Figure 4-14 shows the relative performance costs of various infrastructure-type XML tasks. As shown, when digital signature checking and message encryption/decryption take place, there is a great deal of overhead in processing messages. The XML appliance can off-load this processing from application servers onto dedicated hardware capable of performing these tasks in a fraction of the time.

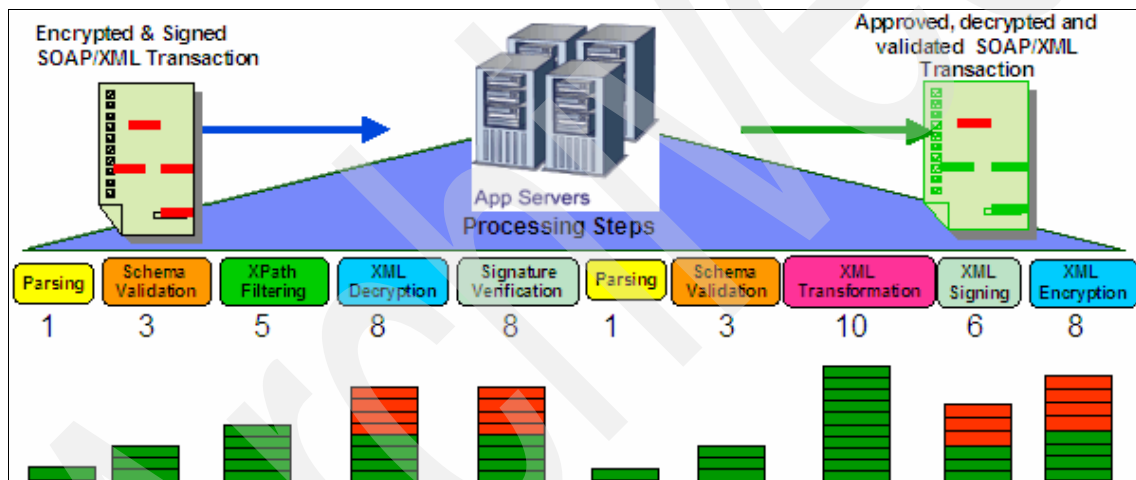


Figure 4-14 Performance costs for XML processing tasks

For more information about DataPower SOA Appliances see:

<http://www-306.ibm.com/software/integration/datapower/index.html>

DataPower in a WebSphere ESB environment

DataPower and WebSphere ESB are very complementary products in that organizations will typically build message mediations and transformations inside of the service bus using WebSphere Integration Developer. The DataPower appliance can then be deployed to the edges of the network to perform message

security, authentication, and authorization. Figure 4-15 illustrates a federated extranet involving XML security appliances that filter inbound and outbound traffic while allowing valid messages to be mediated through the service bus.

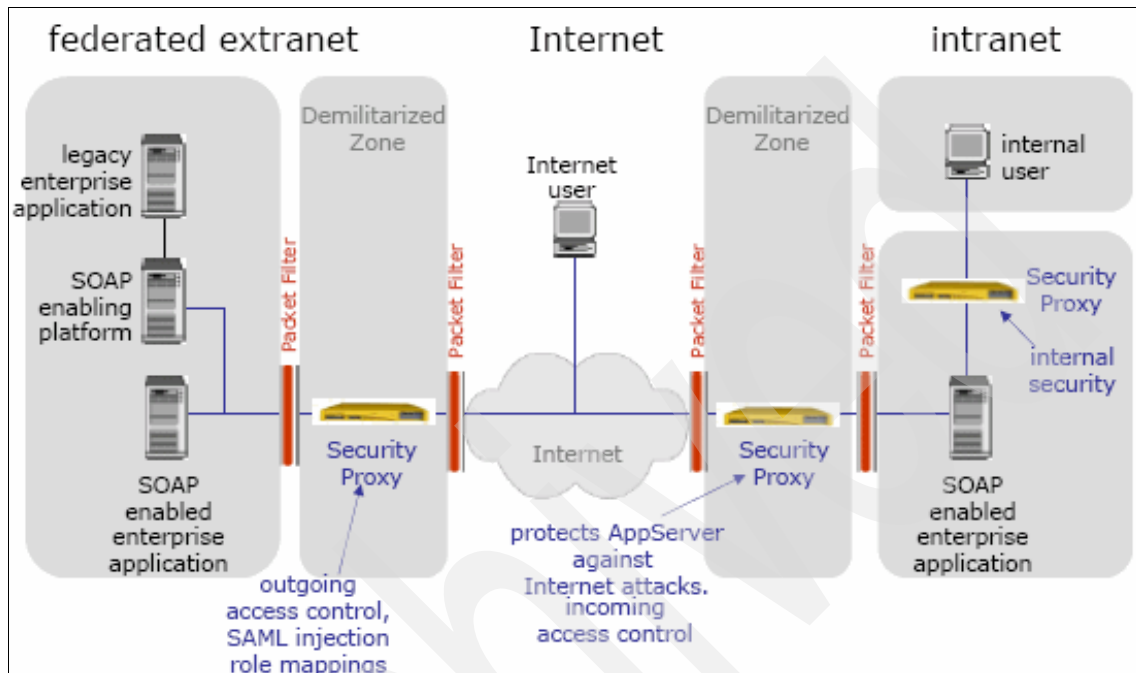


Figure 4-15 A federated extranet with XML security appliances protecting traffic

DataPower appliances can also be used to extend the reach of WebSphere ESB by translating messages consisting of non-XML formats for use by the service bus. DataPower's built-in translation engine allows powerful message transformations between XML and non-XML or binary formats, making it possible for heritage systems to participate in the SOA environment. The appliances can also route requests based on message content to different branch installations of WebSphere ESB. In this way, the appliance acts as a central message concentrator that routes message traffic as appropriate.

4.5.4 Enterprise service bus implementations compared

In summary, IBM offers two ESB products to help build your SOA solution:

- ▶ IBM WebSphere Enterprise Service Bus
- ▶ IBM WebSphere Message Broker (an advanced ESB)

In addition, there are a number of products that work in concert with these ESBs including IBM WebSphere DataPower SOA Appliances. This section is designed

to help solution planners decide which of these products is the most appropriate for their situation.

IBM WebSphere Enterprise Service Bus (WebSphere ESB), a core component of the WebSphere software platform, is built on WebSphere Application Server. WebSphere ESB targets a predominantly Web services based environment. WebSphere ESB adds a mediation layer based on the SCA programming model on top of WebSphere Application Server foundation to provide intelligent connectivity using the native WebSphere Application Server transports.

Building an ESB that is based entirely on WebSphere ESB is an option when Web services support is critical and the service provider and consumer interact via open standards. WebSphere ESB is most suitable for environments that are based on Web services standards and provides facilities to integrate services that are offered via other sources. However, if interaction with non-Web service standards-based services with fast throughput is a major requirement, then WebSphere Message Broker may be better suited for those projects.

IBM WebSphere Message Broker delivers an advanced enterprise service bus providing connectivity and universal data transformation for both standard and non-standards-based applications and services to power your service-oriented architecture. Message Broker intelligently routes and transforms messages and data in real-time from multiple device types, business units, and locations to virtually all systems and applications throughout your enterprise — and beyond. Message Broker offers transactional processing, near universal connectivity over multiple transports including WebSphere MQ, JMS, and HTTP, and reliability and performance unmatched by others in the market.

WebSphere Message Broker is an option when quality-of-service requirements demand the use of mature middleware and deployment requirements go beyond Web services integration. WebSphere Message Broker can support all the ESB capabilities that WebSphere Enterprise Service Bus does, offer broader transformation and mediation features, and support integration assets that may or may not be Web services enabled.

The extensive use of XML and Web services brings new challenges and requirements to your IT environment. WebSphere DataPower simplifies SOA infrastructure with specialized software/hardware devices, expedites SOA XML processing, and enhances SOA security management with XML-level protection. WebSphere DataPower can be used on its own as an ESB Gateway or in conjunction with either WebSphere ESB or WebSphere Message Broker.

4.6 ITSOMart product selection

One of the principal goals of SOA is technology agnosticism. This allows IT to freely choose best-of-breed tools to fulfill their needs in each area, without requiring that services be built using a particular tool, language, or protocol. However, the importance of product selection cannot be ignored and should be considered when planning a solution. IBM offers a broad set of standards-based tools that encompass the SOA life cycle.

This section discusses the set of products selected to design and implement the ITSOMart solution. We have mapped these products to the SOA Foundation Reference Architecture in Figure 4-16.

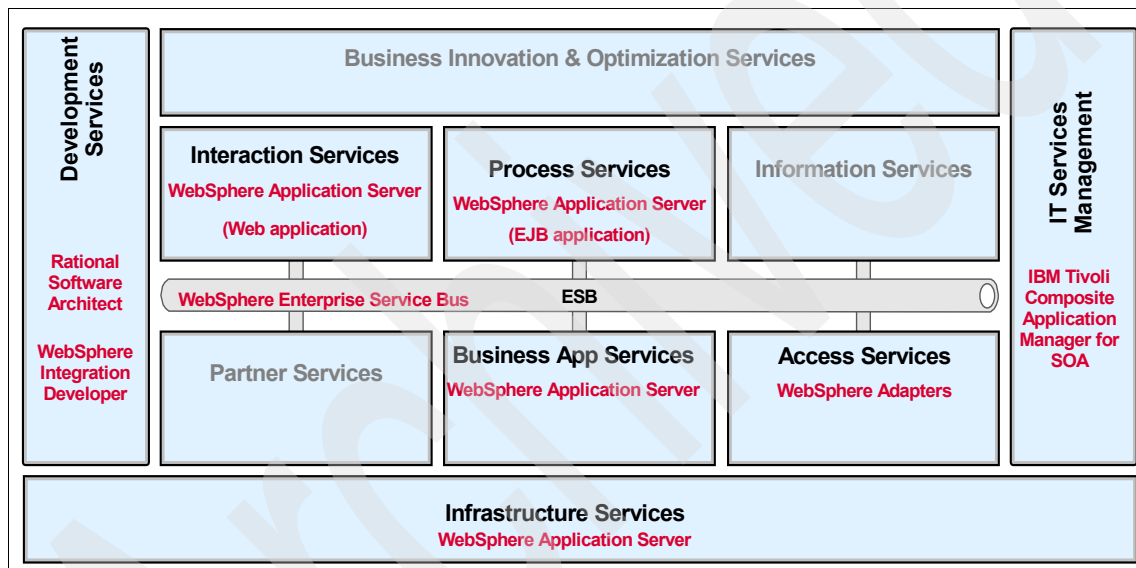


Figure 4-16 ITSOMart product selection

4.6.1 Deployment

Services deployment/runtime support requires containers that can manage runtime concerns such as service invocation, life cycle, and access to underlying resources such as databases. The runtime environment should also be able to support management/monitoring functions or allow plug-ins that enable such functions. The runtime products chosen are driven by the technology choice made for service creation. For example, services created as Web services in Java might run in a J2EE container product, while services created using Microsoft®.NET would run in a .NET application server. Ultimately the

deployment environment is largely based on the technologies chosen during service development.

Server infrastructure

WebSphere Application Server is the base deployment component within IBM services enablement offerings. Developers can create basic Business Application Services using Enterprise Java Beans (EJBs) and Plain Old Java Objects (POJOs) and expose these services via standard Web services methods. In addition to the J2EE 1.4 standards, WebSphere Application Server supports a variety of Web services standards including:

- ▶ Enterprise Web Services (JSR 109)
- ▶ Java API for XML-based RPC (JAX-RPC)
- ▶ WS-Interoperability Basic Profile (WS-I)
- ▶ Web Services Security (WS-Security)
- ▶ Web Services Transactions (WS-TX)

WebSphere Application Server is available in several versions that support various uses. The most basic flavor is the Community Edition, which is based on Apache Geronimo, while the most robust version, WebSphere Application Server Network Deployment, includes enterprise-class features such as high availability, workload balancing, and dynamic failover.

In addition to the basic service hosting role, WebSphere Application Server also provides the underlying layer for both WebSphere Process Server and WebSphere ESB. The tight integration between all these products provides a number of benefits including the possibility of a single point of administration in a mixed product cell. This approach allows deployment processes and procedures to share common elements for all such platforms. The runtime deployment uses the same approach for installation, configuration, clustering, and administration.

Enterprise service bus

ITSOMart has chosen to use WebSphere ESB to implement their solution. They have evaluated their needs, including the connectivity issues of connecting to existing services they plan to use and feel that WebSphere ESB provides all the functionality they require. They anticipate that in the future, though their solution may grow, their connectivity issues will remain fairly simple.

WebSphere ESB provides the ability to abstract transport protocols between two services. WebSphere ESB also includes service bus mediations that can resolve message syntax differences between the service provider and requestor. The product also includes support for content-based routing and message logging for audit purposes.

4.6.2 Modeling and design

ITSOMart has decided to use Rational Software Architect for their application design activities and for the limited J2EE development their solution will require.

Rational Software Architect targets software architects to help modeling application software components and the relationships between components. Rational Software Architect uses the standards-based Unified Modeling Language (UML) for use case analysis, class diagrams, state diagrams, and sequence diagrams.

Rational Software Architect also supports software patterns to encourage software re-use. There is a set of prepopulated patterns, and users may create their own custom patterns as well.

Rational Software Architect also includes structural analysis tools that help software architects navigate and visualize the complex code base of modern applications. The structural analysis also compares the existing code against known software patterns to allow architects to more quickly understand the code.

In addition to the modeling capabilities, Rational Software Architect provides a comprehensive set of tools for developers to create, test, and deploy Java applications, including EJBs, Web, XML, and database applications. These are the same tools found in Rational Application Developer, the Rational product that targets application developers.

For full details on Rational Software Architect see the product page:

<http://www.ibm.com/software/awdtools/architect/swarchitect/index.html>

4.6.3 Development and assembly

Developing a solution consists of two distinct tasks: development and assembly.

- ▶ *Development* refers to the actual creation of services for the solution. This may involve building Web services to implement core Business Application Services or perhaps creating service wrappers around inherited applications. In most cases these tasks involve using an integrated development environment to create and test basic services.
- ▶ *Assembly* involves composing services together in order to deliver the needed functionality. Assembly may involve message mediation between two different systems or protocol transformation between systems or perhaps using application adapters to reach backend systems.

The major difference between development and assembly is that development focuses on creating core business functionality without worrying about who will access that functionality and what methods will be used to access the services.

Assembly focuses only on creating the connections between various services. Service implementations are considered opaque, and application assembly generally does not require detailed knowledge about the specifics of how a particular service has been implemented. By creating this separation between service creation and application assembly, development teams are free to focus on building the needed functionality without being distracted by issues related to integration. This creates a much more efficient programming model when compared to traditional application development where brittle interdependencies between various application components meant that changes in one system propagated throughout the entire system.

Development

ITSOMart will have some limited J2EE development required to implement their solution. Since they have chosen to use Rational Software Architect for their modeling activities, they will also use the extensive J2EE development capabilities to create the elements of their solution that require J2EE or Web services development.

Assembly

Service assembly involves orchestrating the flow of messages between services. Services are created as standalone components that may participate with any other components in a composite application. Service assembly often involves creating message mediation flows that transform message data between systems with different message formats or routing service requests based on the request content.

WebSphere Integration Developer is the integration tool that creates assemblies for both WebSphere ESB and WebSphere Process Server. WebSphere Integration Developer supports a variety of integration standards including Business Process Execution Language (BPEL) and BPEL extensions that allow human tasks, and SCA. Also included are unit test environments that allow integration developers to test the integration flow without involving the actual services being integrated.

ITSOMart has chosen to use WebSphere ESB as the ESB in their solution, making WebSphere Integration Developer the natural choice. Many of the J2EE and Web services features available in Rational Software Architect are also available in WebSphere Integration Developer.

Development and assembly tool coexistence

Rational Software Architect and WebSphere Integration Developer are built on a common foundation. This means that they contain common features and have the same look and feel. It also means that they can coexist in the same installation. WebSphere Integration Developer may be installed into an existing

Rational Application Developer or Rational Software Architect installation. This allows features from both IDEs to be available from a single, integrated development environment.

4.6.4 Monitoring and management

The highly distributed nature of SOA solutions can present difficulties for monitoring and management. Diagnosing application performance issues across a range of distributed systems is difficult without end-to-end monitoring tools. The IBM offering in this space includes the Tivoli Enterprise Monitoring package as well as IBM Tivoli Composite Application Manager. In particular, for Web-services based solutions, IBM Tivoli Composite Application Manager for SOA provides a detailed view of Web services performance within the Tivoli Enterprise Portal and allows basic message filtering and alerts based on performance thresholds.

For more information about IBM Tivoli Composite Application Manager for SOA see Chapter 12, “Service monitoring and management with IBM Tivoli Composite Application Manager SOA” on page 555.

4.7 Installation considerations

Physical topology is an important issue to consider when planning the solution. There must be an adequate number of environments and resources to support on-going development as well as product systems.

The examples in this section illustrate a single line of development for the ITSOMart application. More environments and resources would be needed if other applications were being developed. The physical topology also assumes that the development and deployment environment is based on WebSphere Integration Developer and WebSphere ESB. Physical topology and resources needs would be different for another service bus approach.

Differences between development and production must also be considered when planning the environments. Typically, the mediation modules for WebSphere ESB are created in a workstation development environment and deployed to a server environment. Differences in operating systems, physical resources, and system tuning may yield very different performance results between development and production environments.

Figure 4-17 illustrates a typical WebSphere ESB deployment environment, including development environment, application servers, back-end servers, the service bus runtime, management and monitoring, and client applications.

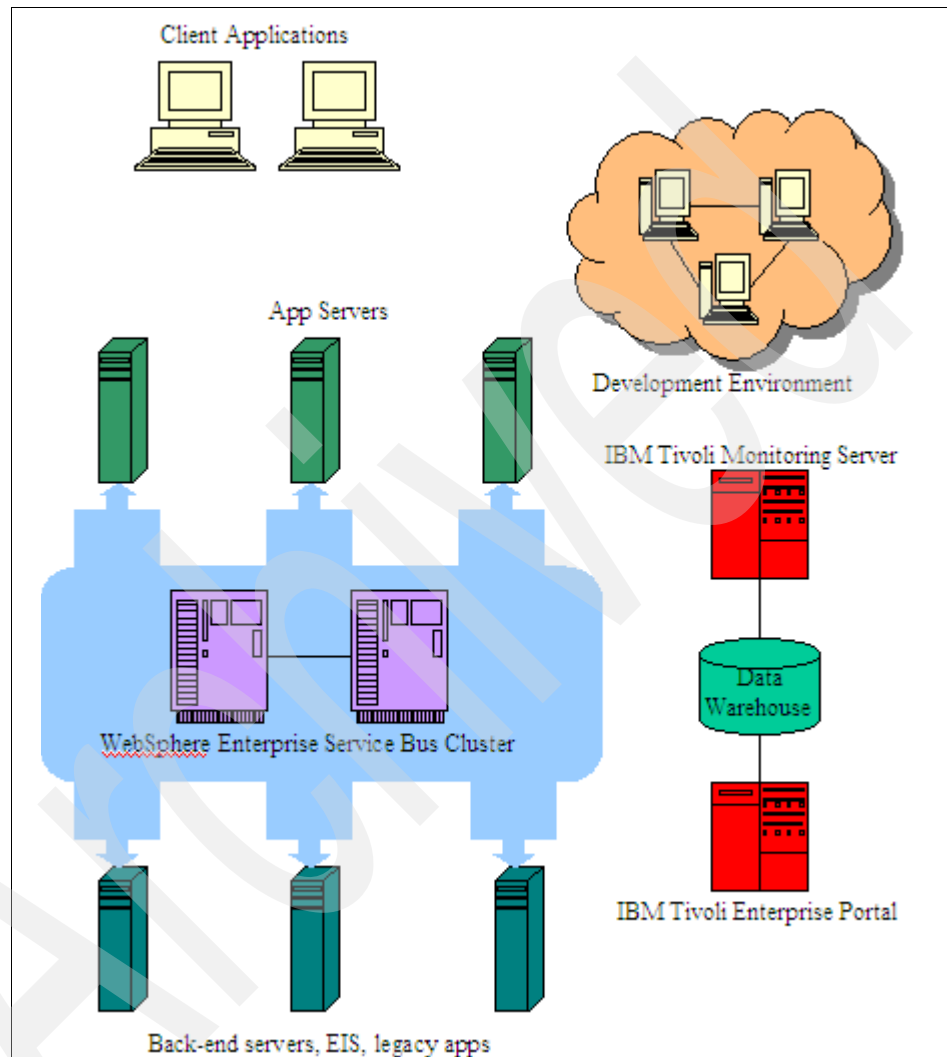


Figure 4-17 WebSphere ESB deployment architecture

In this environment, the development of applications, services, and integration processes occurs independently. Unit testing is performed with the local unit test environment. End user applications are deployed either to client workstations or to application servers in the case of Web applications. These applications then connect to back-end services or inherited applications via a clustered

WebSphere ESB managed by WebSphere Application Server Network Deployment. All service traffic is monitored by IBM Tivoli Monitoring Server, which stores the information in a data warehouse. The IBM Tivoli Enterprise Portal Server then makes this information available to system administrators for system health monitoring, fault detection, and service level monitoring.

A copy of this production environment would be used for testing so that changes can be validated in a production-like environment without affecting the running production environment.

4.7.1 Rational Software Architect, WebSphere Integration Developer

WebSphere Integration Developer is based on the Rational Software Development Platform, which is shared by several IBM products. Note that the Rational Software Development Platform is installed only once when the first product is installed. Subsequent products use the common user interface and add product-specific functionality that is provided by the plug-ins. WebSphere Integration Developer is available for Windows® and Linux® operating systems.

If your current environment has any existing Rational Software Development Platform products installed, the installation of WebSphere Integration Developer will integrate into the existing Rational Software Development Platform.

WebSphere Integration Developer V6.0.1 is based on Rational Software Development Platform V6.0.1 and is only compatible with other products that are based on this level. If you have a product that uses an earlier version of Rational Software Development Platform, you will be required to upgrade that product or uninstall it so that WebSphere Integration Developer V6.0.1 can be installed.

WebSphere Integration Developer can coexist with WebSphere Studio Application Developer Integration Edition V5.1.1 and previous releases. WebSphere Integration Developer V6.0.1 cannot coexist with WebSphere Integration Developer V6.0.

System requirements

WebSphere Integration Developer, Rational Software Architect, and Rational Application Developer are all capable of running on Linux or Microsoft Windows.

For information regarding the hardware and software requirements for each product see:

- ▶ Rational Software Architect system requirements:
<http://www.ibm.com/software/awdtools/architect/swarchitect/sysreq/index.html>
- ▶ Rational Application Developer system requirements:
<http://www-306.ibm.com/software/awdtools/developer/application/index.html>
- ▶ WebSphere Integration Developer system requirements:
<http://www-306.ibm.com/software/integration/wid/>

Installation steps

Below is the recommended order in which to install Rational Software Architect and WebSphere Integration Developer.

Tip: The test server environment uses profiles created under the WebSphere Integration Developer installation directories. If you tend to use long names for mediation modules, you can run into a problem with the 256 URI length limitation in Windows when you deploy to the test environment. One way to avoid this is to choose a short path name for the installation. Another is to create new profiles in a different, shorter directory for use by the test environment (see “Creating a new server in the test environment” on page 640).

The order is:

1. Install Rational Software Architect 6.0.1.
2. Install the Rational Software Architect fixpack 6.0.1.1.
Installation instructions for all Rational Software Architect updates can be found at:
http://www3.software.ibm.com/ibmdl/pub/software/rationalsdp/rsa/60/install_instruction/
3. Install WebSphere Integration Developer 6.0.1. Install into the same directory to which you installed Rational Software Architect so that the WebSphere Integration Developer tools will be available in the same Eclipse workbench.
The steps taken to install WebSphere Integration Developer on a Windows system in our test environment can be seen in “Installing WebSphere Integration Developer” on page 658.

Note: Since the development of the sample in this book, WebSphere Integration Developer 6.0.1.1 became available. The instructions above reflect the product levels used to illustrate and test the samples in this book.

Always check to ensure that the current version of WebSphere Integration Developer is compatible with the installation of the Rational Software Development Platform. You will find a compatibility matrix at:

<http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.install.doc/topics/cinsdp.html>

4. Install the WebSphere Integration Developer interim fixes. Installation instructions for all WebSphere Integration Developer updates can be found at:

http://www3.software.ibm.com/ibmdl/pub/software/websphere/studiotools/wid/install_instructions/

The following interim fixes were available during the development of the ITSOMart sample:

- 6.0.1 interim fix 001
- 6.0.1 interim fix 002
- 6.0.1 interim fix 003

We found that we could also install the WebSphere MQ Explorer as a plug-in to the Rational Software Architect shell. This worked for us. However, you should note that this is an unsupported configuration. For more information see “Installing WebSphere MQ Explorer as a plug-in” on page 654.

4.7.2 WebSphere ESB

WebSphere ESB should be installed separately from the development environment on different physical servers. This separates development from testing and production so that changes in one environment do not affect another. In addition, server tuning may differ between a development environment (for example, hot-deploying code changes) and a testing or production environment.

A WebSphere ESB installation may coexist with any version of WebSphere Application Server, WebSphere Process Server, or WBI Server Foundation Version 5.1.

WebSphere ESB supports a variety of operating systems and hardware architectures. For information regarding supported hardware/software please see WebSphere ESB system requirements at:

<http://www-306.ibm.com/software/integration/wsesb/sysreqs/>

During installation planning, consider the following:

- ▶ Installing WebSphere ESB to a physical disk separate from the disk used by the operating system can help prevent disk contention during mediation execution.
- ▶ WebSphere ESB should not be installed to a directory containing spaces in the directory name. This occasionally causes unexpected behavior.
- ▶ On Microsoft Windows operating systems, ensure that the WebSphere ESB profile is created using a shallow directory structure (that is, C:\PF\ESB instead of C:\IBM\WebSphere\EnterpriseServiceBus\profiles\ESB). Long directory structures can cause you to encounter the Windows limit on path names longer than 256 characters during deployment and mediation execution.

Installation options

A WebSphere ESB server is an application server that has been augmented to run mediation modules. It is built on WebSphere Application Server Network Deployment V6.0.2.3. An installation consists of installing Network Deployment, installing WebSphere ESB in the same directories, and then creating one or more profiles that define the application server environment. The profiles can be done as part of the installation or manually afterward.

When you install WebSphere ESB there are two possibilities for the installation:

- ▶ If you have an existing WebSphere Application Server Network Deployment installation, you can choose to extend this installation with WebSphere ESB (updating Network Deployment to V6.0.2.3 in the process) or you can create a new, separate set of install libraries.

If you choose to extend an existing environment, you can create a new application server as part of the install, create no server during the install, or you can choose to augment an existing application server for WebSphere ESB.
- ▶ If there is no existing WebSphere ESB or Network Deployment environment, the installation process will install Network Deployment first, then WebSphere ESB.

You can choose to create a stand-alone WebSphere ESB server as part of the install, or you can elect to create the application server environment after the install. The following are the options you will see during install:

- Complete install

A complete install includes all the optional features (samples and Javadocs). A stand-alone application server is created using the default values, and the First Steps wizard is launched at the end of the installation process.

- Custom install

No application server is created and the installation of the optional features can be disabled. The Profile creation wizard is launched at the end of the installation to allow you to create the runtime environment.

You can install using the installation wizard or you can perform a silent installation. Sample response files for silent installation are contained on the installation CD.

Topologies for a standalone server

A stand-alone server profile has its own administrative console and all of the sample applications (if you installed the sample applications gallery feature). You can have multiple stand-alone servers on a single machine, each managed independently.

The following are possibilities for installing and configuring stand-alone servers on a single machine:

- ▶ Single-machine installation with one stand-alone server

The simplest scenario is to install WebSphere ESB on a single machine with a stand-alone server profile. Each stand-alone application server profile includes an application server called server1. You can do this by choosing the Complete option during installation.

- ▶ Single-machine installation with multiple stand-alone servers

After installing the WebSphere ESB system files once on a machine, you can use the Profile Creation wizard to create additional stand-alone server profiles on the same machine. This topology lets each profile have unique modules and applications, configuration settings, data, and log files, while sharing the same set of system files. There is no high availability or load balancing among machines. Each server would have its own set of applications and serve its own purpose.

- Single-machine, with multiple installations

Multiple installations of WebSphere ESB on one system are also possible, each with its own set of core files and profiles. This might come in handy when testing new services.

Installing WebSphere ESB creates the set of system files. The Profile Creation wizard is then used, either as part of the installation or started later manually, to create the runtime environment.

Topologies for clustering

WebSphere Application Server Network Deployment includes the ability to cluster application servers for load balancing, failover, and central management. Because WebSphere ESB is built on Network Deployment, the issues for setting up a clustered environment are essentially the same.

Designing a system for workload management and failover can be complex. If you plan to do this, we recommend that you use the following documents to thoroughly understand how clustering works and how to design a clustered environment that includes the service integration bus:

- *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688
- *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392

For examples of clustering topologies specifically geared toward load balancing and failover for mediation modules, see 11.12, “Network Deployment and clustering topologies” on page 531.

4.7.3 WebSphere Application Server

Services used by WebSphere ESB can run in many different environments. These include WebSphere Application Server, other J2EE application servers, .NET servers, mainframe systems, and any number of other systems capable of using the communications protocols understood by WebSphere ESB. Because the mediation hosting layer utilizes WebSphere Application Server, services may be deployed to the same application server that is hosting the mediation modules. We do not recommend this for production, as these two runtimes generally require different tuning parameters, but may be useful in test environments.

4.7.4 ITCAM for SOA

The monitoring and management tools of choice for composite applications is IBM Tivoli Enterprise Monitoring and IBM Tivoli Composite Application Manager. These tools allow system administrators to observe the flow of information through an entire composite application end-to-end instead of observing performance at a single application tier at a time. In particular, ITCAM for SOA is suited for a SOA solution. ITCAM for SOA is a participant in the Tivoli Enterprise Monitoring framework and, as such, requires several supporting components to be installed and configured (see 12.2, “IBM Tivoli Enterprise Monitoring framework” on page 560).

In particular, you will need to install and configure the following:

- ▶ IBM DB2®
- ▶ Tivoli Enterprise Monitoring Server
- ▶ Tivoli Enterprise Portal Server
- ▶ Tivoli Enterprise Portal
- ▶ ITCAM for SOA application support
- ▶ ITCAM for SOA monitoring agents

Installation planning

It is important to have a good understanding of the managed environment and the capability of the product. You will also need to design a topology for the installation, including which servers will be used to host each function (monitoring server, portal server, and so on).

In a production environment, the IBM Tivoli Enterprise Monitoring Server should be installed on a set of servers dedicated for application monitoring and management. The minimum recommended production system consists of three separate servers: a monitoring service, a database server for storing the monitoring data, and an enterprise portal server for viewing and analyzing the collected data. If there is a requirement to install both the monitoring node and the application server node on the same host (such as for demonstration or proof-of-concept purposes), extra consideration must be taken during the install process. Refer to the installation instructions in the online documentation for ITCAM for SOA at:

<http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?toc=/com.ibm.itcamsoa.doc/toc.xml>

Monitoring agents for different application domains such as operating system, application server, Web services, and so on must be installed and configured to report data from the instrumented layers. The presence of these monitoring agents must be included during server planning, and appropriate network access

must be granted in order for the monitoring agents to communicate with the enterprise monitoring server.

In addition to reviewing the product documentation, we recommend that you review the IBM Redbook *IBM Tivoli Composite Application Manager V6.0 Family: Installation, Configuration, and Basic Usage*, SG24-7151.

Installation overview

This is the overall implementation procedure for ITCAM for SOA:

1. IBM Tivoli Enterprise Monitoring Server requires a database to support the historical data warehousing functions. The default database is IBM DB2 UDB V8.2 Fixpack 10. DB2 installation requires various system users depending on the operating system. IBM DB2 needs to be installed on the monitoring server node prior to installing the IBM Tivoli Monitoring Server.
2. Installation of IBM Tivoli Monitoring V6.1 on the node that will act as the monitoring server must be performed before any other ITCAM for SOA component.

The following components will be installed on the server:

- Tivoli Enterprise Monitoring Server (TEMS)
- Tivoli Enterprise Portal Server (TEPS)
- Tivoli Enterprise Portal (TEP)

In a fully distributed production environment, these components would most likely reside on separate nodes.

Tivoli Monitoring installation information can be found in the *IBM Tivoli Monitoring Installation and Setup Guide*, GC32-9407.

3. Install the application support component for ITCAM for SOA on your Tivoli Enterprise Monitoring Server, Tivoli Enterprise Portal Server, and Tivoli Enterprise Portal systems.

Application support augments the enterprise monitoring server with additional capabilities necessary to receive monitoring data from the ITCAM for SOA Monitoring Agent. The application support installation will enable the Web services monitor and management functions in the Tivoli Enterprise Portal.

4. Install and configure the monitoring agents of ITCAM for SOA on the server nodes that you will monitor, for example, on the WebSphere ESB server. If there were other servers hosting Web services, the monitoring agent would need to be installed on those servers as well.
5. Metrics collected by the ITCAM for SOA data collector (DC) can be stored in the Tivoli Data Warehouse. The Data Warehouse Proxy must be configured on the Tivoli Enterprise Monitoring Server in order to enable historical data collection for ITCAM for SOA.

6. The Web Services Navigator allows offline analysis of services flows and patterns. You can install Web Services Navigator into an existing Eclipse environment or into a new environment.

The steps taken to install the ITCAM environment for the ITSOMart solution are shown in Table 4-4.

Table 4-4 ITCAM installation steps for ITSOMart

Step	References
Install IBM DB2 UDB on the monitoring server node.	"IBM DB2 Universal Database installation" on page 672
Install IBM Tivoli Monitoring, including: <ul style="list-style-type: none">▶ Tivoli Enterprise Monitoring Agent Framework▶ Tivoli Enterprise Monitoring Server▶ Tivoli Enterprise Portal Server▶ Tivoli Enterprise Portal Desktop Client	"IBM Tivoli Monitoring installation" on page 678
Install IBM Tivoli Composite Application Manager for SOA installation, including: <ul style="list-style-type: none">▶ ITCAM for SOA Application Support installation▶ ITCAM for SOA Monitoring Agent installation and configuration	"ITCAM for SOA Application Support installation" on page 687 "ITCAM for SOA Monitoring Agent installation and configuration" on page 691
Web services navigator.	"Web Services Navigator installation" on page 698
Test and verify the installation.	"Verify the installation" on page 699

For information about ITCAM for SOA see:

- ▶ IBM Tivoli Composite Application Manager for SOA home page:
<http://www-306.ibm.com/software/tivoli/products/composite-application-mgr-soa/>
- ▶ Information about system requirements and support can be found in the data sheet at:
<ftp://ftp.software.ibm.com/software/tivoli/datasheets/ds-tcam-for-soa.pdf>

For a brief look at monitoring and management with IBM Tivoli Composite Application Manager for SOA see Chapter 12, "Service monitoring and management with IBM Tivoli Composite Application Manager SOA" on page 555.

4.7.5 Environments for testing and production

Discussions about general testing methodology and approach are beyond the scope of this book. However, as a general rule, there should be at least one testing environment that is roughly equivalent to the production system for the purposes of test. This not only helps ensure that the services and mediations will operate in the prescribed manner during production, this approach also helps validate that the production system will be capable of tolerating a product-sized workload. Although development systems include a built-in unit test environment, the integrated test environment should consist of complete deployments of WebSphere ESB, preferably with clustering if clusters are going to be used in production. Finally, care should be taken to properly isolate the testing system from production (using separate, isolated subnets for example) to avoid conflicts and confusion due to the existence of both environments. For example, care should be taken when moving a release from testing to production to ensure that the production mediations do not inadvertently refer to services provided by test systems.

4.8 Security considerations

Enterprise service buses simplify security management by centralizing the point of access for service requests. ESBs can also perform security functions such as digital signatures, message encryption/decryption, and authentication and authorization.

Complex business applications have diverse security requirements that must all be obeyed. Implementing a service bus in the integration environment makes security governance easier by providing a central point of administration. By distributing security policies through the service bus, system administrators can ensure that security standards and procedures are being observed throughout the enterprise without individually securing each application.

4.8.1 Securing communication using WebSphere ESB

WebSphere ESB provides a rich set of security functions because the foundation of WebSphere ESB is the WebSphere Application Server. All of the security mechanisms available in the WebSphere Application Server such as digital signatures and encryption are also available in WebSphere ESB. The service bus can also use authentication features such as Lightweight Third Party Authentication (LTPA) and use Lightweight Directory Access Protocol (LDAP) as its user registry. Also, all of the security products designed to work in conjunction with WebSphere Application Server such as Tivoli Access Manager will also work with WebSphere ESB. By leveraging the WebSphere Application Server

platform, the service bus provides a broad set of standard security features and integrates with a wide array of complementary security products.

4.8.2 Messaging security

Securing WebSphere ESB consists of a complex set of authentication and authorization checks to ensure that protected resources are being accessed by clients with proper credentials. The integrity of communications through the service bus are protected by four separate factors:

- ▶ Authentication and authorization of clients (users or mediations) that initiate connections to the service bus or try to utilize bus resources
- ▶ Ensuring the security and integrity of message transports between the client and the bus and between messaging engines used by the bus.
- ▶ Authentication of messaging engines that are joining the bus
- ▶ Authentication of users who are trying to access the message store

The messaging security model in WebSphere ESB provides client authentication and authorization as well as ensuring message privacy and integrity. When the service bus has security enabled, clients that wish to connect to the bus must supply a user name and a password. These credentials are then checked against the user registry (as configured for the underlying WebSphere Application Server) for authentication.

If the authentication check is successful, then authorization checks are performed to ensure that the client has sufficient access to the bus, as well as any specific resources such as destinations on the bus or foreign buses. A user could be authorized to access the bus (in which case the connection would succeed) but not authorized to access the destination, in which case the operation would fail.

Message bus topics may also be secured in addition to destinations that contain the topic. This is known as topic access checking. When topic access checking is enabled, a further authorization check against the topic is performed once the destination authorization check is complete and, if unsuccessful, the connection is denied.

4.8.3 Transport security using HTTPS

In addition to securing access to protected service bus resources, WebSphere ESB leverages the transport security features within WebSphere Application Server to protect the confidentiality and integrity of in-flight messages. Administrators may configure messaging engines to accept or require SSL when connecting to them by disabling the non-SSL transports. By selectively enabling

and disabling security on the various transports, messages can be protected while en-route over non-trusted paths.

The protocol translation features of WebSphere ESB also extend to security protocols. For example, external services may be offered with security enabled, while internal services operating on secure networks are provided without security. By creating secure versions of internal services and using messaging security, internal services may be externalized without significant re-implementation. As a best practice, however, a security review is always advisable before providing a new internal service to potentially untrusted clients.

4.9 Scalability and performance considerations

WebSphere ESB supports all of the application server clustering features available in WebSphere Application Server Network Deployment. Clustering can provide both workload management and high-availability options.

Clustering WebSphere ESB allows system administrators to distribute the workload of mediations across the cluster, providing better overall throughput.

High availability features help eliminate single points of failure by allowing requests to be processed by multiple servers. WebSphere ESB allows in-flight message processing to fail over to alternate servers if a server stops functioning during message mediation. The messaging engines used by the service bus may also be clustered so that alternate servers in a cluster may process messages if the preferred server ceases to function.

In practical terms, due to the importance of the enterprise service bus to the applications that depend on the services it provides, WebSphere ESB should always be deployed in a clustered configuration to provide workload balancing and fault tolerance. Depending on an organization's down time and data protection needs, automatic failover may or may not be a requirement. If an organization needs very high availability for the service bus, the hosted services, and other non-WebSphere resources, the components of WebSphere ESB may be integrated into other high-availability frameworks such as HACMP. Also, architects may need to consider the need for disaster recovery scenarios that involve alternate geographic sites, standby servers, and network connections.

Examples of clustering for WebSphere ESB can be seen in 11.12, "Network Deployment and clustering topologies" on page 531.

For more information about scalability and high availability for WebSphere Application Server Network Deployment see:

- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- ▶ *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688

4.10 System management and monitoring

The diverse set of environments that comprise a modern, services-oriented integration solution present a great deal of difficulty in terms of management and monitoring. The elevated importance of such applications leads to greater demands on application availability and make resolving performance problems even more critical. However, when such a system is performing poorly, determining the problem area or resource can be very difficult. Is the network behaving properly? Is the services layer responding in adequate time? Is the enterprise service bus performing mediations quickly enough? These are all questions that administrators must answer quickly when faced with performance issues in production. Traditional monitoring tools may not be sufficient to take on this challenge, as they are typically focused on observing individual tiers of the application rather than viewing transactions from end-to-end. They also provide little to no ability to correlate events in one tier with events in another tier, which makes diagnosing issues more difficult.

4.10.1 IBM Tivoli Management Framework

The challenges presented by modern composite applications require a new approach to monitoring such applications. The IBM Tivoli Management Framework provides an approach so that system administrators can gain a complete view of the enterprise from physical server and operating system health to high-level application performance and health.

IBM Tivoli Composite Application Manager integrates into the Tivoli Management Framework to bring increased awareness to application monitoring and management. Tivoli Composite Application Manager works as a series of monitoring agents that report application status to IBM Tivoli Enterprise Monitoring Server. The monitoring server stores performance and event data in a data warehouse, which is then used by the Tivoli Enterprise Portal Server to provide analysis. This approach makes it possible to monitor composite applications end-to-end as a complete environment rather than a collection of disjoint components.

4.10.2 IBM Tivoli Composite Application Manager for SOA

IBM Tivoli Composite Application Manager for SOA is the component of the IBM Tivoli Composite Application Manager family that provides monitoring and management of Web services. IBM Tivoli Composite Application Manager for SOA can:

- ▶ Perform automated service mediation.
- ▶ Proactively recognize and quickly isolate Web service performance problems.
- ▶ Verify that Web services are available and performing to specification.
- ▶ Alert you when Web service performance is degraded.
- ▶ Report results against committed service levels.
- ▶ Visualize entire Web service flows, end-to-end, as they cross the enterprise.
- ▶ Pinpoint the source of service bottlenecks.

Also included is a tool called Web Services Navigator, an Eclipse framework-based tool that allows analysis of Web services usage and gives insight into service flows and invocation patterns.

Web services has become the de facto standard for implementing service-oriented solutions. As organizations adopt an increasingly services-oriented approach to service integration, IBM Tivoli Composite Application Manager for SOA can help manage the growth of Web services. Web services support includes a number of different platforms including WebSphere, Microsoft .NET, and BEA WebLogic.

WebSphere ESB is a fully supported environment that may be monitored and managed by IBM Tivoli Composite Application Manager for SOA. System administrators can view individual service performance as well as mediation performance through the Tivoli Enterprise Portal client. The client can also allow the definition of filters that mediate service requests and responses, discarding or modifying in-flight messages based on established rules.

4.11 Where to find the implementation details

This section highlights where to go next to find the implementation details for the ITSOMart working example, which is an instance of the Service Connectivity scenario. The ITSOMart example implementation chapters illustrate how to model, assemble, deploy, and manage.

- ▶ Model:
 - Chapter 5, “Model with Rational Software Architect” on page 125
- ▶ Assemble:
 - Chapter 6, “Assemble with WebSphere Integration Developer” on page 211
 - Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263
 - Chapter 8, “Building the CRM mediation” on page 347
 - Chapter 9, “Building the Register Shipping mediation” on page 417
 - Chapter 10, “Building Log Registration mediation” on page 449
- ▶ Deploy:
 - Chapter 11., “Deploy with WebSphere ESB” on page 479
- ▶ Manage:
 - Chapter 12, “Service monitoring and management with IBM Tivoli Composite Application Manager SOA” on page 555

4.12 Summary

The Service Connectivity scenario is an important stage in the SOA adoption road map. It allows IT organizations to experiment with SOA without requiring enterprise-wide organization changes necessary to support the efforts of a new IT strategy. Incremental adoption of SOA also delivers value quickly by reducing application construction costs and delivering more agile line of business applications. Finally, SOA benefits from application assembly tools that can make the job of creating new applications and modifying existing applications much less complicated.

When planning an SOA solution, there are many elements to consider. The overall application architecture must meet the current demands of business while remaining flexible and able to respond to changing business needs over time. The product selection for both development and runtime services must deliver on the promise of simpler application creation. Finally, there must be sufficient

management and monitoring tools that can assist system administrators in proactively preventing application downtime and removing performance bottlenecks.

The IBM Soa Foundation Reference Architecture provides a detailed architectural map for developing SOA applications. The entire application integration environment is divided into separate service types, each responsible for a specific functional area. By applying the Soa Foundation Reference Architecture to an SOA solution, architects can build on a robust and flexible architecture capable of meeting current application demands as well as being agile enough to meet future business needs through IT services.

IBM offers a broad range of products serving the needs of SOA design, development, deployment, and management. Development products such as Rational Application Developer, Rational Software Architect, and WebSphere Business Modeler help developers and architects design and develop services and application components. WebSphere Integration Developer allows integration developers to quickly and easily assemble software assets using the Service Component Architecture to create composite applications. WebSphere ESB and WebSphere Message Broker provide a robust runtime environment for mediating service requests across the enterprise, and DataPower SOA appliances can extend the reach of services and can be used to build high-performance, secure service transport infrastructures. Finally, IBM Tivoli Composite Application Manager and IBM Tivoli Enterprise Monitoring give administrators the ability to monitor and diagnose performance problems and monitor message flows and service invocation patterns end-to-end across the entire enterprise application infrastructure.

By combining best-of-breed products with proven architectural patterns, organizations can gain many of the benefits of a service-oriented architecture while still maintaining organizational structure and gradually changing the enterprise IT environment. The Service Connectivity scenario provides an ideal way to enter into SOA while preserving technology investments. The following chapters illustrate the steps to implement a Service Connectivity scenario using the ITSOMart scenario.

4.13 For more information

For information regarding the Enterprise Service Bus pattern see the following:

- ▶ *What Is an ESB, and Do You Really Need One?*
<http://www.computerworld.com/developmenttopics/development/webservices/story/0,10801,108478,00.html>
- ▶ *Simplify integration architectures with an Enterprise Service Bus*
<http://www-128.ibm.com/developerworks/webservices/library/ws-esbia/index.html>
- ▶ *Web services programming tips and tricks: Learn simple, practical Web services design patterns, Part 4*
<http://www-128.ibm.com/developerworks/webservices/library/ws-tip-altdesign4>
- ▶ *SOA programming model for implementing Web services, Part 4: An introduction to the IBM Enterprise Service Bus*
<http://www-128.ibm.com/developerworks/library/ws-soa-progmodel4/>

Model with Rational Software Architect

Modeling business processes helps to accurately capture business requirements and establish a closer alignment of IT delivery with business goals by using a common language. The product of choice for modeling J2EE applications and mediations is Rational Software Architect. This chapter discusses the features of Rational Software Architect and models the sample ITSOMart solution to illustrate the process of modeling a business solution.

This chapter includes the following topics:

- ▶ Introduction to Rational Software Architect
- ▶ Modeling the ITSOMart sample
- ▶ Tools used to model the application
- ▶ Solution requirements
- ▶ Domain analysis
- ▶ Architectural design
- ▶ Modeling business objects: Transform UML to XSD
- ▶ Modeling messaging resources: Transform UML to JACL

5.1 Introduction to Rational Software Architect

Rational Software Architect contains the features found in Rational Application Developer, in addition to features that appeal to software architects. Rational Software Architect is a design and development tool that leverages model-driven development with the Unified Modeling Language (UML) for creating well-architected applications and services.

5.1.1 Rational Unified Process guidance

You can access process guidance content and features directly in the Rational Software Development Platform to guide you and other team members in your software development project. A configuration of the Rational Unified Process® (RUP) platform is provided with topics on software development best practices, tool mentors, and other process-related information.

5.1.2 Model-driven development

Model-driven development (MDD) is the concept of using models as the basis of application development. One approach to model-driven development, called Model Driven Architecture (MDA), is in the process of being defined by the Object Management Group (OMG). MDA defines development using UML models at different levels of abstraction. Transformations are used to take a model at one level and transform it to a model at a different level. You can see the MDA standards at:

<http://www.omg.org/mda>

Rational Software Architect provides full support for MDD with UML 2 modeling, transformations, code generation from models, and patterns. As a part of the model-driven development support, Rational Software Architect also supports the current principles of MDA.

Unified Modeling Language 2.0 editor

Rational Software Architect includes a Modeling perspective (and editors that support the major UML 2.0 diagrams).

UML profile support

UML profiles allow you to customize the language for a particular domain or method. UML profiles introduce a set of stereotypes that extend existing elements of UML for use in a particular context. This technique is used in MDD to allow designers to model using application domain concepts.

Rational Software Architect ships with a set of UML profiles and also supports the creation of new profiles. One of the sample profiles is the Rational Unified Process Analysis profile that provides stereotypes for producing analysis models using the RUP approach.

Transformations

A transformation converts elements of a source model to elements of a target model. For example, the source and target model can be text files, code models, or UML models. When the source and target models are both UML models, the transformation usually converts the elements from one level of abstraction to another. You can apply a transformation to an entire model or a subset of model elements in a model to generate output such as code.

Rational Software Architect comes with the following set of transformations:

- ▶ Business tier transformations
 - UML → EJB Business Tier
- ▶ Integration tier transformations
 - UML → EJB Integration Tier
 - UML → EJB UML
- ▶ Presentation tier transformations
 - UML → IBM Portlet (JSF)
- ▶ RSA transformations
 - EMF Deployment → UML2
 - UML → EMF Deployment
 - UML → C++
 - UML → EJB
 - UML → Java
 - UML → XSD

Additional transformations may be available on the Web.

Patterns

Rational design patterns capture frequently used or complex structures and processes for reuse. They are used to integrate repeatable software design solutions into UML 2.0 models. Rational patterns are a type of transform.

Rational Software Architect provides the tools needed to create design patterns. When developers recognize repeatable structures or processes they can create patterns from them, allowing others to use these designs.

Patterns are stored in an RAS repository as a unique type of reusable asset in the form of a plug-in. Users can browse the repository for useful patterns as they model systems. The pattern user relies on the pattern documentation for information about selecting and applying a pattern. Depending on the pattern design, the pattern applier has the flexibility to apply all or only part of a pattern, as needed.

A set of sample patterns is supplied with Rational Software Architect and can be seen in the RAS perspective. Another set of patterns is included with the product and can be installed as an example.

5.1.3 Modeling

UML modeling provides a way of architecting systems in such a way they can be communicated to the stakeholders. UML models show a specific perspective of a system. Models are visual representations and as such are easily verified and communicated. Models start at the conceptual levels and can be refined down to detailed levels. Rational Software Architect supports UML Version 2 (UML 2).

Rational Software Architect supports modeling through all phases of software development:

1. Capturing system requirements

The first step in any system design is to determine the requirements for the solution. The IBM Rational RequisitePro® solution is a requirements and use case management tool. This tool can be integrated with Rational Software Architect, allowing you to map existing requirements to existing UML model elements. You can also create requirements from existing model elements, or create model elements from existing requirements definitions. The result of this development phase is one or more use case diagrams that describe how the system will be used.

2. Domain analysis

The next step is to build on the use case model by describing the high-level structure of the system based on the system domain requirements. An *analysis model* is used to capture this information. The analysis model consists of class diagrams that model the static structure of the system and of sequence diagrams that model the interactions between participants. The analysis model describes the logical structure of the system but does not define how it will be implemented.

3. Architectural design

The next step is to create a *design model* to define the architecture and implementation choices for the application. The design model builds on the analysis model by adding details about the system structure and how

implementation will occur. Classes that were identified in the analysis model are refined to include the implementation constructs.

You can use a variety of diagrams for this purpose, including sequence, state machine, component, and deployment diagrams. It is during this stage that you can apply proven design patterns and automated model-to-model transformations.

4. Implementation

Developers transition from design to implementation by using automated transformations to convert the model to code (such as Java, EJB, or C++) and by continuing to develop and deploy the application by using software and Web development, debugging, testing, and deployment capabilities.

The Modeling perspective is the primary workbench interface for working with models.

5.1.4 Asset-based development

As business needs are increasingly solved using more and more complex software solutions, it has become apparent that many of these solutions are created using the same integral key components structured in a similar manner. The idea that the same actions can be performed over and over in a variety of ways to create a solution has given rise to many of the fundamental concepts used in software development today, namely the use of patterns to structure solutions, and the reuse of assets within a context to build the key components of a solution.

Asset-based development embodies the idea of developing solutions by reusing defined and documented assets. These assets are made up of software artifacts that detail the requirements, design elements, development and testing process, and deployment requirements. Reusing these assets streamlines the development process and leverages previous investments.

The success of asset-based development within a department, organization, or on a more widespread basis, the community, lies in the ability to identify potential assets for reuse. Once a potential asset is identified, often through repeated experiences during development and deployment processes, it must be defined and made available for reuse by storing it in a central repository.

Potential consumers browse the repository for assets they can use. Documentation, an integral part of each asset, is key to the successful use effectiveness of an asset. The documentation details not only how the asset is to be used, but should give enough information that a potential consumer knows if the asset is appropriate for their use.

As a final step, feedback to the managers of the asset will help in tracking the effectiveness and value of the asset.

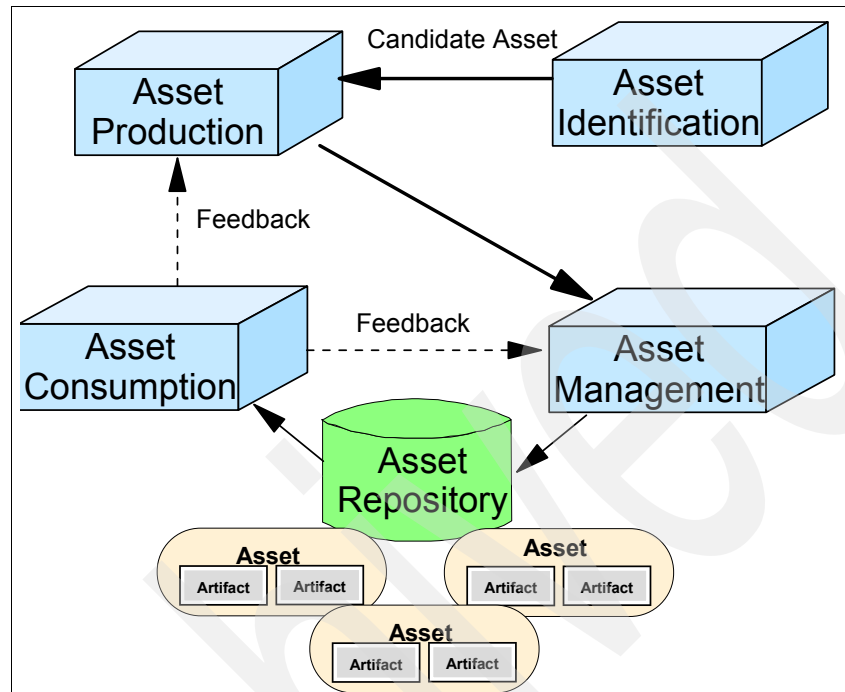


Figure 5-1 Asset-based development cycle

The following platforms deliver asset-based development:

- ▶ Rational Unified Process (RUP) has defined processes for producing and consuming assets.
- ▶ The IBM Software Development platform has incorporated asset-based development capabilities into its products. Rational Software Architect and Rational Software Modeler both contain tools for producing, consuming, and managing assets.

The Reusable Asset Specification (RAS) defines a standard way to package assets and describe their contents. Rational Software Architect provides a *reusable asset* (RAS) perspective for working with reusable assets.

5.2 Modeling the ITSOMart sample

This section describes how we modeled the ITSOMart sample application throughout the software development phases.

While comprehensive instruction on using Rational Software Architect for software analysis and design is out of the scope of this book, we use the modeling tools Rational Software Architect provides to clearly define the application domain and use case scenario we implement throughout the book, and focus primarily on the areas of the solution design that involve the following:

- ▶ Identifying and describing services
- ▶ Identifying and describing mediations for services
- ▶ Using reusable assets provided by Rational Software Architect to transform UML models into implementation and deployment artifacts targeted for WebSphere ESB

5.3 Tools used to model the application

This section identifies the primary tools and modeling structures used to design the ITSOMart solution, including the following:

- ▶ Modeling perspective
- ▶ UML projects
- ▶ UML models
- ▶ UML diagrams

5.3.1 Modeling perspective

The primary perspective used in modeling is the Modeling perspective (Figure 5-2).

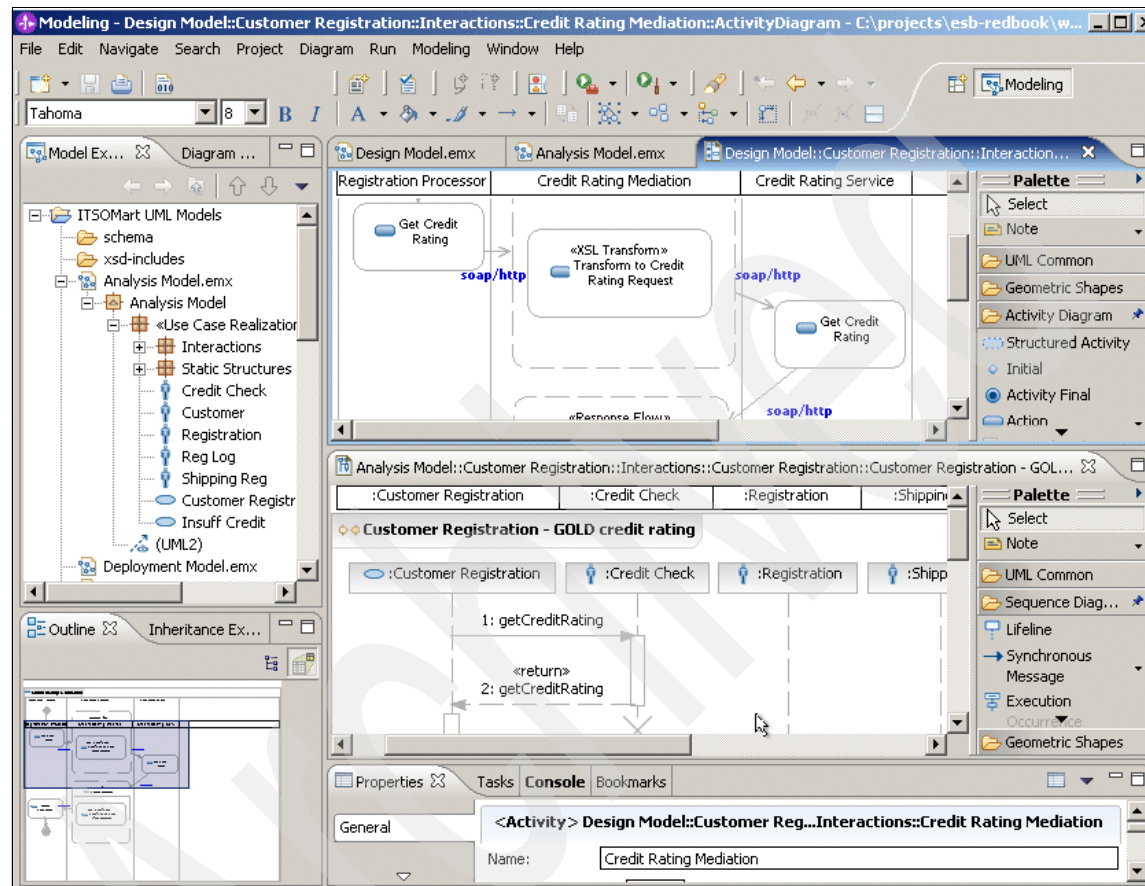


Figure 5-2 Modeling perspective

The modeling perspective is the default perspective when you open a workspace in Rational Software Architect. If you do not have this perspective open, you can open it by selecting **Window** → **Open Perspective** → **Other** → **Modeling**.

Among the views found in the Modeling perspective are:

- ▶ The Model Explorer view, which allows you to navigate your UML models in a tree structure.
- ▶ The main editor view displays models and diagrams in their respective editors.

- ▶ The Properties view shows the individual properties for the model, diagram, or UML element currently selected in the editor or Model Explorer view.
- ▶ The Outline view (Figure 5-3) allows you to navigate through large diagrams by moving a selected area box around with the mouse.

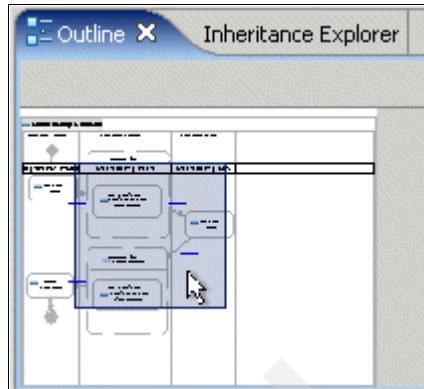


Figure 5-3 Outline view

- ▶ The Diagram Navigator view (Figure 5-4) gives you a view of only the diagrams in your projects.

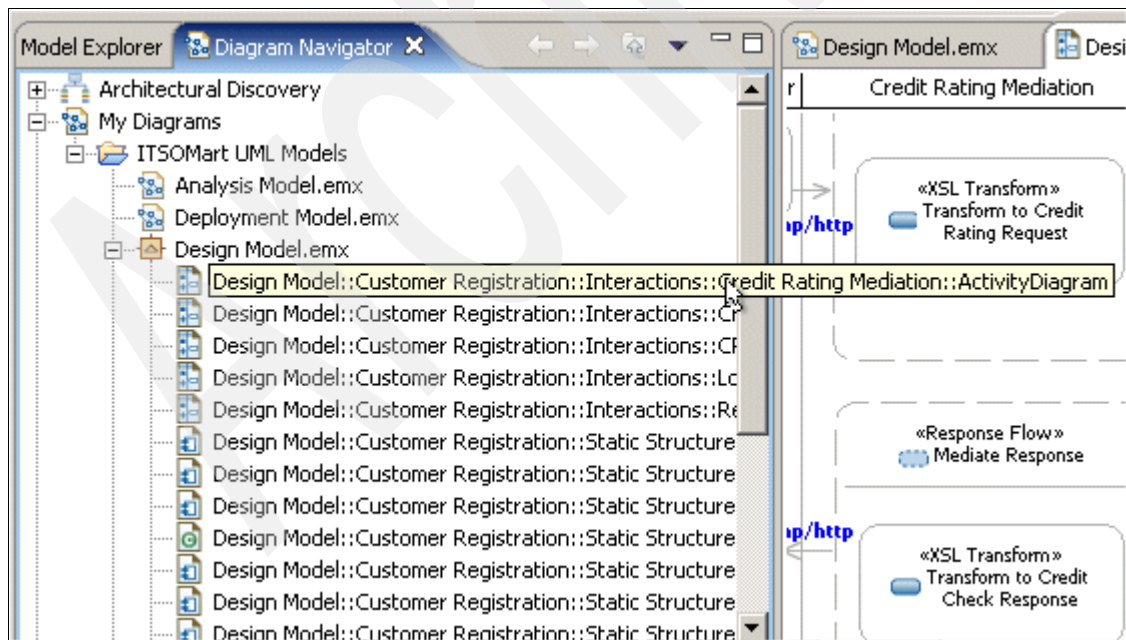


Figure 5-4 Diagram Navigator view

5.3.2 UML projects

You can create a UML model in any type of project. For example, you can place a UML model that contains a class diagram in a Java project to keep your model and Java code together. For the ITSOMart application, we created a separate UML project to contain all of the UML models used to describe the application.

To create a UML project:

1. Right-click anywhere in the Model Explorer view and select **New** → **Project** → **UML Project**.
2. In the UML Modeling project window (Figure 5-5) specify a name for your project and click **Next**.

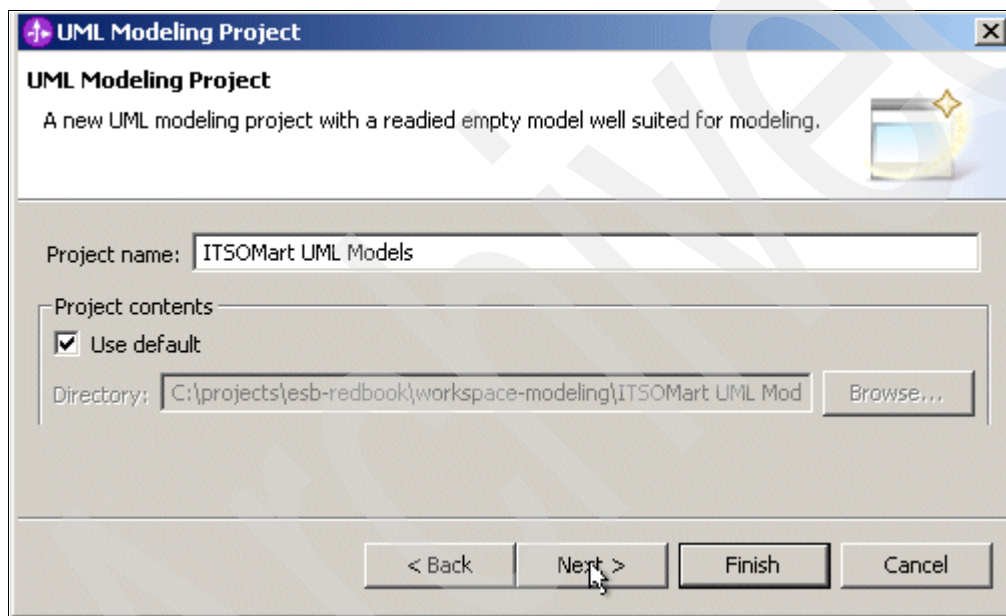


Figure 5-5 UML modeling project

3. In the Create UML Model page (Figure 5-6) you can choose to create a UML model in the new project from a list of UML Model templates and name the model file.

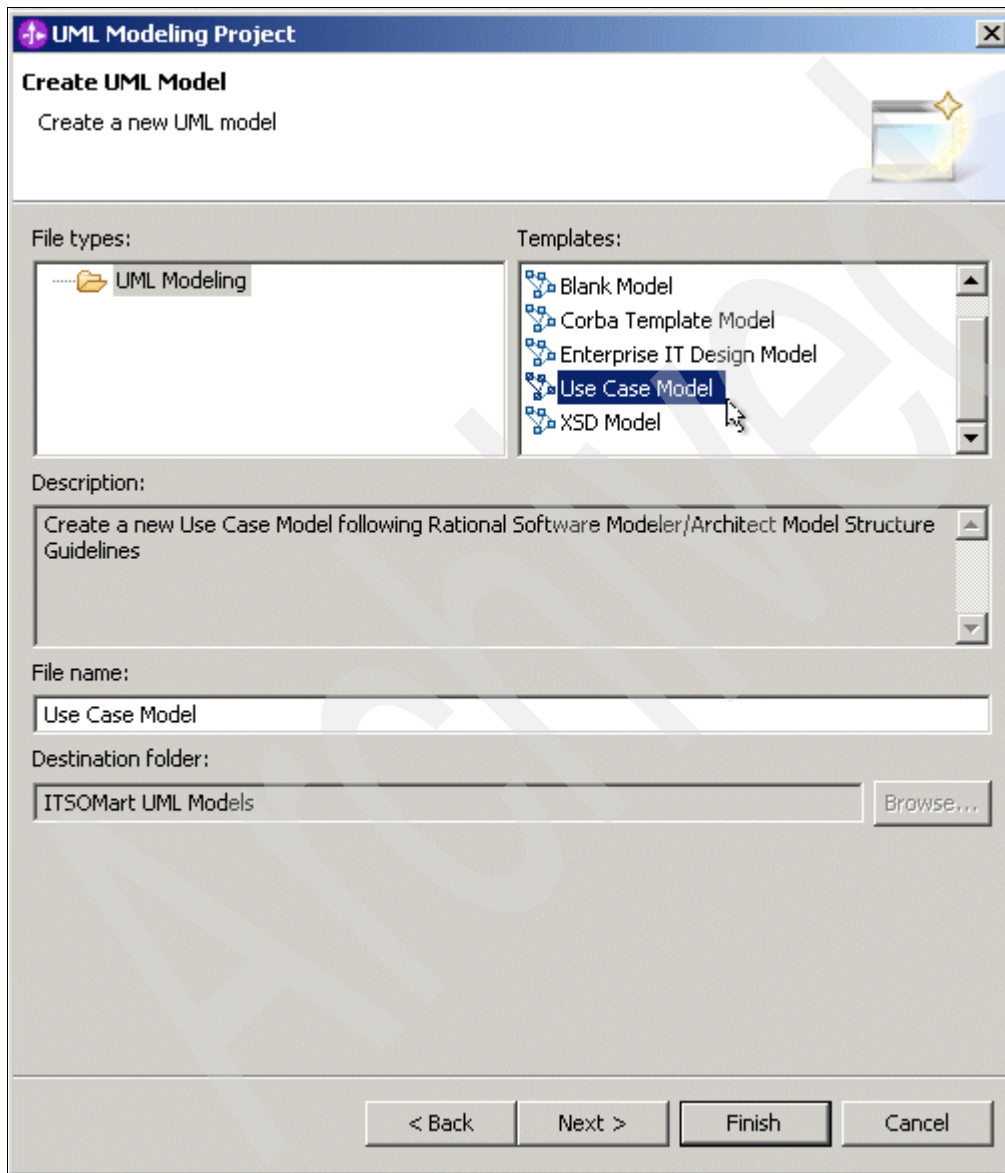


Figure 5-6 UML Modeling Project: create a UML model

5.3.3 UML models

To create a new UML model:

1. In the Model Explorer view, right-click an existing modeling project and select **New** → **UML Model**.
2. In the New UML Model window (Figure 5-7) select the type of UML model to create from a list of templates and name the model file.

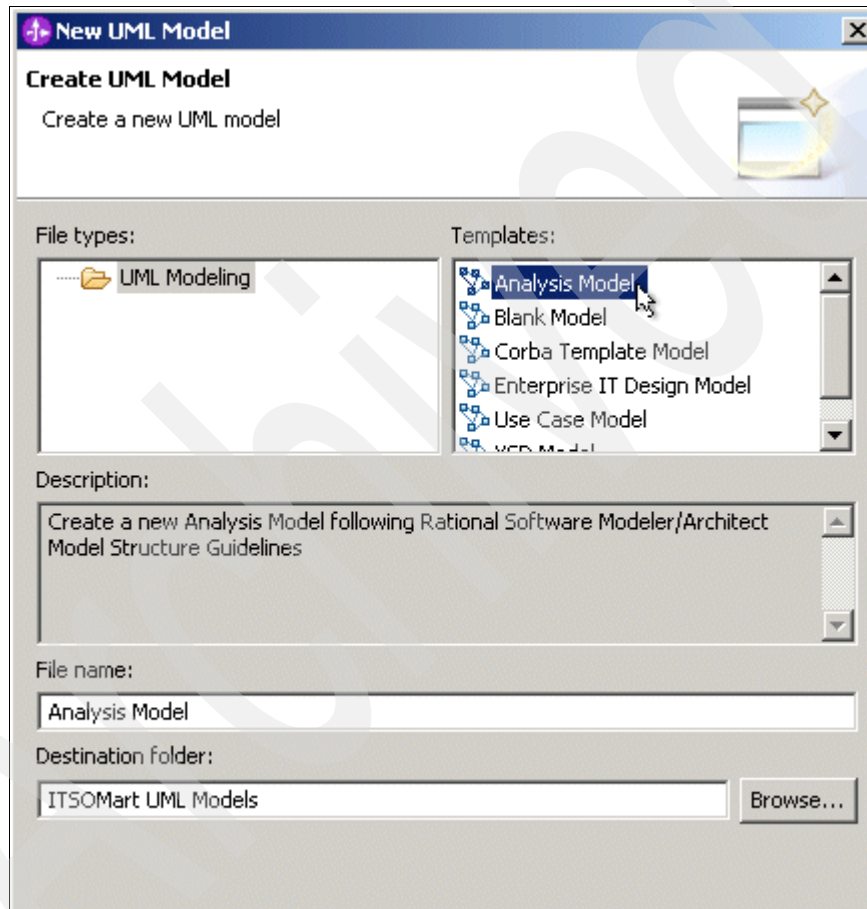


Figure 5-7 New UML Model

When you first create a UML model, it is opened for editing. When you close a model, you only see the model file in the Model Explorer (Figure 5-8).

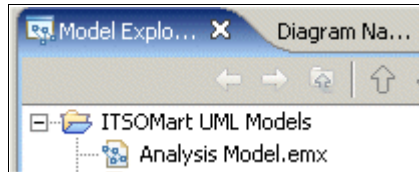


Figure 5-8 Unopened model file

3. To open a model, double-click the model file (Figure 5-9).

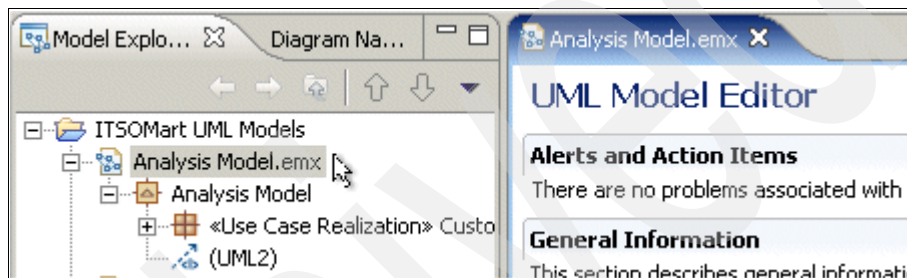


Figure 5-9 Opened model file

For the ITSOMart application, we created the following types of UML models:

- ▶ Use Case model
- ▶ Analysis model
- ▶ Design model
- ▶ XSD model
- ▶ Blank model (to contain the Topology and Deployment model diagrams)

5.3.4 UML diagrams

You can create various types of UML diagrams from the UML elements within a model.

To create a UML diagram:

1. In an opened model, right-click the model or a package in the model, and select **Add Diagram**.

2. Then select the type of UML diagram you want to create (Figure 5-10).

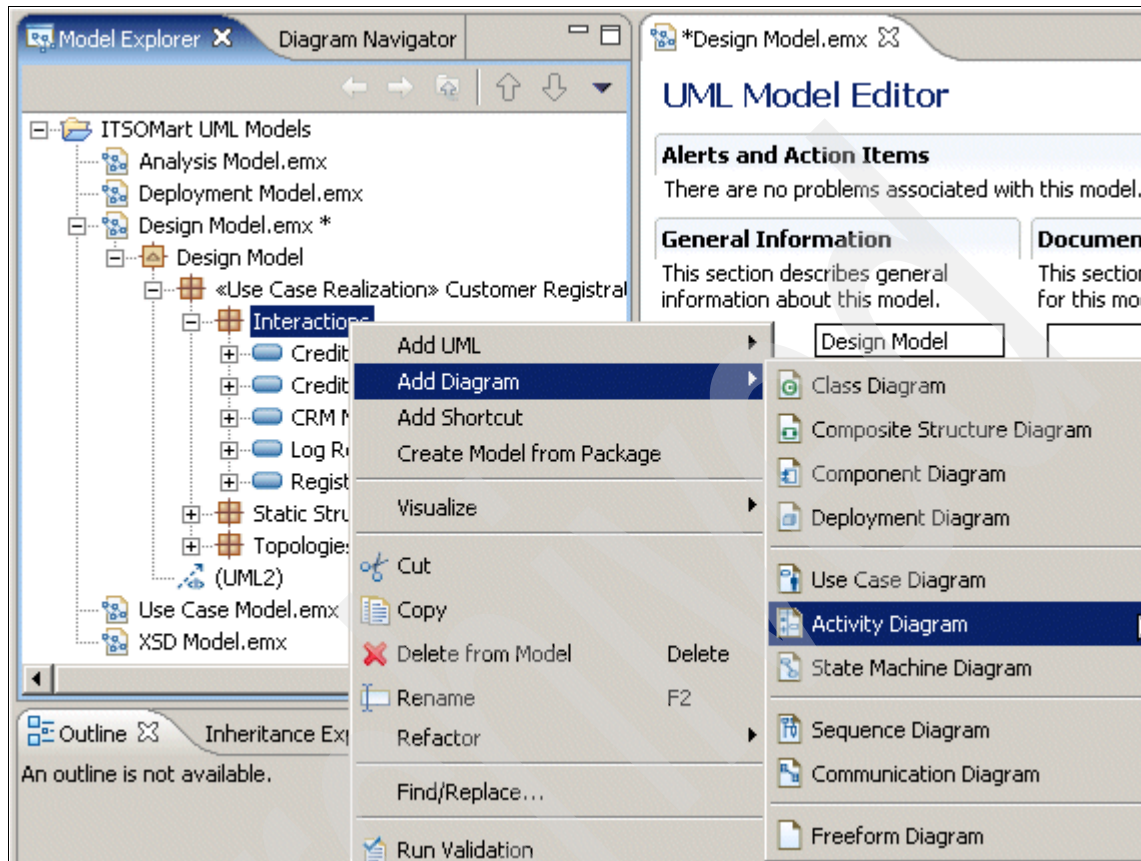


Figure 5-10 Add diagram

The diagram editors provide a palette of UML elements (Figure 5-11) specific for the type of diagram you are working in. For example, if you have a Use Case diagram open in the editor, the palette will provide Use Case modeling elements such as actor, use case, and subsystem that you can drag and drop on the diagram.

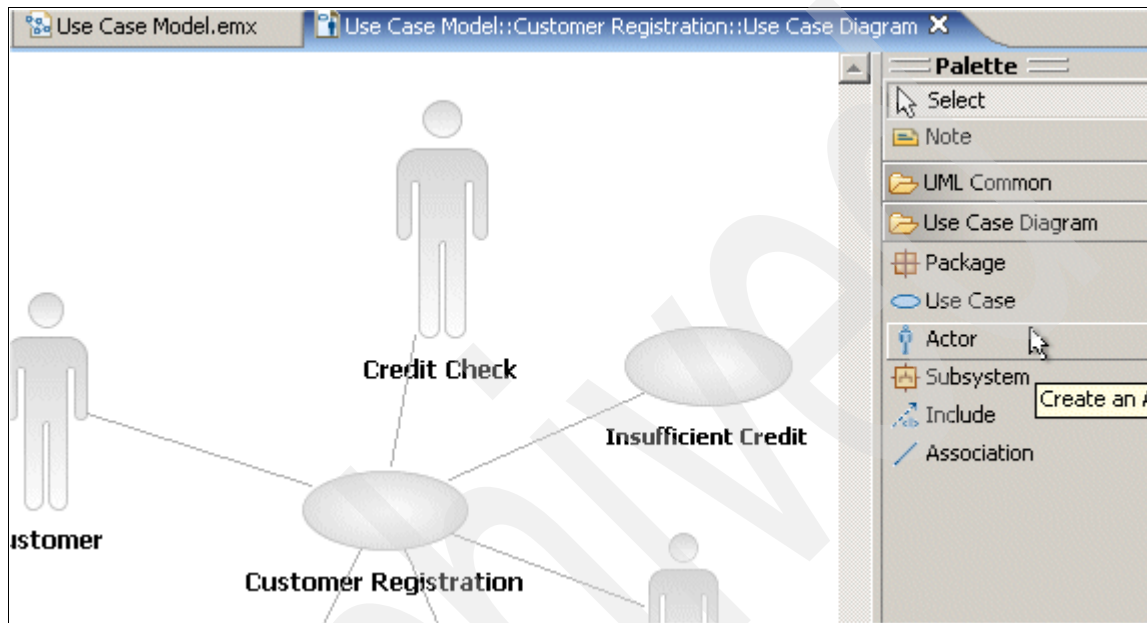


Figure 5-11 Use case diagram palette

Whereas if you are working in an Activity diagram, then the palette will have an Activity Diagram folder (Figure 5-12) with elements such as actions and control nodes specific to that type of diagram.

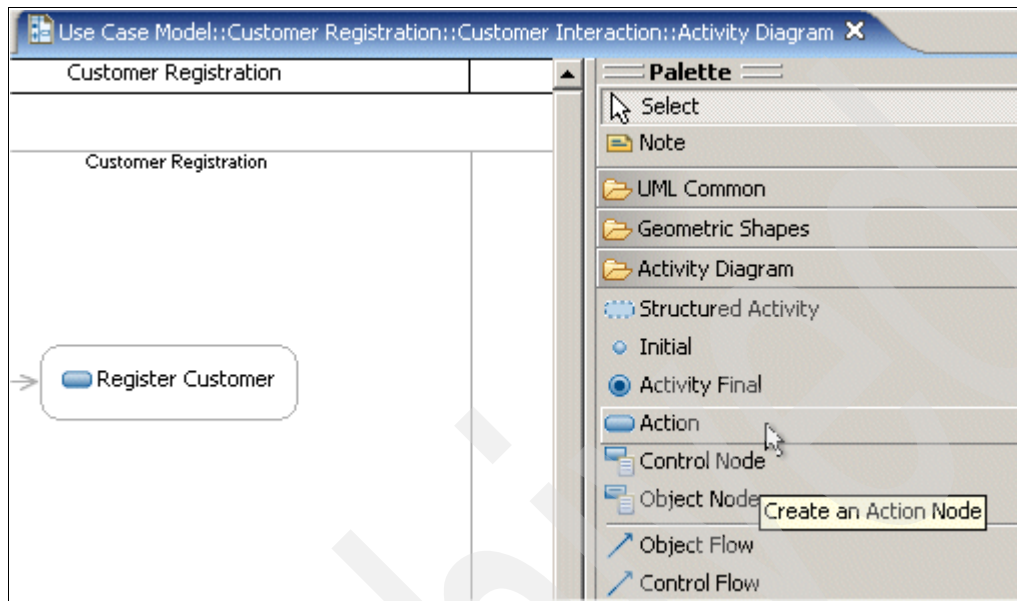


Figure 5-12 Activity diagram palette

In the following sections you will see these types of UML diagrams that we created to model the ITSOMart application:

- ▶ Use Case diagram
- ▶ Sequence diagrams
- ▶ Activity diagrams
- ▶ Component diagrams
- ▶ Class diagrams

5.4 Solution requirements

The business and non-functional requirements of the ITSOMart sample application are described in 4.1, “The ITSOMart scenario” on page 68. In this section we visually describe these requirements in a *use case model*. A use case model describes the scope of the application domain, what actors are involved in the system, and interactions between those actors and the system. Our model contains a use case diagram and activity diagrams.

5.4.1 Use case diagram

The use case diagram we created for the ITSOMart Customer Registration use case (Figure 5-13) shows the five discrete actors that interact with the customer registration system as defined in the ITSOMart business requirements.

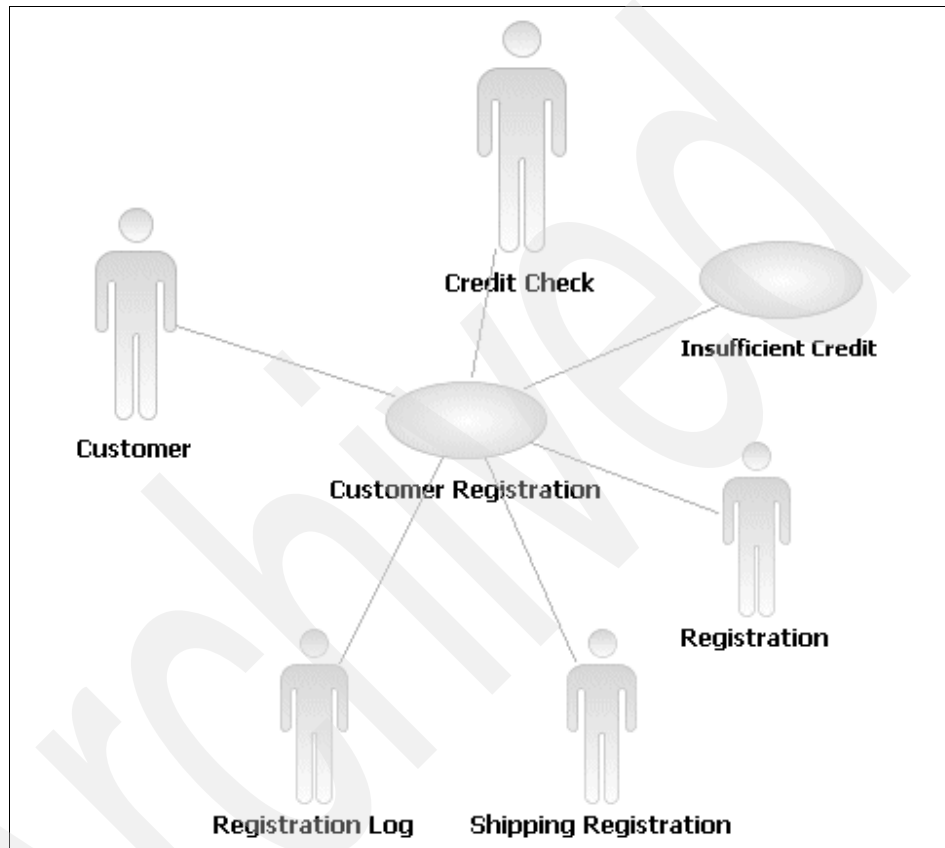


Figure 5-13 ITSOMart Customer Registration use case diagram

The actors are:

- ▶ Customer - represents an ITSOMart customer.
- ▶ Credit Check - represents a third-party credit rating system that determines a customer's credit rating.
- ▶ Insufficient Credit - represents a secondary use case to handle the requirement for insufficient credit processing for customer's who have bad credit. We do not design or implement the solution for this secondary use case, but we show how the customer registration interacts with it.

- ▶ Registration - represents a customer relationship management (CRM) system that ITSOMart owns and maintains in-house.
- ▶ Shipping Registration - represents a system that maintains customer shipping address information for ITSOMart fulfillment centers.
- ▶ Registration Log - represents the ITSOMart customer registration audit log system.

5.4.2 Activity diagrams

Then to better understand the interactions between the customer registration system and the actors involved, we created a couple of activity diagrams.

The first activity diagram shows that the customer's role is to initiate the Customer Registration use case (Figure 5-14).

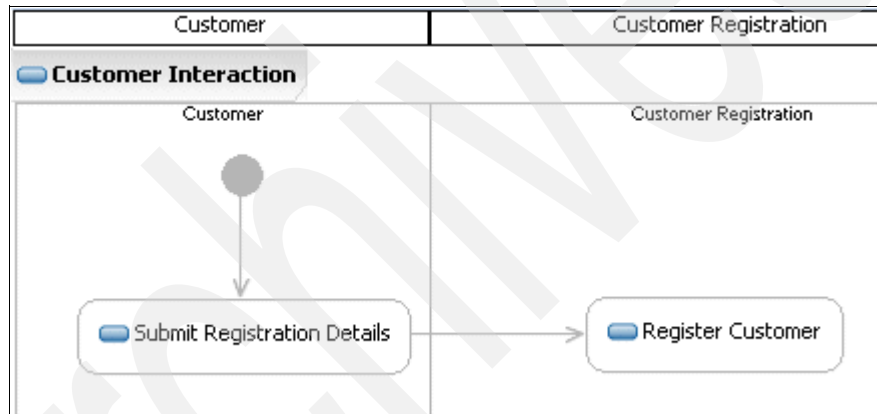


Figure 5-14 Use case model: customer interaction activity diagram

Then the second activity diagram (Figure 5-15) describes the interactions between the Customer Registration system and the rest of the actors. This activity diagram clearly communicates the individual tasks that need to occur to complete the registration process required for ITSOMart to begin offering their online services to a customer.

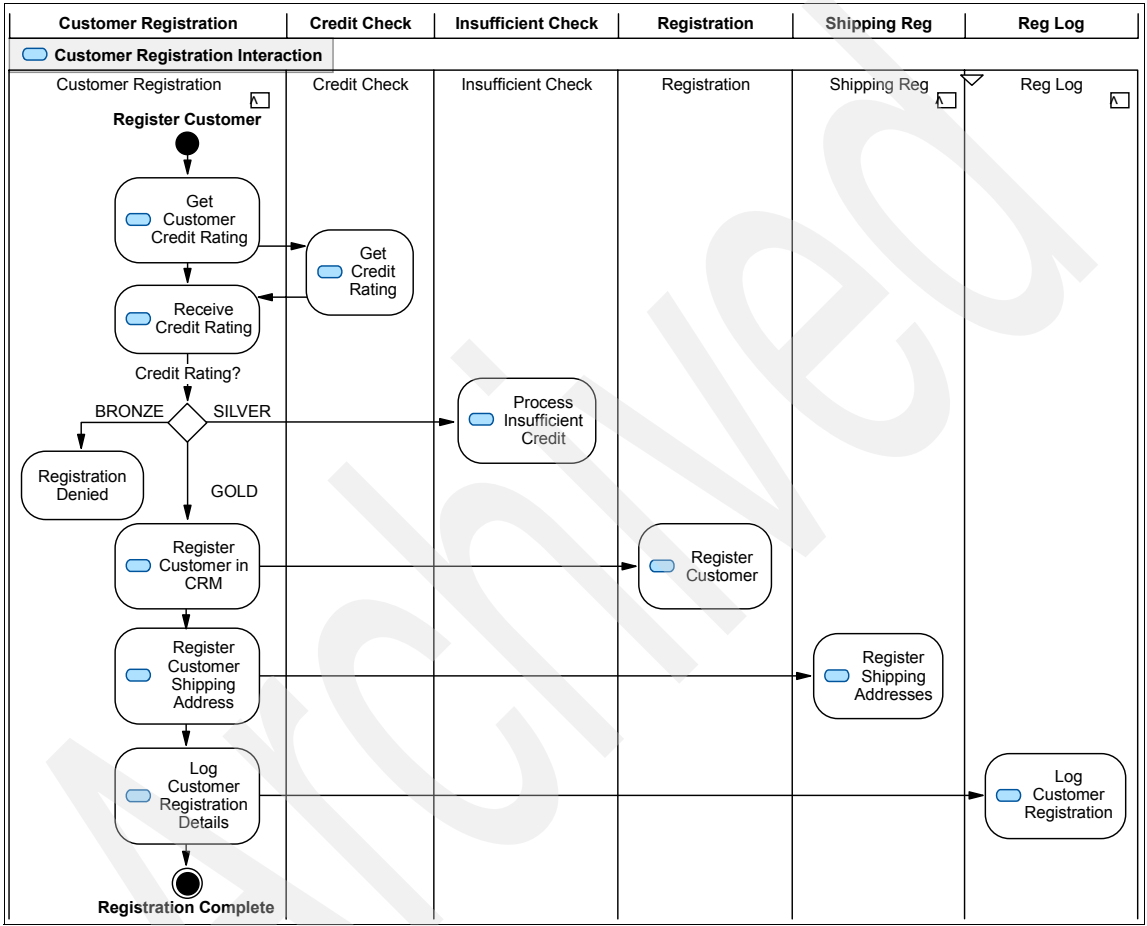


Figure 5-15 Use case model: Customer Registration Interaction activity diagram

The Customer Registration system gets the customer’s credit rating, then makes a decision about how to proceed based on the category that customer rating falls under. If the credit rating category is gold, then the customer is registered in the CRM system, the shipping addresses are registered with the fulfillment centers, and the registration details are logged in the audit system. If the credit rating category is silver, then further credit validation must occur for the customer and

is handled by the Insufficient Credit use case. If the credit rating category is bronze then the customer is denied online registration with ITSOMart.

5.5 Domain analysis

The next step in the solution is to capture the application domain in an *analysis model*, where you model the use case in terms of interactions between actors using sequence diagrams, and describe the static structures of the use case in component and class diagrams.

5.5.1 Sequence diagrams

Sequence diagrams conform to the interactions described by the activity diagrams in the use case model and can be quite similar. However, sequence diagrams define interactions in chronological order and help to identify how the use case actors can be represented as components or services in your solution, what operations need to be defined for those components, and what messages need to flow between them.

We start by creating a sequence diagram that shows the customer's interaction with the customer registration system. You can see from this diagram how the Customer Registration use case itself can be represented as a component with a *register* operation on it. The registration process may take longer to complete than a customer is willing to wait for an immediate response, so we show that this interaction is best represented as a one-way asynchronous message sent to the system. See Figure 5-16.

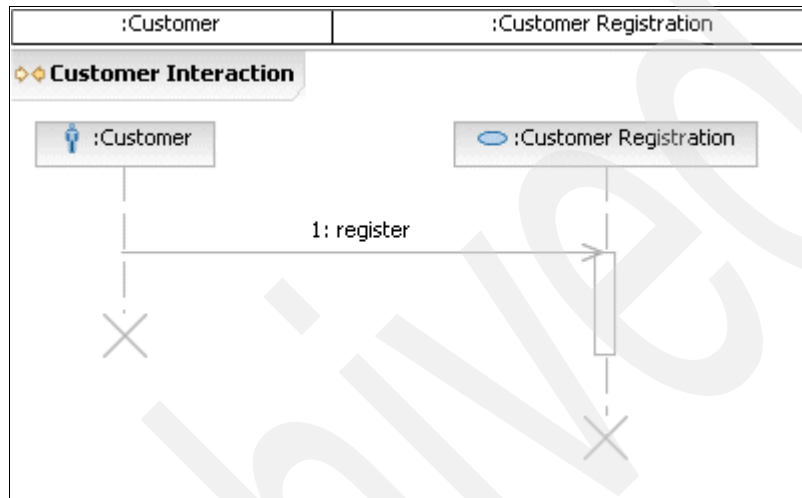


Figure 5-16 Analysis model: Customer Interaction sequence diagram

Then we describe the various interactions that the customer registration system can have with the other actors in the system while processing a registration request. More specifically, the interactions that occur in the customer registration system are different depending on whether the customer's credit rating is gold, silver, or bronze.

The sequence diagram for the case that the credit rating is gold shows a successful customer registration (Figure 5-17).

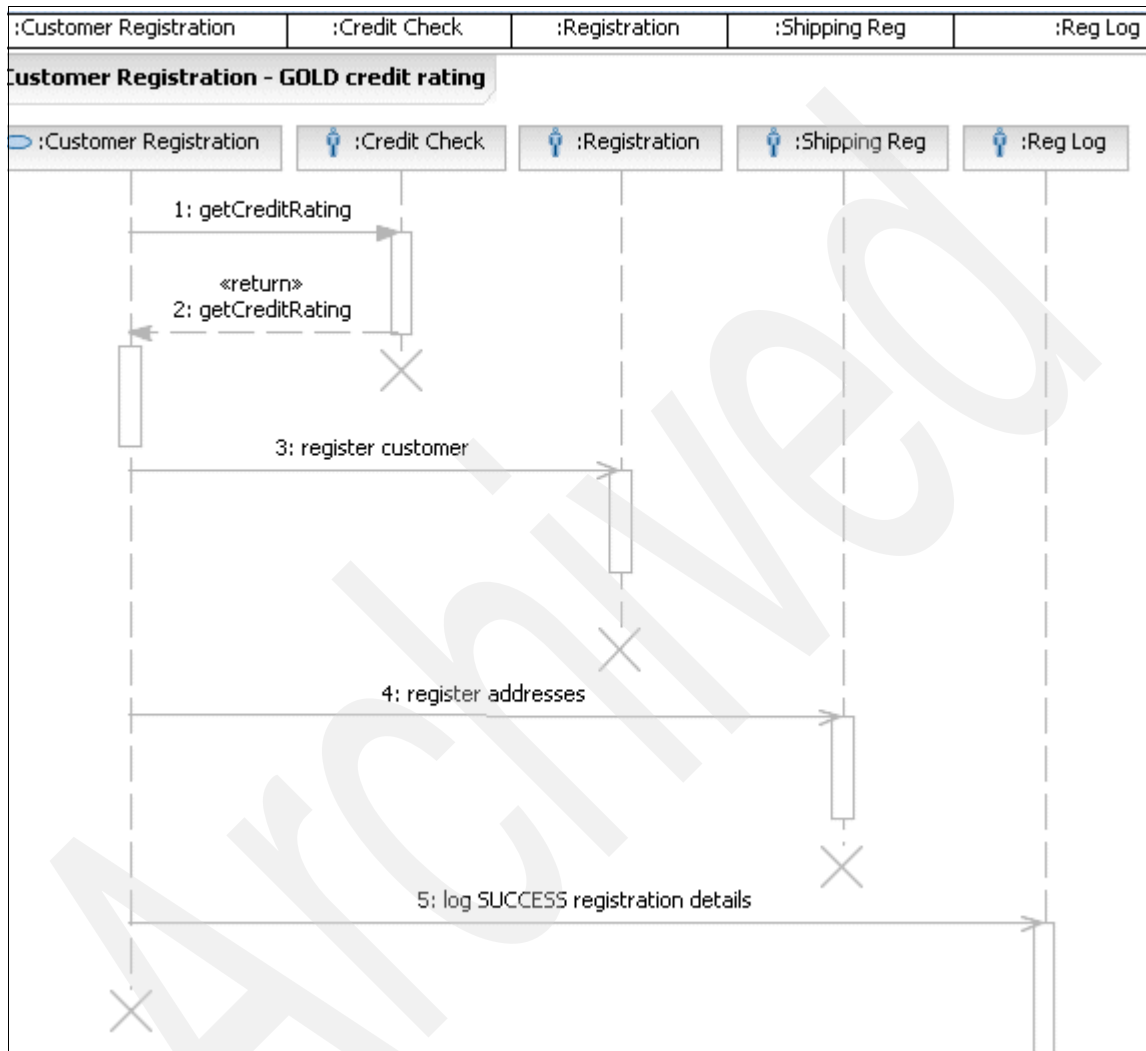


Figure 5-17 Analysis model: customer registration - gold credit rating sequence diagram

The sequence diagram for the case that the credit rating is silver shows that insufficient credit processing must take place for this customer (Figure 5-18).

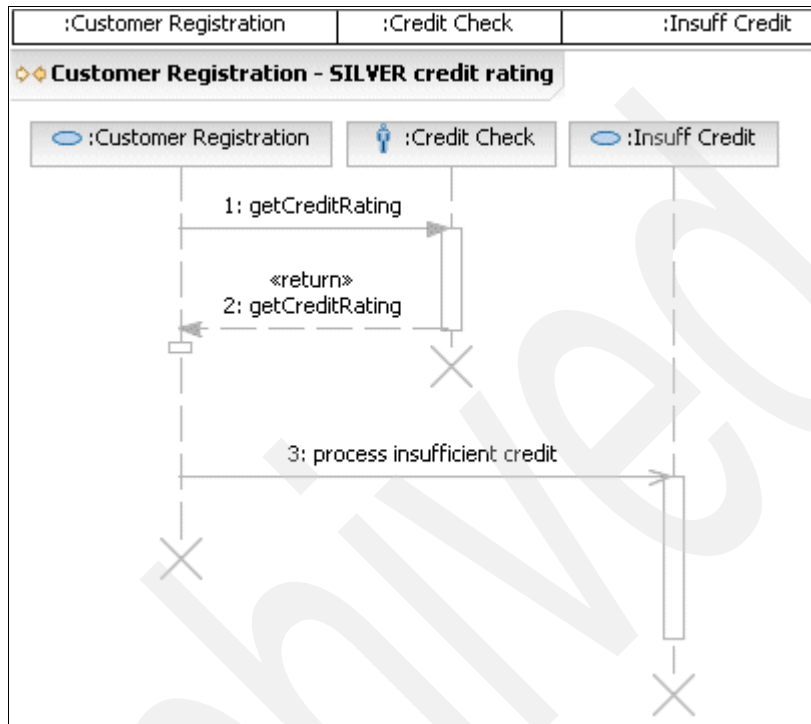


Figure 5-18 Analysis model: customer registration - silver credit rating sequence diagram

The sequence diagram for the case that the credit rating is bronze shows that the customer is denied registration due to terrible credit (Figure 5-19).

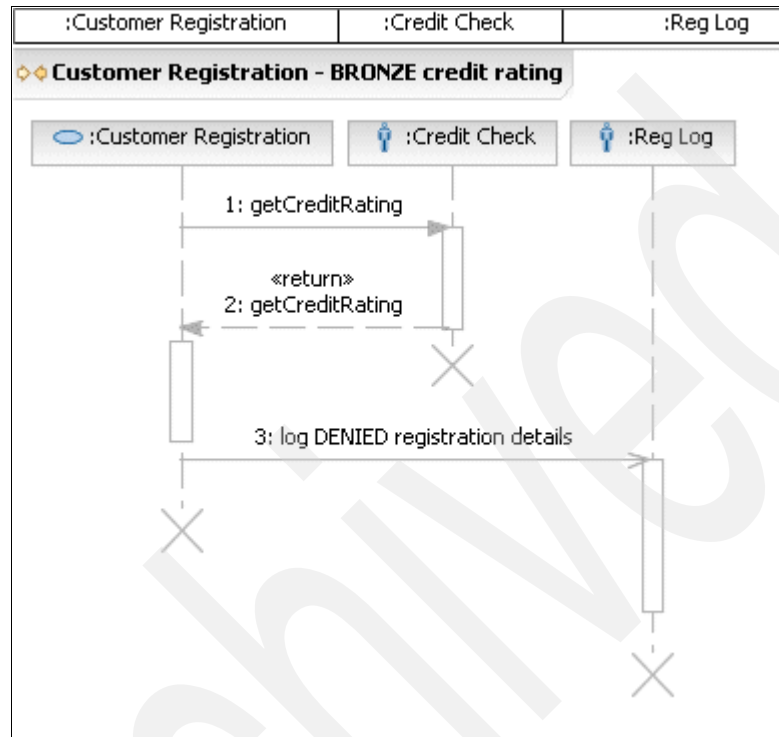


Figure 5-19 Analysis model: customer registration - bronze credit rating sequence diagram

5.5.2 Component diagram

Looking at the sequence diagrams, it is easy to see that each of the actors in the use case can be represented in the solution as high-level components that can be modeled in a component diagram.

In a component diagram, you can apply stereotypes to the components to indicate what types of components you think these might represent. The stereotypes applied to the components here during domain analysis may change as you get deeper into the solution through the architectural design and implementation of the solution.

As you can see in Figure 5-20, we represent the use case itself as a *process component* because it performs the main processing of the customer registration, but we may or may not want to refer to this component as a process in the actual implementation. We represent the customer actor as a *boundary component* because there will need to be some component in the application that interfaces with and represents the customer. The Insufficient Credit component case is described as a boundary component as well because it encapsulates the interaction with the Insufficient Credit use case. The Registration component has a stereotype of *subsystem* because it represents the CRM system that ITSOMart runs in-house. The rest of the actors in the use case are simply described as *services*.

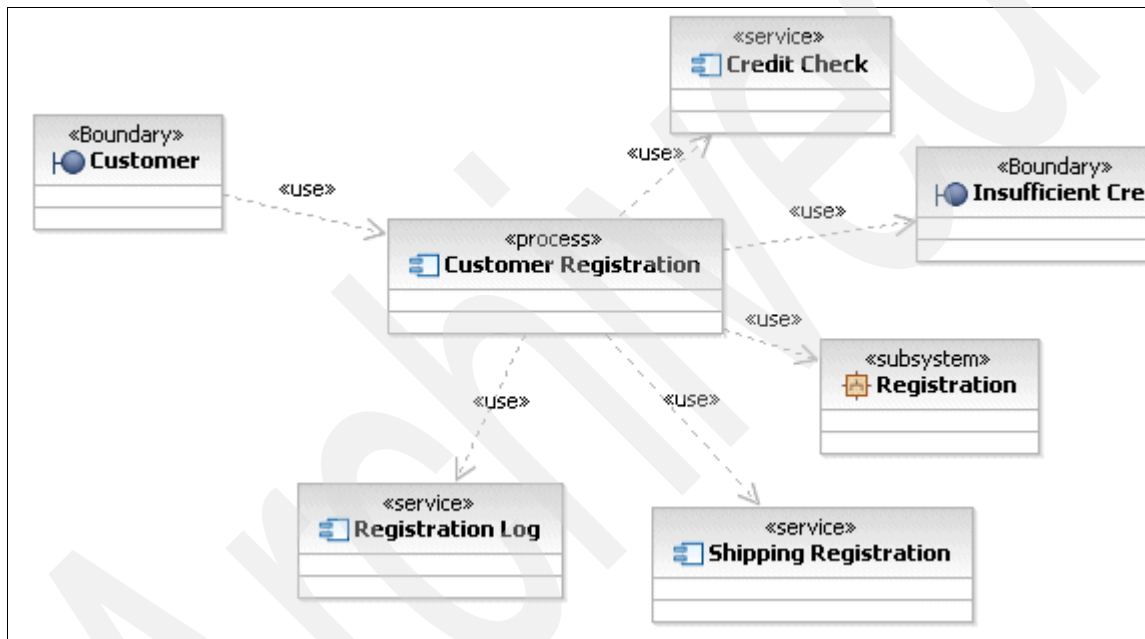


Figure 5-20 Analysis model: component diagram

5.6 Architectural design

The architectural design phase of a solution is where you create a *design model* that provides more details about how the system will be implemented such as what interfaces look like, what messages flow between components, what actual systems the application communicates with, what protocols are used, and where and how your application is deployed.

The decisions around the use of an ESB most likely will occur during the architectural design of your solution. These will be decisions about what services or components will be deployed to your ESB and what mediations, if any, are required for those services.

For designing services in the ESB layer, you can create models to describe implementation details such as what protocols will be used to invoke services, what types of specific transformations need to take place in the mediations for those services, and what mediation primitives will be used to implement the mediations.

For the design of the ITSOMart sample application, we looked at each of the components individually in the use case and the interactions between them and came up with the design described in the following sections.

5.6.1 Service components

We modified the component diagram from the analysis model. In that diagram, all the components used for registration processing were defined as *services*. We changed the name of the Customer Registration use case component to Registration Processor, which we think better reflects its function of processing customer registration requests. Then we added a new component called Customer Registration, which consolidates the functionality for deciding whether to insert the customer information into the CRM Registration Service or to send it to the insufficient credit service for further credit validation. The changes can be seen in Figure 5-21.

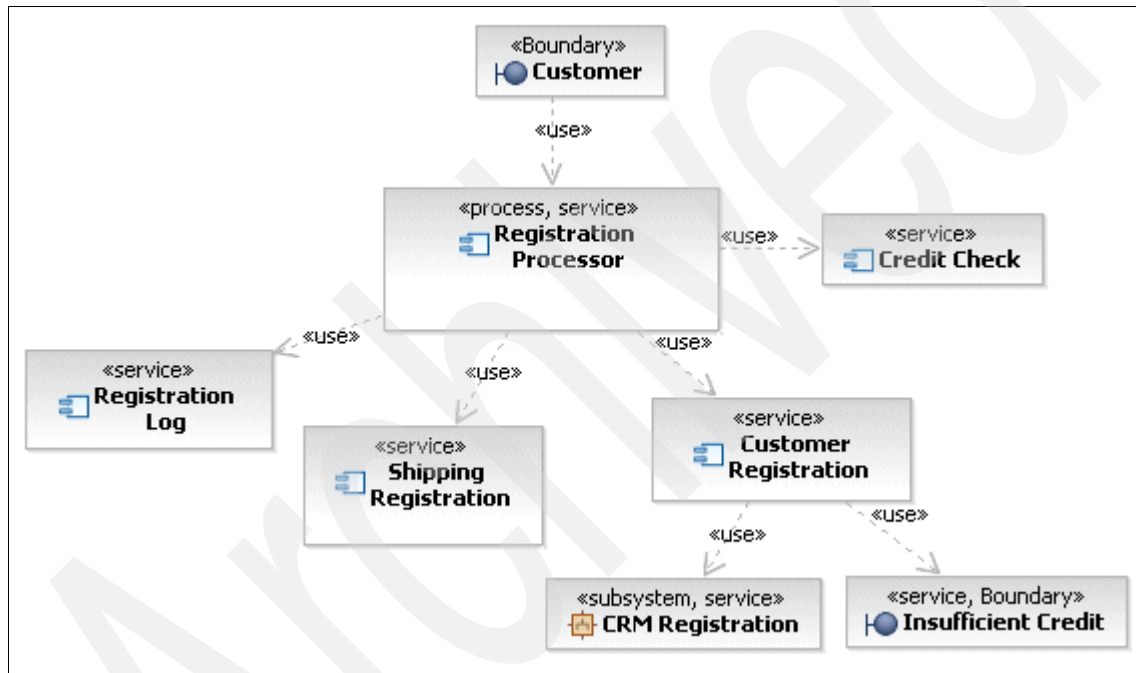


Figure 5-21 Modified component diagram

5.6.2 Connecting services through the ESB

For our design, we decided to access the services used by the Registration Processor component through the ESB. This sample is contrived, of course, to highlight the use of the ESB since that is what this book is about, but using an ESB as illustrated by the sample scenario provides a lot of value in a solution that focuses on connectivity.

We view the components that the Registration Processor interacts with as boundary services for communicating with systems that may or may not be maintained by ITSOMart. Accessing these services through an ESB allows the ITSOMart solution to use well-defined interfaces and protocols to fulfill a customer registration request. The actual services that implement those interfaces are virtualized by the ESB. For example, the interface and protocol that is required by the third-party provider that ITSOMart chooses to use for its Credit Rating Service is hidden from the registration processor, and thus from the implementation of ITSOMart's internal customer registration application. In addition, we can now apply mediations to these services that handle message transformations, augmentation, routing, and logging.

Virtualizing these services on the ESB (Figure 5-22) also allows them to be easily re-used by other applications. These services provide discrete but very important coarse-grained functionality that ITSOMart needs in its enterprise. The interfaces we define for these services are specific for the sample scenario (what the customer's credit rating is, register a customer in the CRM system), but by specifying additional operations for these services, such as retrieving and updating the customer information in the CRM system, you can easily see the value of providing this type of functionality in a central architectural layer in the enterprise.

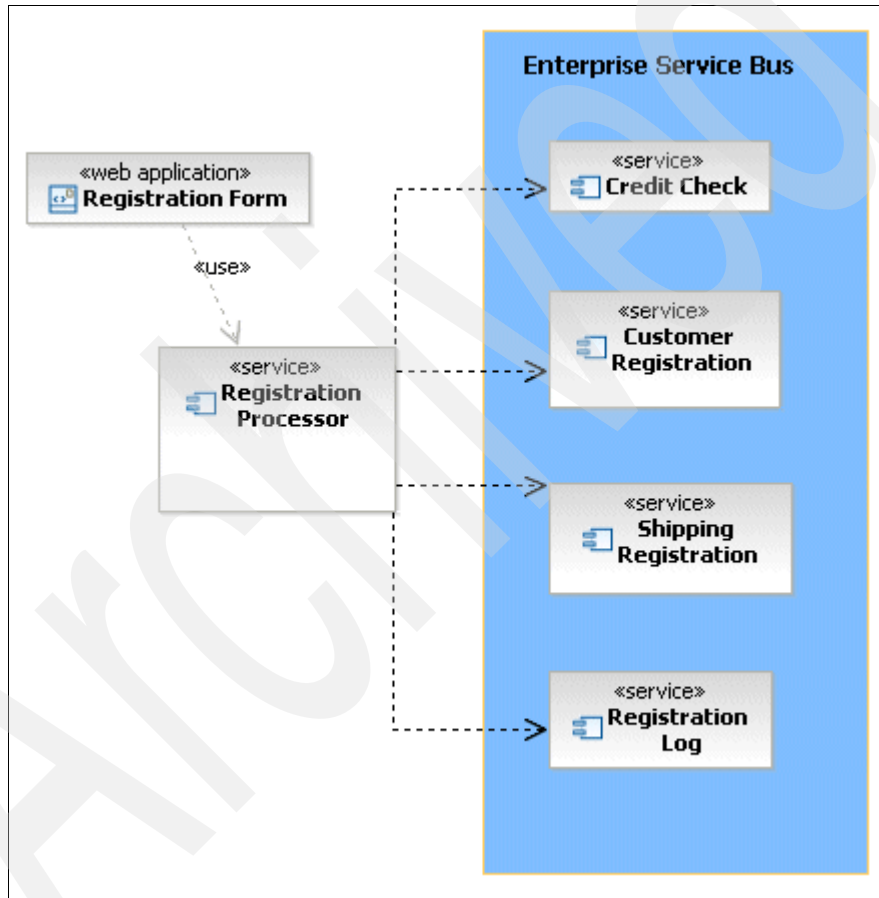


Figure 5-22 Services defined to the ESB

We can go one step further and implement the registration processor as a service, accessible via the ESB (Figure 5-23 on page 154). This service provides the orchestration of the other services on the ESB.

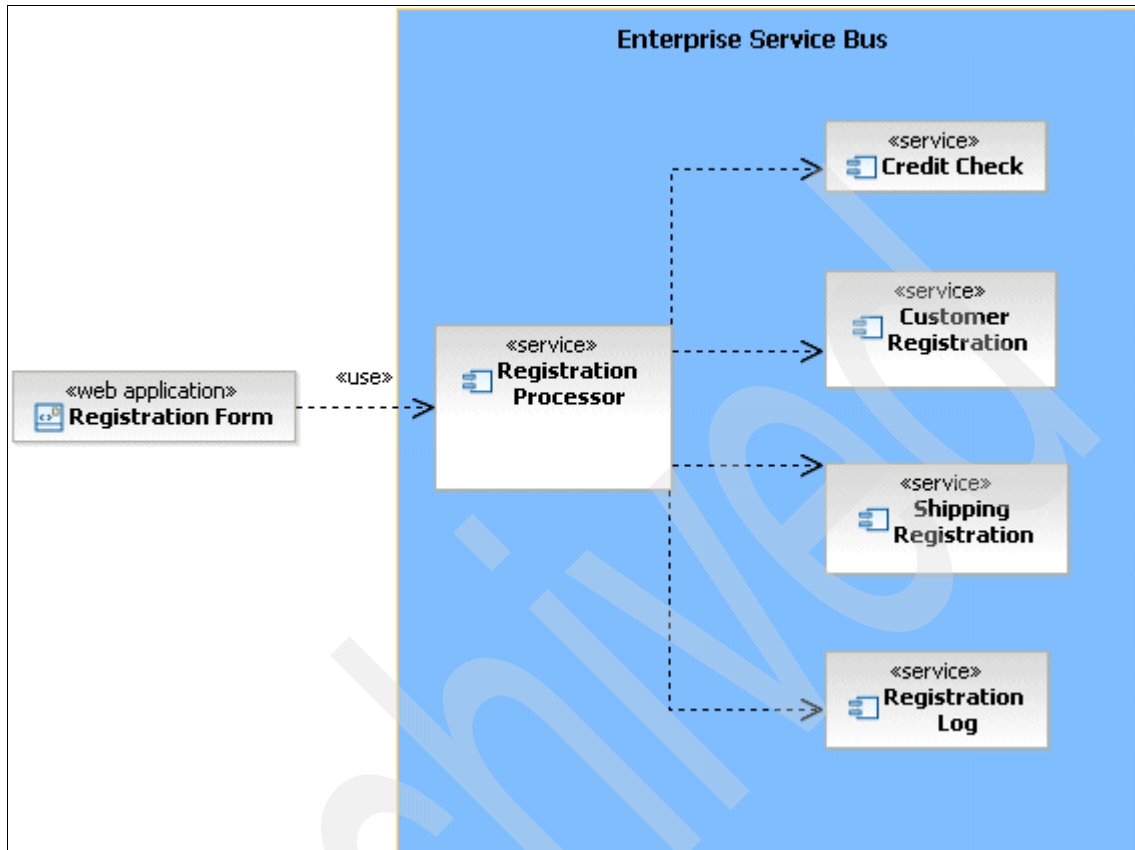


Figure 5-23 Services defined to the ESB

5.6.3 Mediations on the ESB

So far in the design we have identified the services required for the application and have determined that we will access them through the ESB. We have not indicated how these services will be implemented. Each of these services provides an interface, or a facade, through which applications can access systems ITSOMart maintains in-house, as well as third-party systems. In the sample application, we define these services to the ESB and implement mediations that route requests to the required services, or systems, outside of the ESB.

Figure 5-24 shows the mediations and the services invoked from the mediations. The implementation of each of these mediations and the purposes for them are discussed in the following sections.

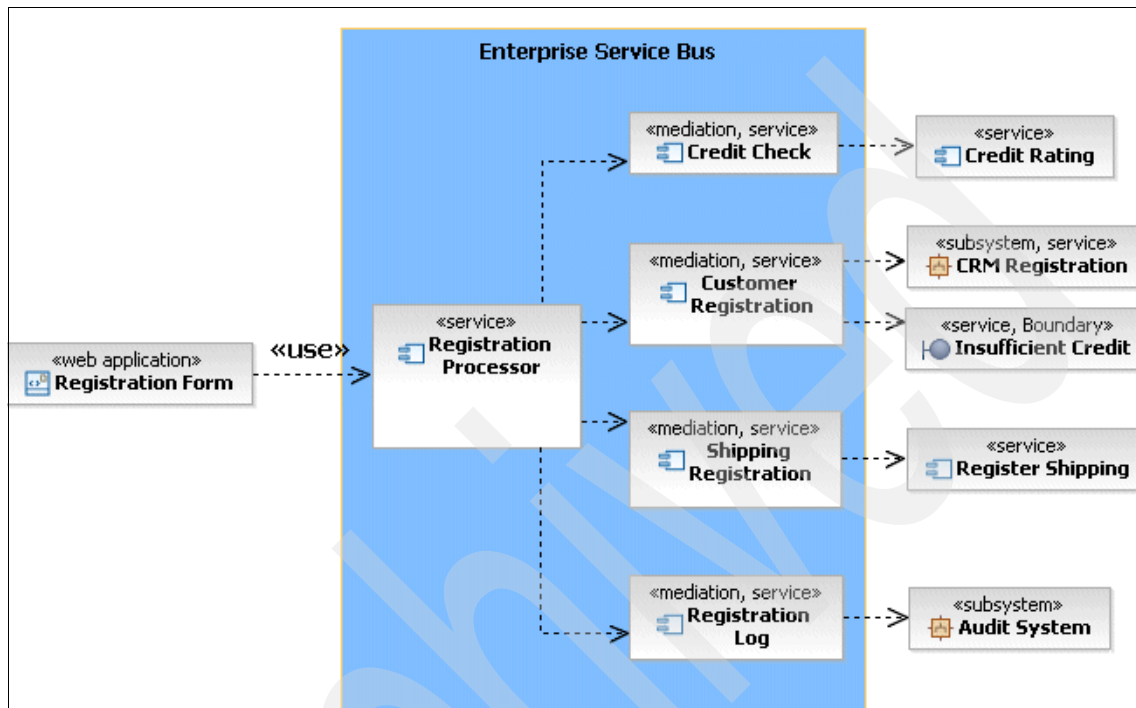


Figure 5-24 Mediations on the ESB

Credit Rating mediation

We want to provide a mediation on the ESB that will virtualize the interaction with whatever Credit Rating Service provider ITSOMart chooses to use. In addition, we can easily keep a log of interactions with the Credit Rating Service provider by logging requests through this mediation layer. This mediation will be called the Credit Rating mediation.

This mediation is an example of the Directly Integrated Single Channel application pattern (3.4.1, “Directly Integrated Single Channel application pattern” on page 44) because it provides a direct one-to-one connection between the client request and the service invocation, as shown in Figure 5-25.

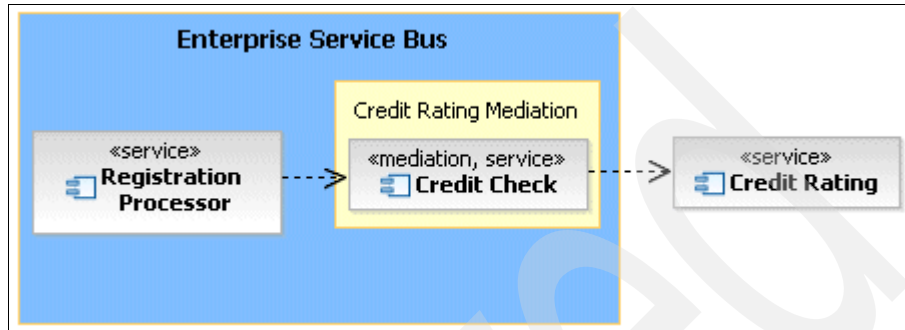


Figure 5-25 Credit Rating mediation on the ESB

The activity diagram in Figure 5-26 shows this mediation separated into request and response flows. The registration processor sends a SOAP/HTTP request for the customer's credit rating, the mediation transforms the request message into the appropriate format that the Credit Rating Service requires, and then invokes the Credit Rating Service using SOAP/HTTP. The opposite occurs on the response, where the mediation transforms the response message from the service to the correct response message specified in the interface for the mediation.

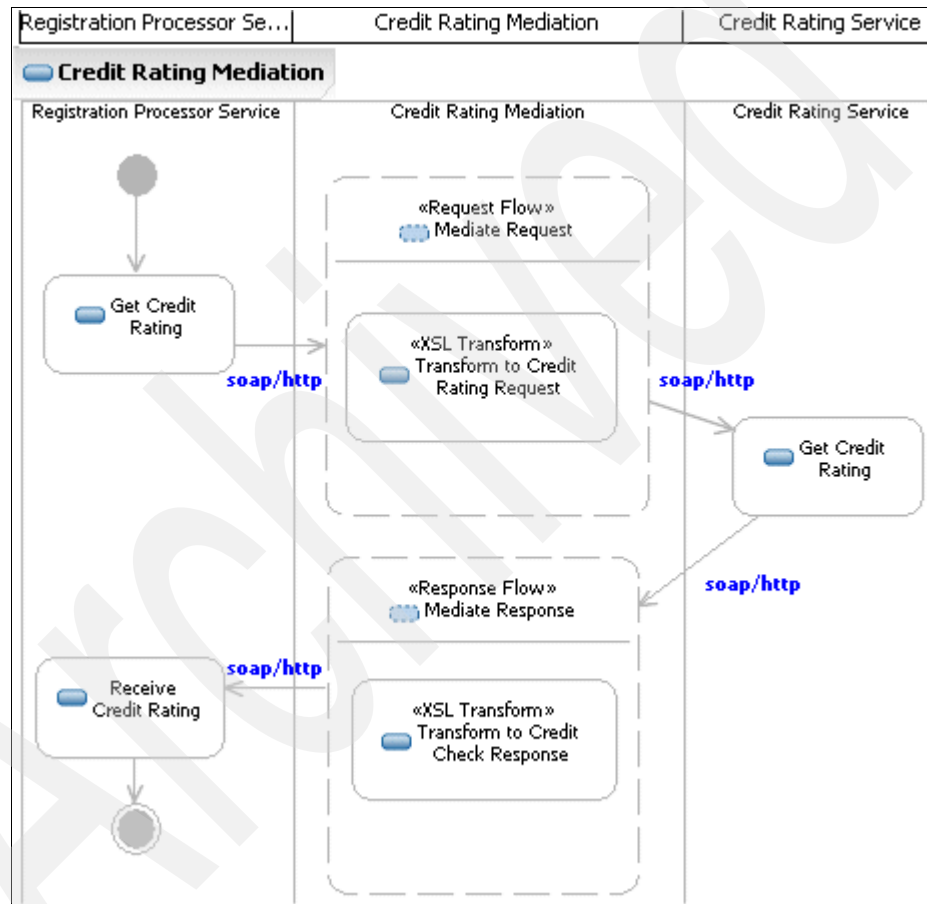


Figure 5-26 Credit Rating mediation activity diagram

The Credit Rating mediation itself is exposed as a SOAP/HTTP Web service with the credit check interface, so client applications will always use the credit check interface to retrieve a customer's credit rating no matter what the actual Credit Rating Service interface looks like.

The component diagram in Figure 5-27 shows the relationships between the mediation and the Credit Rating Service and how each of the interfaces are realized, or implemented.

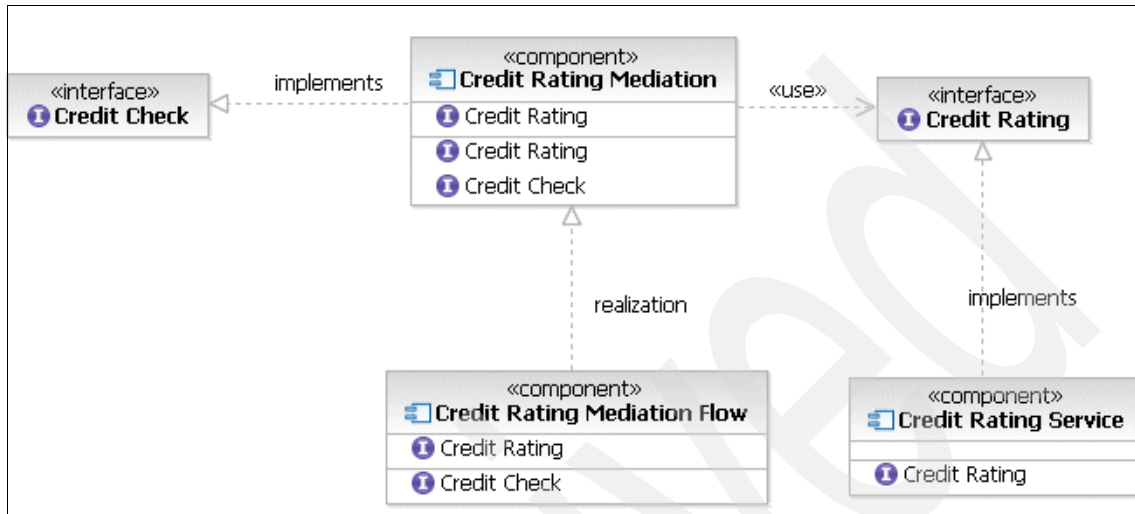


Figure 5-27 Credit Rating mediation component diagram

You can see how this simple component diagram easily fits into the SCA when we overlay the diagram with how these components will be represented in an SCA mediation module.

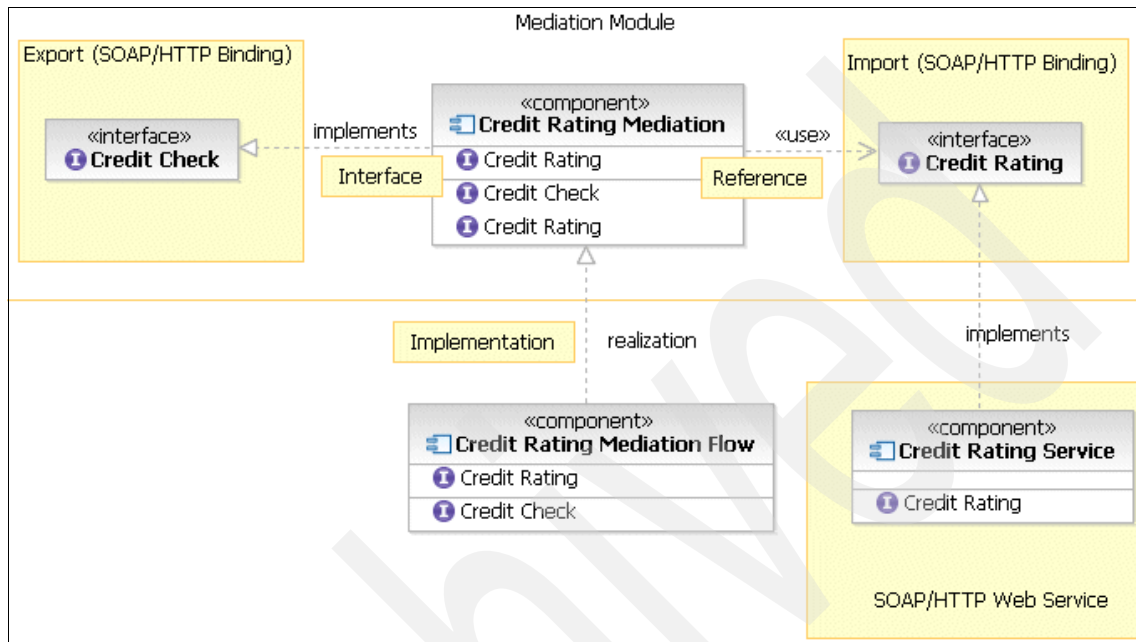


Figure 5-28 Credit Rating mediation SCA component diagram

The Credit Rating mediation implements the Credit Check interface, which is exposed as an export with a SOAP/HTTP Web service binding. It has a reference to the SOAP/HTTP Credit Rating Service through an import component. The actual implementation of the mediation is a mediation flow component.

Credit Score mediation

For the ITSOMart sample, we wanted to show how easy it is to actually switch out the use of an existing Credit Rating Service with a service from another provider if ITSOMart should decide to use another credit rating provider, as well as show an example of chaining together mediations, so we created a second mediation called the Credit Score mediation.

For this scenario, the new Credit Rating Service provider that ITSOMart has chosen provides a service called the Credit Score Service. This service returns a credit score number, rather than a rating category (gold, silver, bronze) that the original Credit Rating Service returned. The new Credit Score mediation (Figure 5-29) takes care of this difference between the old and the new services.

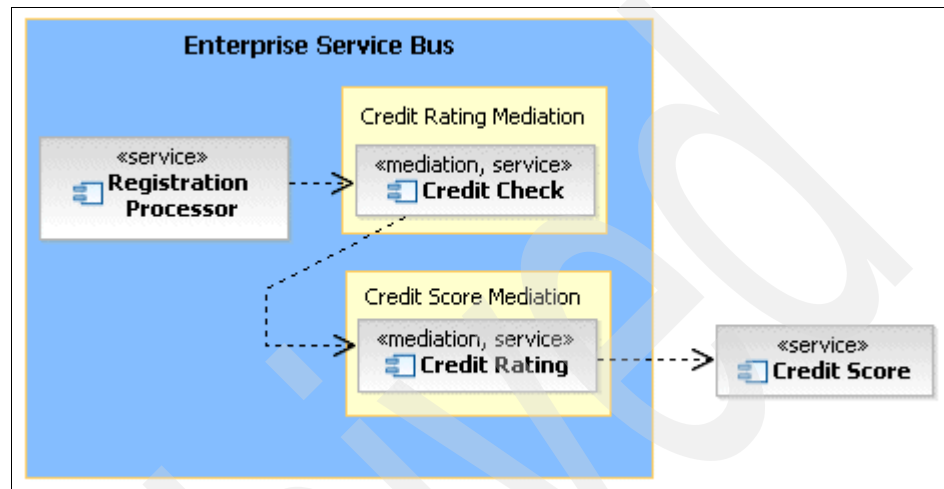


Figure 5-29 Credit Score mediation on the ESB

The Credit Score mediation has the same interface as the Credit Rating Service so that it can logically take the place of the Credit Rating Service. The original Credit Rating mediation invokes the Credit Score mediation, which in turn mediates the request to and response from the new Credit Score Service, as shown in Figure 5-29.

Note: The decision as to whether you should create a new mediation to accommodate the new service or to extend the original mediation with a second is an architectural choice that will rely on factors such as performance, ease of creating a new mediation, simplicity of mediation structure, and future maintenance. In the ITSOMart case, adding a second mediation is a simple solution to the problem and illustrates a technique you can use to call one mediation from another. In many real-life cases, a new mediation would probably be the better choice.

The assembly diagram in Figure 5-30 shows the implementation of the Credit Score mediation. In the response flow of the mediation, a database call is performed to look up the credit rating category that matches the credit score number returned from the Credit Score Service.

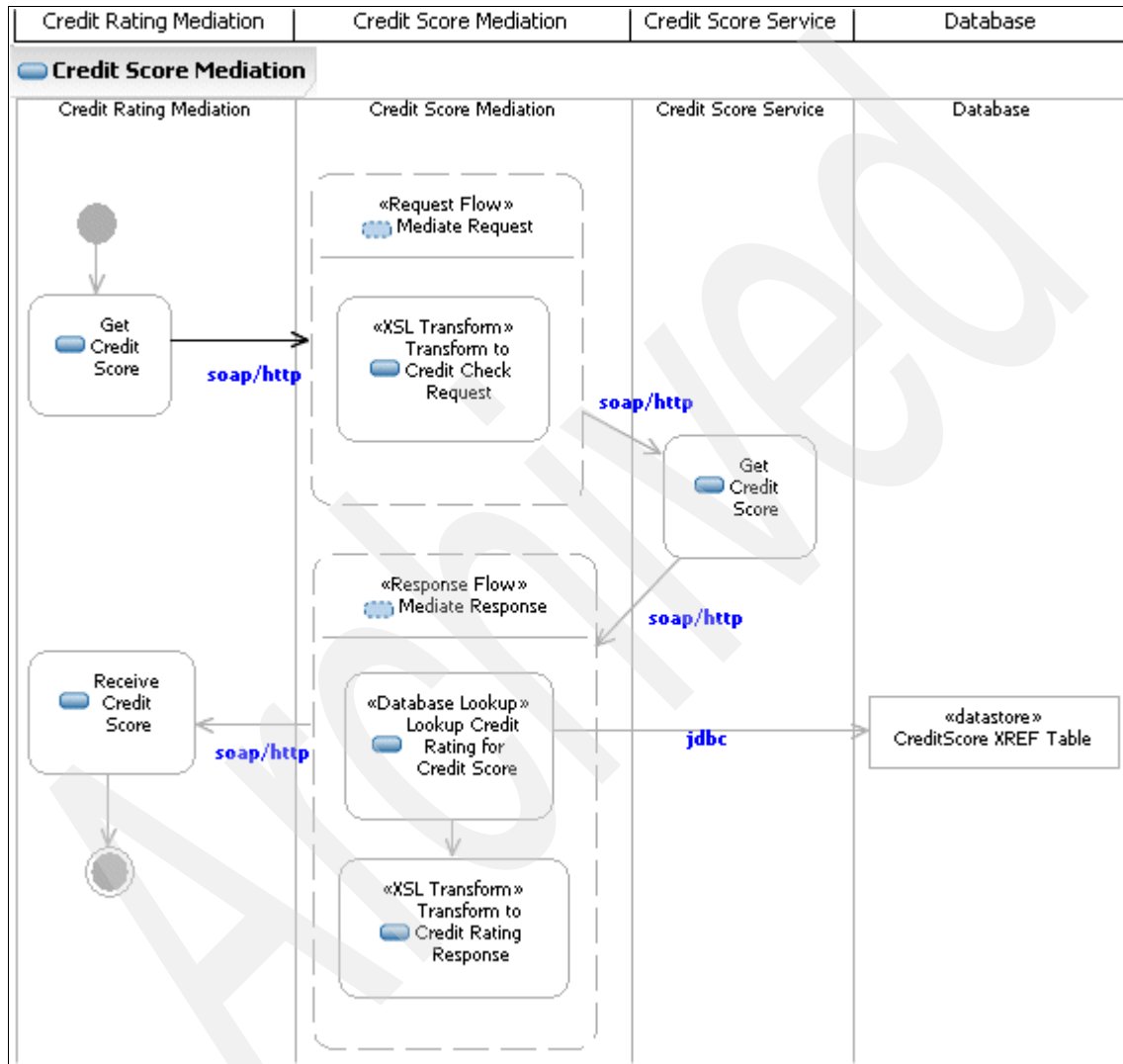


Figure 5-30 Credit Score mediation activity diagram

CRM mediation

A mediation for the customer registration service, called CRM mediation, routes registration requests depending on the customer's credit rating. This mediation is

an example of the Router application pattern discussed in 3.4.2, “Router application pattern” on page 47.

Figure 5-31 shows this mediation. The registration processor sends a registration request to the mediation synchronously via SOAP/HTTP. The mediation then routes the request to the CRM registration system using the Siebel J2C Adapter if the credit rating is gold and returns a registration status of success to the registration processor. If the credit rating is silver, then the registration request is written to the file system using the Flat File J2C Adapter to be handled by the insufficient credit service. If the credit rating is bronze, then the customer is denied registration.

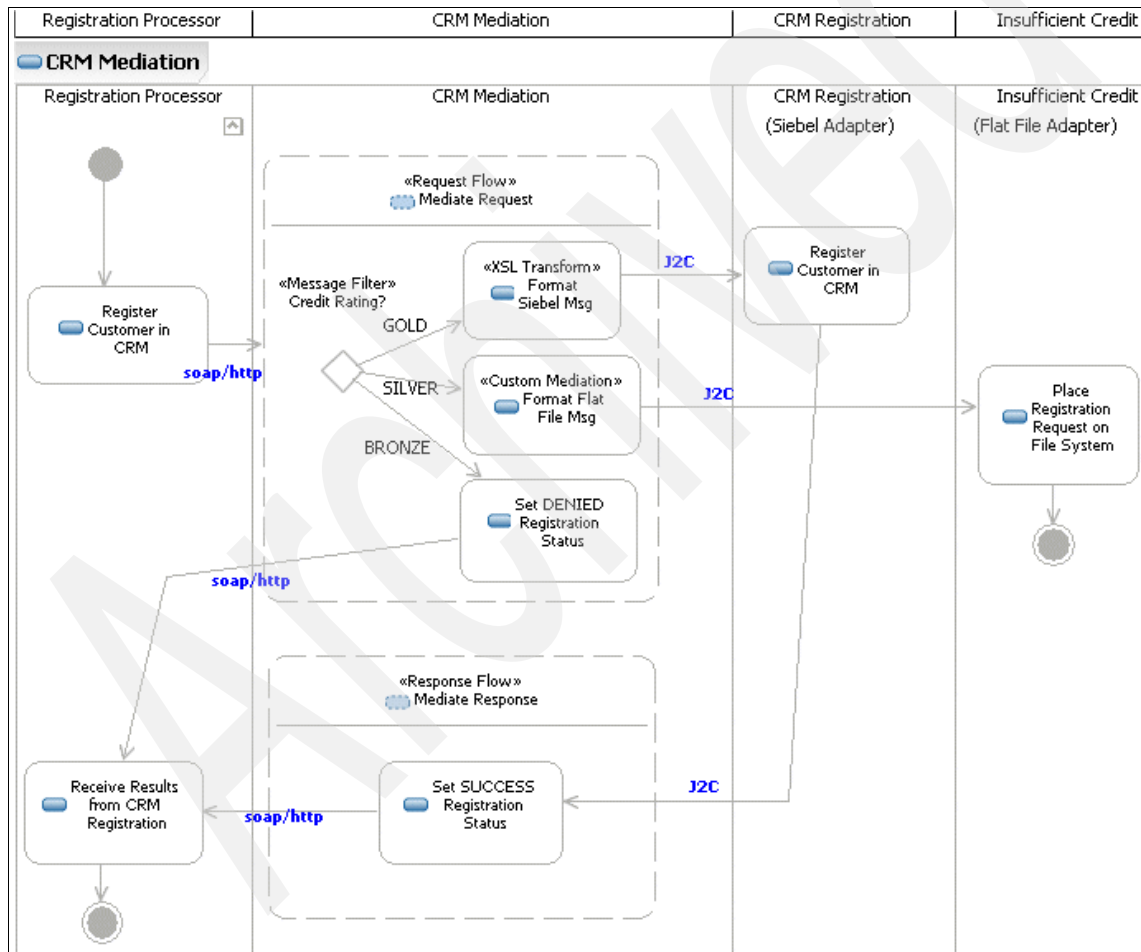


Figure 5-31 CRM mediation activity diagram

Register Shipping mediation

The mediation for the shipping registration service, called Register Shipping mediation, takes an input parameter of an array of shipping addresses to register. Because the Register Shipping Service can only register a single shipping address at a time, the mediation will decompose the message into separate messages, each containing a single address from the array to be passed as request messages to the Register Shipping Service.

The activity diagram in Figure 5-32 shows this mediation and indicates that all communication with the mediation service and the Register Shipping Service is asynchronous with the use of SOAP/JMS.

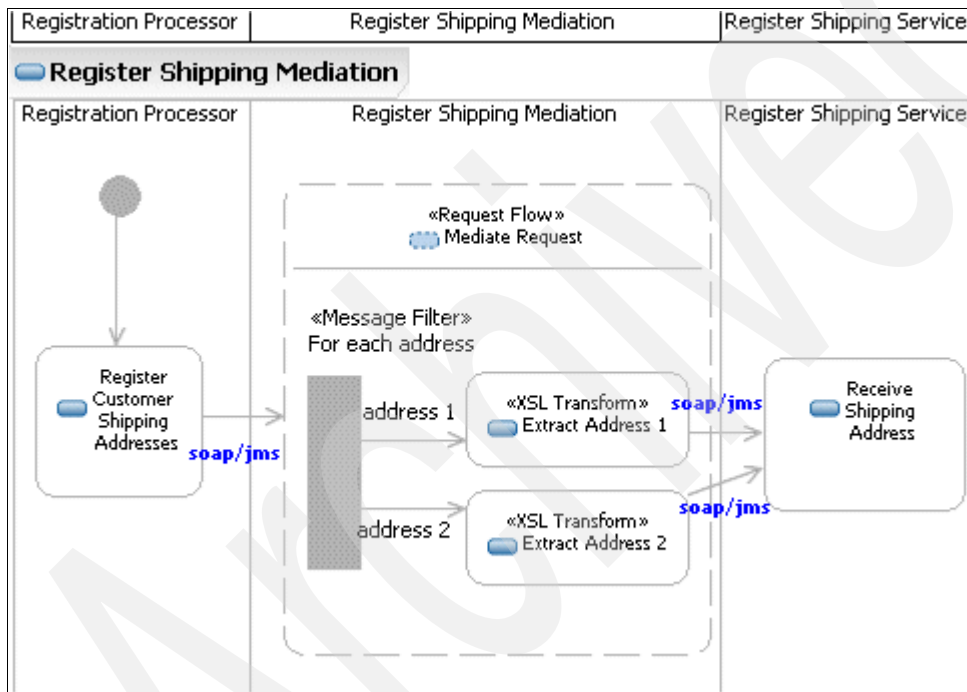


Figure 5-32 Register Shipping mediation activity diagram

Log Registration mediation

The last step in the customer registration process is to log the registration details in the audit system. The audit system maintains registration log messages in three different queues, depending on the results of the registration process. We use a mediation to provide a single service interface with one operation that takes the customer registration details and routes the log message to the appropriate queue based on the registration status.

This mediation is another example of the Router application pattern. Also, by providing an ESB mediation between the caller and the audit system, we can effectively expose the audit system via an asynchronous SOAP/JMS Web service, while the actual communication to the audit system is through the JMS API.

The activity diagram in Figure 5-33 shows how this mediation routes messages to either the success, denied, or failure queues, depending on the registration status in the registration log message.

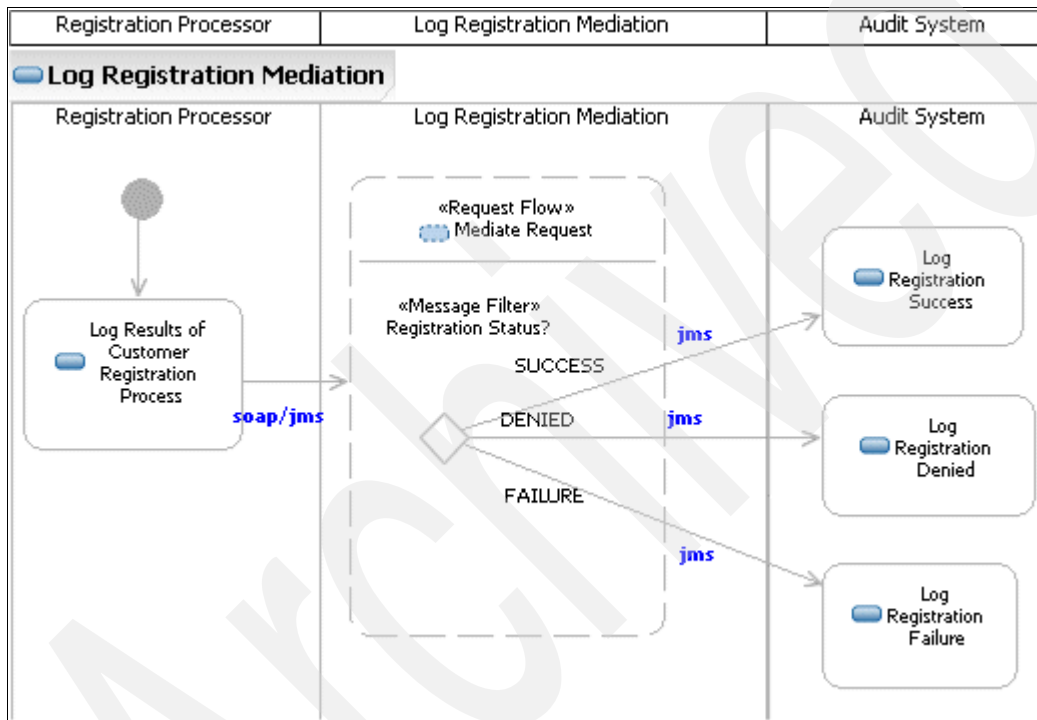


Figure 5-33 Log Registration activity diagram

5.7 Modeling business objects: Transform UML to XSD

Rational Software Architect provides an XSD model template for defining your XML schema in UML. An XSD model has a UML profile called *XSDProfile* applied to it. This profile allows you to apply a XSD *schema* stereotype to packages and specify XSD types (primitives, simple types, complex types) as attribute types in your classes. Then once you have created a model of the classes that logically represent the data entities required for your application, you

can apply a UML to XSD transformation to your model, which will generate an XSD representation of your model.

This type of model is perfect for modeling data, or *business objects*, that will be passed between components in your ESB. You can describe your data entities in a normal UML class diagram, then transform them into XML schema that can be imported directly into WebSphere Integration Developer, where these entities will be represented as business objects and can be used in the assembling of ESB mediations. The diagram in Figure 5-34 is an UML diagram of the Customer class and its associated classes. The Customer class is the main data entity used through the ITSOMart sample application.

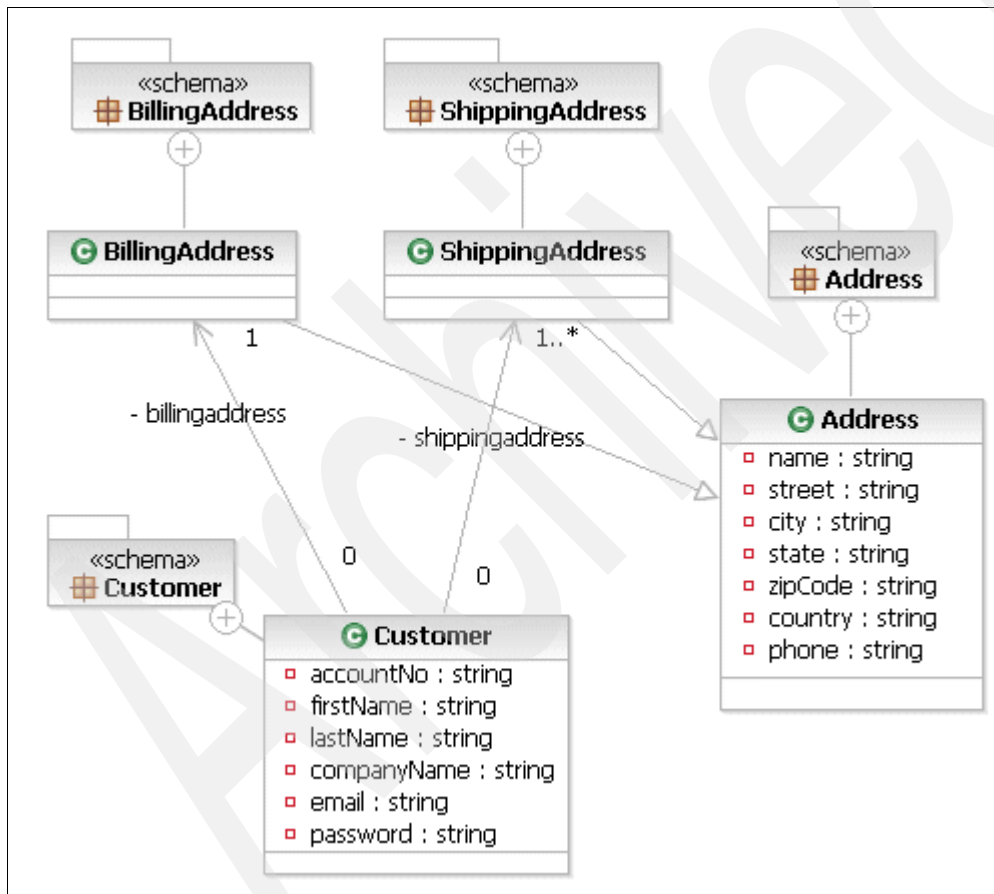


Figure 5-34 XSD model

Notice that you can have all the normal UML notations for object-oriented design such as generalizations (or inheritance) and associations between classes.

These classes are contained in packages that have a stereotype of *schema*. You can specify the XML namespace for schemas in the properties of the schema stereotype.

In this example, each class is contained in a separate package. This is not a requirement for the UML-to-XSD transformation tooling. We only represented our model this way to show that a separate XSD file will be generated for each package in the model, and where there are associations across packages, the XSD will correctly import the associated schemas.

Example 5-1 is what the XSD looks like that was generated by applying the UML-to-XSD transformation on this model.

Example 5-1 Customer.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:billingAddress="http://ITSMartLib/BillingAddress" xmlns:customer="http://ITSMartLib/Customer"
xmlns:shippingAddress="http://ITSMartLib/ShippingAddress" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://ITSMartLib/Customer">
  <xsd:import namespace="http://ITSMartLib/ShippingAddress" schemaLocation="ShippingAddress.xsd"/>
  <xsd:import namespace="http://ITSMartLib/BillingAddress" schemaLocation="BillingAddress.xsd"/>
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element name="accountNo" type="xsd:string"/>
      <xsd:element name="firstName" type="xsd:string"/>
      <xsd:element name="lastName" type="xsd:string"/>
      <xsd:element name="companyName" type="xsd:string"/>
      <xsd:element name="email" type="xsd:string"/>
      <xsd:element name="password" type="xsd:string"/>
      <xsd:element name="billingaddress" type="billingAddress:BillingAddress"/>
      <xsd:element maxOccurs="unbounded" name="shippingaddress" type="shippingAddress:ShippingAddress"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

The following sections show you the basic steps for creating a XSD model of your data entities, generating the XSD representation of those entities, and importing the XSD into WebSphere Integration Developer where the entities can be used as business objects. The steps are:

1. Create an XSD model.
2. Create a package.
3. Create a class.
4. Create a class diagram.
5. Run the UML to XSD transformation.
6. Figure 5.7.6 on page 178.

5.7.1 Create an XSD model

To create an XSD model:

1. Open the Modeling perspective.
2. Create a UML Project if you do not already have one in your workspace by selecting **File** → **Project** → **UML Project**.

3. Right-click your UML Project in the Model Explorer view and select **New** → **UML Model**.
4. In the New UML Model window, under Templates, select **XSD Model**, then **Finish** (Figure 5-35).

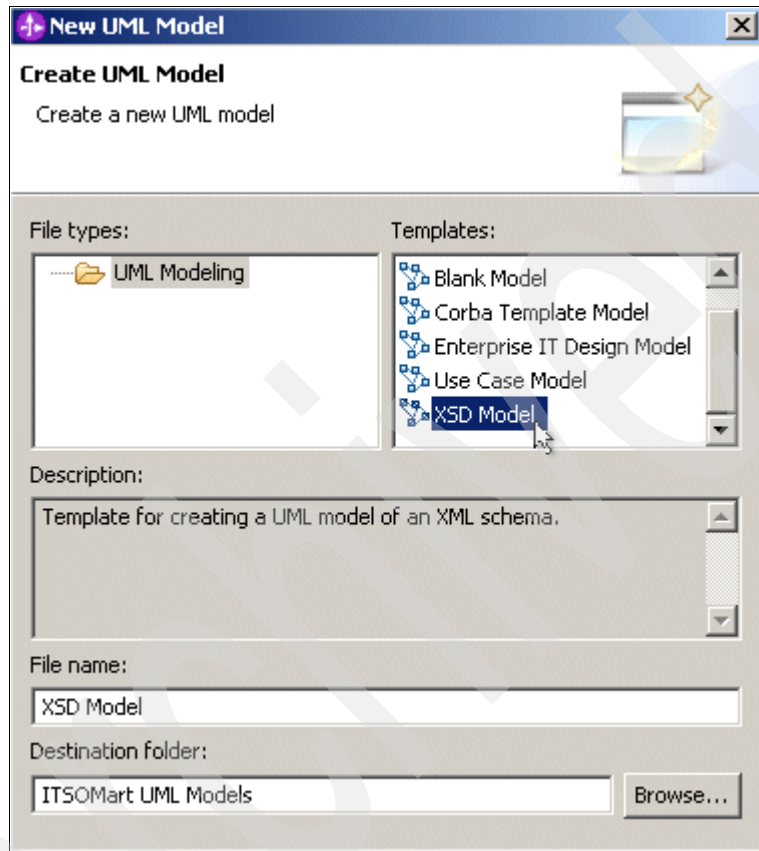


Figure 5-35 New XSD model

The model is created and opened. In the UML Model Editor, you can see under Applied Profiles that the XSDProfile is applied to this model.

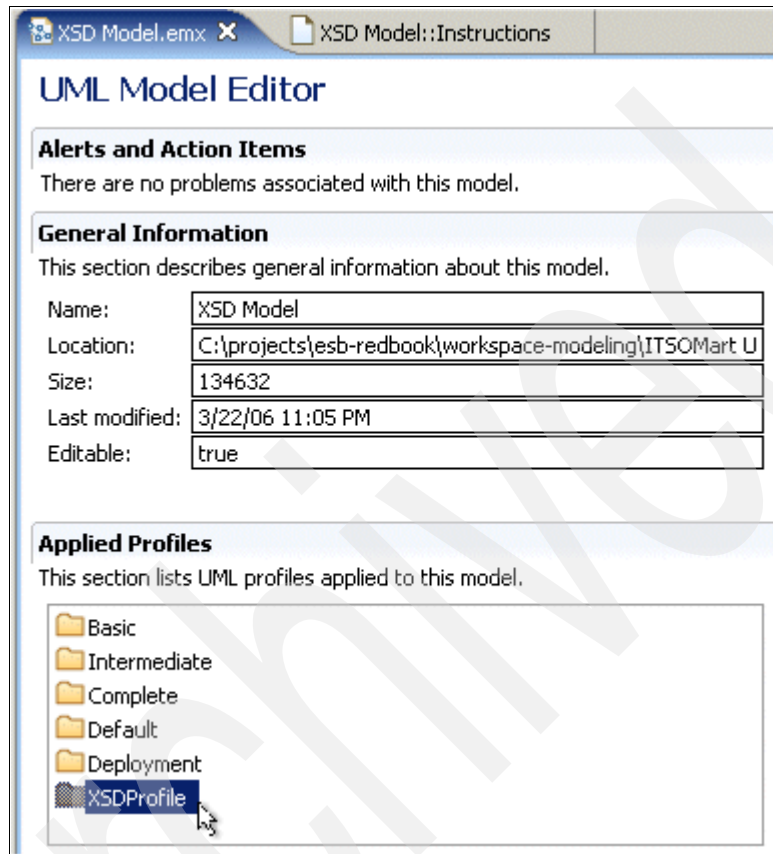


Figure 5-36 XSD profile

5.7.2 Create a package

To create a package:

1. Right-click **XSD Model.emx** → **XSD Model**, and select **Add UML** → **Package**. Name the package **Address**.
2. In the properties for the address package, select the **Profiles** tab, then **Add Profile**, and select the profile **XSD Transformation** (Figure 5-37).

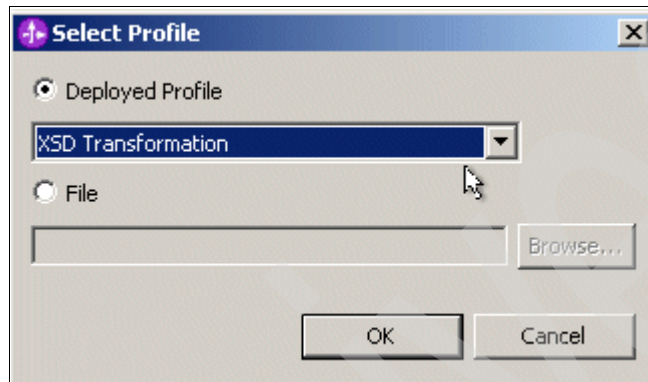


Figure 5-37 XSD Transformation profile

You will now see the XSDProfile under Applied Profiles.

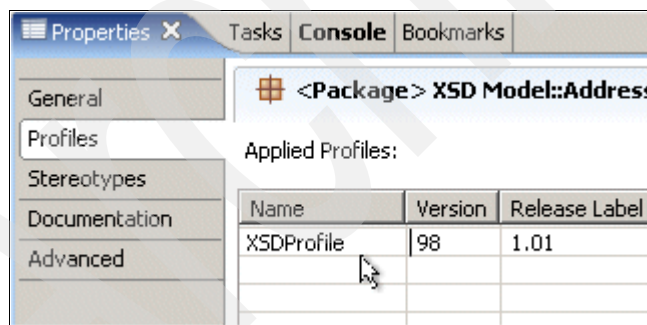


Figure 5-38 XSD profile

3. Still in the properties for the address package, select the **Stereotypes** tab, then **Add Stereotypes**. In the Apply Stereotypes window, select **schema** (Figure 5-39).

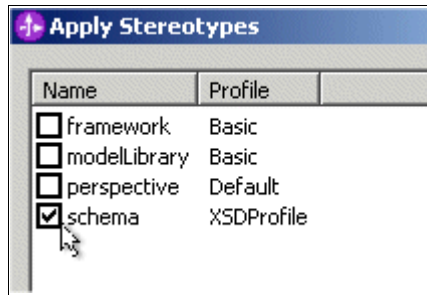


Figure 5-39 Apply schema stereotype to package

4. In the stereotype properties, set the schema target namespace properties (Figure 5-40):
- targetNamespace: http://ITSOMartLib/Address
 - targetNamespacePrefix: address

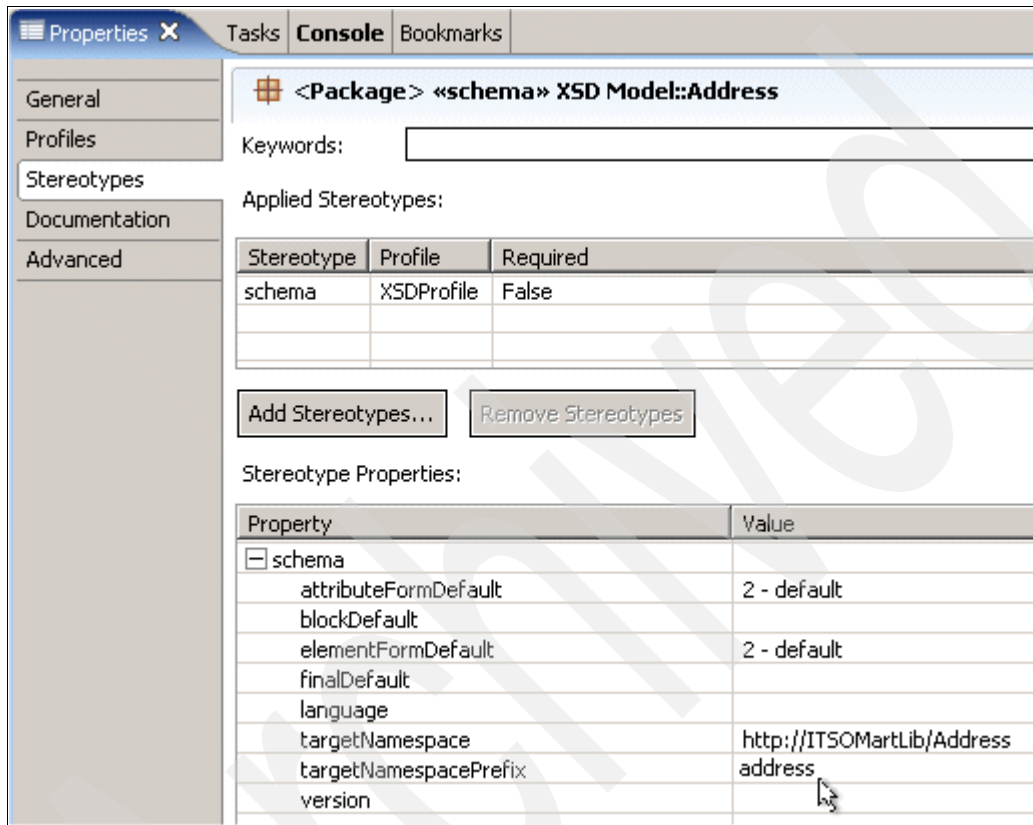


Figure 5-40 Schema target namespace

5.7.3 Create a class

To create a class:

1. Right-click <<schema>> **Address** and select **Add UML → Class**. Name the class Address.

2. In the Properties for the Address class, select the **Attributes** tab, then click the little red square to the right to add an attribute to the class (Figure 5-41).

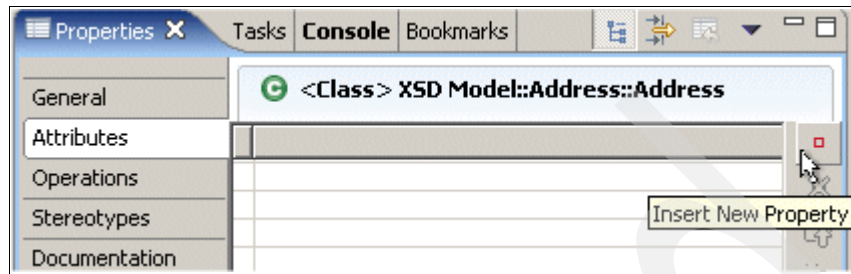


Figure 5-41 Insert new property

3. Give the attribute a name of name. Set the type by clicking the box under Type and select **string** under XSDDataTypes.
Notice that all of the standard XSD types are available.

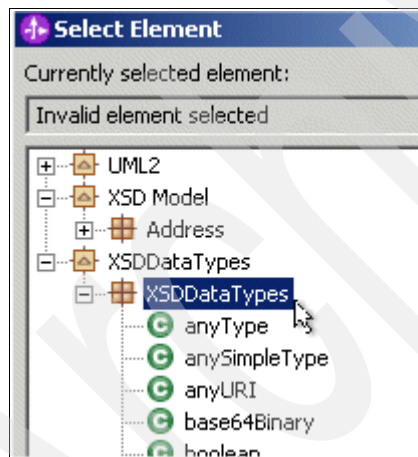


Figure 5-42 XSDDataTypes

The name attribute will look like that shown in Figure 5-43 in the Attribute properties.

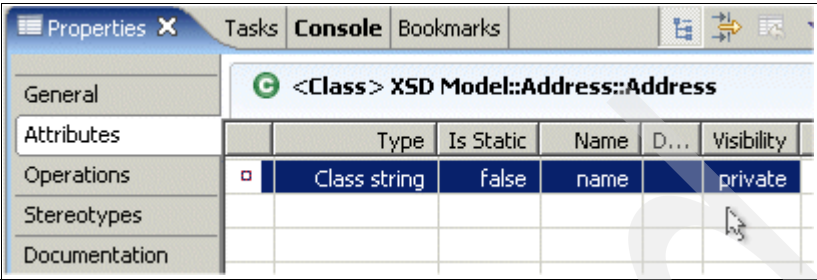


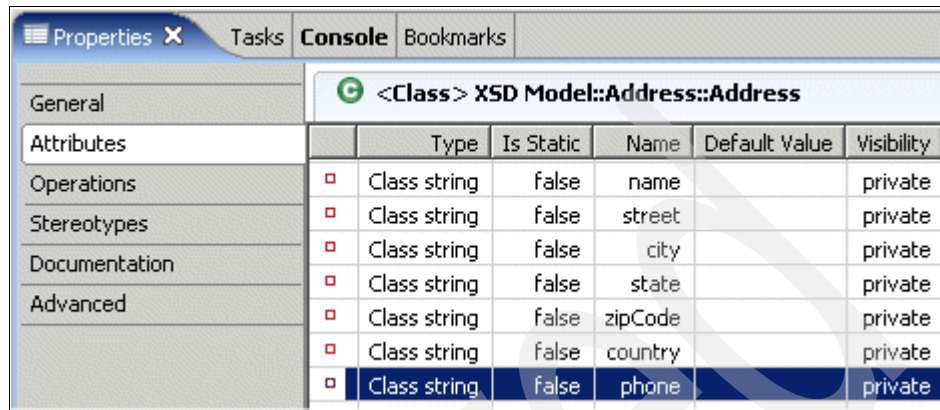
Figure 5-43 Name attribute

4. Add the attributes listed in Table 5-1 to the Address class.

Table 5-1 Attributes

Name	Type
street	string
city	string
state	string
zipCode	string
country	string
phone	string

The attribute properties for the Address class will look like those shown in Figure 5-44.



The screenshot shows the 'Properties' window of an IDE. The 'Attributes' tab is selected, displaying a table of attributes for the class '<Class> XSD Model::Address::Address'. The table has columns for 'Type', 'Is Static', 'Name', 'Default Value', and 'Visibility'. There are seven attributes listed, all of type 'Class string' and 'private' visibility. The 'phone' attribute is highlighted in blue.

	Type	Is Static	Name	Default Value	Visibility
<input type="checkbox"/>	Class string	false	name		private
<input type="checkbox"/>	Class string	false	street		private
<input type="checkbox"/>	Class string	false	city		private
<input type="checkbox"/>	Class string	false	state		private
<input type="checkbox"/>	Class string	false	zipCode		private
<input type="checkbox"/>	Class string	false	country		private
<input type="checkbox"/>	Class string	false	phone		private

Figure 5-44 Address attributes

5.7.4 Create a class diagram

To create a class diagram:

1. Under your model project, right-click **XSD Model.emx** → **XSD Model**, and select **Add Diagram** → **Class Diagram**. Name the diagram B0 Diagram.

2. Drag and drop the **<<schema>> Address** package and the **Address** class from the Model Explorer view onto the BO Diagram. The diagram will now look like Figure 5-45.

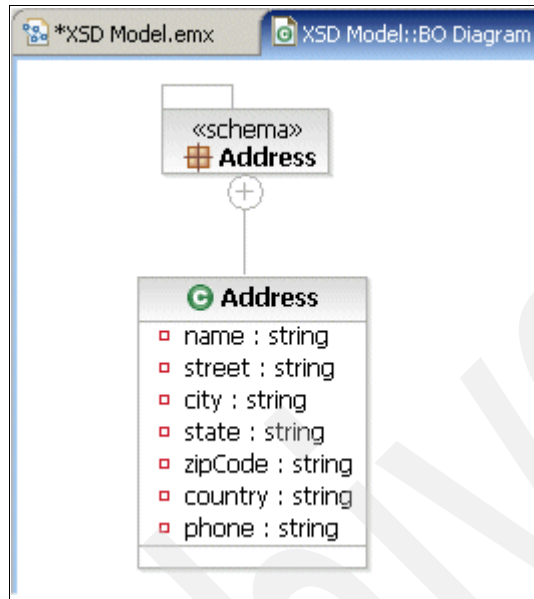


Figure 5-45 BO Diagram

5.7.5 Run the UML to XSD transformation

To run the transformation:

1. Right-click anywhere inside the BO Diagram and select **Transform** → **Run Transformation** → **UML to XSD**.

2. In the Run Transformation window (Figure 5-46) select a target project to generate the XSD into, then click **Run**.

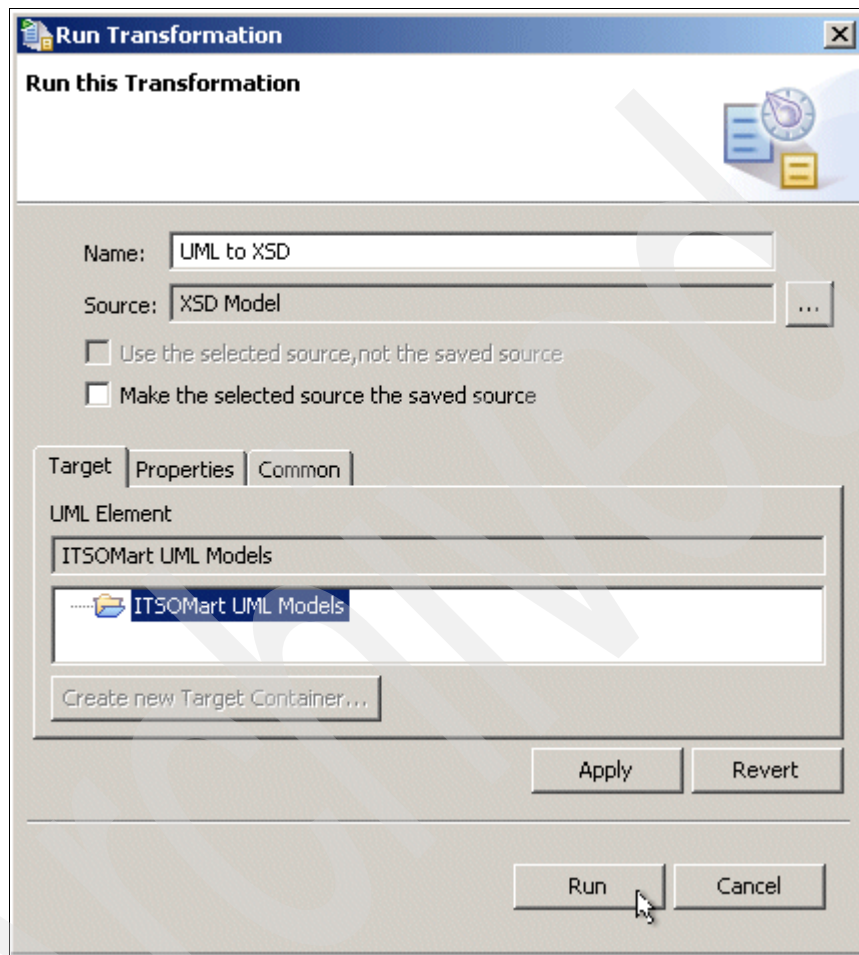


Figure 5-46 Run UML-to-XSD transformation

If you have the XSD Building Blocks package that was automatically added to your XSD model when you created it, then you will see the error shown in Figure 5-47 when running the UML-to-XSD transformation.

Click **OK** and ignore the error. This package does not have anything inside of it to generate a XML schema from. The XSD for the address package will still be generated.

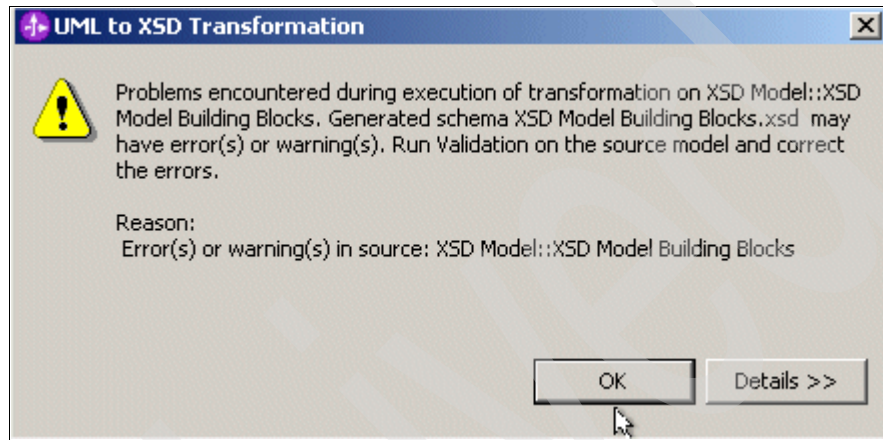


Figure 5-47 Error generating XSD for XSD Model Building Blocks package

Look at the generated XSD

XSD files are not displayed in the Model Explorer, so you need to switch to a different view to see the XSD that was generated for your model.

1. Add the Navigator view to your Modeling perspective by selecting **Window** → **Show View** → **Other** → **Basic** → **Navigator**. The Navigator view gets added to the bottom pane.

2. The XML schema for your model will now be displayed in the Navigator view, under your UML project under a directory called schema.

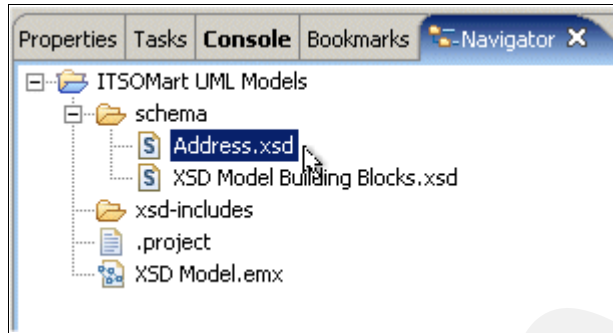


Figure 5-48 Generated XSD files

3. Open Address.xsd.

Notice that the schema has the targetNamespace and prefix that you specified in the schema stereotype properties, and the Address class is represented as a complex type.

Example 5-2 Address.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:address="http://ITSOMartLib/Address"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ITSOMartLib/Address">
  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state" type="xsd:string"/>
      <xsd:element name="zipCode" type="xsd:string"/>
      <xsd:element name="country" type="xsd:string"/>
      <xsd:element name="phone" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

5.7.6 Import the XSD into WebSphere Integration Developer

Now you have XML schema that you can bring into the your libraries or mediation modules in WebSphere Integration Developer where the complex types within this schema will be represented as business objects.

If you have Rational Software Architect and WebSphere Integration Developer installed into the same Eclipse workbench, then you can simply copy the Address.xsd from your UML project to a library or mediation module.

Otherwise, you can export the Address.xsd to the file system and then import it into WebSphere Integration Developer.

Either way, once you place the XSD file into a library or mediation module, the Business Integration view will display Address under Data Types.

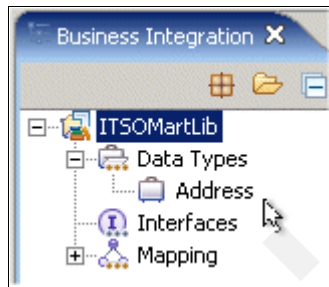


Figure 5-49 XSD in WebSphere Integration Developer

And if you double-click **Address**, it will be opened in the Business Object editor (Figure 5-50).



Figure 5-50 Address business object

5.8 Modeling messaging resources: Transform UML to JACL

The Registration Processor Service in the ITSOMart sample application is a SOAP/JMS Web service, which means that there must be JMS resources configured in WebSphere ESB to support this service. For the sample application, the Registration Log mediation will also need JMS queues to represent the audit system with which this mediation is interacting.

In this section, you will learn how to model JMS resources and how they are configured to use the service integration bus within WebSphere ESB.

Note: You must be connected to the Internet to install the WebSphere Platform Messaging Patterns Reusable Asset.

5.8.1 Import the WebSphere Platform Messaging Patterns asset

Support for modeling the service integration bus and JMS queues is not provided by default in Rational Software Architect. This support is provided by a reusable asset called WebSphere Platform Messaging Patterns that you can import into your workbench from the publicly available developerWorks® Repository.

1. Select **File** → **New** → **Other**, then select **RAS** → **DeveloperWorks Repository** (Figure 5-51).

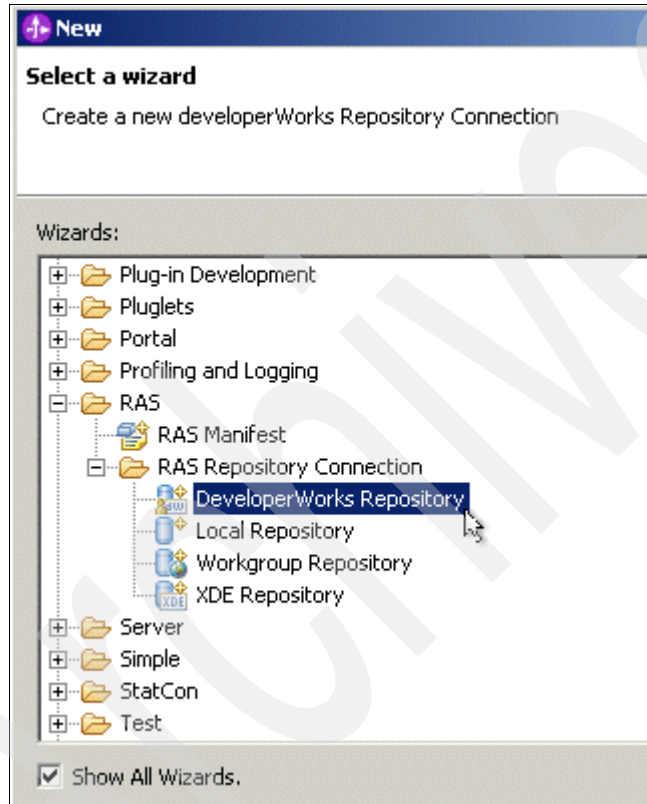


Figure 5-51 DeveloperWorks Repository

2. Click **Next**. You will be prompted to enable the Reusable Asset Management capability if you do not already have this enabled for your workspace. Click **OK** to enable it (Figure 5-52).

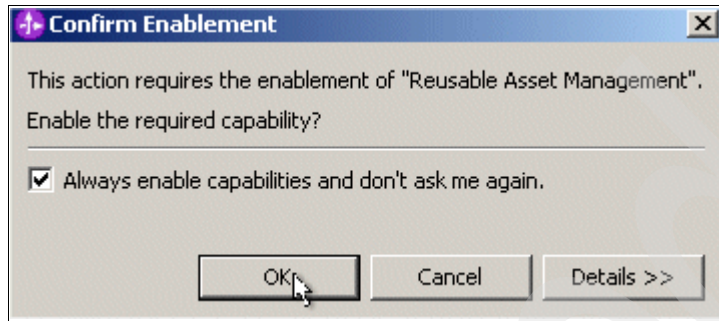


Figure 5-52 Enable Reusable Asset Management capability

3. Click **Finish** on the New IBM Rational developerWorks Repository Connection page (Figure 5-53) to create the connection to this repository.

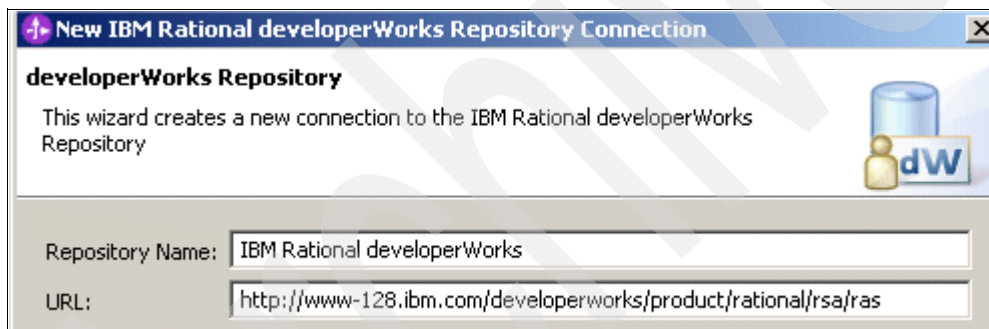


Figure 5-53 New developerWorks Repository connection

4. Open the Reusable Assets perspective by selecting **Window → Open Perspective → Other → RAS (Reusable Assets)**.

5. In the Asset Explorer, expand **IBM Rational developerWorks** → **design_soa**. Right-click **WebSphere Platform Messaging Patterns** and select **Import** (Figure 5-54).

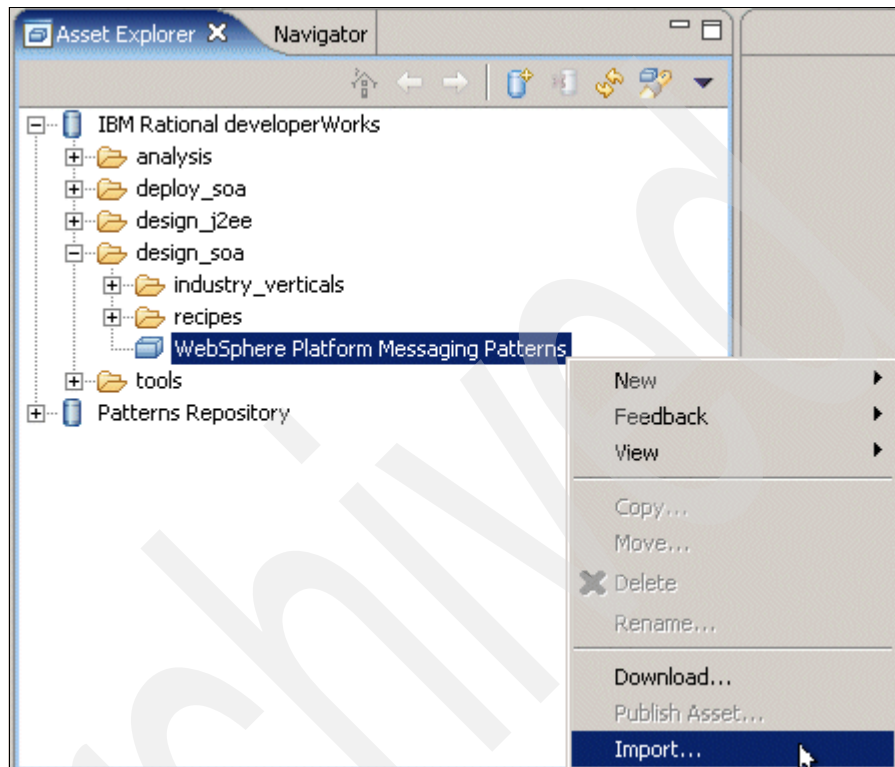


Figure 5-54 Import WebSphere Platform Messaging Patterns

6. An Information window displays and shows the plug-ins that will be imported into your Eclipse workbench (Figure 5-55). Click **OK**.

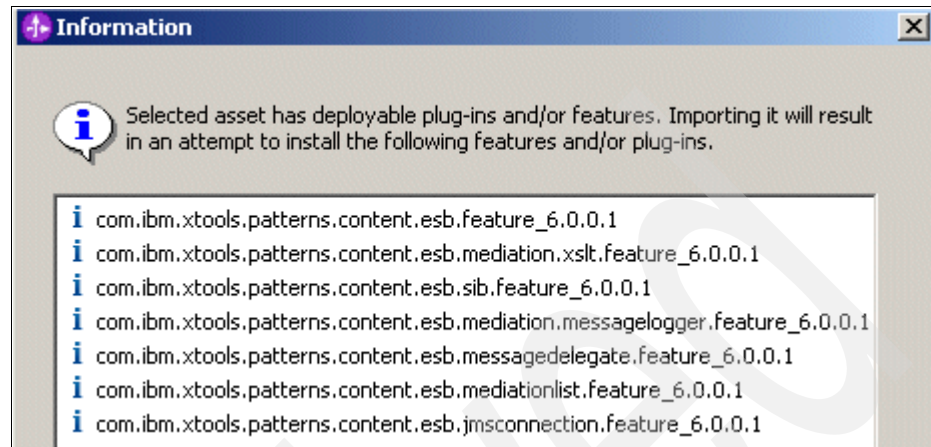


Figure 5-55 Asset plug-ins

7. In the Import RAS Asset window (Figure 5-56), click **Next**.

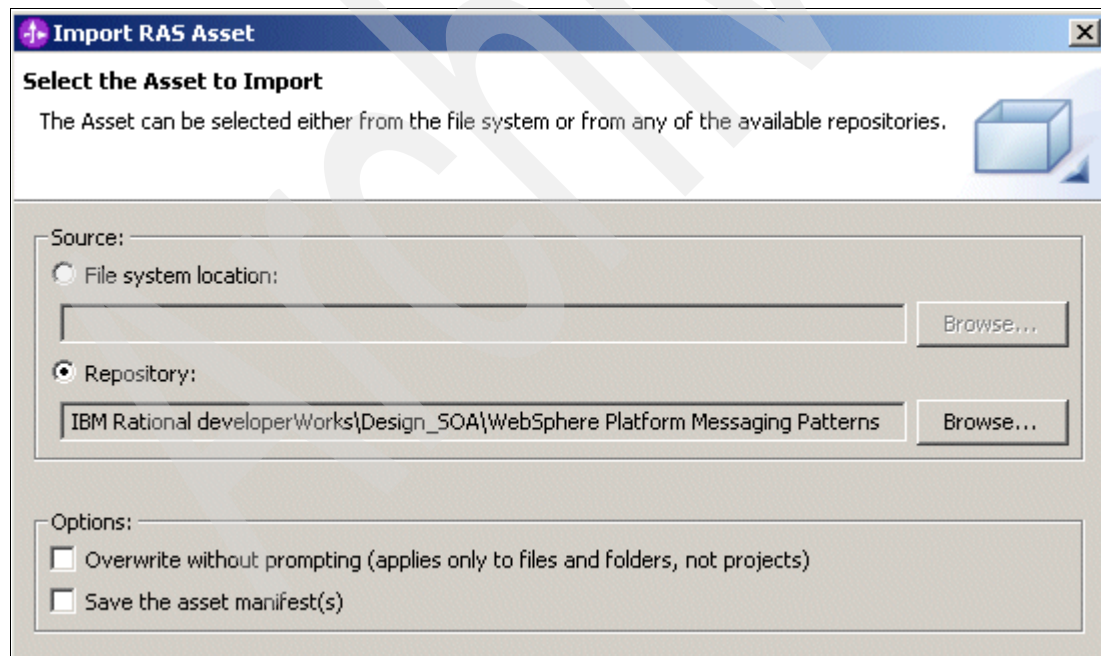


Figure 5-56 Import RAS Asset: Select the Asset to Import

- On the next page (Figure 5-57) accept the license agreement for the asset by selecting the check box, then **Next**.

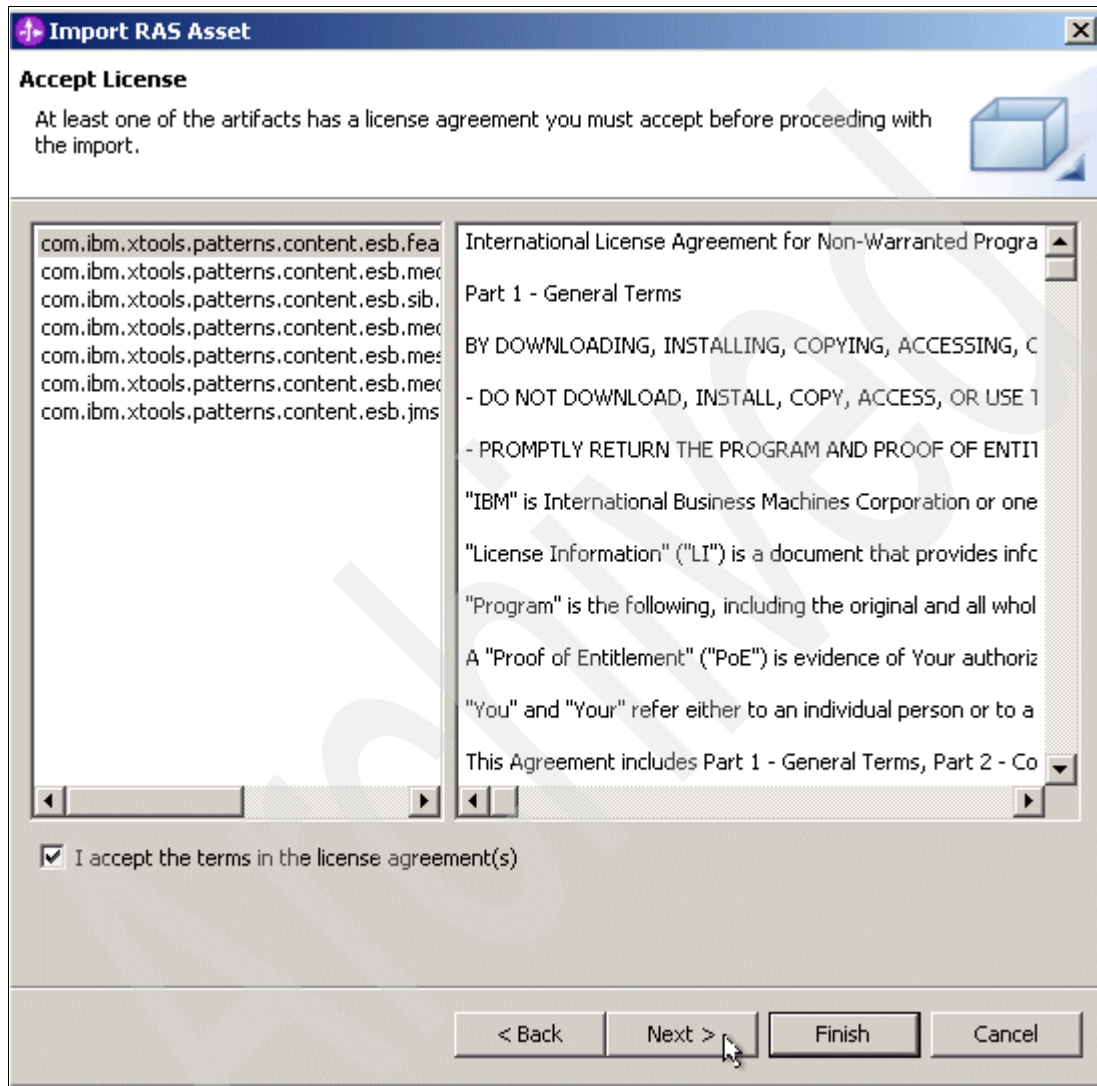


Figure 5-57 Accept license

9. The next page (Figure 5-58) shows the location in which the plug-ins for this asset will be installed to the file system. You can change this location if you like, or accept the default, and select **Finish**.

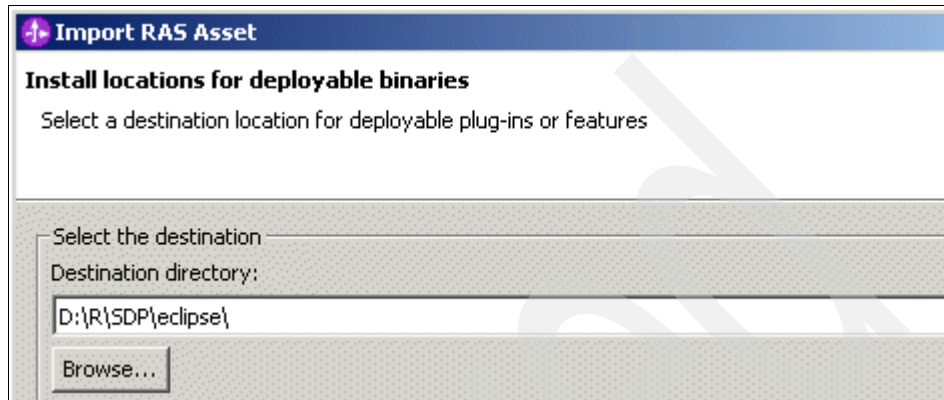


Figure 5-58 Install location

10. When the asset is finished importing, you will see a window containing messages about the import (Figure 5-59). One message indicates that you need to restart Eclipse for the asset features to be available. Click **OK**.

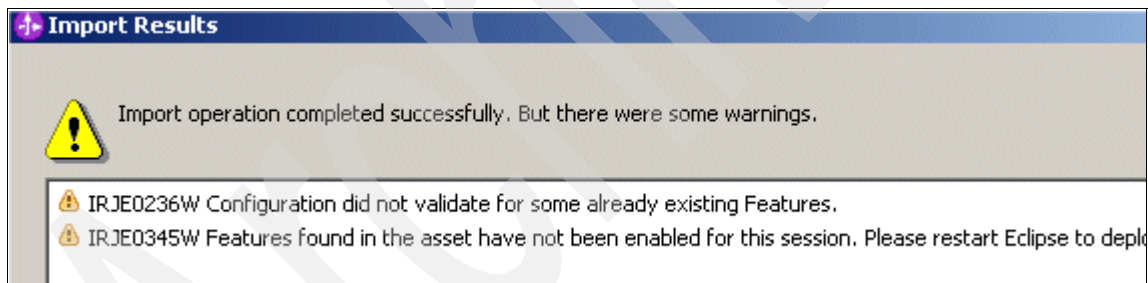


Figure 5-59 Import results

11. Another window will prompt you to restart the workbench. Click **OK**.

5.8.2 Model messaging resources

The following instructions show you how to model the service integration bus for the ITSOMart sample and the JMS resources required for the registration processor SOAP/JMS Web service.

1. Open the Modeling perspective.

2. In a UML project, create a new UML model using the Blank Model template and name it something like Deployment Model.
3. Add the Pattern Explorer view to the Modeling perspective. Select **Window** → **Show View** → **Pattern Explorer**.
4. In the Pattern Explorer view, expand **WebSphere Platform Messaging Patterns** → **Topology** and you can see pattern components for defining JMS resources and service integration bus resources. See Figure 5-60.

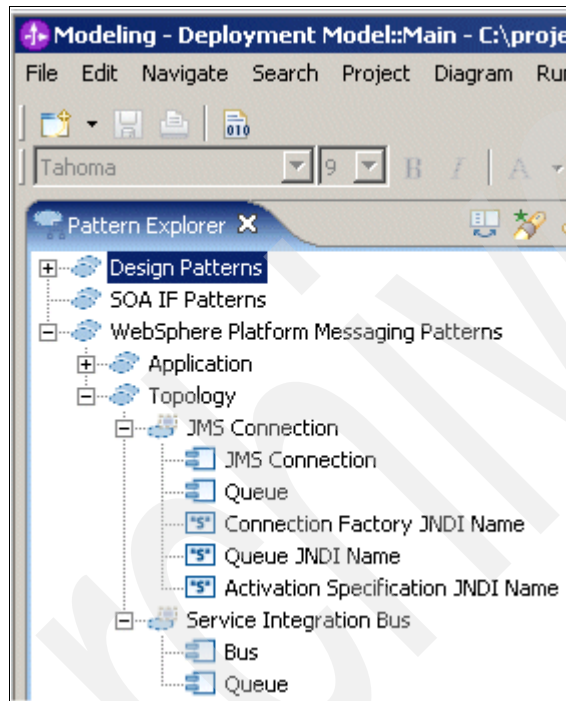


Figure 5-60 Pattern Explorer

5. Select the **Service Integration Bus** collaboration under Topology and drag and drop it onto your model diagram.

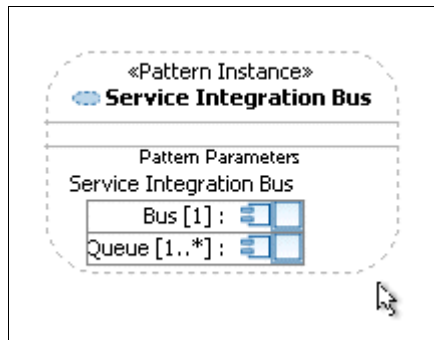
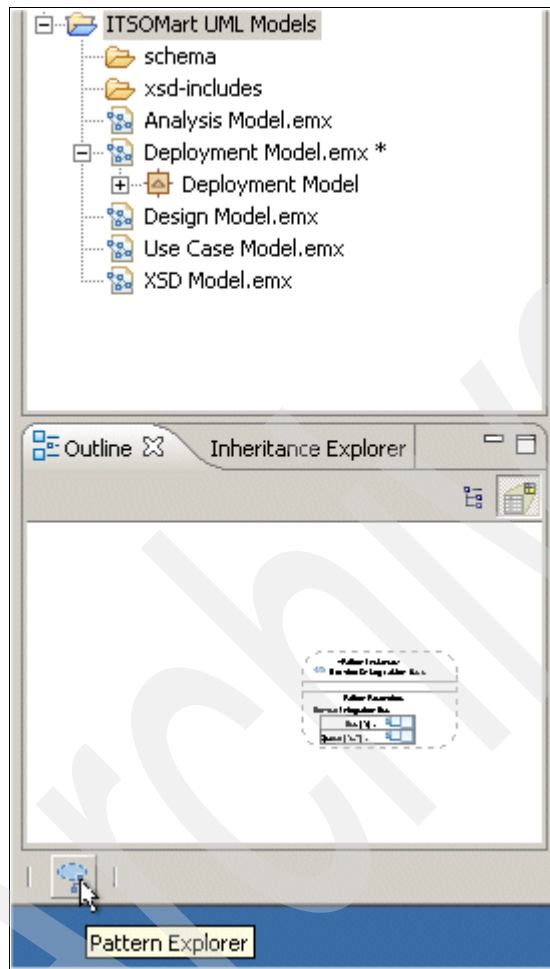


Figure 5-61 Service Integration Bus pattern

The Pattern Explorer view minimizes every time you click outside of it. To see this view again, click the small blue Pattern Explorer icon in the bottom left corner of the workspace (Figure 5-62 on page 190).



6. The Service Integration Bus pattern requires a *Bus* component and at least one *Queue* component as parameters. To add a Bus component:
 - a. In the Service Integration collaboration on the diagram, click the component icon next to the Bus parameter. Then in the pop-up menu, select the component icon.

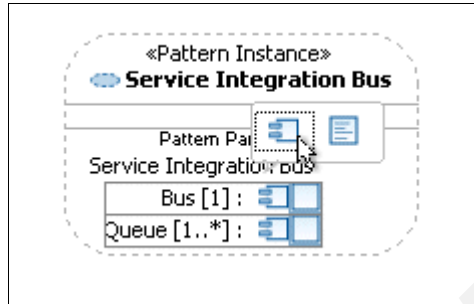


Figure 5-62 Add a Bus component

Now you will have a Bus component created in your model and it will show up in the Model Explorer view (Figure 5-63). This Bus represents the actual service integration bus instance that will be configured in your WebSphere ESB environment.

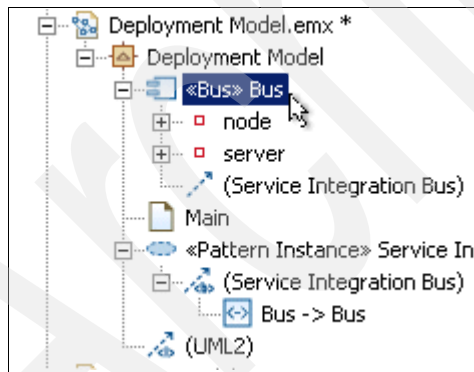


Figure 5-63 Bus component in the Model Explorer

- b. Select the **Bus** component and in the Properties view, select the **General** tab and change the component name to ITSOMartBus (Figure 5-64).

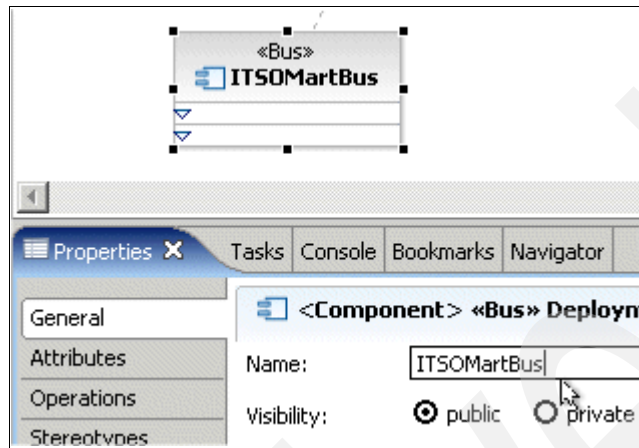


Figure 5-64 Bus name

- c. In the Model Explorer view, under the Bus component, select the **node** property. Switch to the General tab of the Properties view.

In the Default Value field, enter a name for the ESB node, for example, esbNode. This is the WebSphere node that your service integration bus will be configured on. See Figure 5-65.

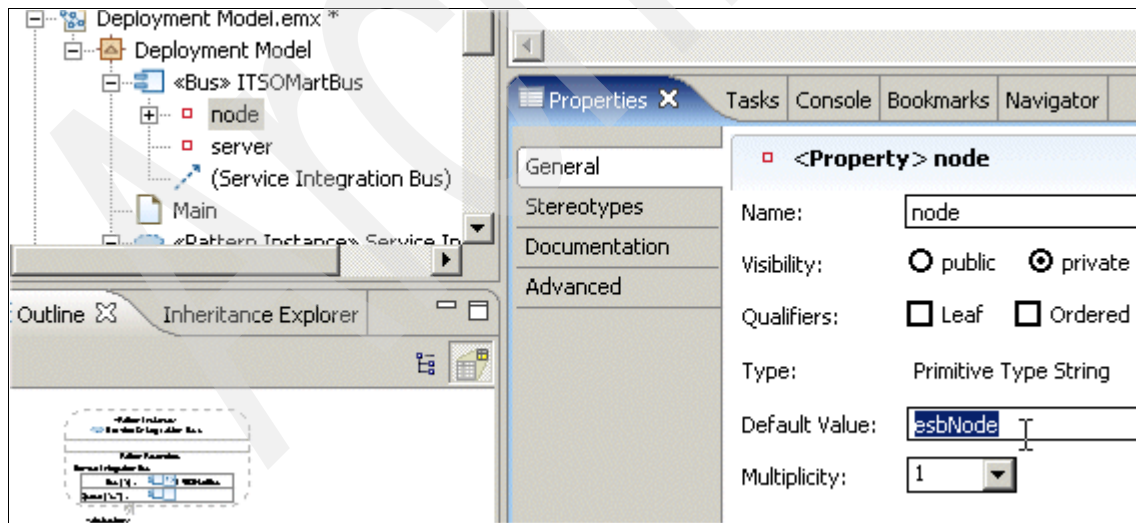


Figure 5-65 Bus node name

- d. Select the **server** property of the Bus component. In the Default Value field, set the name of the application to server1 (Figure 5-66).

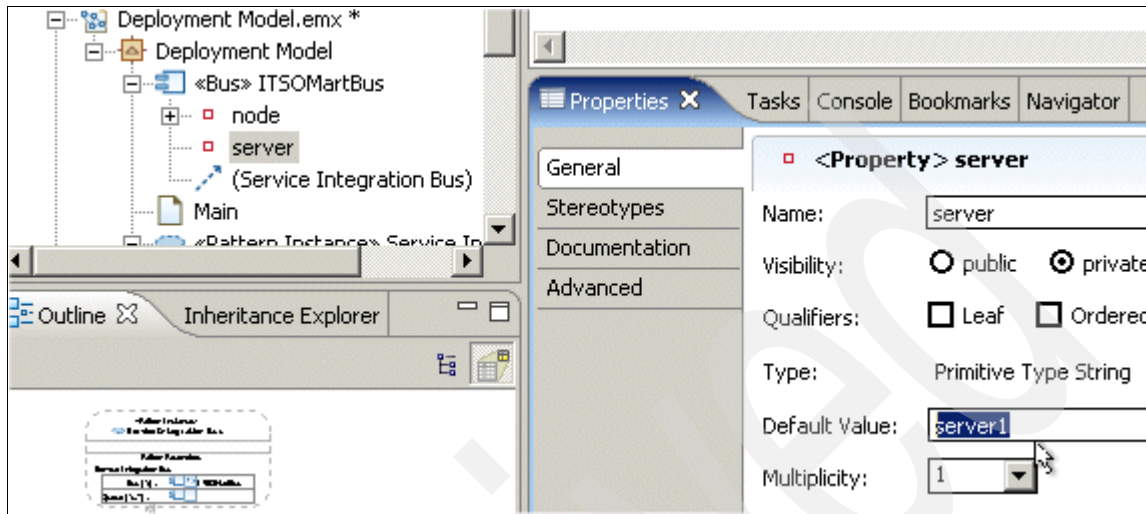


Figure 5-66 Bus server name

- e. Drag and drop the Bus component from the Model Explorer view onto your diagram.

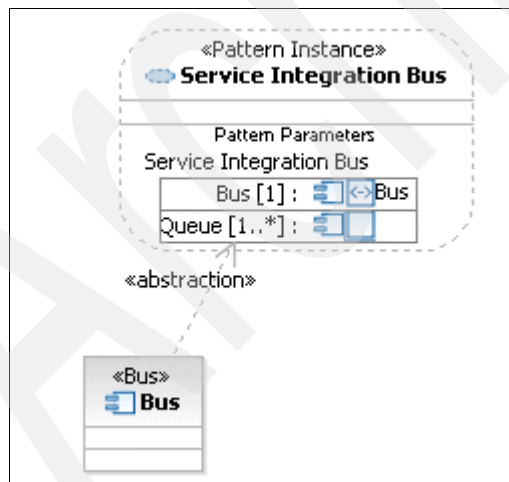


Figure 5-67 Bus component in the diagram

7. Add a Queue component to the Service Integration Bus pattern similar to how you added a Bus component as a parameter in step 6. A queue on the service integration bus is a physical destination where messages are stored.

- a. In the Service Integration collaboration on the diagram, click the component icon next to the Queue parameter. Then in the pop-up menu, select the component icon (Figure 5-68).

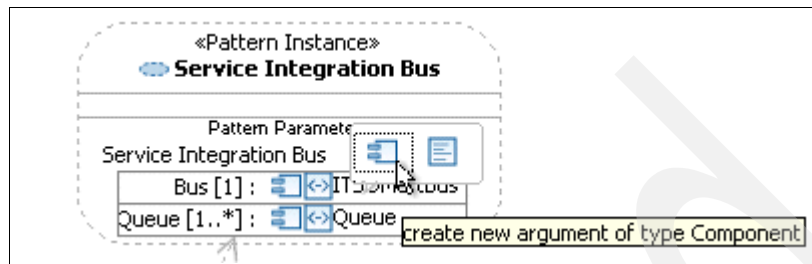


Figure 5-68 Add a Queue component

- b. Select the **Queue** component in the Model Explorer view. Use the Properties View to give it a name of ITSOMart.RegistrationProcessorServiceQ (Figure 5-69).

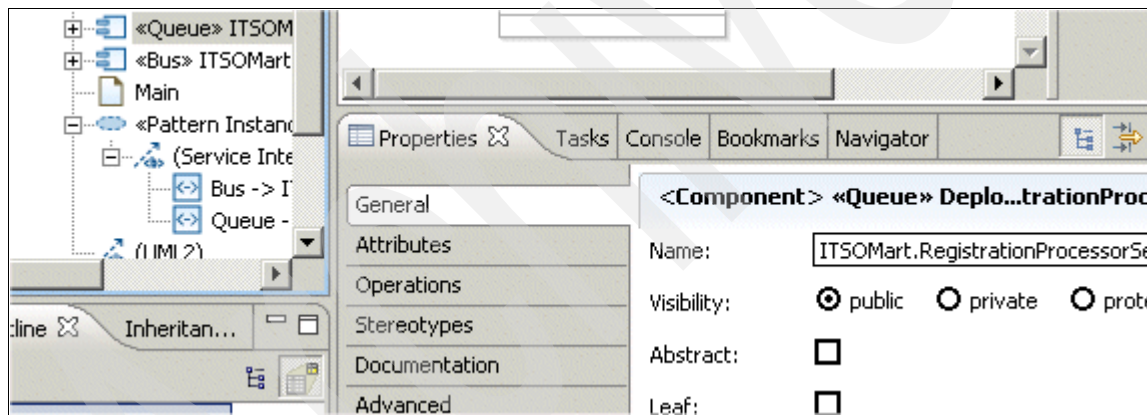


Figure 5-69 Queue name

- c. Drag and drop the Queue component onto the diagram.

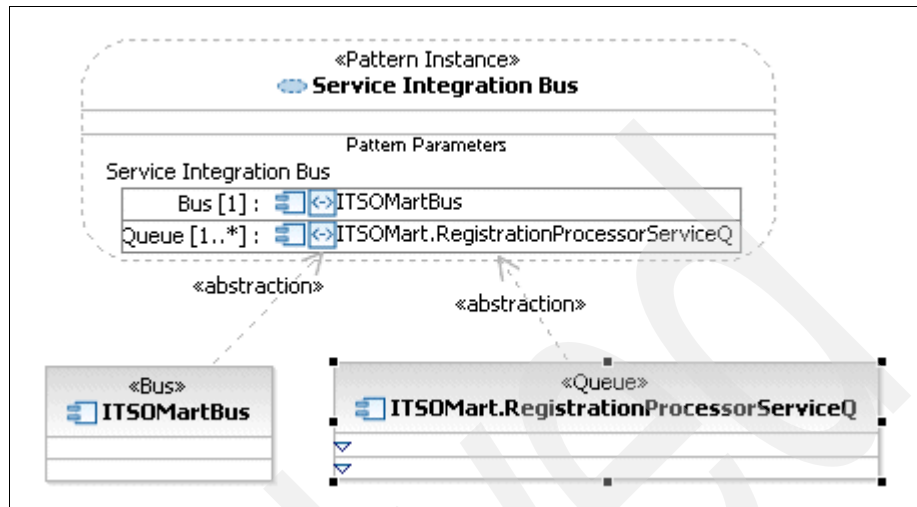


Figure 5-70 Queue component in the diagram

8. Add a JMS Connection pattern to the model.
- a. In the Pattern Explorer view, drag and drop a JMS Connection pattern onto the diagram.

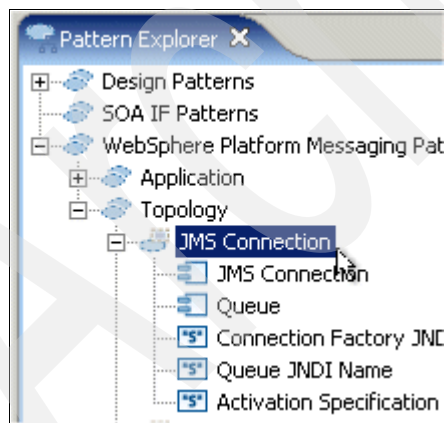


Figure 5-71 JMS Connection pattern

In the diagram, you can see that the JMS Connection pattern takes parameters for configuring the resources a J2EE application needs for connecting to a JMS queue. These parameters are:

- JMS Connection - This component is specific to the JMS Connection pattern and will not result in any actual configuration component in WebSphere ESB.
- Queue - This is a Queue component on the Service Integration Bus.
- Connection Factory JNDI Name - This is the name by which the JMS Connection Factory will be registered in JNDI.
- Queue JNDI Name - This is the name by which the JMS Queue will be registered in JNDI.
- Activation Specification JNDI Name - This is the name by which an activation specification will be registered in JNDI that will provide a listener for the queue specified for this JMS Connection pattern.

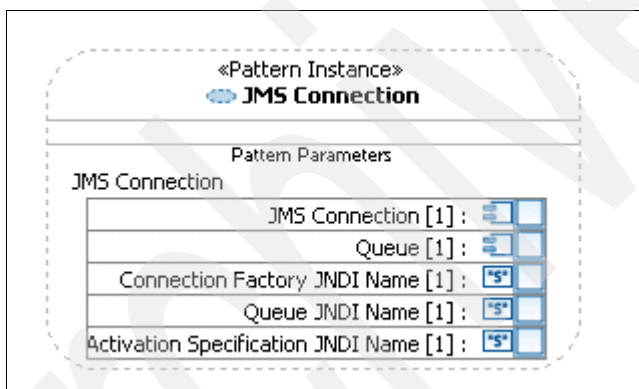


Figure 5-72 JMS Connection pattern

9. Create the JMS Connection component parameter.
 - a. Click the component icon next to the JMS Connection parameter, then in the pop-up menu, select the component icon (Figure 5-73).

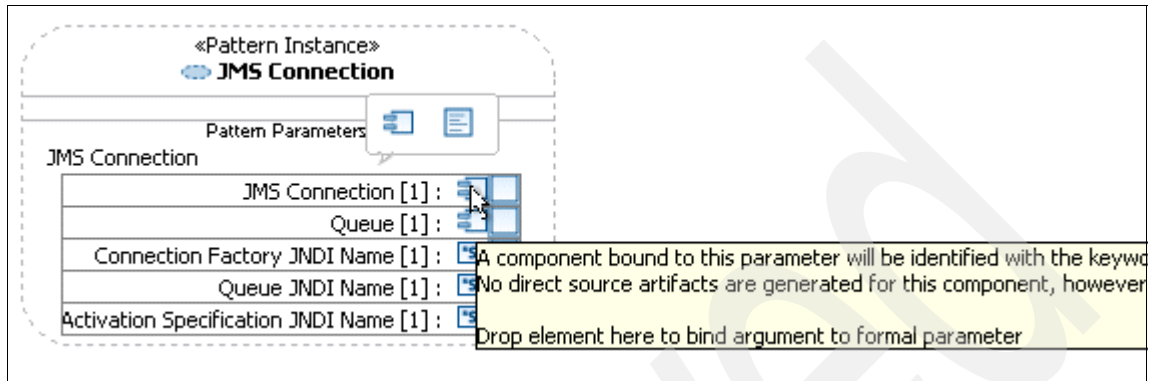


Figure 5-73 Create JMS Connection component

- b. Select the new JMS Connection component in the Model Explorer view. In the Properties view, change its name to `ITSOmart.RegistrationProcessorServiceJMSConnection`.
 - c. Drag and drop the JMS Connection component you just created onto the diagram (Figure 5-74).

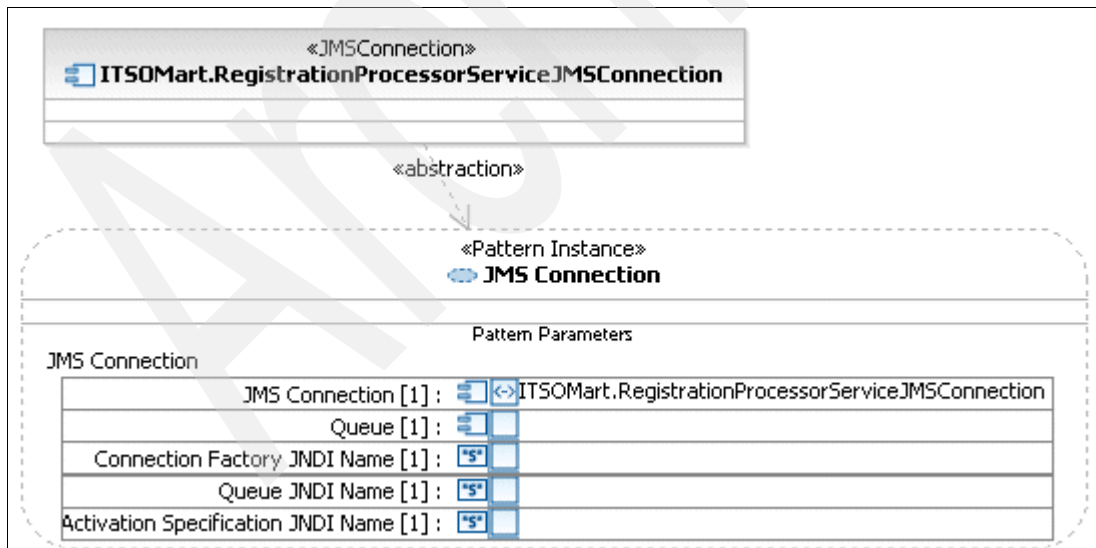


Figure 5-74 JMS Connection component in the diagram

10. Specify the Queue component parameter.

- a. Click the component icon next to the Queue parameter, then in the pop-up menu, select the icon on the right (Figure 5-75). The hover text for this icon is Enter argument name/value.

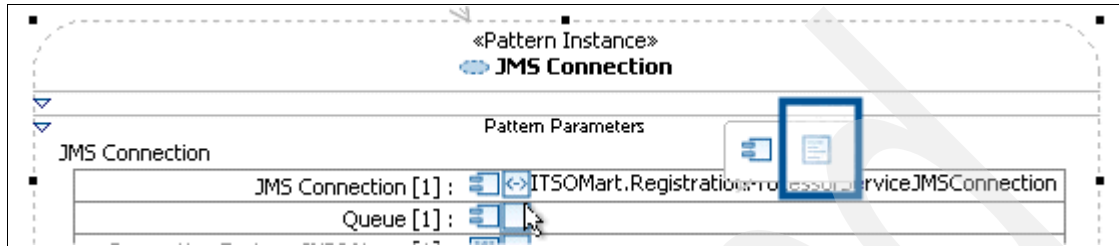


Figure 5-75 Specify Queue component

- b. In the Queue parameter field, type in the name of the queue that you defined on the service integration bus in step 7 above (Figure 5-76):
ITSOMart.RegistrationProcessorServiceQ

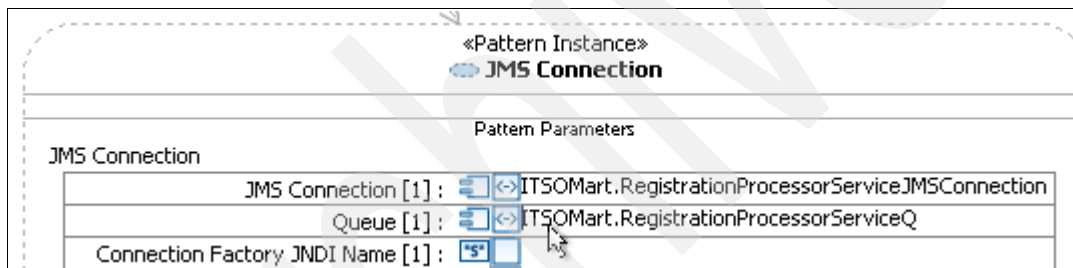


Figure 5-76 Queue name

- c. To show the association between the Queue component and the JMS Connection pattern in the diagram, right-click the JMS Connection pattern and select **Filters** → **Show/Hide Relationships**.

In the Show/Hide Relationships window, make sure Associations is checked then click **OK**.

The diagram will now show the association with the Queue component (Figure 5-77).

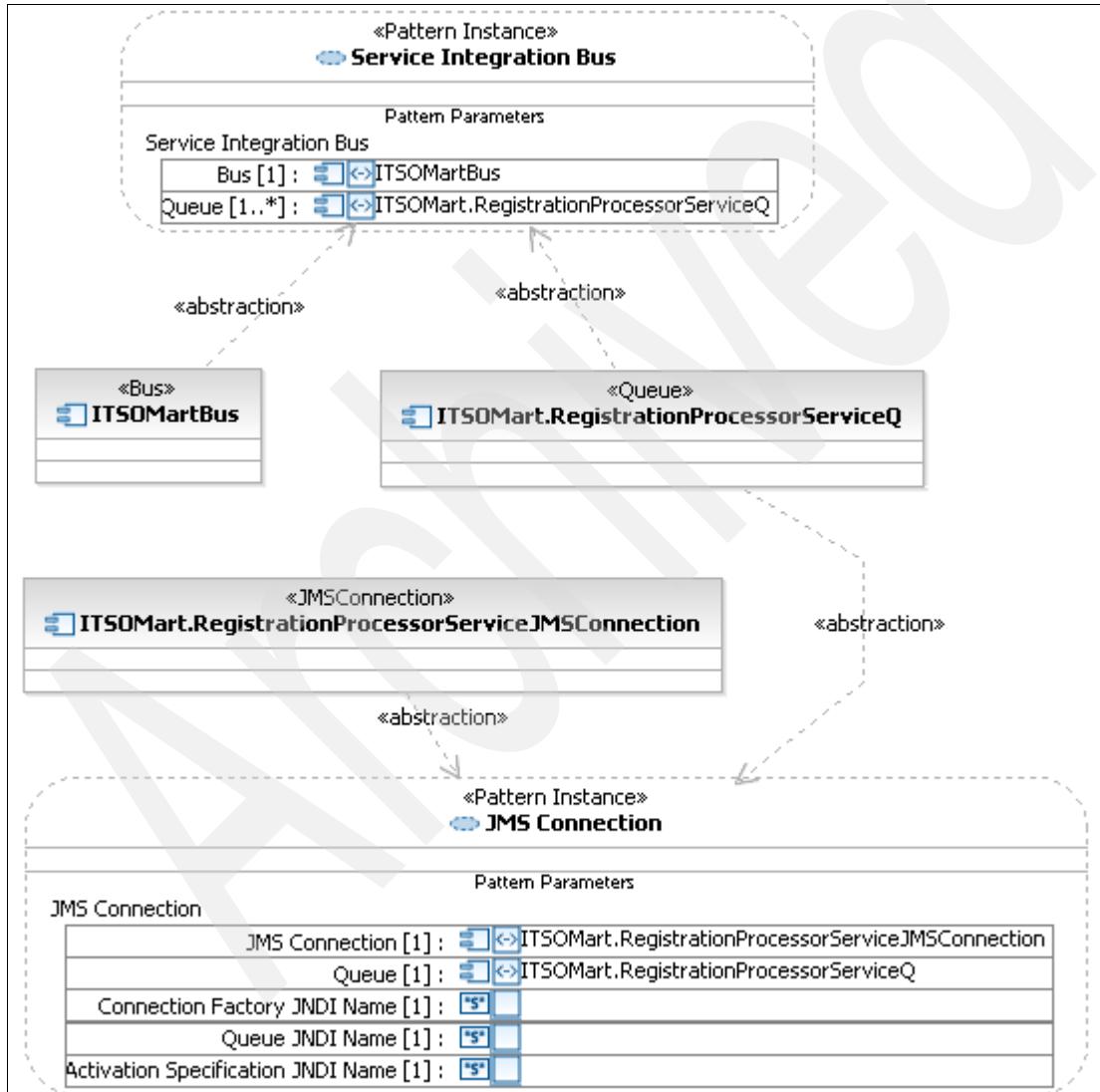


Figure 5-77 Association between the Queue component and JMS Connection pattern

11. Specify the Connection Factory JNDI Name parameter.

- a. Click the “S” icon next to the Connection Factory JNDI Name parameter, then in the pop-up menu, select the “S” icon to create a literal string for the parameter (Figure 5-78).

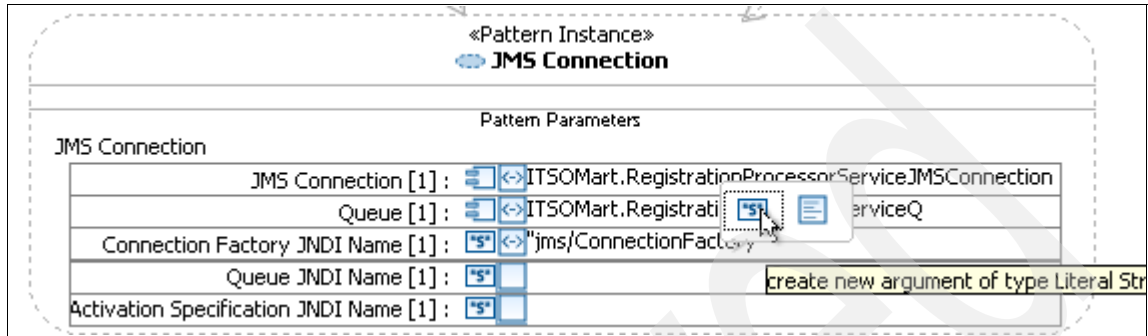


Figure 5-78 Specify Connection Factory JNDI Name

- b. Select the Connection Factory JNDI Name field, then in the Properties view, select the **Advanced** tab, and under the UML properties, change the Value field to `jms/RegistrationProcessorServiceQCF` (Figure 5-79).

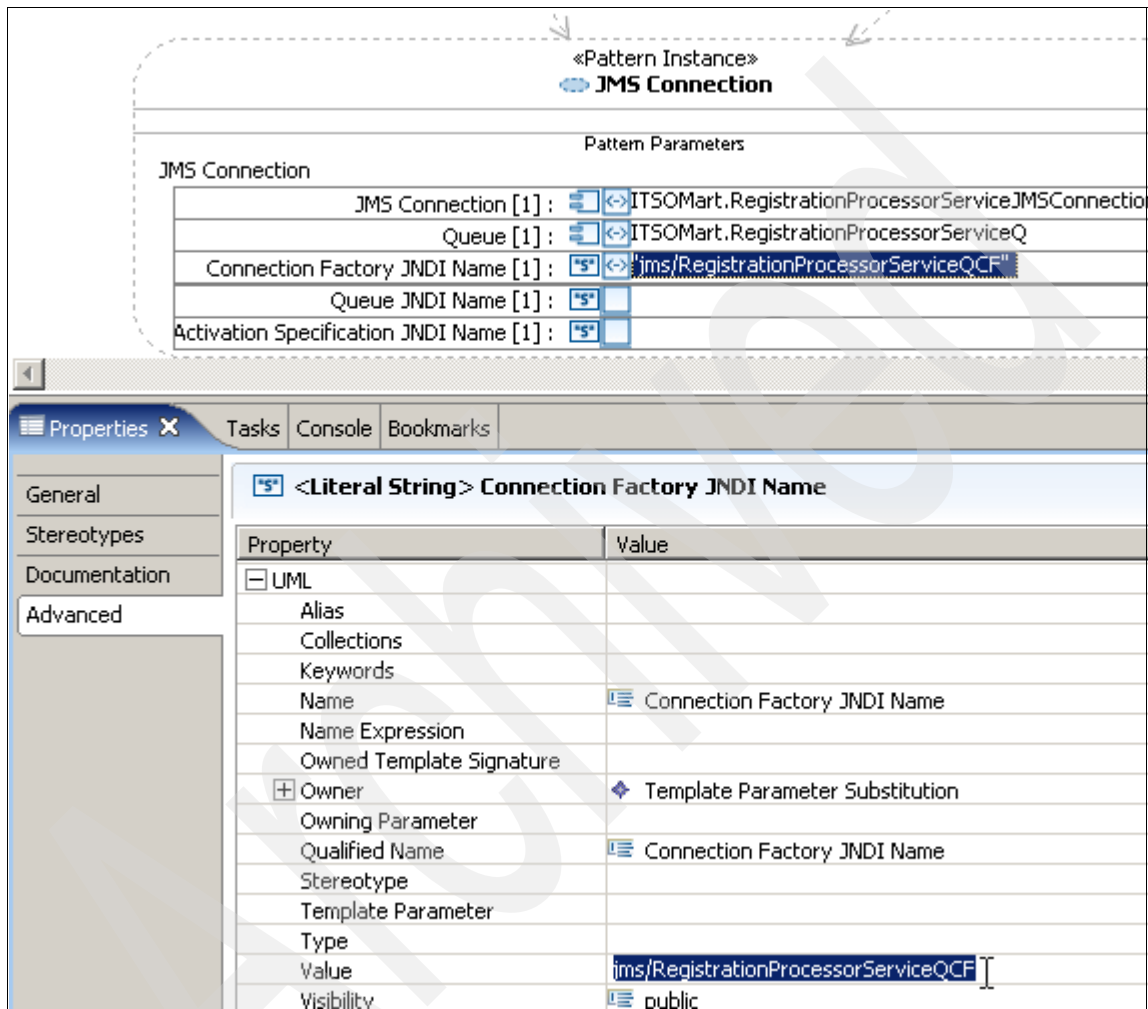


Figure 5-79 Connection Factory JNDI Name value

12. Specify the Queue JNDI Name parameter the same way you specified the Connection Factory JNDI Name in the previous step.
 - a. Click the “S” icon next to the Queue JNDI Name parameter, then in the pop-up menu, select the “S” icon to create a literal string for the parameter.

- b. Select the Queue JNDI Name field, then in the Properties view, select the **Advanced** tab, and under the UML properties, change the Value field to `jms/RegistrationProcessorServiceQ` (Figure 5-80).

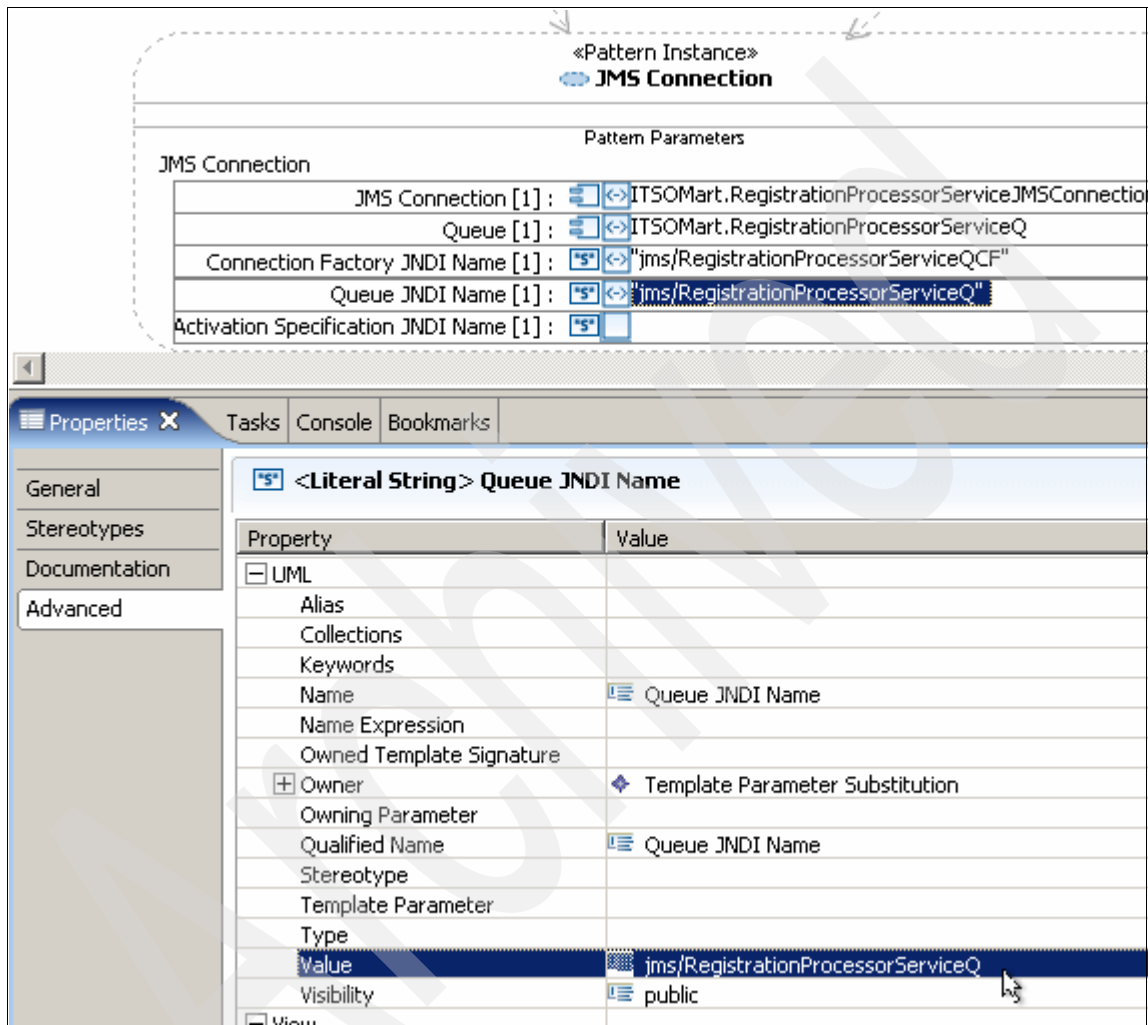


Figure 5-80 Queue JNDI Name value

13. Specify the Activation Specification JNDI Name parameter.
 - a. Click the “S” icon next to the Activation Specification JNDI Name parameter, then in the pop-up menu, select the “S” icon to create a literal string for the parameter.

- b. Select the Activation Specification JNDI Name field, then in the Properties view select the **Advanced** tab, and under the UML properties, change the Value field to `jms/RegistrationProcessorServiceActivationSpec` (Figure 5-81).

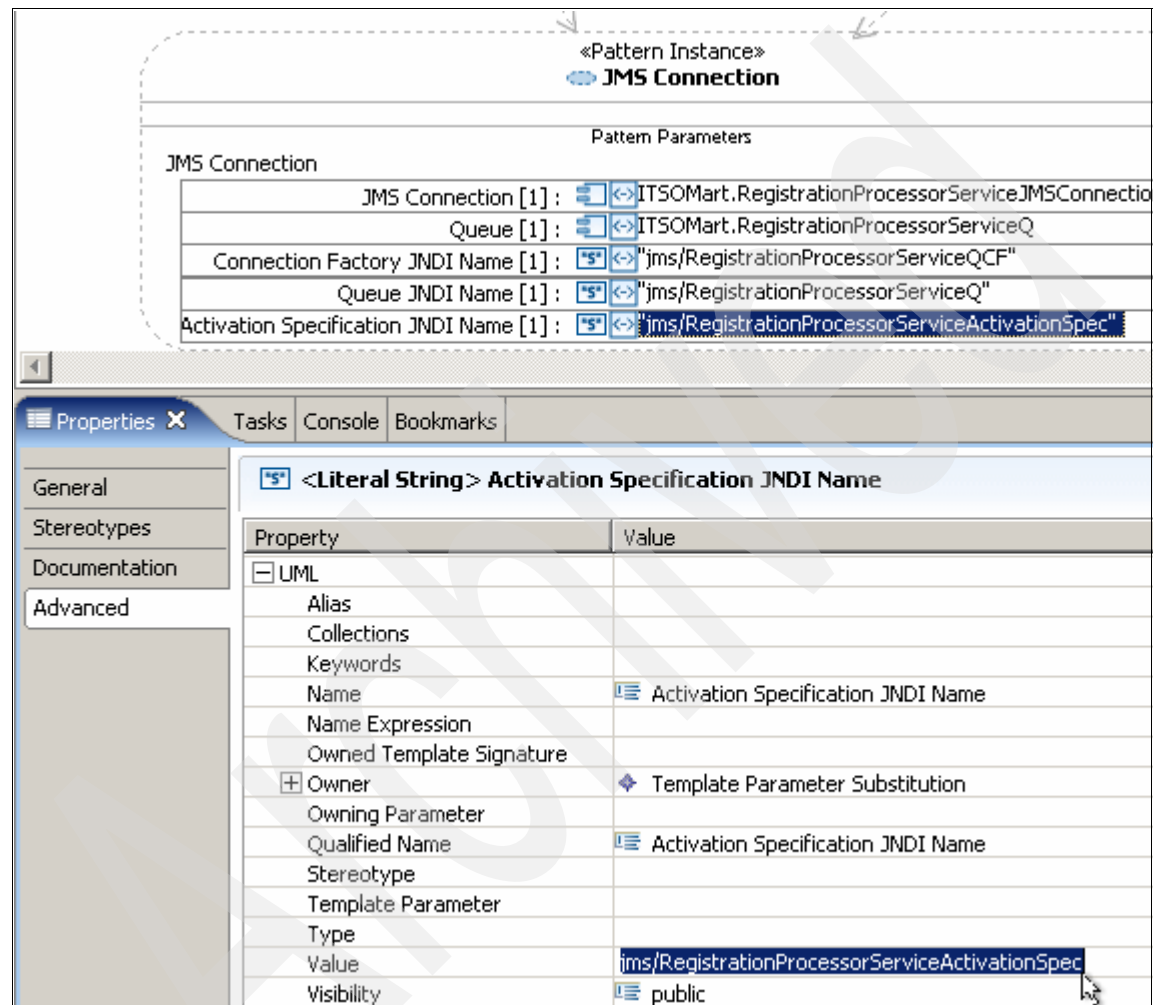


Figure 5-81 Activation specification JNDI name value

The complete diagram should now look like Figure 5-82.

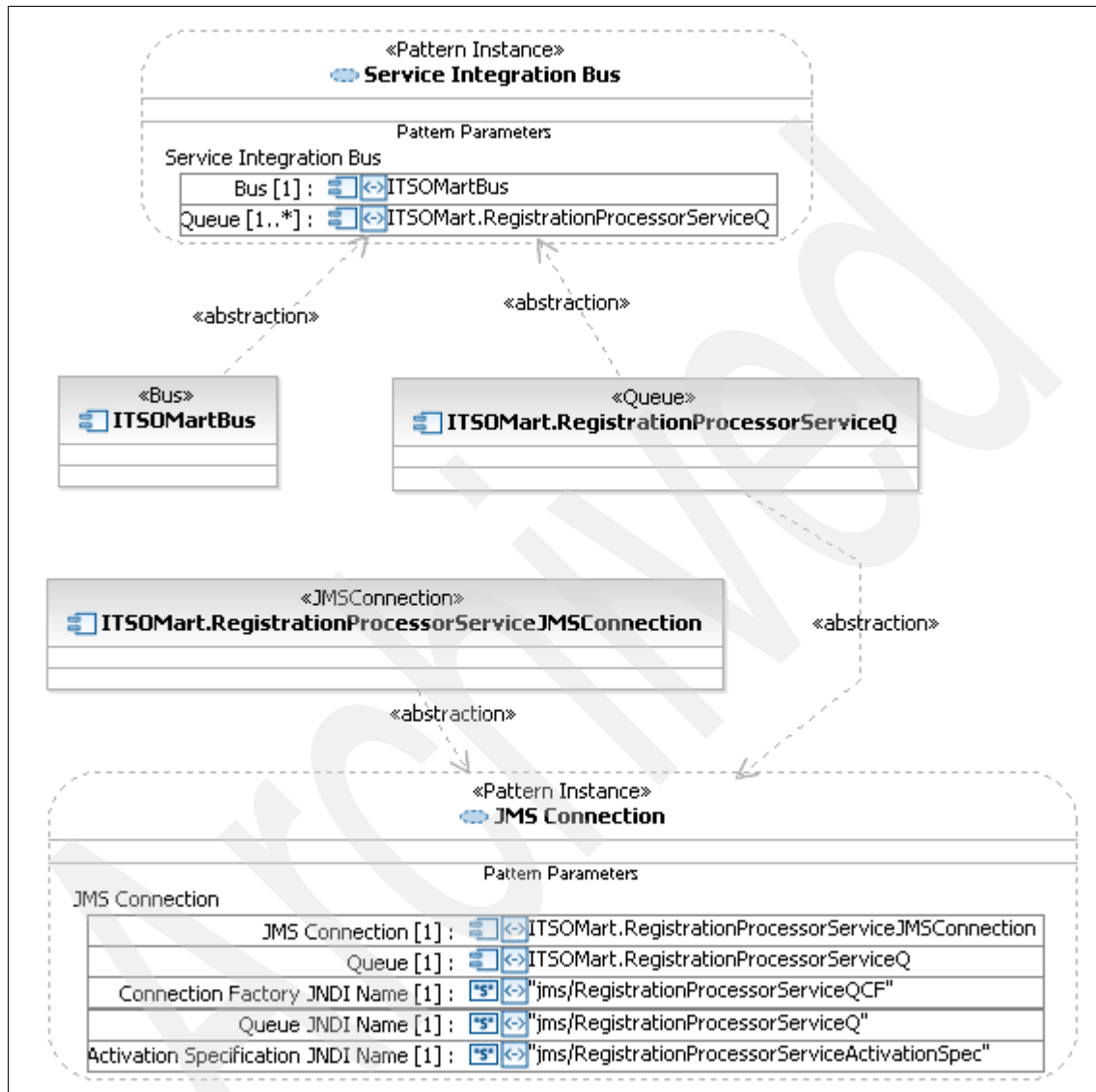


Figure 5-82 Deployment model diagram

5.8.3 Run the UML-to-JACL transformation

The UML-to-JACL transformation that is provided by the WebSphere Platform Messaging asset will generate a JACL script that you can use to configure the service integration bus and JMS resources in WebSphere ESB.

1. Right-click your model in the Model Explorer and select **Transform** → **Run Transformation** → **UML to JACL**.
2. In the Run Transformation window, specify the source model on which the transformation will run. This should be filled in already if you selected the model in step 1, but if it is not, then next to the Source field, click the square button and browse to select your model. See Figure 5-83.

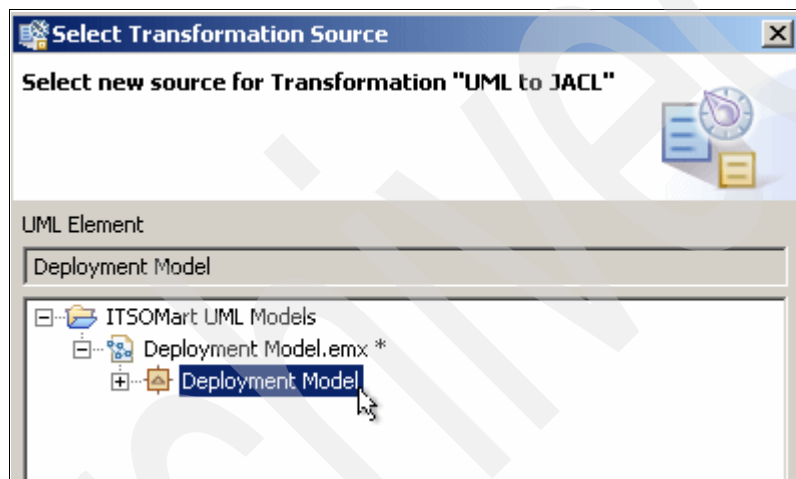


Figure 5-83 Select Transformation Source

3. In the Run Transformation window, select a project for the target where you want the JACL script to be generated (Figure 5-84).

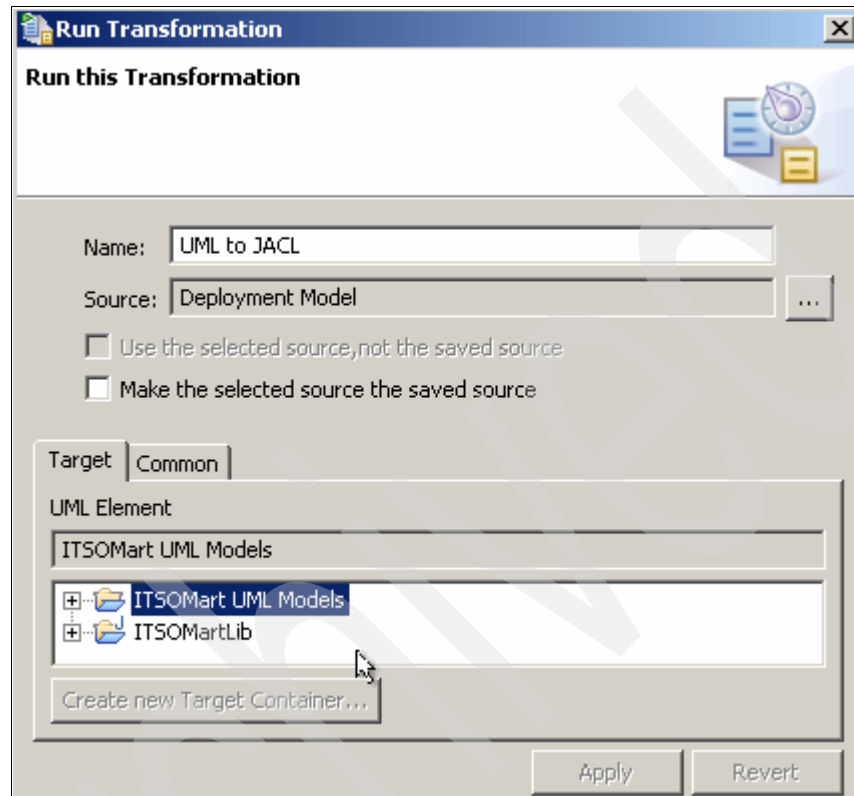


Figure 5-84 Run Transformation - Target

4. Click **Run** to run the transformation and generate the JACL script.

Look at the generated JACL script

Files with an extension of JACL are not displayed in the Model Explorer view, so you must switch to a different view to see the JACL file that was generated for your model:

1. Add the Navigator view to your Modeling perspective by selecting **Window** → **Show View** → **Other** → **Basic** → **Navigator**. The Navigator view gets added to the bottom pane.

2. The JACL file will now be displayed in the Navigator view, under your UML project. This file will have the name of the Bus component in your model. For example, for the model built for the sample, the file will be named ITSOMartBus.jacl (Figure 5-85).

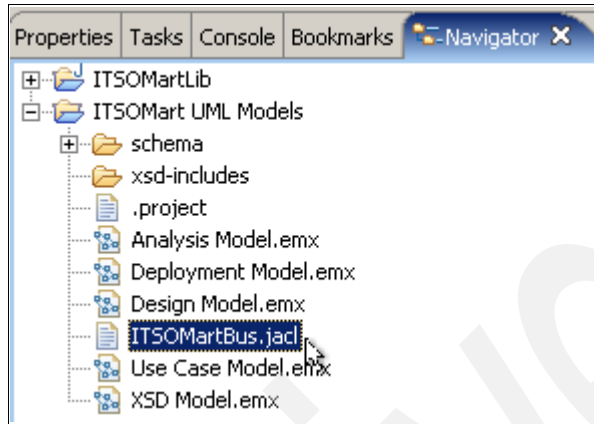


Figure 5-85 ITSOMartBus.jacl in the Navigator view

3. Open the ITSOMartBus.jacl file (Example 5-3) and you can see the set of commands for the configuration each component you defined in the model for the service integration bus and JMS resources.

Example 5-3 ITSOMartBus.jacl

```
# Setup basic variables for which server etc to create bus on
set node esbNode
set server server1
set scope [$AdminConfig getid /Node:$node/Server:$server/]

# Create Bus
set busName ITSOMartBus
puts "Create messaging bus and assign the server as a member"
set params [list -bus $busName]
puts "\$AdminTask createSIBus $params"
$AdminTask createSIBus $params
set params [list -bus $busName -node $node -server $server -createDefaultDatasource true]
puts "\$AdminTask addSIBusMember $params"
$AdminTask addSIBusMember $params

# Create destination
set queueName ITSOMart.RegistrationProcessorServiceQ
set queueJNDIName jms/RegistrationProcessorServiceQ
```



```

puts "Create SIB Queues"
set params [list -bus $busName -name $queueName -type queue -node $node -server $server]
puts "\$AdminTask createSIBDestination $params"
$AdminTask createSIBDestination $params
puts "Create JMS JNDI Resources"

# Create queue
set params [list -name $queueName -jndiName $queueJNDIName -queueName $queueName -busName
$busName]
puts "\$AdminTask createSIBJMSQueue $scope $params"
$AdminTask createSIBJMSQueue $scope $params

# Create Connection Factory
set cfName RegistrationProcessorServiceQCF
set cfJNDIName jms/RegistrationProcessorServiceQCF
set params [list -name $cfName -jndiName $cfJNDIName -busName $busName -type queue]
puts "\$AdminTask createSIBJMSConnectionFactory $scope $params"
$AdminTask createSIBJMSConnectionFactory $scope $params

# Create JMS Activation specification to be used with MDB's
set asName RegistrationProcessorServiceActivationSpec
set asJNDIName jms/RegistrationProcessorServiceActivationSpec
set params [list -name $asName -jndiName $asJNDIName -destinationJndiName $queueJNDIName
-busName $busName -destinationType javax.jms.Queue]
puts "\$AdminTask createSIBJMSActivationSpec $scope $params"
$AdminTask createSIBJMSActivationSpec $scope $params

# Save the configuration
$AdminConfig save
puts "Configuration saved."

```

4. Notice the following naming conventions the transformation used to generate the JACL script:
 - The JMS queue name is the same as the Queue destination on the Service Integration Bus: ITSOMart.RegistrationProcessorServiceQ.
 - The JMS connection factory name is taken from the Connection Factory JNDI Name specified in the model. The prefix *jms* is removed from the JNDI name `.jms/RegistrationProcessorServiceQCF` to form the name for the JMS connection factory that will be displayed in the WebSphere administration console.
 - The JMS activation specification name is formed also from the JNDI name specified in the model with the prefix *jms* removed.

5.8.4 Running the JACL script from a command line

You can now use the generated JACL script to configure the service integration bus and JMS resources for your application.

1. Start your WebSphere ESB server.
2. Copy the generated JACL script to somewhere on the file system where WebSphere ESB is installed.
3. Run the following command from the bin directory of your WebSphere ESB profile.

```
wsadmin -f ITSOMartBus.jacl
```

If you are using the default WebSphere ESB profile in WebSphere Integration Developer, this directory will be <wid_install>\pf\esb\bin.

You should see the following messages in your command shell (Figure 5-86).

```
WASX7209I: Connected to process "server1" on node esbNode using SOAP connector; The type of process
is: UnManagedProcess
Create messaging bus and assign the server as a member
$AdminTask createSIBus -bus ITSOMartBus
$AdminTask addSIBusMember -bus ITSOMartBus -node esbNode -server server1 -createDefaultDatasource true
Create SIB Queues
$AdminTask createSIBDestination -bus ITSOMartBus -name ITSOMart.RegistrationProcessorServiceQ -type
queue -node esbNode -server server1
Create JMS JNDI Resources
$AdminTask createSIBJMSQueue
server1(cells/esbCell/nodes/esbNode/servers/server1|server.xml#Server_1140500861281) -name
ITSOMart.RegistrationProcessorServiceQ -jndiName jms/RegistrationProcessorServiceQ -queueName
ITSOMart.RegistrationProcessorServiceQ -busName ITSOMartBus
$AdminTask createSIBJMSConnectionFactory
server1(cells/esbCell/nodes/esbNode/servers/server1|server.xml#Server_1140500861281) -name
RegistrationProcessorServiceQCF -jndiName jms/RegistrationProcessorServiceQCF -busName ITSOMartBus
-type queue
$AdminTask createSIBJMSActivationSpec
server1(cells/esbCell/nodes/esbNode/servers/server1|server.xml#Server_1140500861281) -name
RegistrationProcessorServiceActivationSpec -jndiName jms/RegistrationProcessorServiceActivationSpec
-destinationJndiName jms/RegistrationProcessorServiceQ -busName ITSOMartBus -destinationType
javax.jms.Queue
Configuration saved.
```

Figure 5-86 wsadmin message

In the WebSphere Administration console, you can see the JMS resources (connection factory, queue, and activation specification) that were created under Resources → JMS Providers → Default Messaging at the Server scope. The JMS connection factory is actually created as a JMS queue connection factory.

5.9 Resources

For more information see the following:

- ▶ Rational Software Architect
<http://www-306.ibm.com/software/awdtools/architect/swarchitect/>
- ▶ *Patterns: Model-Driven Development Using IBM Rational Software Architect*, SG24-7105
- ▶ Rational Software Architect: SOA design resources
<http://www-306.ibm.com/software/info/developer/solutions/soadev/dtoolkit2.jsp>
- ▶ UML Profile for Software Services, RSA Plug-In
http://www-128.ibm.com/developerworks/rational/library/05/510_svc/
- ▶ Architecting on demand solutions, Part 11: Build ESB connectivity with Rational Software Architecture (RSA) WebSphere Platform Messaging Patterns
<http://www-128.ibm.com/developerworks/ibm/library/i-odoebp11/>
- ▶ Modeling Web services, Part 1
http://www-128.ibm.com/developerworks/rational/library/05/1129_johnston/
- ▶ Modeling Web services, Part 2
http://www.ibm.com/developerworks/rational/library/06/0411_johnston/

Assemble with WebSphere Integration Developer

WebSphere Integration Developer is the development environment for building integrated business applications targeted for WebSphere ESB and the WebSphere Process Server. One of the primary purposes of WebSphere Integration Developer is to provide the appropriate tools to easily build and test SCA-based applications.

This chapter discusses WebSphere Integration Developer key concepts and common tasks in terms of mediation module development for deployment to WebSphere ESB. Subsequent chapters provide more details about building and deploying mediations using the ITSOMart scenario as an example.

This chapter includes the following topics:

- ▶ Technology overview
- ▶ Introduction to WebSphere Integration Developer
- ▶ Development environment settings
- ▶ Development process
- ▶ Testing mediations
- ▶ Packaging the mediation for deployment

For a complete look at WebSphere Integration Developer, WebSphere ESB, and the process of building and deploying mediations, see *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212.

6.1 Technology overview

This book focuses on the use of WebSphere ESB as the enterprise service bus in a solution. Understanding the following technology is key to designing efficient solutions. This section gives a brief introduction to each technology.

6.1.1 Service Component Architecture

Service Component Architecture (SCA) and Service Data Objects (SDOs) provide the underpinnings for the SOA programming model used to build business processes in WebSphere Process Server and mediations in WebSphere ESB.

SCA is a universal model for business services that publish or operate on business data. It separates business logic from infrastructure logic so that application programmers can focus on the business problem. The implementation of business processes is contained in service components. SDO defines a model for the information exchanged between these components.

Figure 6-1 shows the main terms of an SCA service component.

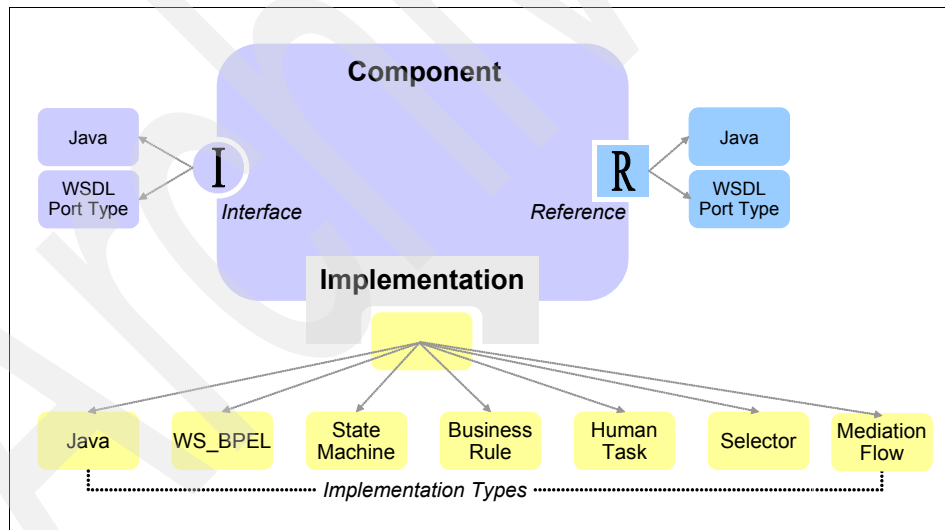


Figure 6-1 Service component overview

WebSphere ESB: The service component in WebSphere ESB is referred to as a *mediation flow component*.

Interfaces

A component exposes business-level interfaces so that the service can be used or invoked. The interface of a component defines the operations that can be called and the data that is passed, such as input arguments, returned values, and exceptions.

A service interface is defined by a Java interface or WSDL port type. Arguments and return values are described with Java classes, simple Java types, or XML schema. SDO-generated Java classes are the preferred form of Java class because of their integration with XML technologies. Arguments described in XML schema are exposed to programmers as SDOs. All components have interfaces of the WSDL type. Only Java components support Java-type interfaces. If a component has more than one interface, all interfaces must be the same type.

A component can be called synchronously or asynchronously. This is independent of whether the implementation is synchronous or asynchronous. The component interfaces are defined in the synchronous form and asynchronous support is also generated for them. You can specify a preferred interaction style as synchronous or asynchronous. The asynchronous type advertises to users of the interface that it contains at least one operation that can take a significant amount of time to complete. As a consequence, the calling service must avoid keeping a transaction open while waiting for the operation to complete and send its response. The interaction style applies to all the operations in the interface.

You can also apply a role-based permission qualifier to an interface so that only authorized applications can invoke the service with that interface. If the operations require different levels of permission for their use, you must define separate interfaces to control their access.

Implementation

A service can be implemented in a range of languages, for example, Java, BPEL, state-machine definitions, and so on. When implementing a service, the focus is on the business purpose and less on infrastructure technology.

SCA and non-SCA services can use other service components in their implementations. They do not hard-code the other services they use. They declare soft links called service references. Service wires resolve service references. You can use SCA wiring to assemble components and create an SCA application.

WebSphere ESB: The implementation of a mediation flow component is a *mediation flow*.

References

When a component wants to use the services of another component, it must have a partner reference or simply a reference. An in-line reference means that the referenced service component is defined within the same scope of the referencing component. In other words, both components are defined within the same module.

Applications that are not defined as SCA components (for example, JSPs) can still invoke SCA components. They do so through the use of stand-alone references. Stand-alone references contain partner references that identify the components to call. Alone, stand-alone references do not have any implementation or interface.

Service modules

Components are assembled in a module, as shown in Figure 6-2, which is a basic unit of deployment in the WebSphere Process Server or WebSphere ESB.

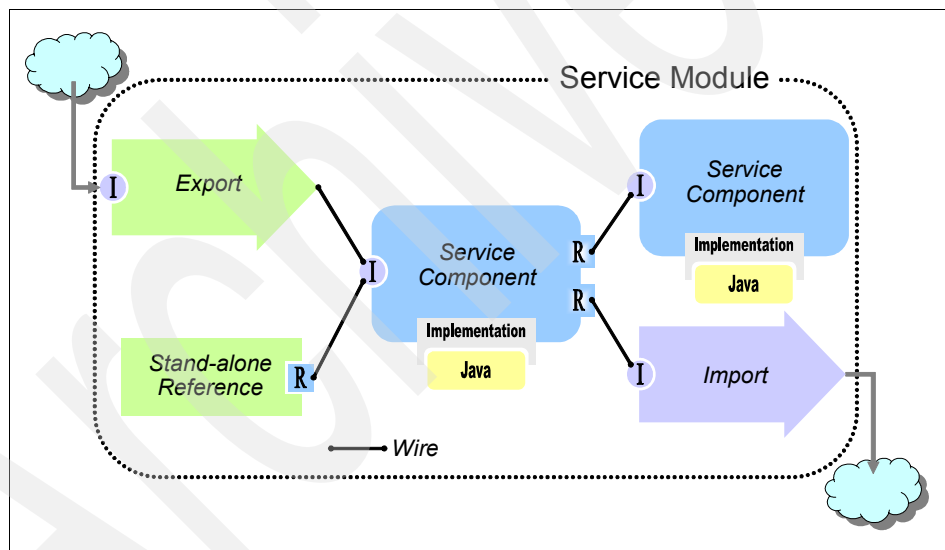


Figure 6-2 Service module overview

The *module assembly* is a diagram of the integrated business application, consisting of components and the wires that connect them. In WebSphere Integration Developer, the module assembly is exposed as an assembly diagram where components can be assembled.

The implementations of components that are used in a module assembly can reside within the module or can be external. Components that belong to other modules can be used through *imports*. Components in different modules can be

wired together by publishing the services as *exports* and dragging the exports into the assembly diagram to create imports.

When wiring components, you can also specify quality of service qualifiers on the implementations, partner references, and interfaces of the component.

WebSphere ESB: A *mediation module* is a type of SCA module for service requests over an enterprise service bus. Mediation modules consist of imports, exports, mediation flow components, and (optionally) Java SCA components.

Imports

An import allows you to use functions that are not part of the module that you are assembling. Imports can be from components in other modules or non-SCA components such as stateless session EJBs and Web services. Available function or business logic that is implemented in remote systems such as Web services, EIS functions, EJBs, or remote SCA components is modeled as an *imported service*.

Imports have interfaces that are the same as or a subset of the interfaces of the remote service that they are associated with so that those remote services can be called. Imports are used in an application in exactly the same way as local components. This provides a uniform assembly model for all functions, regardless of their locations or implementations. The import binding does not have to be defined at development time; it can be done at deployment time.

WebSphere ESB: WebSphere ESB supports the following import bindings:

- ▶ Web service bindings
 - SOAP over HTTP (SOAP/HTTP)
 - SOAP over JMS (SOAP/JMS)
- ▶ SCA bindings (to connect SCA modules to other SCA modules).
- ▶ Java Message Service (JMS) 1.1 bindings

JMS can exploit various transport types, including TCP/IP and HTTP(S).
- ▶ WebSphere Adapter bindings

WebSphere Adapters enable interaction with Enterprise Information Systems (EIS).
- ▶ Stateless session bean binding

This allows you to invoke a stateless session EJB as an SCA component.

Export

An *export* is a published interface from a component that offers the component business service to the outside world, for example, as a Web service. Exports have interfaces that are the same as or a subset of the interfaces of the component that they are associated with so that the published service can be called. An export dragged from another module into an assembly diagram automatically creates an import.

The service component details are stored in an XML file using a new definition language, Service Component Definition Language (SCDL).

WebSphere ESB: WebSphere ESB supports the following export bindings:

- ▶ Web service bindings
 - SOAP/HTTP
 - SOAP/JMS
- ▶ SCA bindings to connect SCA modules to other SCA modules
- ▶ Java Message Service (JMS) 1.1 bindings
- ▶ WebSphere Adapter bindings

6.1.2 Service Data Objects

Business data that is exchanged in an integrated application in WebSphere ESB is represented by business objects. The objects are based on SDO, which is a new data access technology.

SDO unifies data representation across disparate data stores. It supports a disconnected programming model and is integrated with XML. SDO provides dynamic and static (strongly typed) data APIs. The SDO proposal was published jointly by IBM and BEA as JSR 235. SDO Version 1.0 support is introduced in WebSphere Application Server V6 and IBM Rational Application Developer V6. The SDO V2.0 specification is currently available.

In addition to providing a programming model that unifies data access, there are several other key design features to note about SDO. First, built into the SDO architecture is support for some common programming patterns. Most significantly, SDO supports a disconnected programming model. Typically, with this type of pattern, a client might be disconnected from a particular data access service (DAS) while working with a set of business data. However, when the client has completed processing and needs to apply changes to a back-end data store by way of a DAS, a change summary is necessary to provide the appropriate level of data concurrency control. This change summary information has been built into the SDO programming model to support this common data access scenario.

Another important design point to note is that SDO integrates well with XML. As a result, SDO naturally fits in with distributed service-oriented applications where XML plays a very important role.

Finally, SDO has been designed to support both dynamic and static data access APIs. The dynamic APIs are provided with the SDO object model and provide an interface that allows developers to access data even when the schema of the data is not known until runtime. In contrast to this, the static data APIs are used when the data schema is known at development time, and the developer prefers to work with strongly typed data access APIs.

Data objects

The fundamental concept in the SDO architecture is the data object. In fact, the term SDO is often used interchangeably with the term data object. A data object is a data structure that holds primitive data, multi-valued fields (other data objects), or both. The data object also has references to metadata that provide information about the data found in the data object. In the SDO programming model, data objects are represented by the `commonj.sdo.DataObject` Java interface definition. This interface includes method definitions that allow clients to obtain and set the properties associated with `DataObject`.

Data graph

Another important concept in the SDO architecture is the data graph shown in Figure 6-3. A data graph is a structure that encapsulates a set of data objects. From the top-level data object in the graph, all other data objects can be reached by traversing the references from the root data object. In the SDO programming model, data graphs are represented by the `commonj.sdo.DataGraph` Java interface definition.

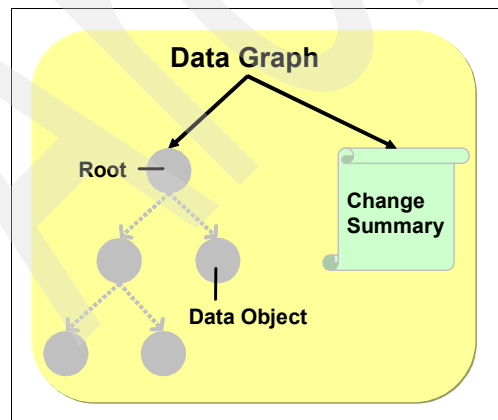


Figure 6-3 Data graph consisting of data objects and change summary

An important feature of the data graph is a change summary that is used to log information about what data objects in the graph have changed during processing. The change summary information is defined by the `commonj.sdo.ChangeSummary` interface.

Business objects and the business object framework

The business object framework provides a data abstraction for SCA. Business objects are based on SDO v1.0 technology but provide additional functionality not found in SDO.

Both SCA and SDO, the basis of business objects, have been designed to be complimentary service-oriented technologies. Figure 6-4 illustrates how SCA provides the framework to define service components and to compose these services into integrated applications, and it further shows that business objects represent the data that flows between each service. Whether the interface associated with a particular service component is defined as a Java interface or a WSDL port type, the input and output parameters are represented by business objects.

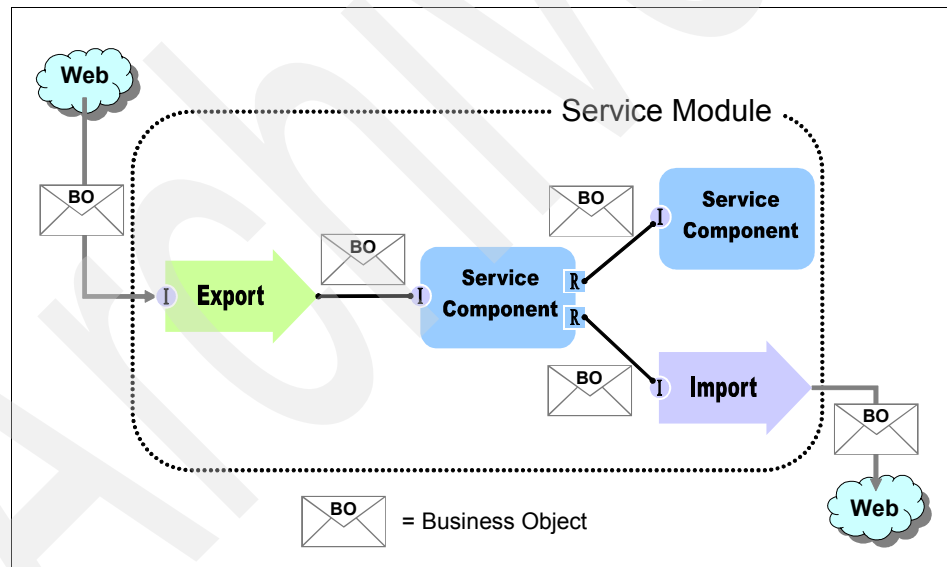


Figure 6-4 Exchanging data in an SCA runtime

The business object framework consists of the following four concepts:

Business object (BO) Fundamental data structure for representing business data

Business graph (BG)	Wrapper for a business object or hierarchy of business objects to provide enhanced information such as change summary, event summary, and verb
BO Type Metadata	Metadata that provides the ability to annotate business objects with application-specific information
BO Services	A set of services provided to facilitate working with business objects, available in addition to the capabilities already provided by SDO V1.0

Note: The term *business object* is occasionally used to refer to the entire framework. However, here the term refers to the fundamental data structure for representing business data and *not* to the overall architecture. For the overall architecture, the term *business object framework* is used.

The business object is directly related to the SDO data object concept. In fact, business objects in WebSphere Process Server and WebSphere ESB are represented in memory with the SDO type `commonj.sdo.DataObject`. Business objects are modeled as XML schema. Two types of business objects are found in the business object framework:

- ▶ Simple business objects, which are composed only of scalar properties
- ▶ Hierarchical business objects, which are composed of attributes that refer to nested business objects

In the business object framework, a business graph is used to wrap a top-level business object and provide additional information that can enhance the data. In particular, the business graph includes the change summary for the data in the graph similar to the SDO change summary information, the event summary, and verb information used for data synchronization between EISs.

The business graph is similar to the SDO data graph. However, the event summary and the verb portion of the enhanced information is not included with the SDO data graph concept.

6.1.3 Service Message Objects

Within WebSphere ESB, mediation flows process messages as Service Message Objects (SMOs). SMOs are enhanced SDOs.

All SMOs have the same basic structure, consisting of the following:

- ▶ A *root* data object called a `ServiceMessageObject` that contains other data objects representing header, body, and context data.

- ▶ A *body* that contains the message payload. The payload is the application data (business object) exchanged between service endpoints.
- ▶ *Header* information that originates from a specific import or export binding. The message headers handled by WebSphere ESB are:
 - Web service message header
 - SCA message header
 - JMS message header
 - WebSphere Adapter message header
- ▶ *Context* information, which is data other than the message payload.

All of this information is accessed as SDO DataObjects, and there is a schema declaration that specifies the overall structure of the SMO. The schema is generated by the WebSphere Integration Developer.

6.2 Introduction to WebSphere Integration Developer

WebSphere Integration Developer is the development environment for WebSphere Enterprise Service Bus and WebSphere Process Server. It provides an environment for building and testing integrated applications based on a services-oriented architecture.

The application development in WebSphere Integration Developer is based on SCA. Besides developing SCA components and modules, WebSphere Integration Developer is also used to assemble mediations, components using mediation primitives, and to create mediation modules. WebSphere Integration Developer includes an integrated unit test environment for WebSphere Process Server, WebSphere ESB, and WebSphere Application Server, allowing developers to deploy their modules to the integrated test server and perform unit testing using the integration test client.

6.2.1 Starting WebSphere Integration Developer

To start WebSphere Integration Developer, perform the following tasks:

1. Click **Start** → **Programs** → **IBM WebSphere** → **Integration Developer V6.0.1** → **WebSphere Integration Developer V6.0.1**.
2. This launches the Workspace Launcher window shown in Figure 6-5 on page 221. A *workspace* is a directory where your work is stored. You can create many workspaces and choose which one to work on at any time. A common scenario is to have separate workspaces for different projects you may be working on.

Enter a path where a workspace should be created and click **OK**.

Note: On Windows systems, we recommend using short path names for the workspace directory, to minimize the chances of encountering the Windows limit on path names longer than 256 characters.

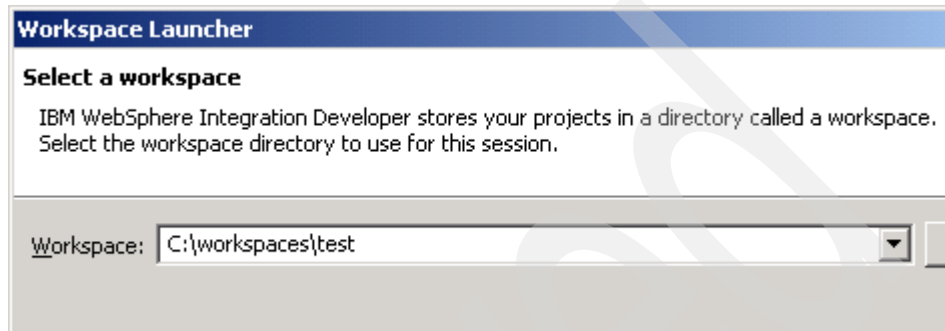


Figure 6-5 Workspace Launcher

3. WebSphere Integration Developer will start and open the Welcome page (shown in Figure 6-6). From this screen you can access information such as the product overview, tutorials, samples, migration information, and Web resources.

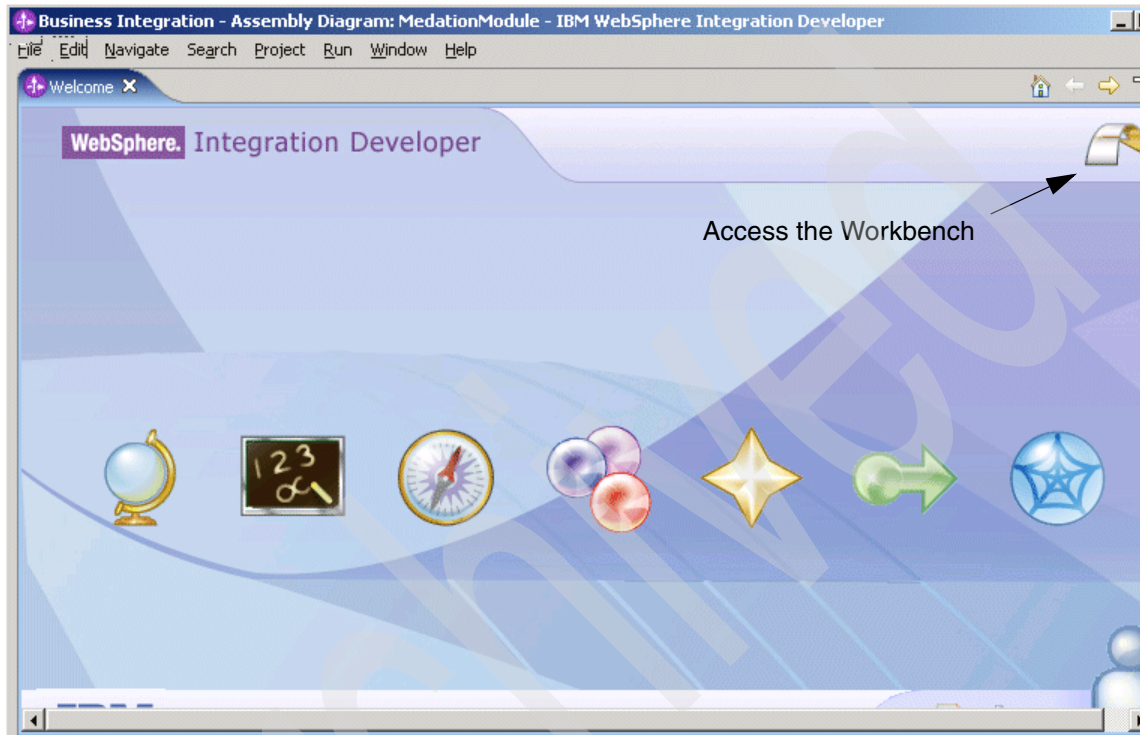


Figure 6-6 WebSphere Integration Developer welcome page

4. Clicking the arrow in the top right corner will close the Welcome screen and open the workbench using the *Business Integration perspective*.

The workbench is where you will spend most of your time developing mediation modules. It offers the developer a choice of perspectives and an array of toolbars and menu items that are used to accomplish a variety of tasks.

A *perspective* is a role-based collection of views and editors. The primary WebSphere Integration Developer perspective is the *Business Integration perspective*. We will use this perspective almost exclusively because it contains all the tools we need to create, develop, and manage business integration projects. Figure 6-7 shows the Business Integration perspective on the workbench.

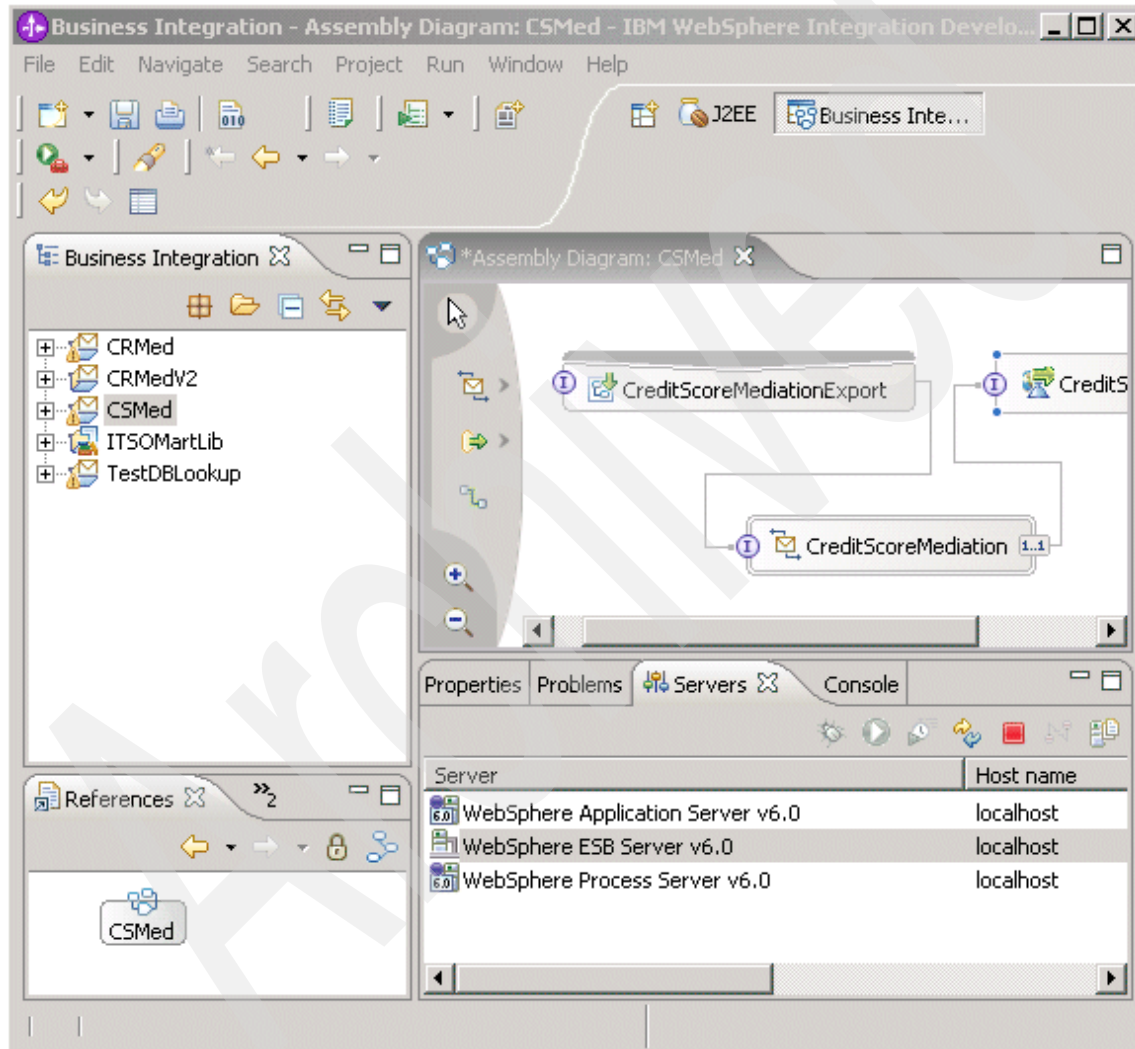


Figure 6-7 The Business Integration perspective

Each frame represents one or more views. A *view* is used to present information about a resource. Views are also used for navigating the

information in the workspace. Views might appear by themselves or stacked with other views in a tabbed notebook.

For example, in the lower right portion of the screen in Figure 6-7 on page 223 you can see the Servers view, which is used to manage the test environment application servers. Clicking the **Properties** tab at the top of the view will switch that frame to show the Properties for a selected resource.

The frame in the upper right corner contains resources that you are editing. One way to open a resource for editing is to double-click it in the Business Integration view. For example, in Figure 6-7 on page 223 you can see that an assembly diagram of a module assembly is open for editing. Each resource type is associated with an editor specifically designed for editing that object type.

Tip: You can maximize any frame by double-clicking the view titles in the frame or on the bar at the top of the frame. To return to the previous size, double-click the bar or title once more. This is particularly useful when working with editors or the server console.

As we go through the development of the ITSOMart mediations, you will see how many of the views, wizards, and editors make up the Business Integration perspective are used.

6.3 Development environment settings

As the Redbook team built the sample mediations, we discovered that certain preference settings in WebSphere Integration Developer made our tasks easier.

6.3.1 Disable automatic build

The automatic build feature of WebSphere Integration Developer is designed to keep resources current as you work with them. If you do not turn off this feature, the workspace will rebuild every time you save a resource. This takes time, and if you save additional resources before the build completes, could cause problems.

To turn off this feature, select **Project** → **Build Automatically**. A check to the left of the option (as shown in Figure 6-8) means that it is enabled. Selecting this option will turn the feature off or on.

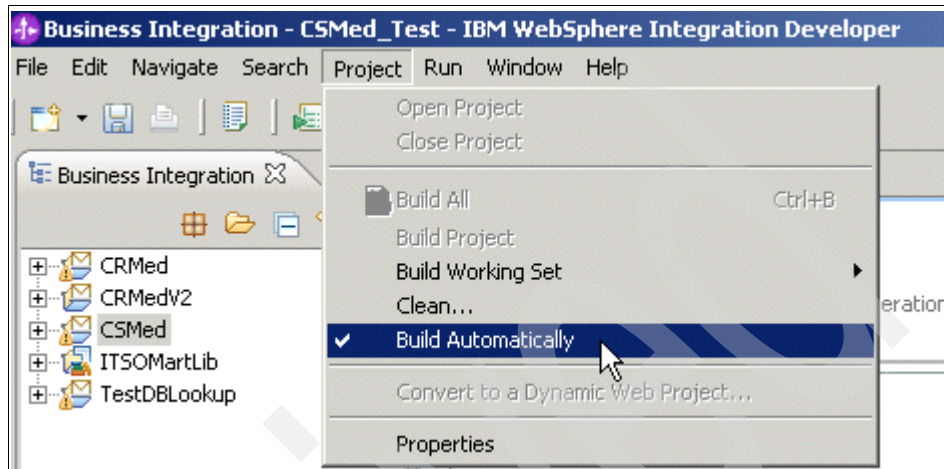


Figure 6-8 Automatic build option

With the feature turned off, you will need to periodically build the projects in your workspace.

- ▶ You can build the project you are working in by selecting **Project** → **Build Project**.
- ▶ You can build all the projects in the workspace by selecting **Project** → **Build All**.
- ▶ You can force a clean build of all projects by cleaning your workspace. Performing a workspace clean deletes all derived artifacts and staging projects, forcing an automatic build and regeneration to occur. To do this select **Project** → **Clean** from the menu bar, select **Clean all projects**, and click **OK**.

The instructions in this book assume that you have turned off the automatic build feature and will occasionally prompt you to build your projects.

6.3.2 Set the default target runtime

Select the WebSphere ESB Server v6.0 runtime as your target test server as follows:

1. Open your workspace preferences by selecting **Window** → **Preferences**.
2. Expand **Server** → **Installed Runtimes**.
3. Check the box to the left of WebSphere ESB Server v6.0.
4. Click **OK**.

6.3.3 Configure Web services workspace preferences

It is almost inevitable that you will end up working with Web services when building mediations. Whether you are creating a Web service, generating a Web service client, or testing a SOAP/HTTP mediation, you will require some level of Web Services Developer support.

We recommend that you set the following preferences before you begin:

1. Open your workspace preferences by selecting **Window** → **Preferences**.
2. Under Workbench → Capabilities, enable the Web Service Developer capability, as shown in Figure 6-9.

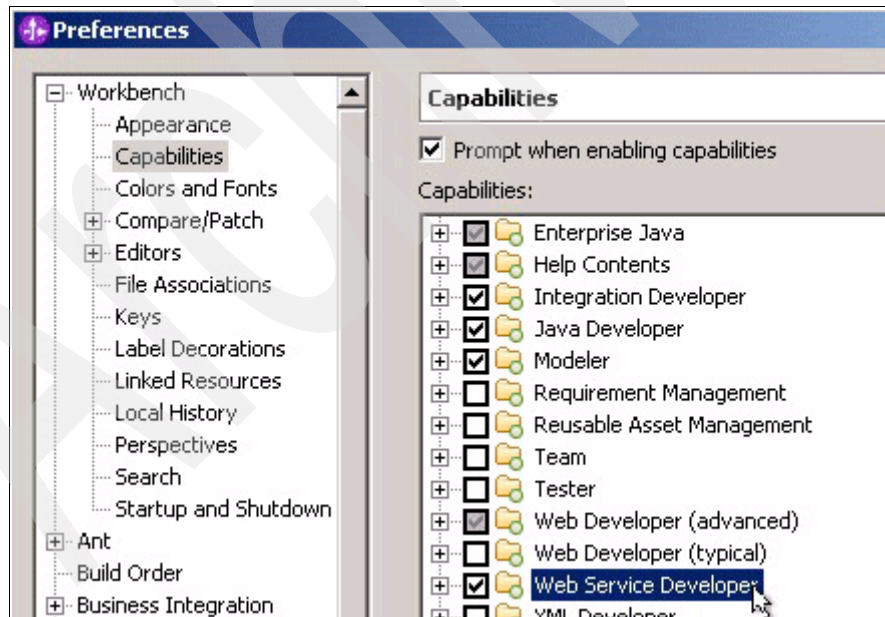


Figure 6-9 Preferences: Web Service Developer capability

Click **OK** to save the changes and close the preferences.

3. Re-open the preferences so that the Web Services preferences option will be available.
4. Select **Web Services** → **Server and Runtime** and specify the following values, as shown in Figure 6-10:
 - Server: WebSphere ESB Server v6.0
 - Web service runtime: IBM WebSphere
 - J2EE version: 1.4

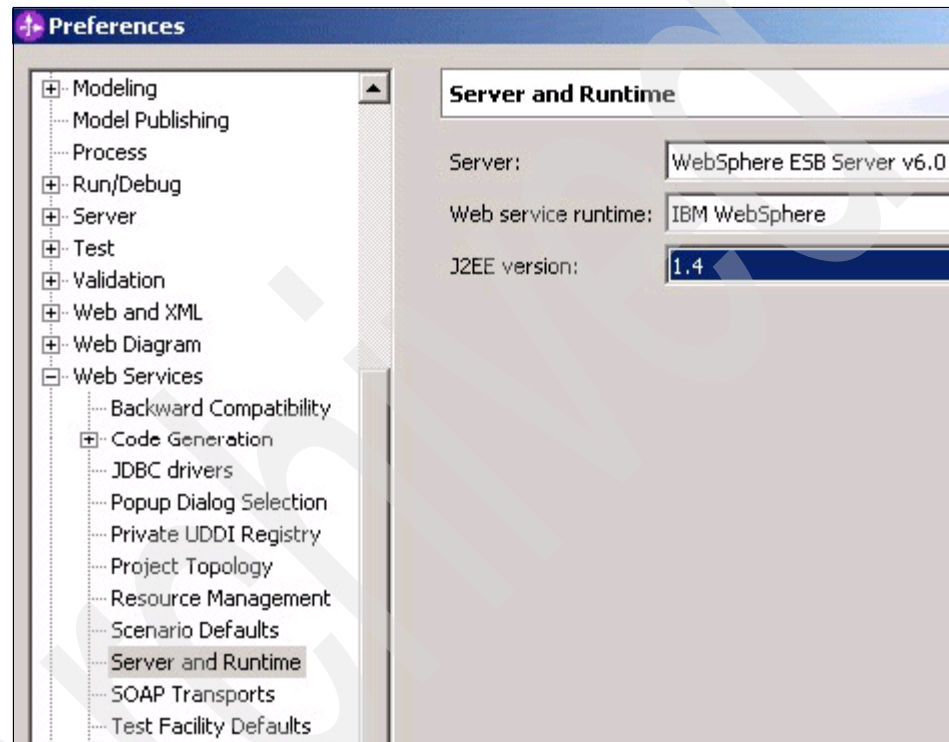


Figure 6-10 Preferences: Web Services → Server and Runtime

5. Select **Web Services** → **SOAP Transports**. Select **JMS** as the preferred transport when using the Web Services wizard, as shown in Figure 6-11.

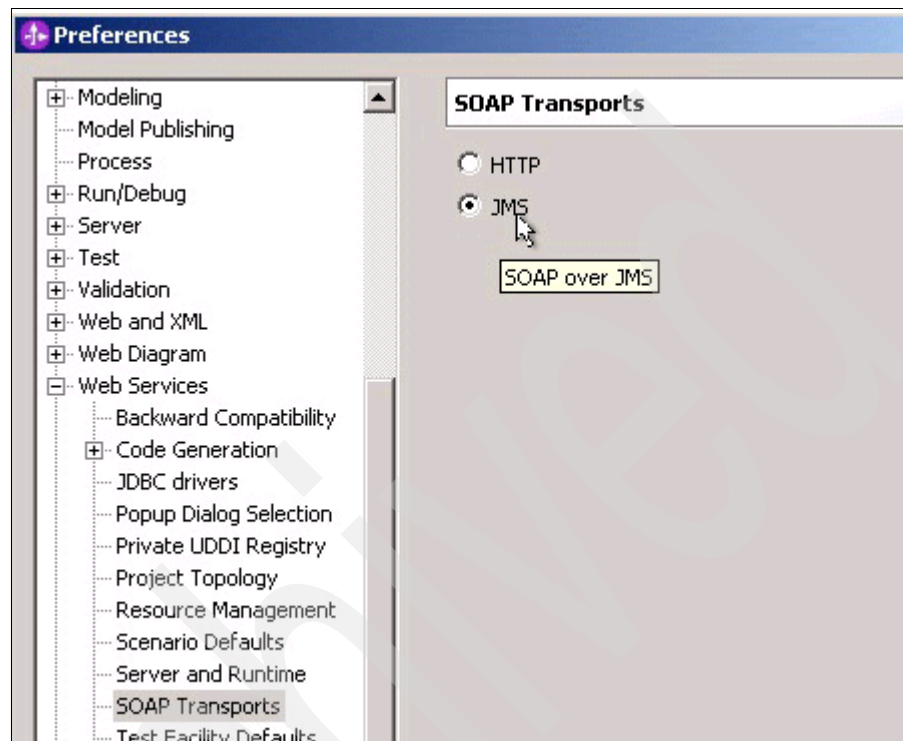


Figure 6-11 Preferences: Web Services → SOAP Transports

6. Select **Web Services** → **WS-I Compliance**. For the WS-I SSBP compliance level select **Suggest compliance**, as shown in Figure 6-12.

This preference will validate WSDL files against the WS-I Simple SOAP Basic Profile and show warnings in the Problems view if anything is out of compliance with this specification.

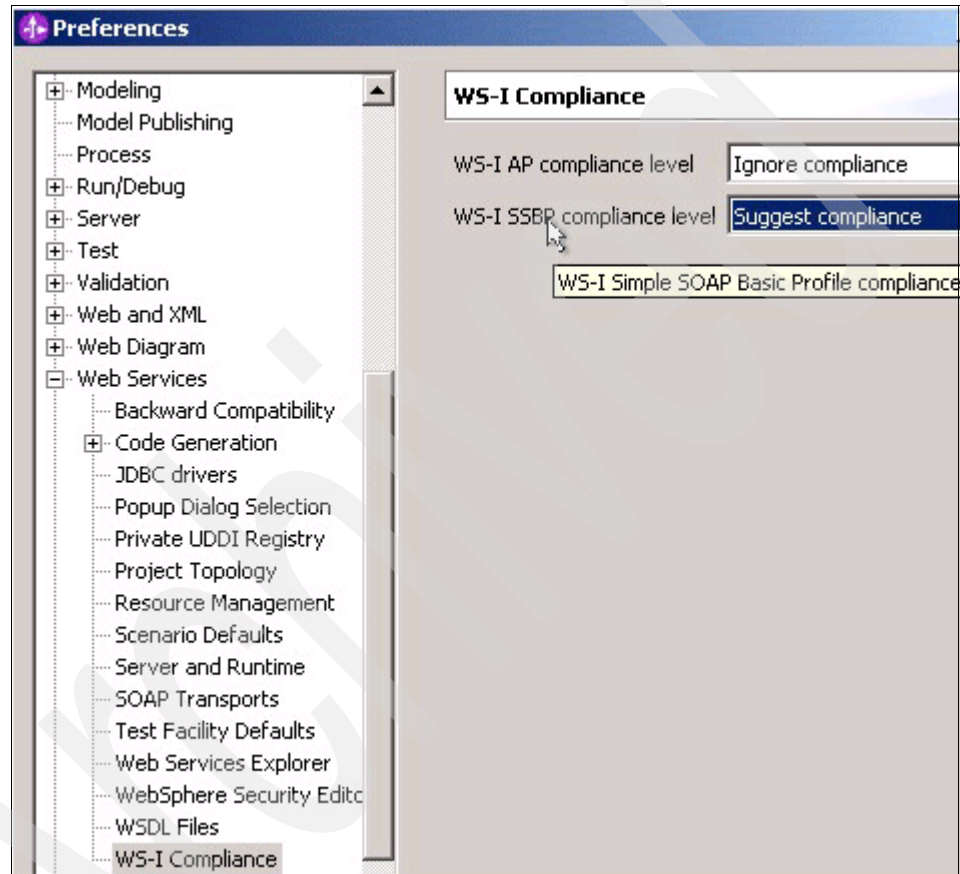


Figure 6-12 Preferences: Web Services → WS-I Compliance

Click **OK** to save your workspace preferences changes.

6.3.4 Workspaces and test environment

We also found that it was easier to organize our test efforts by creating a new workspace and corresponding WebSphere ESB test server profile for each mediation. This is not necessary, but intended to simplify things.

To create a new server profile and associate it with the workspace, see “Creating a new server in the test environment” on page 640.

6.4 Development process

This section gives a very brief overview of the mediation development process. Details about the development process and the various options can be found in the WebSphere Enterprise Service Bus 6.0.1 Information Center. In the following chapters you will also see examples of the process.

A module project represents a single deployable unit and encapsulates SCA components, J2EE projects, Java projects, and required libraries. When deploying to WebSphere ESB your choice of module type is limited to *mediation* modules.

Figure 6-13 shows a typical structure for a mediation module (ITSO_CreditRatingMed), including a dependent library (ITSOMartLib). The view is the Business Integration view in the Business Integration perspective.

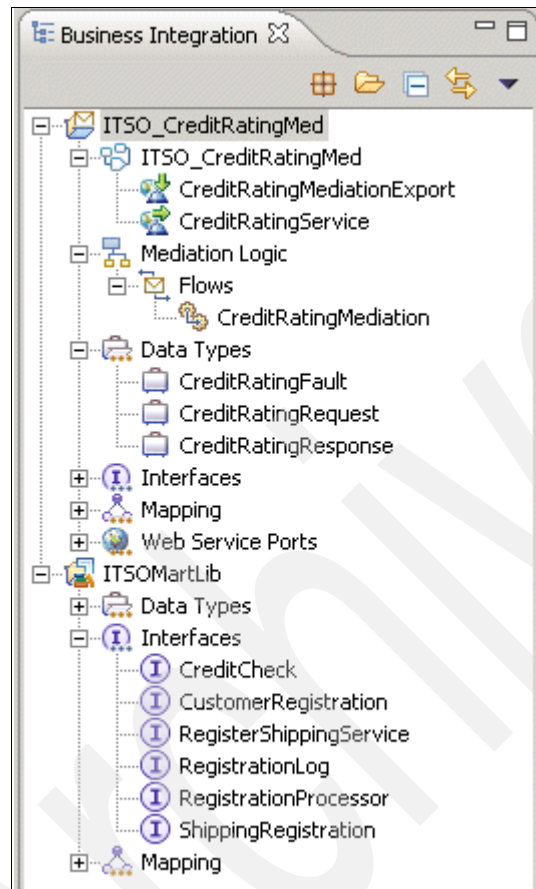


Figure 6-13 Mediation module and library structure

There are many ways to go about building a mediation module. Your path through the process may vary depending on the type of mediation, existing resources you will use, connectivity options, and individual preferences. The following steps will provide general guidance on the process:

1. Create a library.
2. Create business objects.
3. Define interfaces.
4. Create a mediation module.
5. Complete the module assembly.
6. Build the mediation flow.

6.4.1 Create a library

A library can be used to hold resources for use by one or more mediation modules. During development, the library is defined as a dependency for the mediation modules. At runtime, the libraries are deployed with each module that depends on it.

If you are deploying to WebSphere ESB, you can create two types of artifacts in a library: business objects and interfaces. The *mapping* folder only applies to WebSphere Process Server projects. Additionally, you can use libraries to hold WSDL bindings in a Web Service Bindings folder, which is created when you copy WSDL files into your library.

To create a library in WebSphere Integration Developer, do the following:

1. Right-click in the white space of the Business Integration view and select **New** → **Library**, as shown in Figure 6-14.

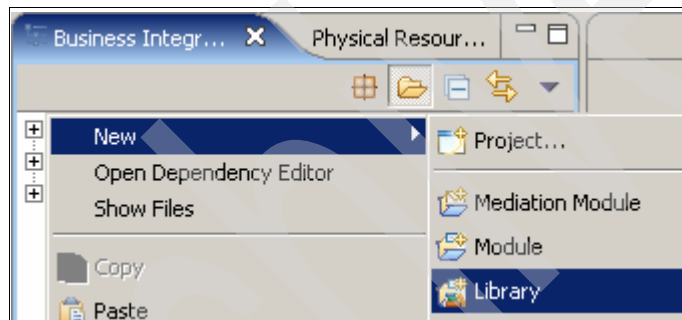


Figure 6-14 Create new library

2. Enter a name for the new library and click **Finish**, as shown in Figure 6-15.

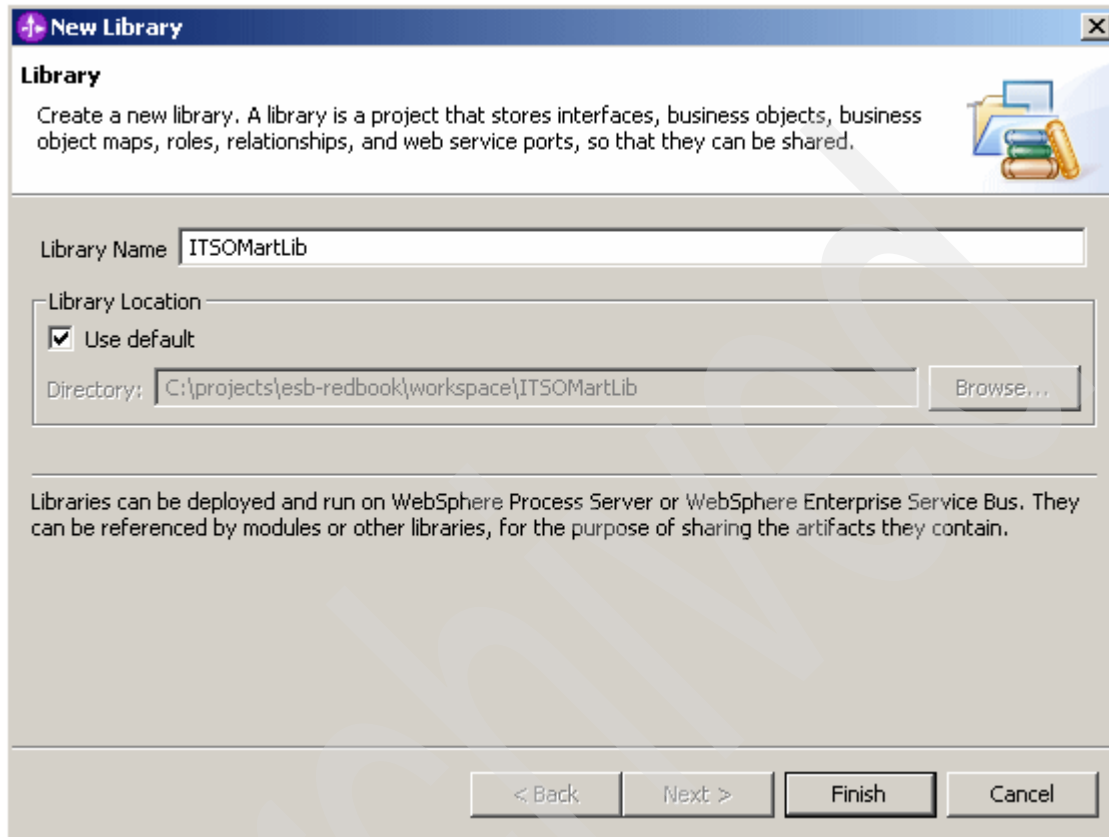


Figure 6-15 Naming a new library

6.4.2 Create business objects

Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice. A business object contains attributes, each of which has a name, a type (scalar type or another business object), a default value (for scalar types), and cardinality. Business objects can extend (define a super-set of attributes) other business objects through parent/child relationships. However, a business object can only inherit from a single parent.

Business objects can be created in the Data Types folder of mediation modules or their dependent libraries. If the business object is to be shared between modules then it should be created in a library.

To create a business object, do the following:

1. Select the **Data Types** folder in the library or project you want to create the objects in.
2. Right-click and select **New** → **Business Object**, as shown in Figure 6-16.

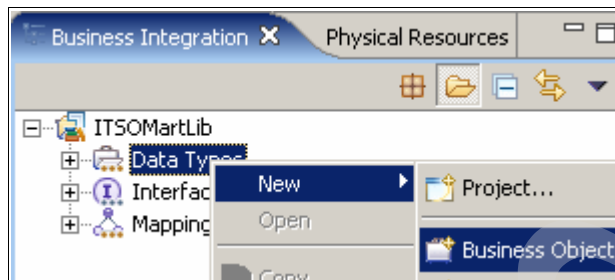


Figure 6-16 Create new business object

3. Enter the name for the business object in the Name field, as shown in Figure 6-17, and click **Finish**.

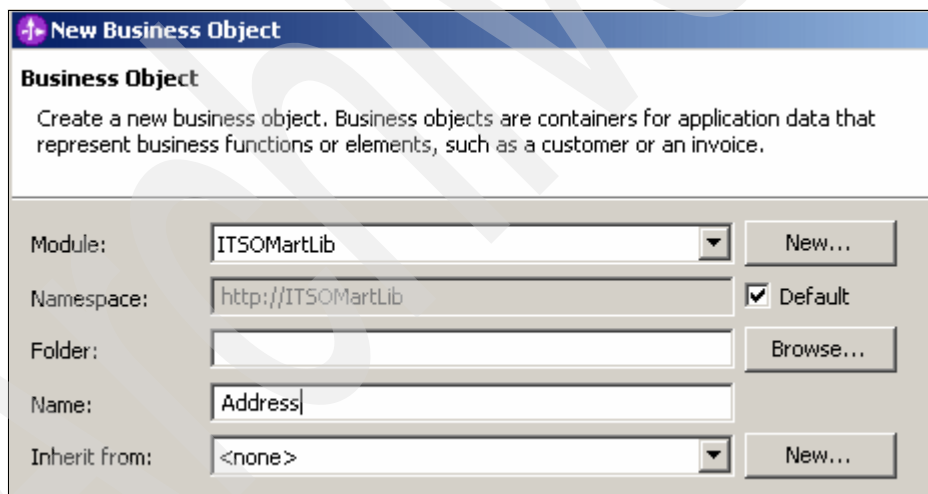



Figure 6-17 Defining the new business object

The new business object will be created and opened for editing.

To add an attribute to a business object, perform the following steps:

1. Select the object, right-click, and select **Add Attributes**. You can also select the  icon in the canvas. A new attribute entry will appear in the business object with the default name attribute1 and default type string.

2. In the new entry, type over the name of the attribute with the new name or change it in the Properties view. In Figure 6-18 you can see that we have named the new attribute *name*.

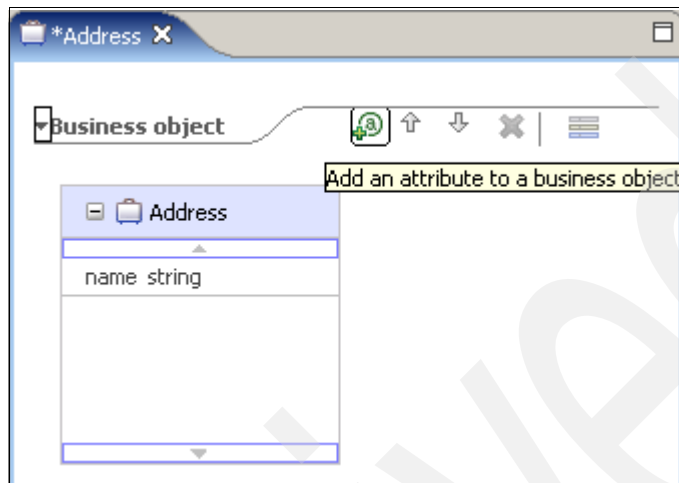


Figure 6-18 Add attribute to address business object

3. Select the correct type of attribute. The default is string, but you can change this by clicking **string** and selecting the new type from a drop-down menu, as shown in Figure 6-19.

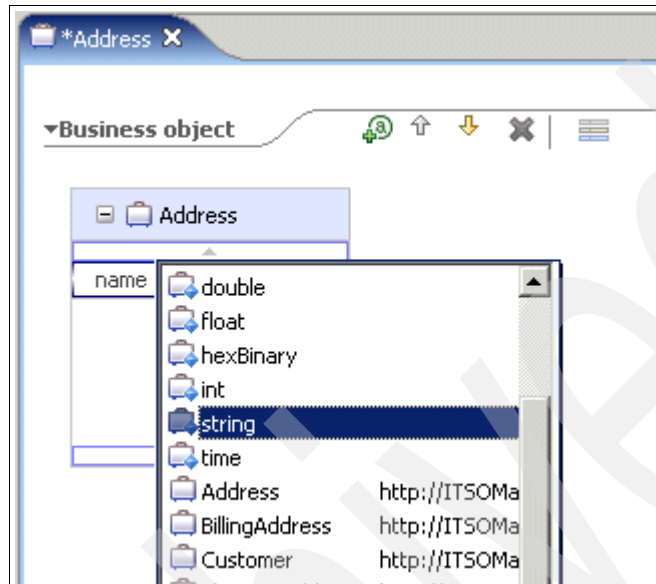


Figure 6-19 Select the attribute type

4. Continue to add attributes until the object is defined and save and close the object.

Note: You can save a file in WebSphere Integration Developer by selecting **File** → **Save** or pressing Ctrl+S. You can close an open file by selecting **File** → **Close** or clicking the **X** beside the object name at the top of the view. If you close a file and it needs to be saved, you will be prompted to save it.

6.4.3 Define interfaces

An interface provides the input and output of a component. It is created independently of the internal implementation of the component. Interfaces can be created in mediation modules or libraries. If the interface is common to more than one module then it should be created in a library.

An interface consists of one or more operations and a Web Services Description Language (WSDL) type.

An operation is a description of an action implemented by the component. An operation may be a request/response type, meaning a request is sent and a response returned to the interface, or a one-way type, meaning only an input is sent and there is no response needed. Each operation in the interface defines the data that can be passed in the form of input to and output from the component when the operation is invoked. Each operation may have one or more faults to handle error conditions. A one-way operation only has an input. The WSDL type specifies the protocol and data format of the operation.

To build an interface, perform the following tasks:

1. Select the **Interfaces** folder in the library or module where you want to create it.
2. Right-click and select **New** → **Interface**, as shown in Figure 6-20.

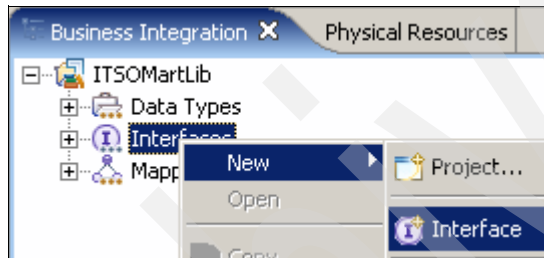


Figure 6-20 Create an interface

3. Enter the name for the interface in the Name field and click **Finish**.

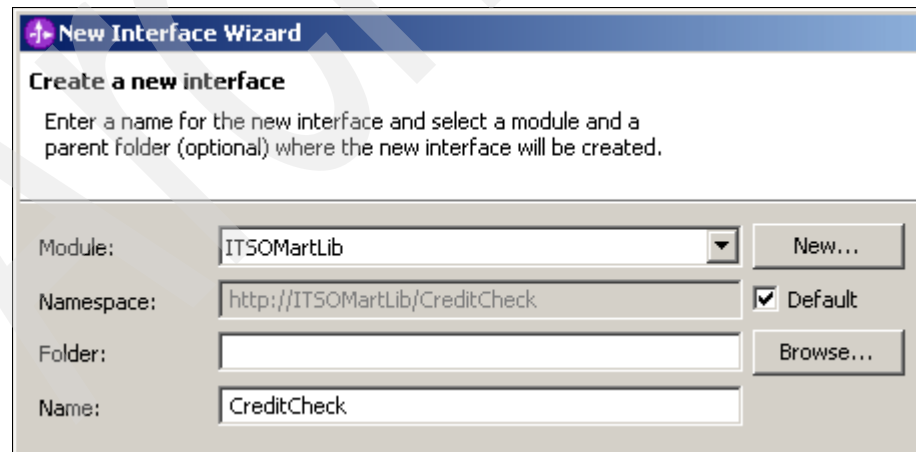


Figure 6-21 Naming the new interface

The new interface will open in the editor area.

4. Add a new operation (one-way or request/response) by selecting the appropriate icon (Figure 6-22).

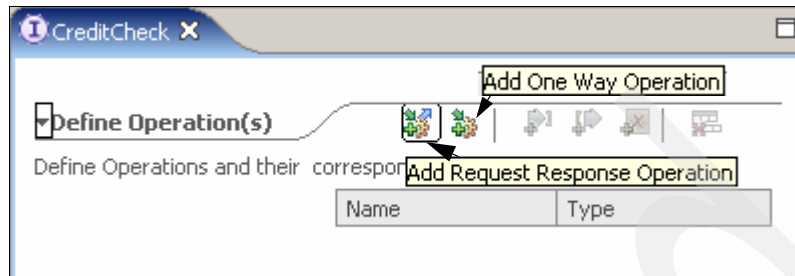


Figure 6-22 Add request response operation

A new operation will be added to the interface with the default name of Operation1. Change this name to an appropriate name by typing over it, as shown in Figure 6-23. When the new operation has been successfully added, the rest of the icons at the top of the panel will be activated.

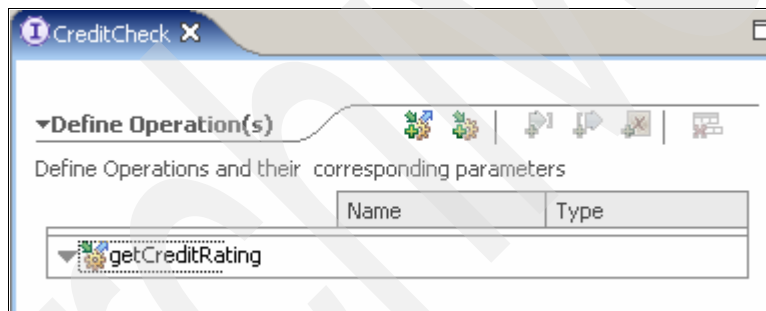


Figure 6-23 Give the operation a name

5. Add input, output, or fault information to the operation by clicking the appropriate icons and completing the information.

A request/response operation will have an input describing the data that flows into the component from the interface and an output describing the data that flows back from the interface. It can also have a fault to handle error conditions.

A response operation will only have an input defined.

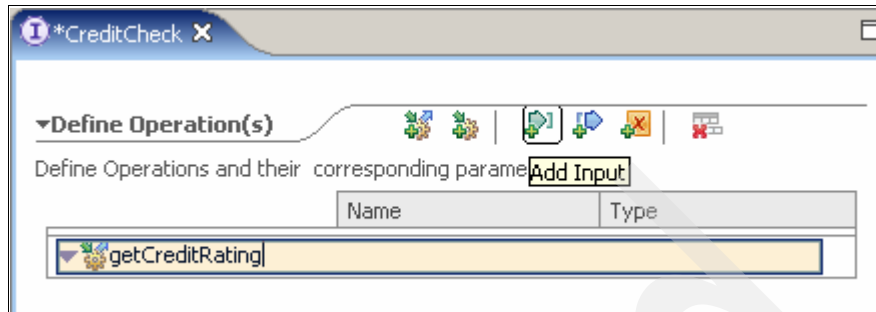


Figure 6-24 Add input

Regardless of the type (input, output, or fault), a new entry will appear on the canvas under the new operation, as shown in Figure 6-25. It will have a default for the name and type specified.

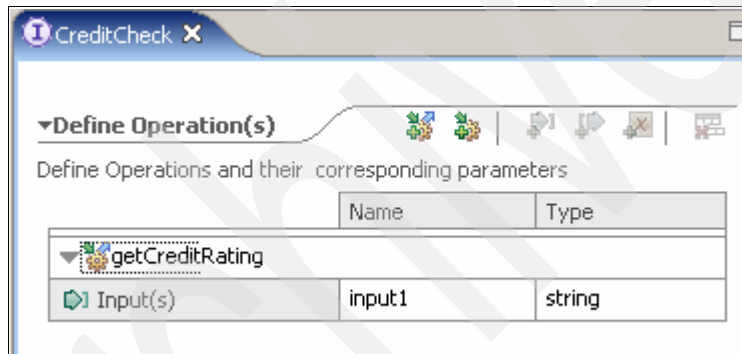


Figure 6-25 New input

Change the name by typing over the default name.

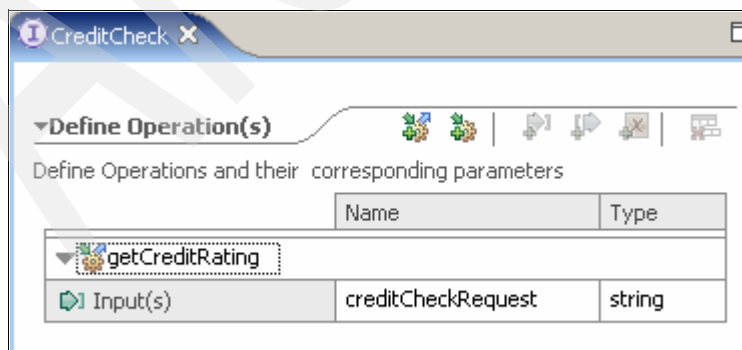


Figure 6-26 Give the input a name

Change the type by clicking the default type (string) and selecting a new type from the pull-down. This will contain a list of legitimate types and business objects to choose from (Figure 6-27).

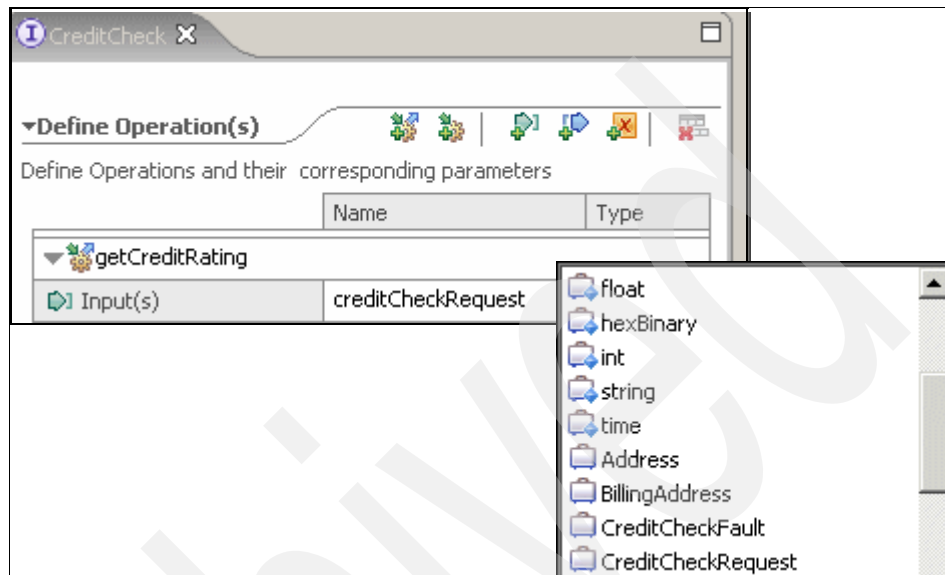


Figure 6-27 Select the type field

You can also use the Properties view to change these values.

6. Save and close the interface.

6.4.4 Create a mediation module

The wizard provided in WebSphere Integration Developer to create the mediation module allows you to do several things during the process:

1. Name the module.
2. Specify that the wizard also create the mediation flow component. Select this option.
3. Specify the target server. Select **WebSphere ESB Server**.
4. Select required libraries. Be sure to select your library with the business objects and interfaces.

To create a mediation module, perform the following tasks:

1. Right-click in the Business Integration view and select **New** → **Mediation module**, as shown in Figure 6-28.



Figure 6-28 New mediation module

2. Name the mediation module and allow the wizard to create the mediation flow component. This option is checked by default (Figure 6-29). Click **Next**.

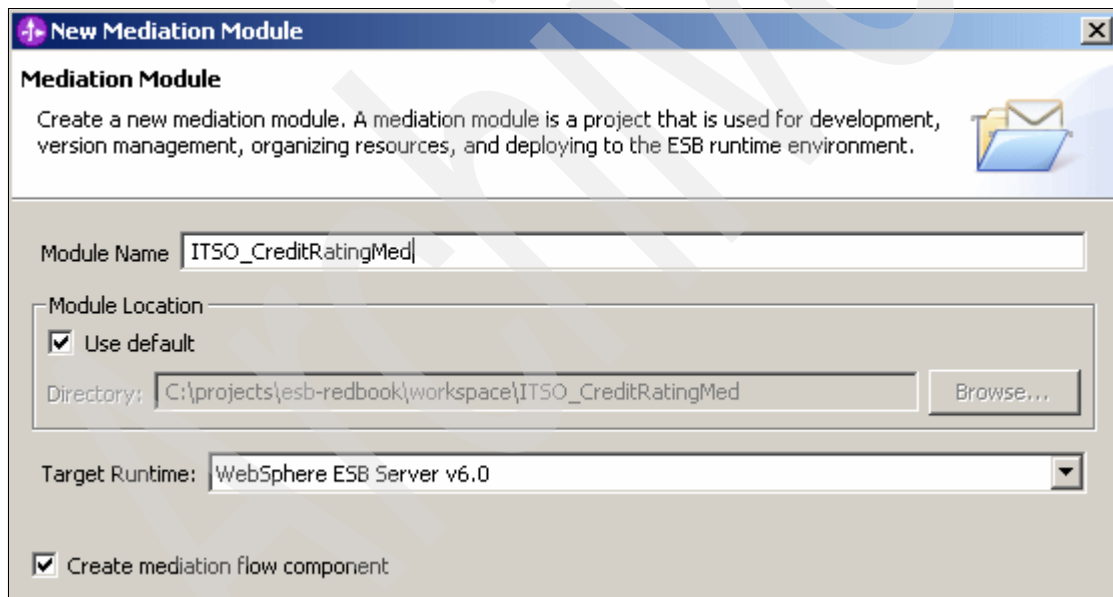


Figure 6-29 Name the mediation module

3. Select the libraries you want to include as a dependency, as shown in Figure 6-30, and click **Finish**.

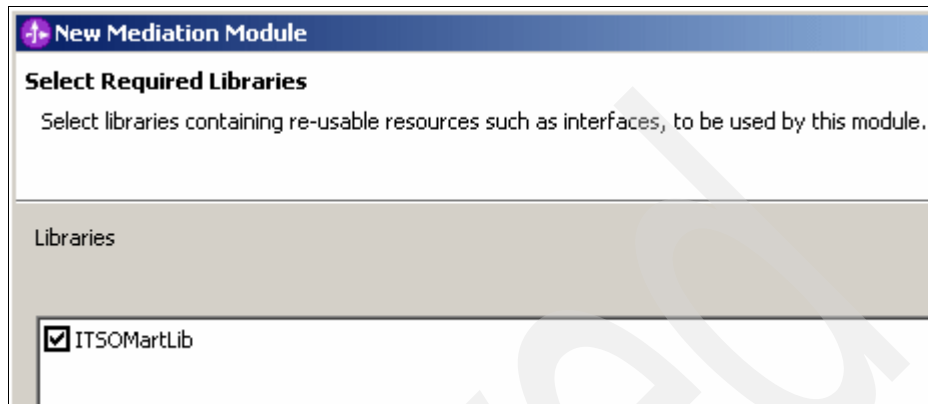


Figure 6-30 Add library

The new mediation module will be created. You will see the new structure in the Business Integration view, including an entry for the module assembly, as highlighted in Figure 6-31.

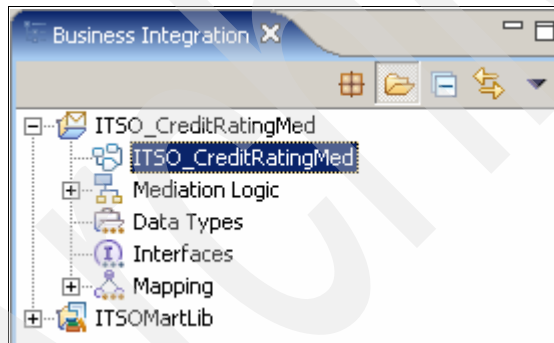


Figure 6-31 Module assembly

Adding dependent libraries to a mediation module

As you just saw, dependent libraries can be added to a mediation module when you create it. If you want to add more libraries in the future after creating the mediation module, perform the following steps:

1. Double-click the mediation module in the Business Integration view. This will open the module dependencies in the editing area, as shown in Figure 6-32. Click **Add** under the Libraries section.

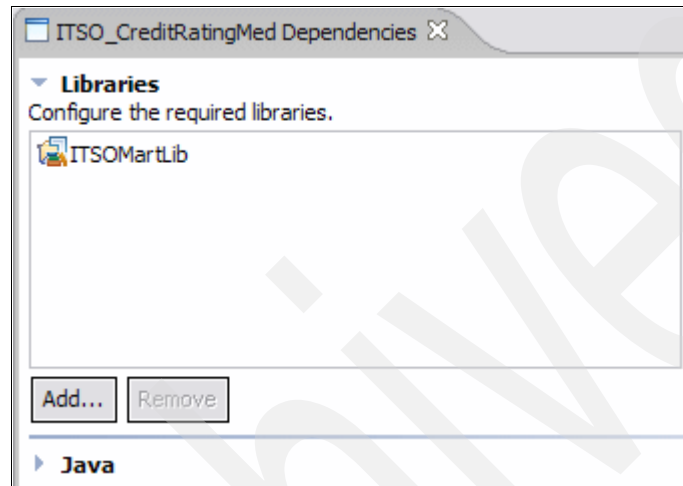


Figure 6-32 ITSO_CreditRatingMed Dependencies

2. Select the library you want to add in the next window, as shown in Figure 6-33, and click **OK**.

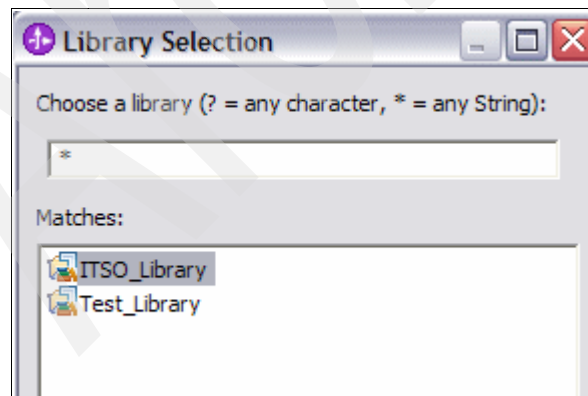


Figure 6-33 Library selection

You will see the library added in the list of required libraries, as shown in Figure 6-34.

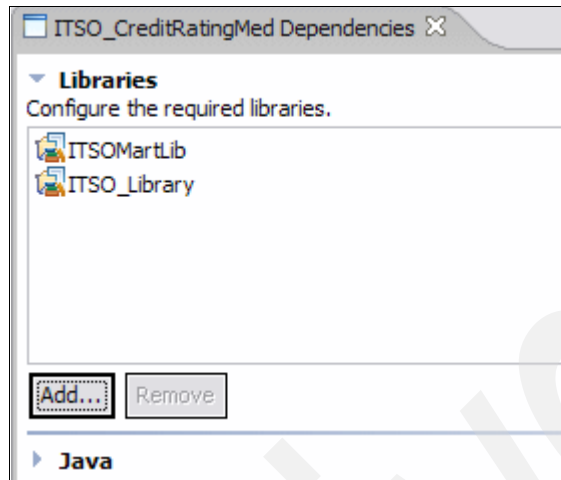


Figure 6-34 Required libraries

3. Save the changes and close dependencies.

6.4.5 Complete the module assembly

Once the new module is created you can double-click the module assembly in the Business Integration view to open it. Initially, the assembly will consist of a mediation flow component labeled Mediation1, as shown in Figure 6-35.



Figure 6-35 Mediation flow component

Completing the module assembly consists of adding an export for the module and imports for services you plan to use, and wiring these components together to create the necessary interfaces and references. Imports and exports are then bound to a particular transport with the details required for the runtime environment.

Adding and binding imports and exports

There are a variety of methods you can use to add import and export components to a module assembly:

- ▶ You can drag and drop a Web service port to the assembly diagram.
- ▶ You can drag and drop an interface to the assembly diagram.
- ▶ You can create an export using the context menu of the mediation flow component.
- ▶ You can use the Import and Export tools to add a component.

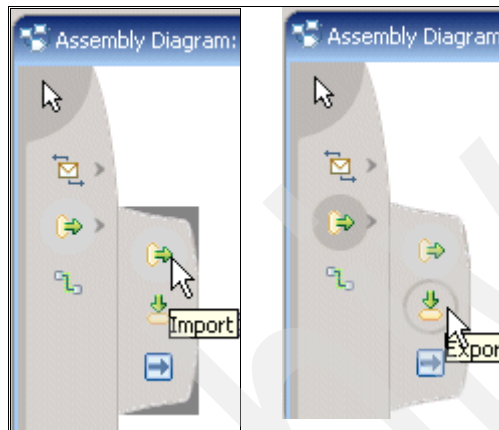


Figure 6-36 Import and export tool

Most of these methods allow you to create a binding at the same time, or you can delay and add the binding later using the context menus of the import and export components.

Here are common ways to add and bind imports and exports.

Import with Web service binding

When you are working with Web services as imports, you will need to obtain a copy of the WSDL describing the service and copy it to your mediation module or library. When you do this, new entries for the data types, interfaces, and Web service ports required by the Web service are created automatically. You will see an example of this later in 7.3.3, “Define the interface to the Credit Rating Service Web service” on page 278.

To create an import for a Web service, drag and drop the Web service port to the canvas. You will be prompted to select a component and binding type, as shown in Figure 6-37.

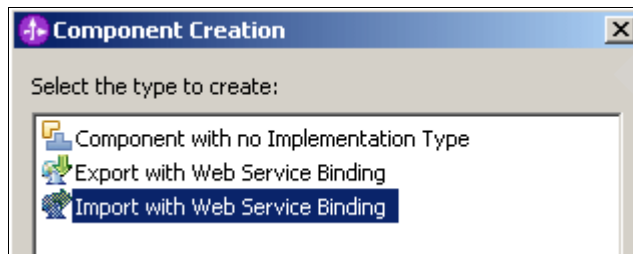


Figure 6-37 Select component and binding type

Export with Web service, SCA, or JMS binding

You can create an export from the context menu of the mediation flow component, as shown in Figure 6-38.

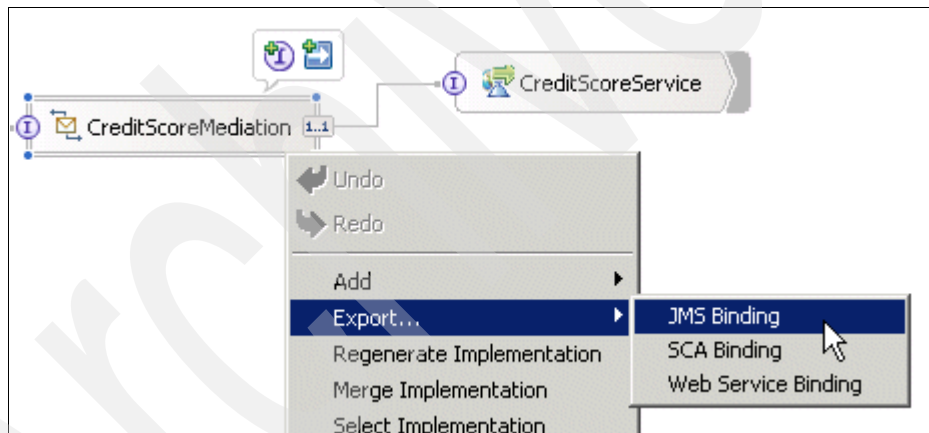


Figure 6-38 Create an export from the mediation flow component

The new export will be wired automatically to the mediation flow component, and the appropriate interface and reference will be automatically added.

Import or export with JMS binding

To create an import and bind it to JMS, drag and drop an interface onto the canvas. You will be prompted for the component and binding type, as shown in Figure 6-39.

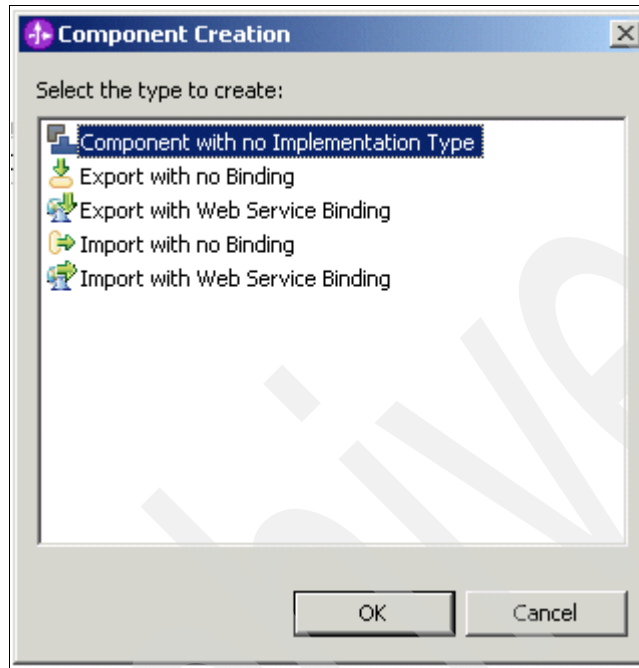


Figure 6-39 Select component type and binding for a new component

To use a JMS binding, select the import or export with no binding option and click **OK**.

Once the import has been added, you can generate the binding by selecting **Generate Binding** → **JMS Binding** from the context menu for the component.

Note: If you drop the interface onto a mediation flow component, you will only get the interface defined to the component, but not a new import or export component.

Import with adapter binding

For adapter services, use the Enterprise Service Discovery wizard to create an import or export. The wizard also creates the bindings.

Adding binding details

The binding contains implementation-specific details for the transport. The details are added using the Properties view for an import or export component that has been bound. For example, a Web service binding would provide the port and service information for the Web service, as shown in Figure 6-40. This information is generated automatically from the WSDL information.



Figure 6-40 Web service binding

For JMS bindings, you can enter data that includes information such as the JNDI lookup name of the destination queue and queue connection factory, as shown in Figure 6-41. Many of these fields reflect resources defined on the WebSphere ESB server. If they are not filled in, values will be automatically generated for you and the required server configuration will be performed when you deploy the mediation module.

The screenshot displays the 'Import: RegistrationLog - Success (JMS Binding)' configuration window. It features a tabbed interface with 'JMS Import Binding', 'Connection', 'Resource Adapter', 'JMS Destinations', and 'Method Bindings'. The 'JMS Destinations' tab is active, showing 'Send Destination Properties'. The configuration fields are as follows:

Property	Value
JNDI Lookup Name:	jms/ITSO_Mart/LogSuccessQ
Type:	javax.jms.Queue
Destination Properties	
Destination	
Bus Name:	
Delivery Mode:	Application
Time To Live:	
Priority:	Default
Advanced Messaging	
Read Ahead:	AsConnection
Queue	
Queue Name:	ITSO_RegLogMed.RegistrationLog-Success_SEND_D

Figure 6-41 JMS binding details

Adapter binding information shown in Figure 6-42 on page 250 is usually filled in by the Enterprise Service Discovery wizard.

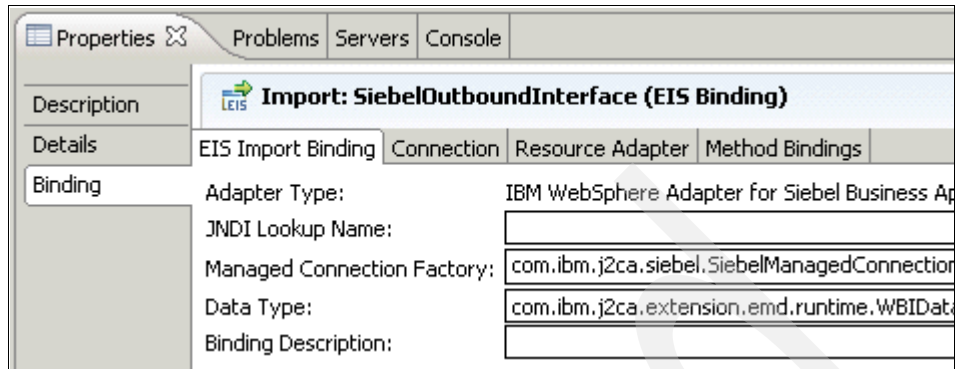



Figure 6-42 Adapter binding information

Wiring components

To complete the assembly diagram, any remaining unwired components need to be wired together.

1. Click the wire icon  and then click the first component.
2. Drag the wire from one component to the interface of the next.
3. You will be prompted and told that by adding the wire, a matching reference will be created on the source node. Click **OK**.

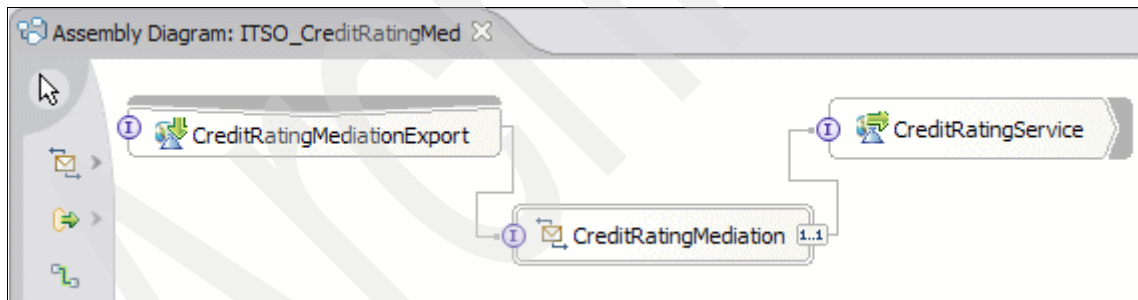


Figure 6-43 Wire the components

4. Click the arrow at the top left of the palette to get out of the wiring mode.

The assembly diagram is now complete. It contains the export that exposes the mediation to users, a mediation flow component that can be implemented to contain a mediation flow, and one or more imports to invoke services used by the mediation. At this stage we have only defined the flow of information, but not what happens to it as it passes through the mediation module. We do that next by generating the implementation for the mediation flow component.

6.4.6 Implement the mediation flow component

The implementation for the mediation flow component is the mediation flow. To generate the mediation flow, perform the following steps:

1. Right-click the mediation flow component and select **Generate Implementation**, as shown in Figure 6-44.

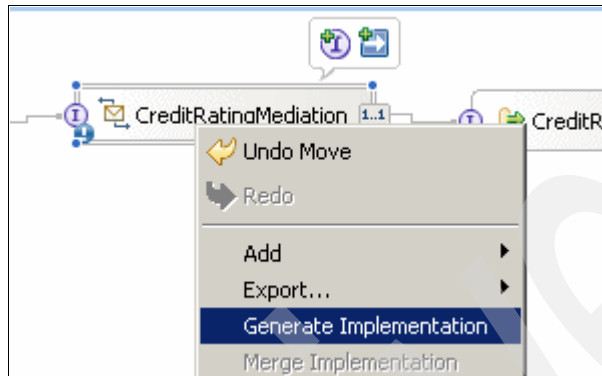


Figure 6-44 Generate implementation

2. Click **OK** in the next window. The Mediation Flow Editor will open with the new flow, as shown in Figure 6-45.

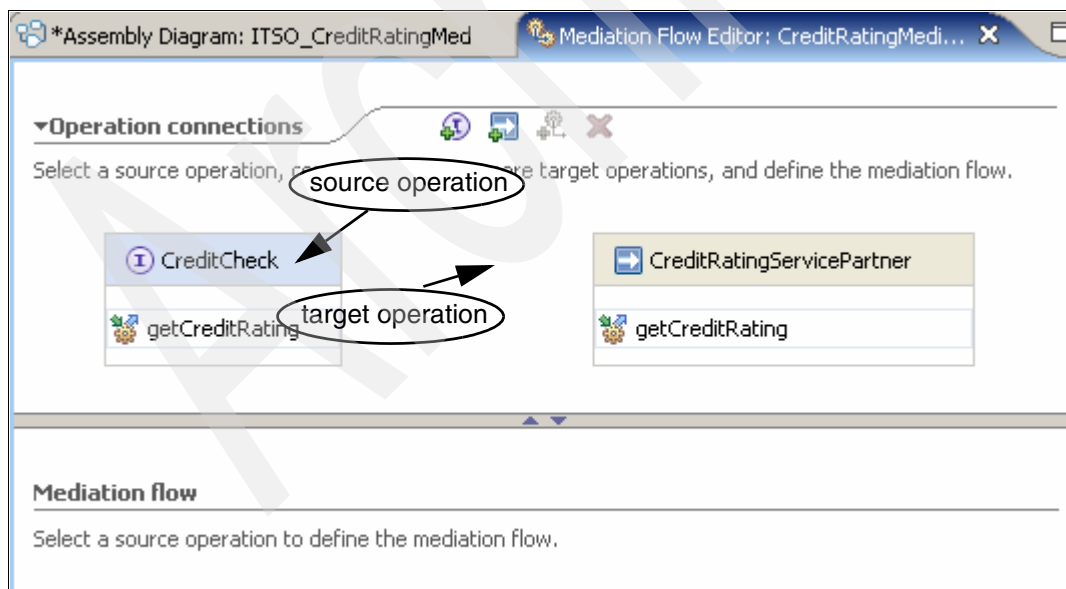


Figure 6-45 Mediation flow editor

3. The starting and endpoints of the mediation flow component are defined by source interfaces and target references. Source operations are connected to target operations and a flow is defined for each connection.

In the Operation connections window, you will see the interface and the reference used in this flow.

Click the **source operation** (you will see an input node in the Mediation flow pane) and drag it to the target operation.

This will create an operation connection between the two and generate the starting point and end-point nodes in the flow. The nodes generated will depend on the source and target. See Figure 6-46.

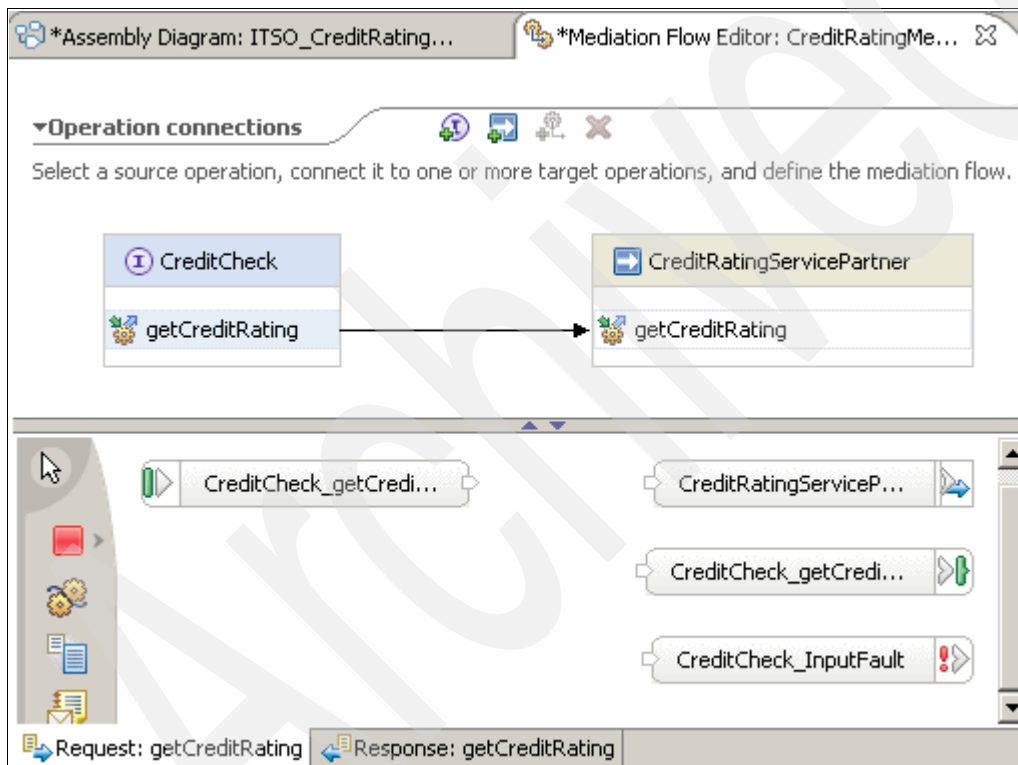


Figure 6-46 Mediation flow editor








If the interface to the mediation contains a request/response operation, you will have a request and response flow generated. You can switch between the two in the editor by clicking the appropriate tab at the bottom of the screen. In the case of a one-way operation (request), you will only have the request flow.

6.4.7 Build the mediation flow

A mediation flow consists of mediation primitives wired together. As a message flows through it, actions are performed by the mediations.

Table 6-1 lists the primitives supplied with WebSphere ESB and their toolbar icon and description.

Table 6-1 Mediation primitive types

Mediation primitives	Symbol	Description
Message Logger		To log message information to a database.
Message Filter		To filter messages, selectively forwarding them on to output terminals based on a simple condition expression.
Database Lookup		To access information in a database and store it in the message.
XSLT		To manipulate or transform messages using XSL transformation.
Stop		To stop a path in the flow without generating an exception.
Fail		To stop a path in the flow and generate an exception.
Custom		For custom processing of a message. Uses a custom SCA Java component for custom message processing.

Adding primitives

To add a mediation primitive to the canvas, use the toolbar on the left side of the mediation flow pane. Select the primitive and drop it on the canvas, as shown in Figure 6-47.

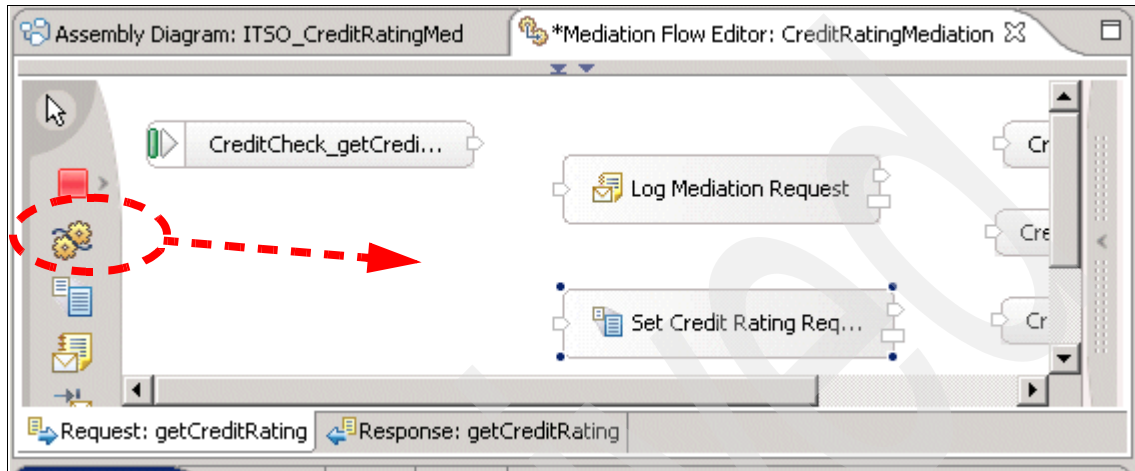


Figure 6-47 Adding a mediation flow primitive

Each mediation primitive has properties that detail what it does and how it does it. To set the properties for a primitive, select the primitive. The properties appear in the Properties view. The view will have the properties organized within tabs. You can rename the primitive on the Description tab. The Details tab, as shown in Figure 6-48, will have properties that detail what the primitive is to do.

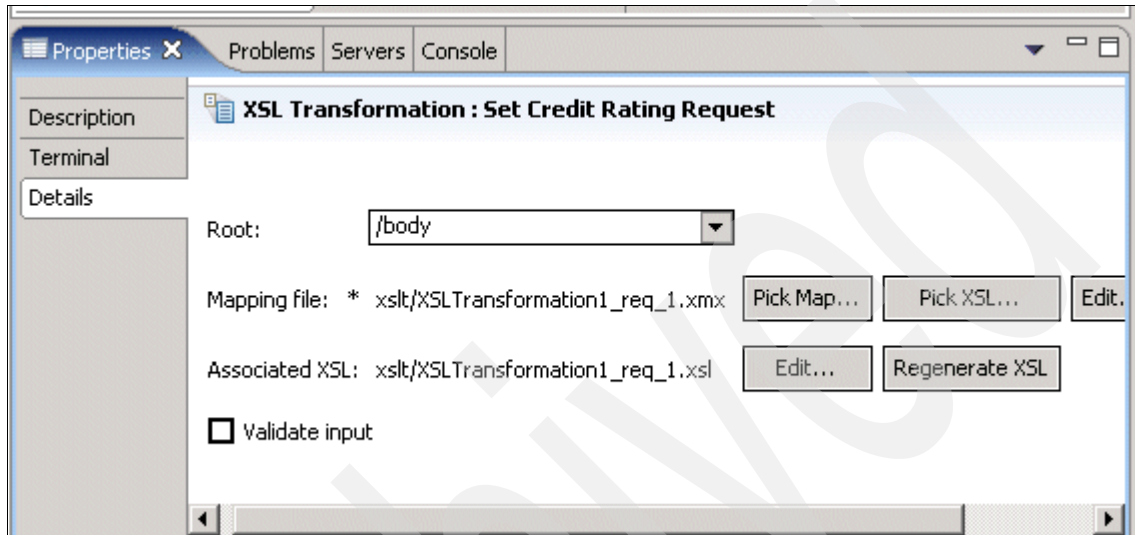


Figure 6-48 Properties for a mediation primitive

Wiring a mediation flow

Wiring the mediation flow defines the sequence in which mediation primitives are executed and assigns their terminal's message type.

Each primitive has an input terminal and one or more output terminals. Most primitives have at least an out and a fail terminal. In the case of the Message Filter primitive, the number of out terminals can vary depending on the filter implementation. Wiring connects an output terminal of one primitive or input node to the input terminal of another primitive or callout node.

As primitives are added to the mediation flow, they have no assigned message type. As one primitive is wired to the next, the next primitive is assigned the message type.

In a request flow, the input node represents the entry point to the mediation flow. In a response flow, the callout response node represents the entry point to the mediation flow.

To wire one node to another, click the out terminal of the first node and drag the wire to the in terminal of the next node. A wired mediation flow will look like Figure 6-49.

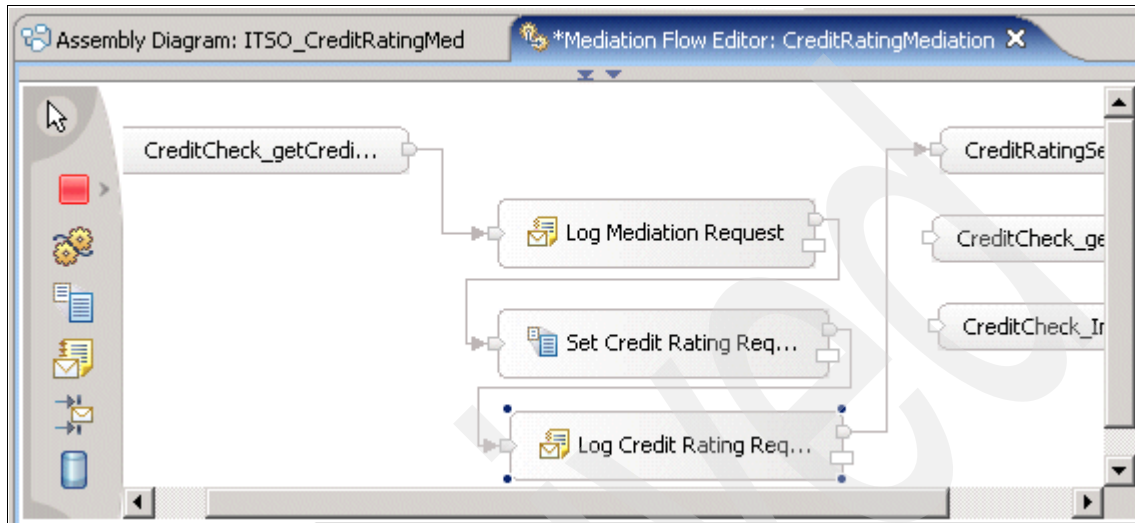


Figure 6-49 Wired mediation flow

6.5 Testing mediations

WebSphere Integration Developer provides a full array of test server environments and clients that support testing for integration and J2EE modules.

6.5.1 Test servers

WebSphere Integration Developer provides a test environment for integration modules. Three server configuration modes are supported:

- ▶ Local test environments

At installation time you have the option of installing the integrated test environment and associated profiles. Each workspace that you start will have a pointer to the profiles that you install.

- ▶ Local separate installations of WebSphere ESB

You can use a separate installation of the runtime as your test environment. If you have installed a separate instance of WebSphere ESB, WebSphere Application Server, or WebSphere Process Server on your local machine, you can create a new workspace server configuration within WebSphere Integration Developer that points at the profile of your choice.

► Remote test environments

When configuring a test environment, the server can be either a local integrated server or a remote server. Once the server itself is installed and configured, the server definition within WebSphere Integration Developer is very similar for local and remote servers.

You can see the pointers to the local test environment profiles when you start the Workspace by switching to the Servers view, as shown in Figure 6-50.

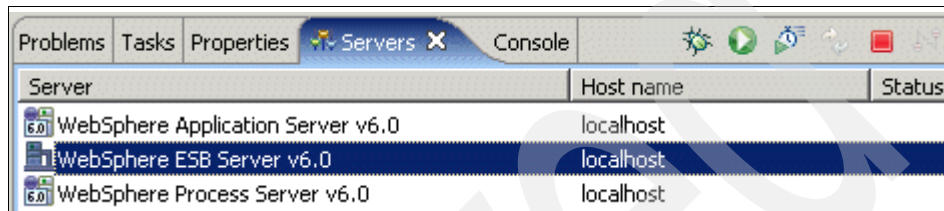




Figure 6-50 Local test servers

Tip: The profile is independent of the workspace, meaning that changes you make or applications you install using one workspace will be there even if you switch workspaces. If you would like to use a separate profile (server) for each workspace, you can create a new profile and change the workspace pointer to the profile. See “Creating a new server in the test environment” on page 640 for information about how to do this.

Mediation modules can run in the WebSphere Process Server test server or in the WebSphere ESB test server. The testing for the ITSOMart solution was done using the local WebSphere ESB Server V6.0 test server environment.

Using icons or the context menu, you can manage the servers. You can start  or stop  a server from this view, add or remove a project, open the administrative console, and do other common administrative tasks.

Applications and integration modules can be automatically packaged and deployed to a test server using the context menu for the server. Simply right-click the server and select **Add and remove projects**. The screen shown in Figure 6-51 will open. Select the projects you want to run on the server and click **Add**. WebSphere Integration Developer will start the server and deploy the applications to it.

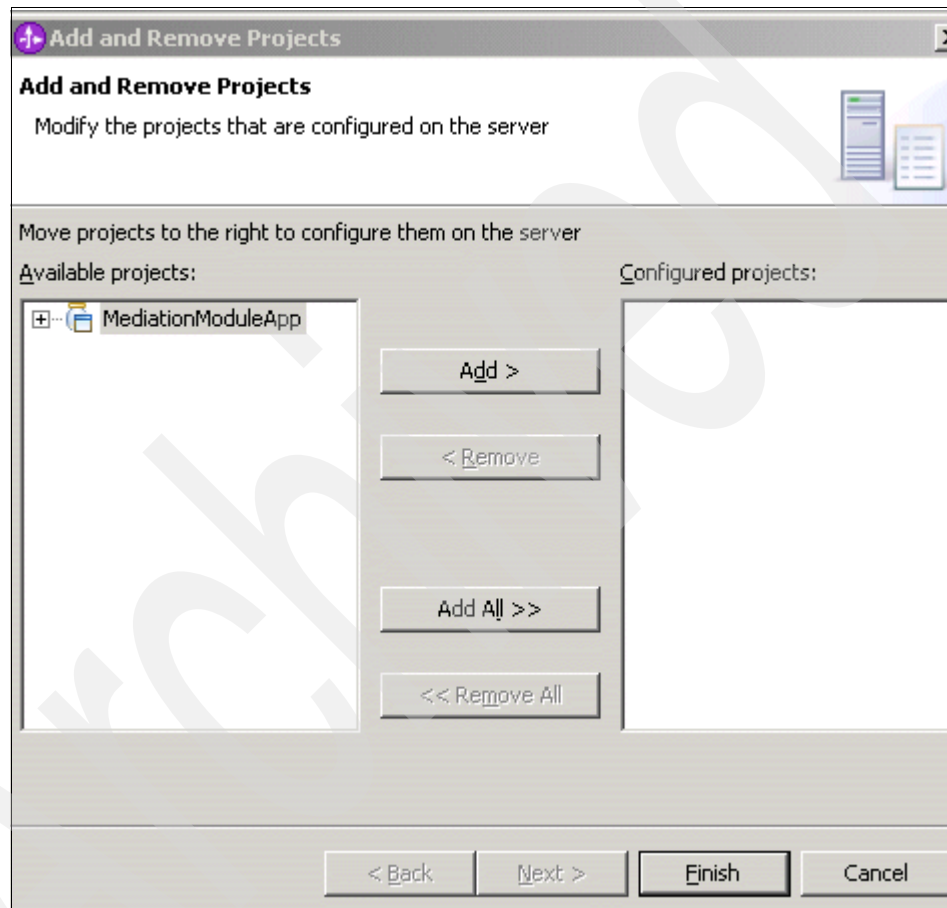


Figure 6-51 Add projects to the server

The servers are full servers running in a normal environment, and as such, can also be managed using the standard WebSphere Application Server administrative tools. Once the application has been deployed, you can open the WebSphere administrative console and view or manage the application.

The Console view will show the runtime messages produced by the server, as shown in Figure 6-52.

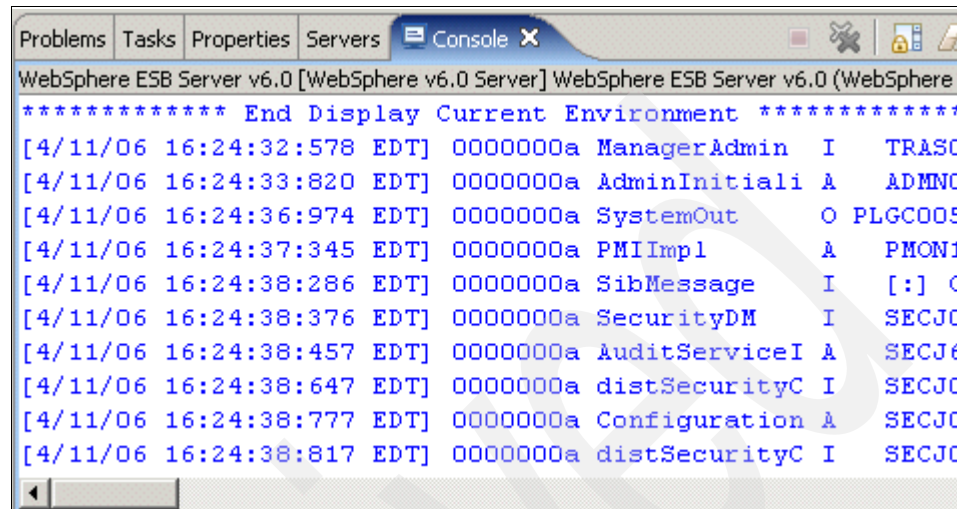


Figure 6-52 Test server console

Remember, you can maximize this view by double-clicking the bar at the top.

6.5.2 Test client

Once the module is deployed or published to the test server you can test it with the Integration Test Client. Typically you will perform module tests and component tests.

Module test

In the Business Integration view, right-click the module and select **Test** → **Test Module**. This will launch the Integration Test Client with all emulation disabled.

Component test

In the module assembly diagram, right-click the **mediation flow component** and select **Test Component**. This will launch the Integration Test Client with emulators configured to emulate any component references so you can test the component in isolation.

You will see examples of these later as the ITSOMart solution is developed and tested.

6.6 Packaging the mediation for deployment

Mediation modules are deployed to the WebSphere ESB server as EAR files. To deploy to the server you must first export the module as an EAR file and make it available to the server. Then you can install the module as an application using the WebSphere administrative console.

To export modules as EAR files:

1. Select **File** → **Export**.
2. Select **Integration module** and click **Next**.
3. Check the box to the left of the mediation module. Select **EAR files for server deployment** and click **Next**.

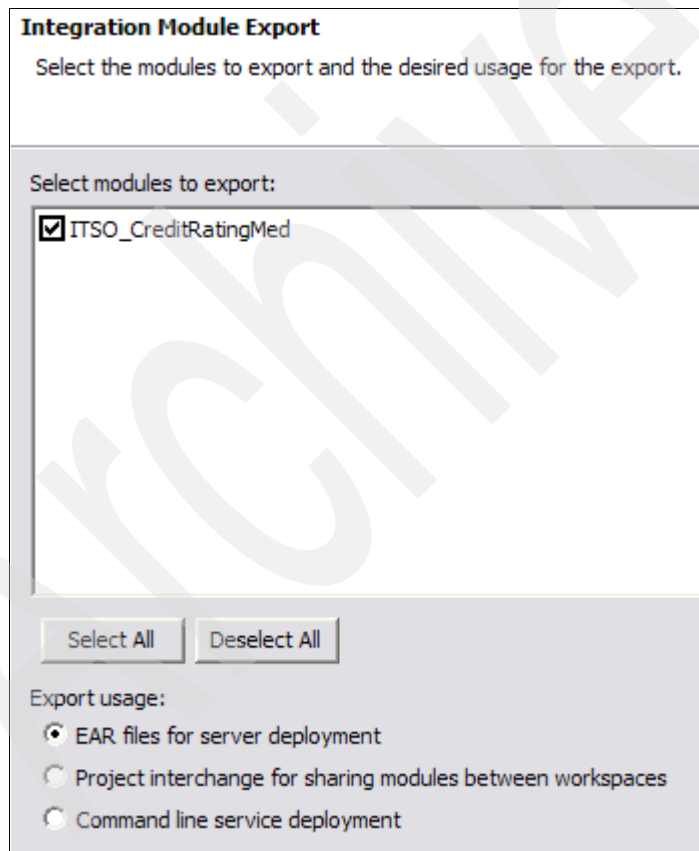


Figure 6-53 Integration module export

4. In the Target directory field, type the path and name of the target directory where you want to export the EAR file. Note the EAR file name and click **Finish**.

The exported EAR file will be stored in the target directory and is ready to be installed on a WebSphere application server or cluster. If the application server does not have access to the target directory, you will need to move the EAR file to a location where the server has access.

For information about deploying EAR files to WebSphere Application Server, see 11.10.2, “Deploy an EAR file” on page 524.

Archived

Building the Credit Rating and Credit Score mediations

This chapter shows an example of using the capabilities of WebSphere ESB to enhance direct connectivity between a service requester and a service provider. Direct connectivity implies a one-to-one connection between requester and provider, and in a SOA environment that connection can benefit from the use of an ESB for protocol and message transformation, security, and shielding the requester from the connection details of the service provider. The mediation built for this example illustrates the following:

- ▶ SOAP/HTTP transport
- ▶ Request/response operation
- ▶ XML transformation of messages
- ▶ Message logging
- ▶ Database access
- ▶ Invoking one mediation from another
- ▶ Fault handling

This chapter includes the following topics:

- ▶ Scenario overview
- ▶ Preparing for the ITSOMart mediations
- ▶ Developing the Credit Rating mediation
- ▶ Developing the Credit Score mediation
- ▶ Calling the service from the application

7.1 Scenario overview

This chapter focuses on the Get Credit Rating scenario of the ITSOMart solution.

Note: The samples included in this chapter can be downloaded from the Web. See Appendix A, “Sample application install summary” on page 589, for instructions on downloading and importing the sample projects.

7.1.1 Business scenario

The first step in the Register Customer process is to evaluate the customer credit rating. The process will use the rating to determine how to proceed with the registration. The expected format of the rating is one of three values: gold, silver, or bronze.

In the first stage of the solution implementation, ITSOMart uses a credit service that returns these values. In the second stage, a new credit service is substituted. The new service returns a numerical value that must be converted to the expected value of gold, silver, or bronze. this conversion will be done in the ESB based on values stored by ITSOMart financial executives in a database.

The activity diagram for the Get Credit Rating scenario of the ITSOMart solution can be seen in Figure 7-1. The activity diagram for the entire process can be seen in Figure 5-15 on page 143.

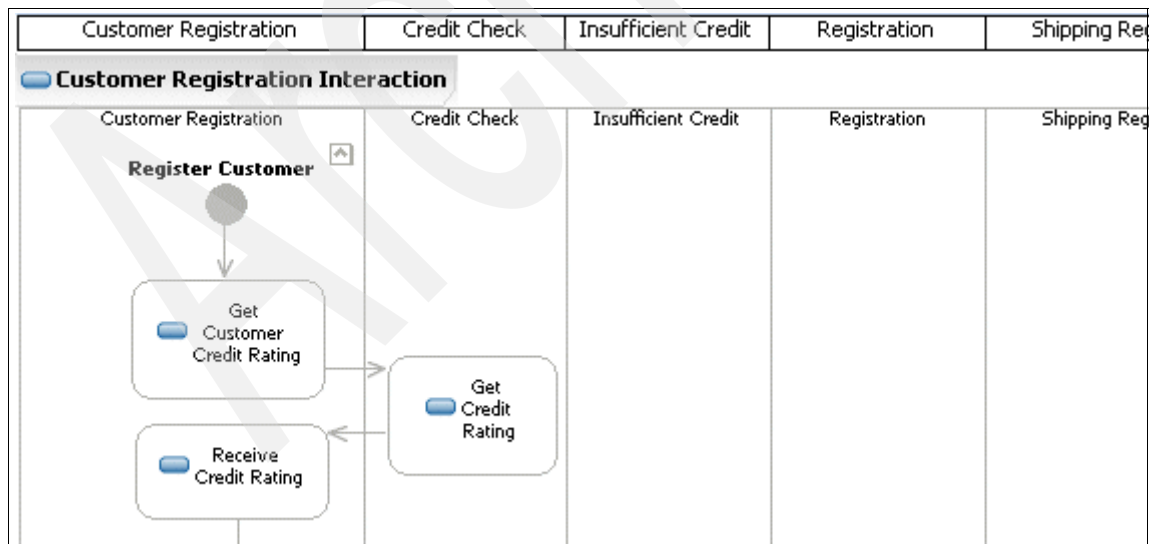


Figure 7-1 Get Credit Rating scenario

7.1.2 Get Credit Rating scenario stage 1

The call from the Register Customer process will call the Credit Rating Service via the ESB. The call will be mediated in the ESB to transform the request into the proper format for the service.

Figure 7-2 shows the activity diagram for the mediation.

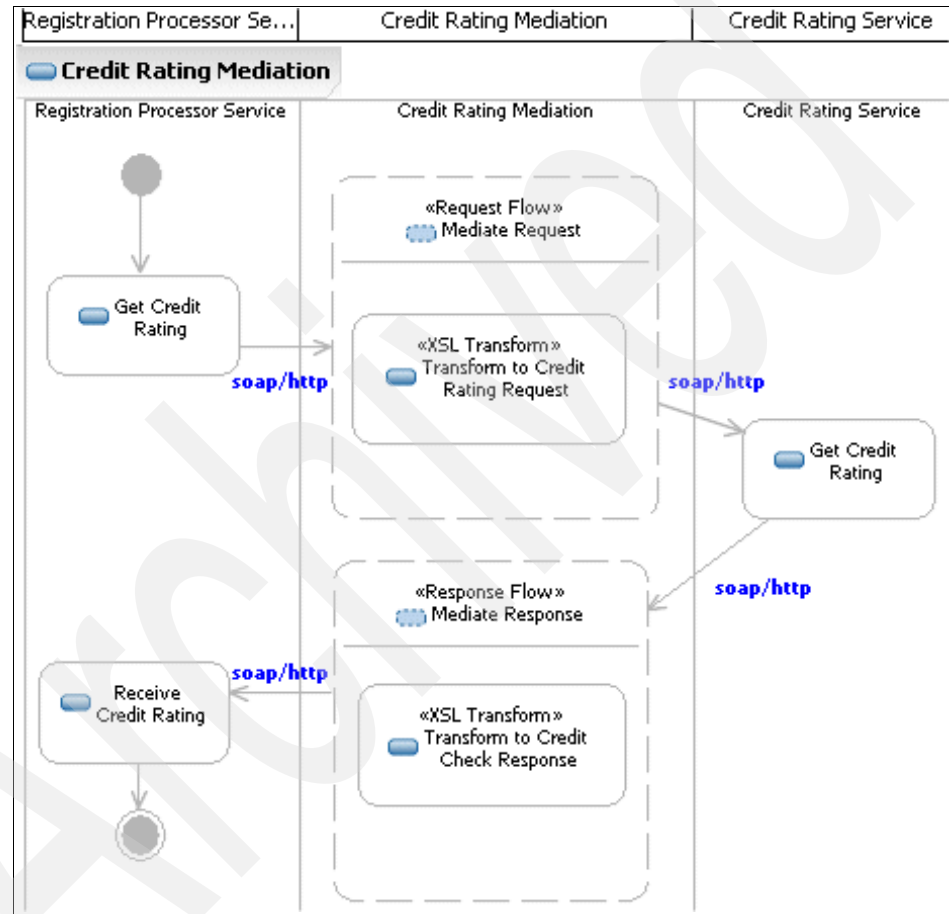


Figure 7-2 Credit Rating mediation activity diagram

As you can see, SOAP/HTTP will be used as the transport protocol for invoking the mediation and for calling the Credit Rating Service.

The interface for the mediation will define the input and output for the mediation. The input will consist of customer registration data. The response will consist of a string (gold, silver, or bronze) that describes the customer's credit rating.

7.1.3 Get Credit Rating scenario stage 2

In stage 2, ITSOMart has switched Credit Rating Services. The new service returns a numerical value for the credit rating. The Register Customer process expects a text value. Rather than change the process, ITSOMart will use a mediation in the ESB to convert the numerical value to an acceptable text rating value.

Rather than call (import) the Credit Rating Service, the Credit Rating mediation will now call the Credit Score mediation, which will call the Credit Score Service to get the rating and convert the response, sending it back to the Credit Rating mediation. You can see this new flow in Figure 7-3.

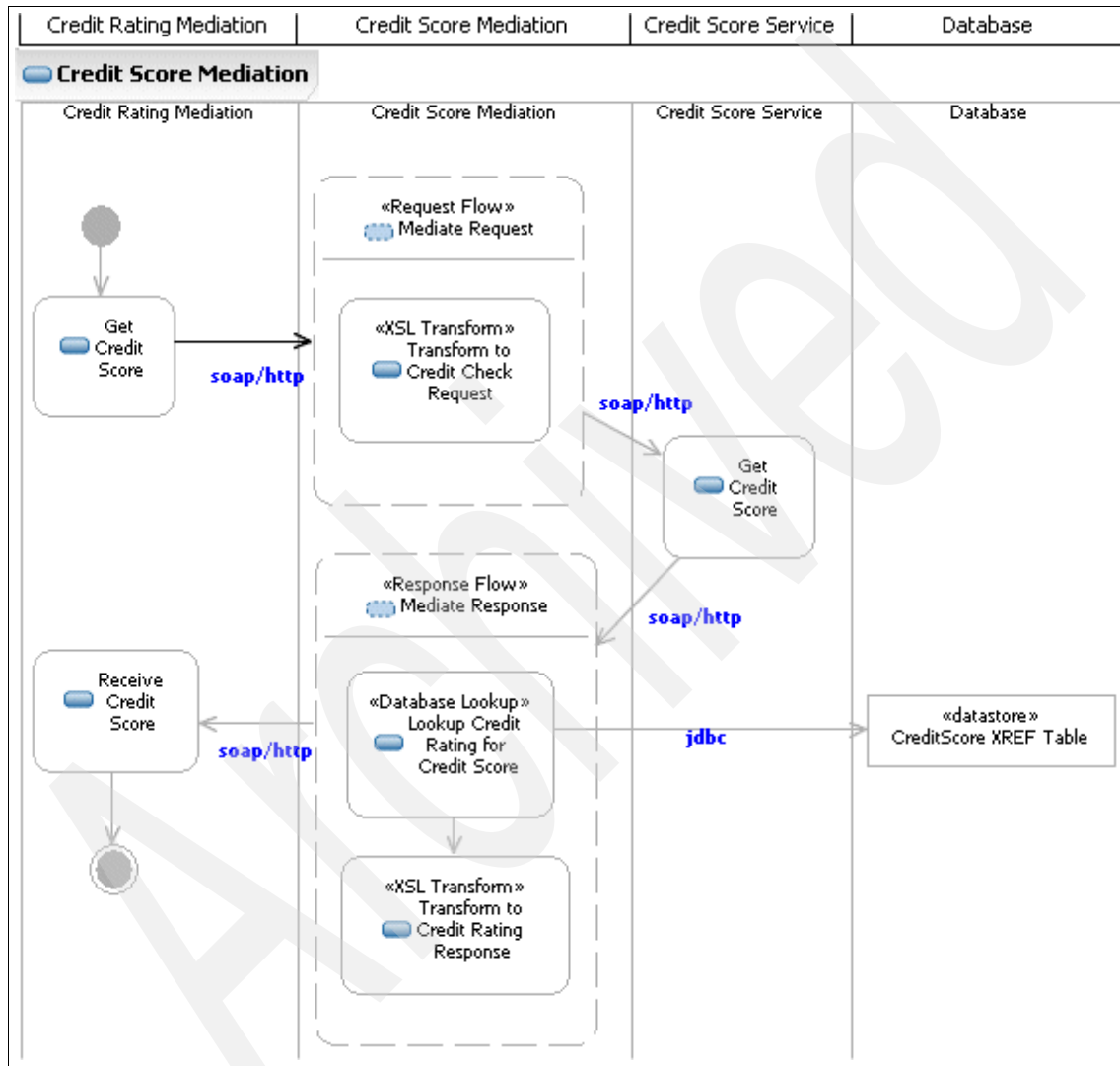


Figure 7-3 Credit Score mediation activity diagram

Note: The decision on whether to call one mediation from another is an architectural decision. In this case we are attempting to illustrate the capabilities of mediations and not necessarily the best practices.

Once again, the mediation and service are invoked using SOAP/HTTP. A JDBC call is used in the mediation to access the database containing the numerical credit score to credit rating comparison tables.

7.2 Preparing for the ITSOMart mediations

Note: If you are not familiar with using WebSphere Integration Developer to develop mediations, review Chapter 6, “Assemble with WebSphere Integration Developer” on page 211, before proceeding.

To build and test this scenario you will need the following projects in your workspace:

- ▶ **ITSOMartUtils:** This Java project contains a utility class that is required for some of the XSLT functions.
- ▶ **ITSO_CreditRatingService:** This project contains a Web service that returns a credit rating (gold, silver, or bronze).
- ▶ **ITSO_CreditScoreService:** This project contains a Web service that returns a numerical credit score.
- ▶ **MessageLogApp:** This project contains a J2EE application that can be used to display the contents of the logs populated by the Message Logger mediation primitives.

To prepare for the mediation, we are going to do the following:

- ▶ Create a library.
- ▶ Create the common business objects.

These resources are common to all the mediations illustrated in this book.

7.2.1 Create a library

The first step in preparing for building the mediation is to create a library to hold common resources that will be used across the mediations in this sample. The library can be added as a dependency library to any mediation modules that require its resources.

To create the library, do the following using the Business Integration perspective in WebSphere Integration Developer:

1. Right-click in the white space of the Business Integration view and select **New → Library**.
2. Enter ITSOMartLib as the library name and click **Finish**.

7.2.2 Create the common business objects

Next we take you through the process of creating the business objects listed under the Data Types folder shown in Figure 7-4. These data objects provide the common basis for the data that flows through all of the mediations.

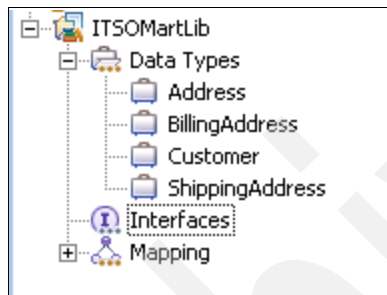



Figure 7-4 Create business objects

Create the Address business object

The Address business object contains fields that hold customer billing or shipping address data. This object will be used as the basis for several other business objects.

1. Select the **Data Types** folder in ITSOMartLib, right-click, and select **New → Business Object**.
2. Enter Address in the Name field.
3. Click **Finish**. The business object will open in the editing area.
4. Add the following attributes to the Address business object. All have an attribute type of string.
 - name
 - street
 - city
 - zipcode
 - country
 - phone

To add an attribute:

- a. Click the  icon above the business object in the editing area.
- b. Type over the name of the attribute with the new name or change it in the Properties view.
- c. Click the type and select the new type from the pull-down. Since string is the default, you will not need to do this for the attributes in Address.

The final results for the Address business object should look like Figure 7-5.

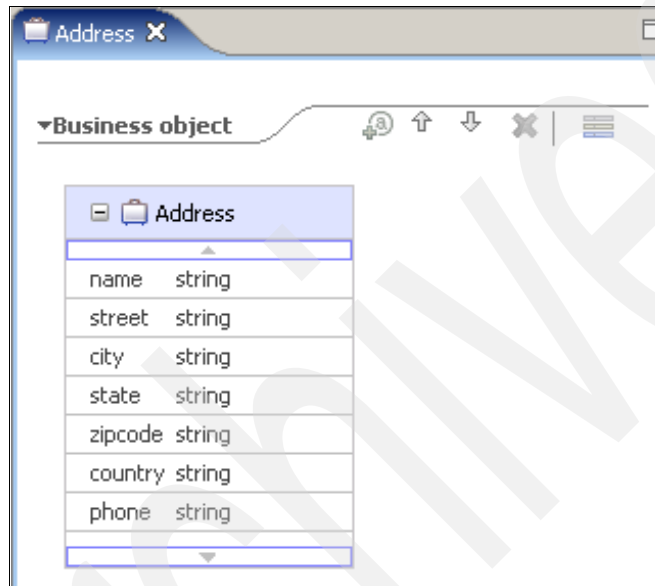


Figure 7-5 Address business object attributes

5. Save and close the business object.

Create the BillingAddress and ShippingAddress objects

The steps to create the BillingAddress and the ShippingAddress business objects are the same. We will take a look at one of them, and the other is built following the same steps. Both business objects will inherit the characteristics from the Address business object.

1. Select the **Data Types** folder in ITSOMartLib and right-click.
2. Select **New → Business Object**.
 - a. Enter BillingAddress in the Name field.

- b. Select **Address** in the Inherit from field (Figure 7-6).

New Business Object

Business Object

Create a new business object. Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice.

Module: ITSOMartLib New...

Namespace: http://ITSOMartLib ☒ Default

Folder: Browse...

Name: BillingAddress

Inherit from: Address { http://ITSOMartLib } New...

Figure 7-6 Defining the new business object

3. Click **Finish**. The new business object will have the same attributes as Address. We will not be adding new attributes. The results should look like Figure 7-7.

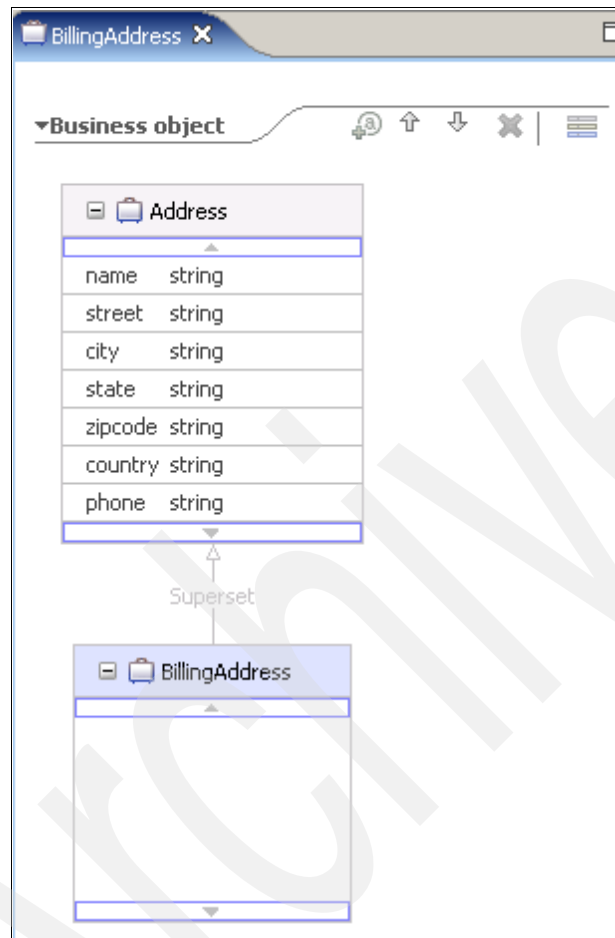


Figure 7-7 BillingAddress business object attributes

4. Save and close the business object.
5. Use the same steps to create the ShippingAddress business object.

Note: Although the BillingAddress and the ShippingAddress both inherit their attributes from the Address business object and do not add new ones, they are different in concept and we may want to add new attributes to one of them later.

Create the Customer business object

The last business object will be used to hold customer registration data.

1. Create a new business object and name it customer.
2. Add the following attributes to the Customer business object:
 - The following String attributes:
 - accountNo
 - firstName
 - lastName
 - companyName
 - email
 - password
 - An attribute called billingAddress with an attribute type of BillingAddress. You can select the BillingAddress business object from the list of drop-downs for type (click **string** to get the list; see Figure 7-8).

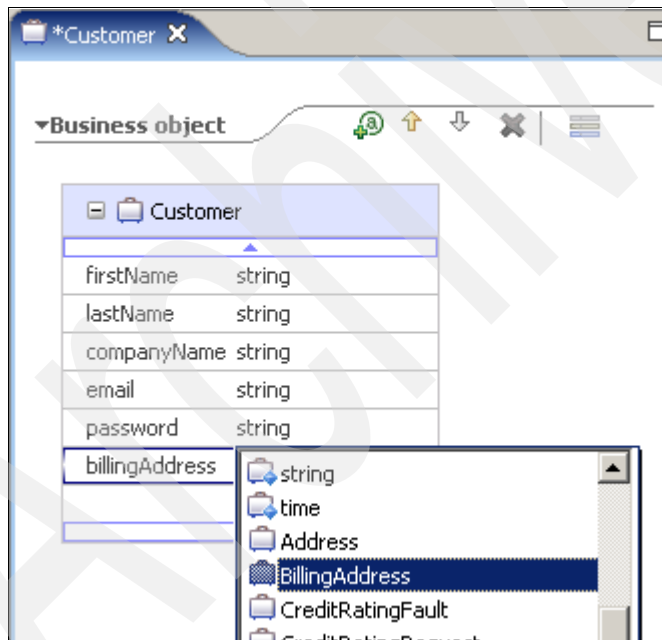


Figure 7-8 Adding attributes to Customer business object

- An attribute called shippingAddress with an attribute type of ShippingAddress[]. The ShippingAddress attribute will represent an array. First, add the shippingAddress attribute and select the ShippingAddress business object as the type. Then, with the attribute selected, check the **Array** setting in the Properties view (Figure 7-9).

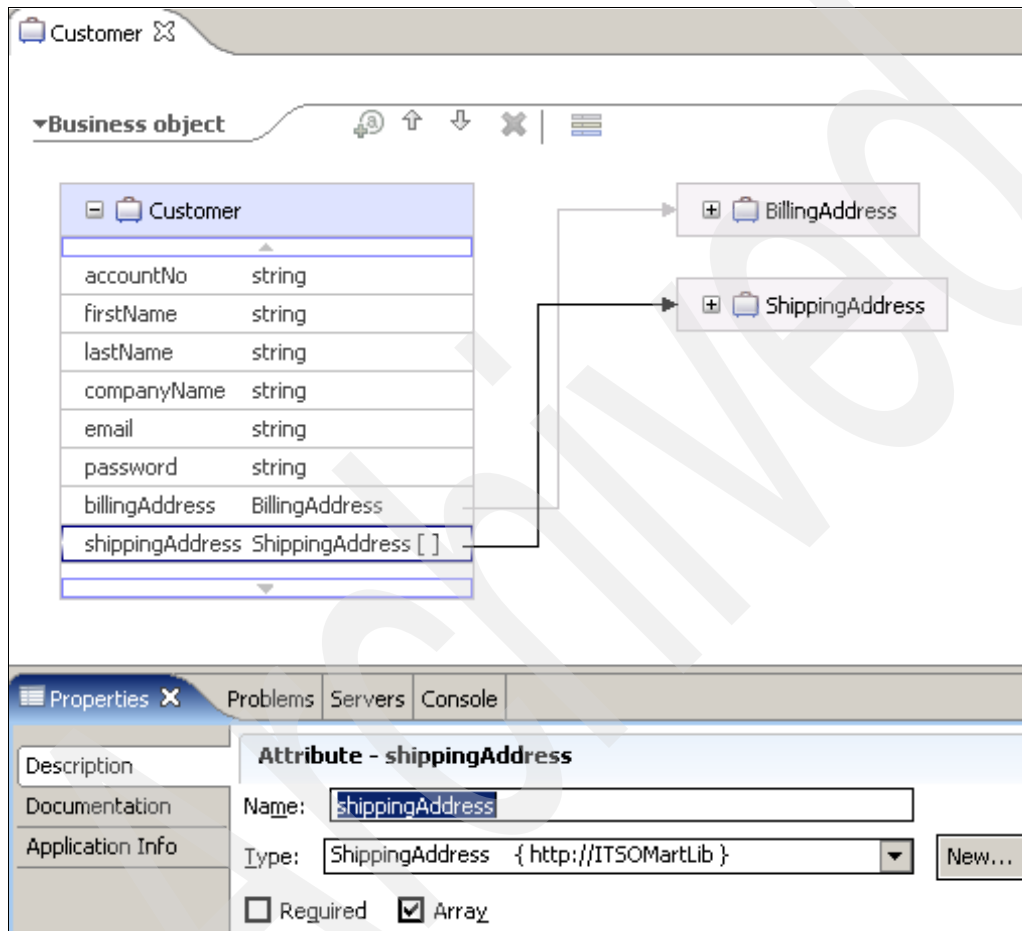


Figure 7-9 Adding the ShippingAddress attribute to the Customer business object

3. Save and close the business object.

7.3 Developing the Credit Rating mediation

In this section we describe how to build the Credit Rating mediation described in 7.1.2, “Get Credit Rating scenario stage 1” on page 265. This mediation contains a simple mediation flow that transforms a message in one XML format to another XML format, a function that is provided by the XSL Transformation primitive. We also use the Message Logger primitive to see how the message changes at every stage. These primitives will be removed before deploying the mediation in a production environment.

Note: The sample zip file contains the ITSOMartUtils Java project containing utility classes used in our mediations.

7.3.1 Mediation development steps

Figure 7-10 shows the mediation module contents we are going to build.

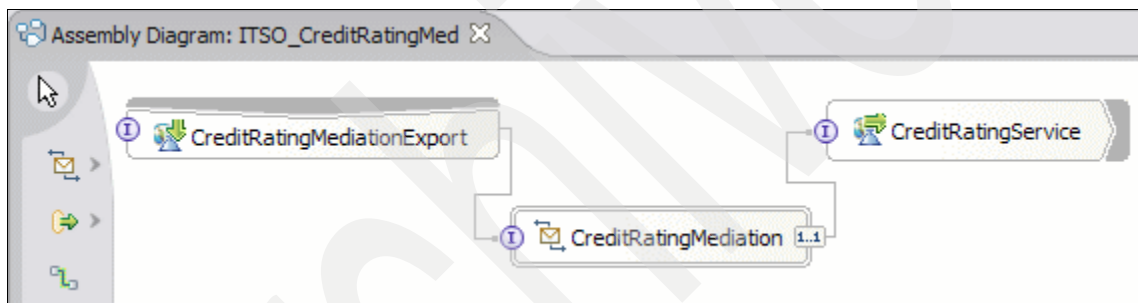


Figure 7-10 Credit Rating mediation assembly diagram

The steps required to build the mediation in WebSphere ESB are the following:

1. Define the interface for the mediation.
2. Define the interface to the Credit Rating Service Web service.
3. Create the mediation module.
4. Add the components to the module assembly.
5. Build the mediation flow.
6. Test the mediation.

Note: Bindings for the import and export will be generated when they are added to the module assembly.

7.3.2 Define the interface for the mediation

An interface to a component contains one or more operations that describe the action implemented by the component. An operation may be a request/response type (which means that a request is sent and a response returned to the interface) or a one-way type (which means that only an input is sent and there is no response needed).

Define the business objects

Each operation in the interface defines the data that can be passed in the form of inputs to and outputs from the component when the operation is invoked. This step will create the business objects that will represent the data as it flows through the mediation.

1. Create a business object called `CreditCheckRequest`. Add one attribute called `customer` with an attribute type of `customer`. The results should look like Figure 7-11.

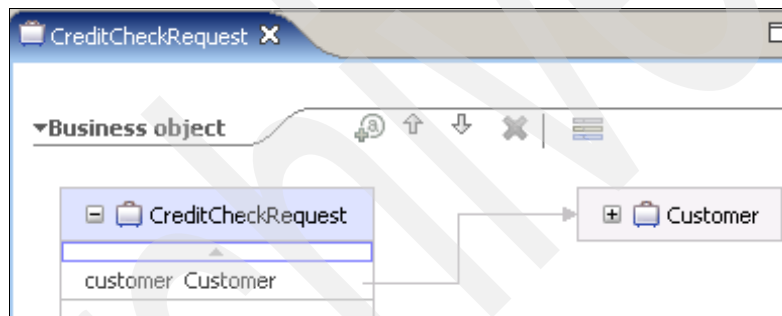


Figure 7-11 Create the `CreditCheckRequest` business object

2. Create a business object called `CreditCheckResponse` with the attribute type of `string`.
3. Create a business object called `CreditCheckFault`. Add the following attributes:
 - `errorMessage` with an attribute type of `string`.
 - `creditCheckRequest` with an attribute type of `CreditCheckRequest`.

The results should look like Figure 7-12.

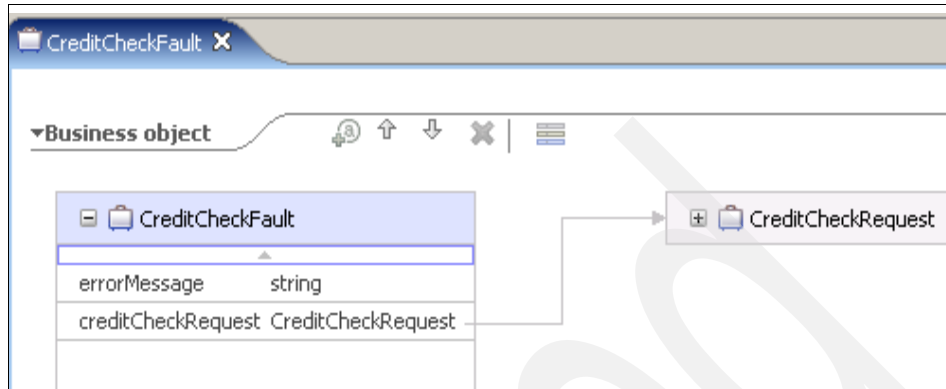


Figure 7-12 Create the CreditCheckFault business object

Build the interface

In this scenario we define an interface that represents the functions provided by the mediation. It includes a request/response operation called `getCreditRating`. The input to the interface will be defined by the `CreditCheckRequest` business object, while the output will be defined by the `CreditCheckResponse` object. The new interface will look like Figure 7-13.

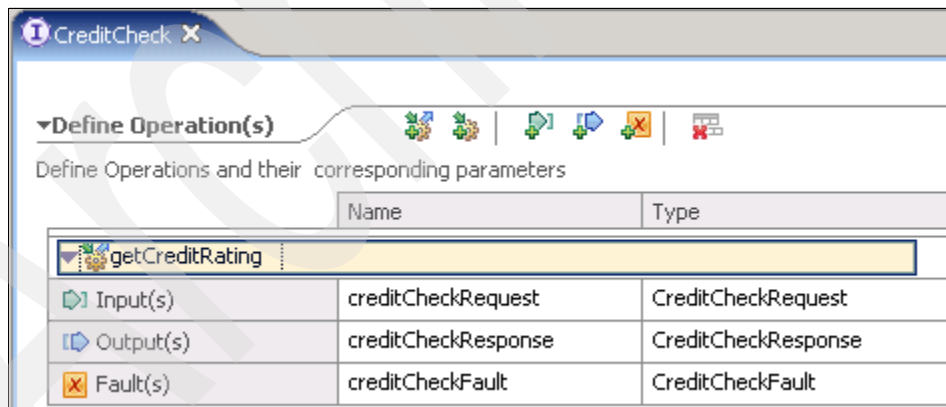





Figure 7-13 Mediation interface

To build the interface:

1. Select the **Interfaces** folder in ITSOMartLib. Right-click and select **New** → **Interface**.
2. Enter CreditCheck in the Name field.

3. Click **Finish**. The new interface will open in the editor area.
4. Since our example is expecting a response to be returned, the operation is a request response operation. In the edit area, click the Add Request Response Operation icon .
5. A new operation will be added to the interface with the default name of Operation1. Change this name to `getCreditRating` by typing over it.
6. Click the `getCreditRating` operation and click the Add Input icon . A new input entry will appear on the canvas.
 - Enter `creditCheckRequest` in the Name field.
 - Click **string** in the Type field and select the **CreditCheckRequest** business object.
7. Click the **`getCreditRating`** operation and click the Add Output icon . A new output entry will appear on the canvas.
 - Enter `creditCheckResponse` in the Name field.
 - Click in the Type field and select the **CreditCheckResponse** business object.
8. Click the **`getCreditRating`** operation and click the Add Fault icon. A new Fault entry will appear on the canvas.
 - Enter `creditCheckFault` in the Name field.
 - Click in the Type field and select the **CreditCheckFault** business object.
9. Save and close the interface.

7.3.3 Define the interface to the Credit Rating Service Web service

The mediation will use an external Web service that provides the credit rating for the customer. We will be using a Web Service provided by the `ITSO_CreditRatingService` project.

Note: We do not need any new business objects. When building the interfaces, we will be using a WSDL file from an existing Web service. When copying the WSDL file, we will see that the data types used by the interface are also copied.

To create an interface to the Web service, we will first need to get a copy of the Web service WSDL file and make it available to our mediation module. In this case, we have the Web service project in our workspace so we can simply copy the WSDL file from one project to another.

1. Right-click in the Business Integration view and select **Show Files** (Figure 7-14).

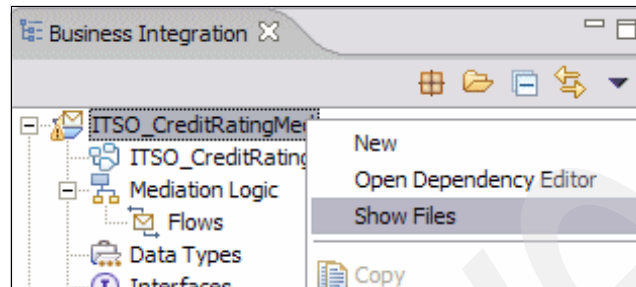


Figure 7-14 Open Physical Resources view

2. This will open a different view where all of the available resources can be seen. Select the **ITSO_CreditRatingService** project and navigate to the **CreditRatingService.wsdl**.
3. Right-click **CreditRatingService.wsdl** and select **Copy** (Figure 7-15).

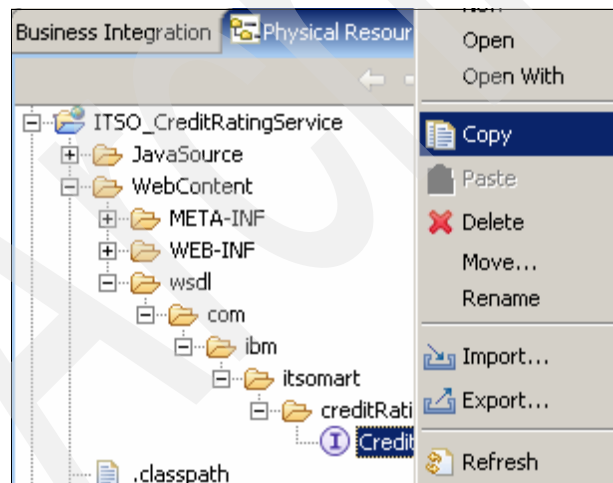


Figure 7-15 Copy CreditRatingService.wsdl

4. Since this Web service will be used by more than one mediation module, we will be copying it to the ITSOMartLib library. This way, all the modules can have access to it.

Select the **ITSOMartLib** library, right-click, and select **Paste**.

This will add the service to ITSOMartLib. If we change to the Business Integration view, we will see that the resources circled in Figure 7-16 were added. In addition to the CreditRatingService interface and Web service port, the CreditRatingFault, CreditRatingRequest, and CreditRatingResponse data types are also copied because they are used by the CreditRatingService interface.

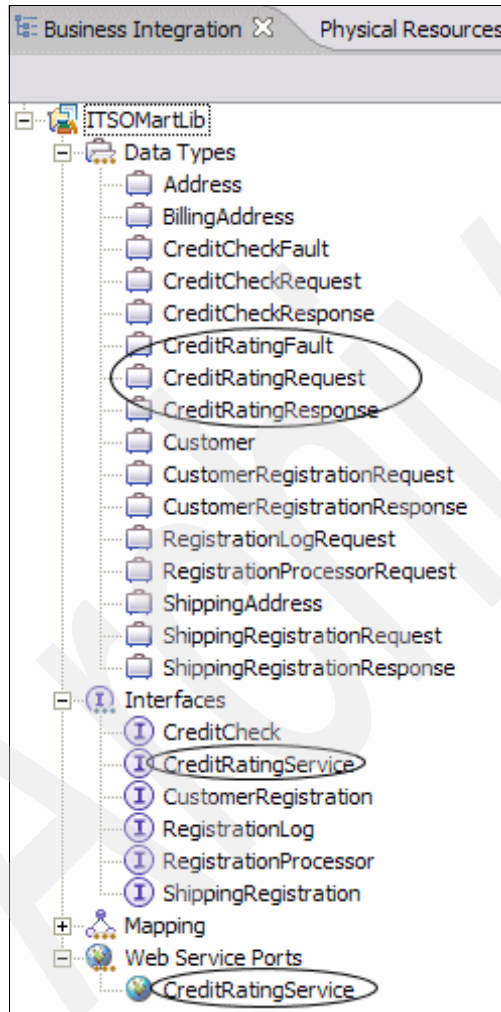


Figure 7-16 CreditRatingService

7.3.4 Create the mediation module

With the business objects and interfaces in place, the next step is to create the mediation module.

1. Create a mediation module by right-clicking in the Business Integration view and selecting **New** → **Mediation module**.
 - Enter `ITSO_CreditRatingMed` for the module name.
 - Select **WebSphere ESB Server v6.0** for the target runtime.
 - Check the **Create mediation flow component box**.
2. Click **Next**. Select **ITSOMartLib** as a dependency.
3. Click **Finish**.

Note: This module is named CRMed in the sample zip file.

7.3.5 Add the components to the module assembly

Next we populate the mediation module. The mediation flow component was added when the mediation module was created. First we add the import that invokes the Credit Rating Service. Then we add the export to be used to invoke the module. Then we wire the components together. The final assembly looks like Figure 7-17.

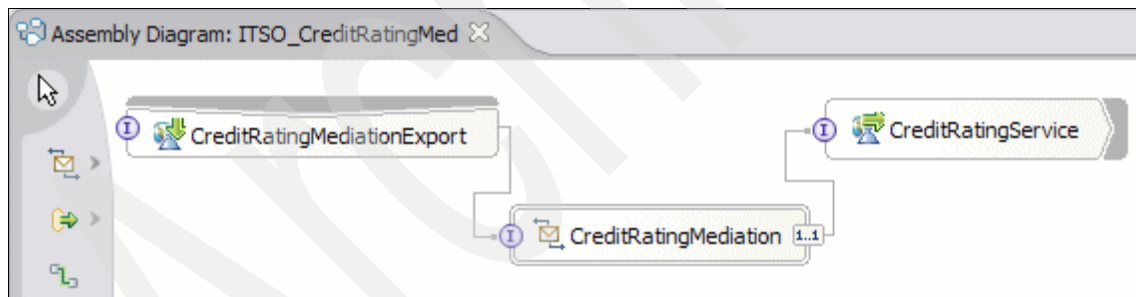


Figure 7-17 *ITSO_CreditRatingMed module assembly*

The steps are:

1. Navigate to the module assembly in the Business Integration view (Figure 7-18). This was created automatically when you created the mediation module. Open the module assembly by double-clicking it.

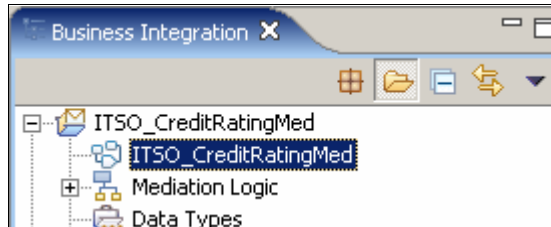


Figure 7-18 Module assembly

The assembly editor will open and you will see that a mediation flow component called Mediation1 has been added for you.

2. Select the mediation flow component. In the Description tab of the Properties view, change the display name from Mediation1 to CreditRatingMediation. Later we will generate an implementation for this component. That implementation will be the mediation flow that will perform the mediation functions.

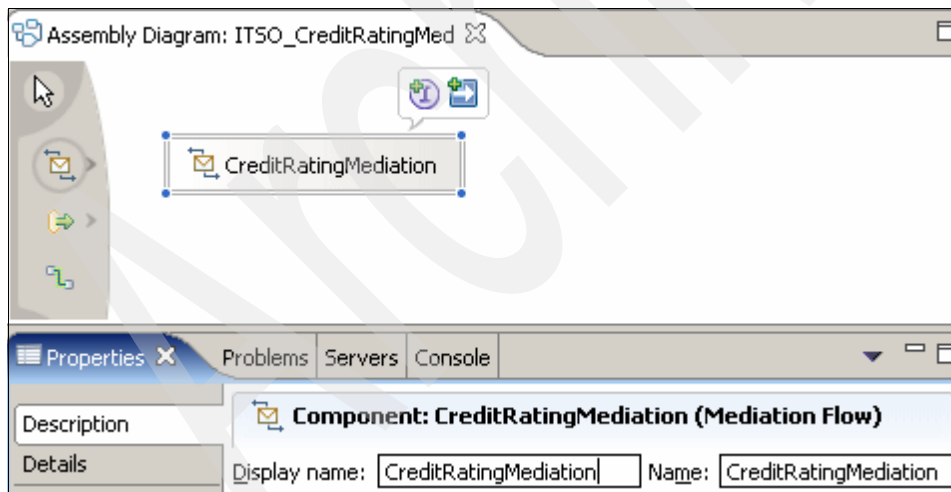


Figure 7-19 Change the name of the mediation flow component

3. Add an import that allows the mediation to invoke the Credit Rating Service:
 - a. Select the **CreditRatingService** Web service port from ITSOMartLib and drag and drop it onto the assembly diagram to the right of the mediation flow component (Figure 7-20).

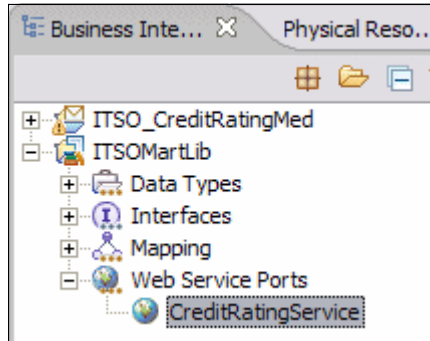


Figure 7-20 Select CreditRatingService service

- b. When you drop the service on the assembly diagram, you will be asked to choose which type of component you want to create. Select **Import with Web Service Binding** and click **OK**.
 - c. An import component will appear on the assembly diagram. Move the new import (Import1) to the right of the mediation flow component. Note that the interface for the import has been automatically defined.

Select the import component and in the Properties view, change the display name to CreditRatingService (Figure 7-21).

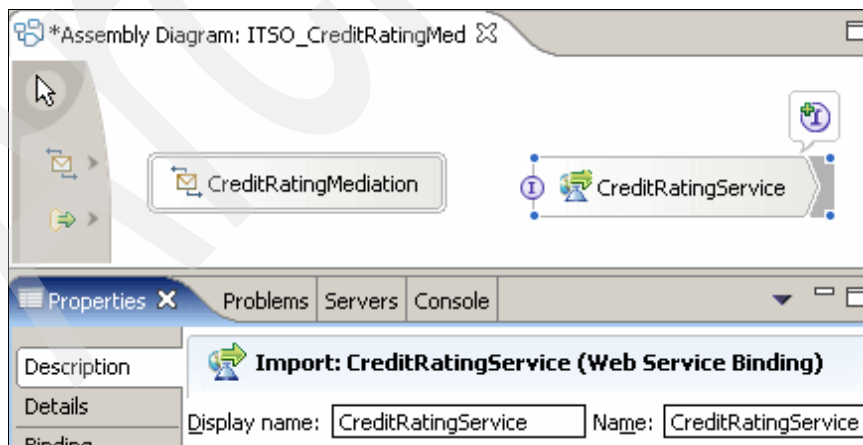


Figure 7-21 Change import component name

4. Right-click the **CreditRatingMediation** component and select **Add** → **Interface**. Select **CreditCheck** and click **OK**.
5. Add an export component that makes the mediation available for use:
 - a. Right-click the **CreditRatingMediation** component and select **Export** → **Web Service Binding**.
 - b. You will be asked whether you want to have a WSDL file with binding/service/port elements defined inside generated for you. Click **Yes**.
 - c. Select **soap/http** in the Select Transport window.
 - d. The export component and its interface will be added for you and will be already wired to the **CreditRatingMediation** component.
6. To complete the assembly diagram, wire the mediation flow component to the import for the **CreditRatingService**.
 - a. Click the Wire icon and then click the **CreditRatingMediation** component.
 - b. Drag the wire onto the interface on **CreditRatingService**.
 - c. You will be prompted and told that by adding the wire a matching reference will be created on the source node. Click **OK**.
 - d. Click the arrow at the top left of the palette to get out of the wiring mode.

This completes the assembly diagram (Figure 7-17 on page 281). Leave it open, and in the next step we use it to generate the implementation (mediation flow) for the mediation flow component, **CreditRatingMediation**.

7.3.6 Build the mediation flow

Next we generate and refine the implementation for the mediation flow component. The implementation is the mediation flow.

1. Right-click **CreditRatingMediation** and select **Generate Implementation** to generate the mediation flow. Click **OK** in the next window. The Mediation Flow editor will open with the new flow (Figure 7-22).

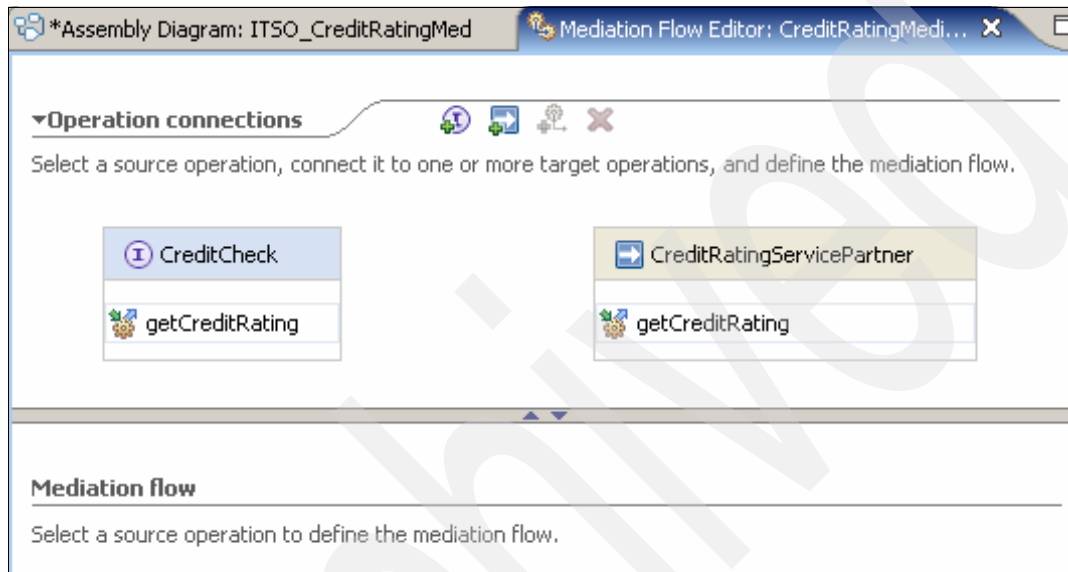


Figure 7-22 Mediation flow editor

2. Click the **getCreditRating** operation under CreditCheck and drag it to the getCreditRating operation under CreditRatingServicePartner.

This will create an operation connection between the two and generate nodes needed to represent the source and target operations. If you do not see these nodes, select the wire between the operations.

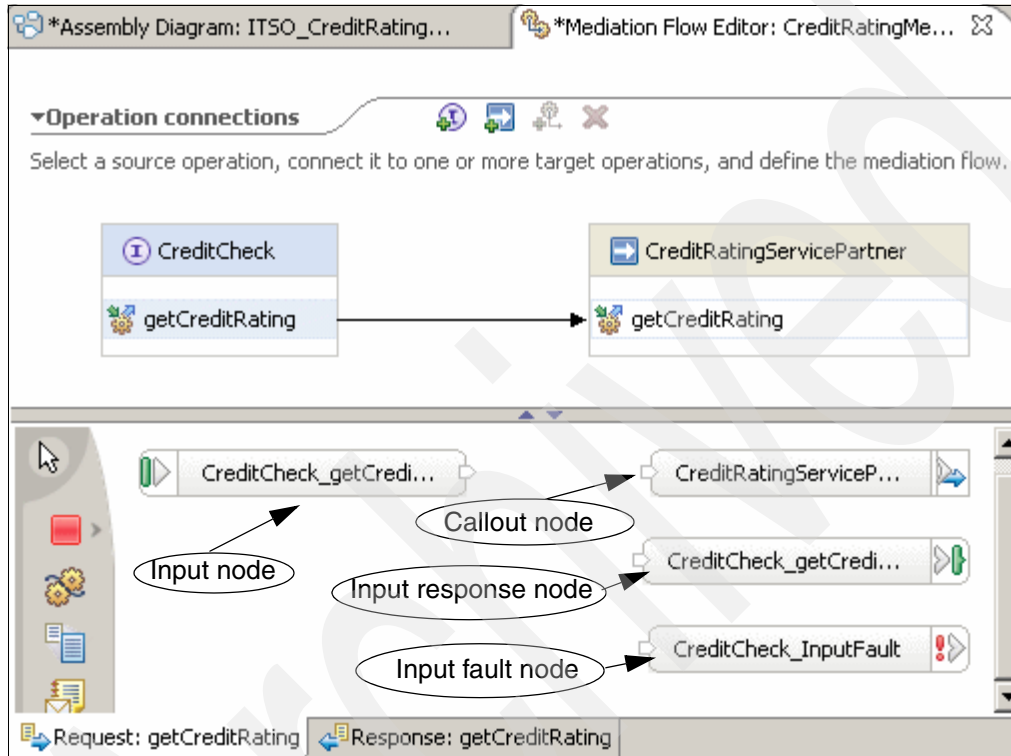


Figure 7-23 Mediation flow editor

Note that there are two flows, a request flow labeled Request: getCreditRating and a response flow labeled Response: getCreditRating. This is because the getCreditRating operation on the CreditCheck interface is a request/response operation. You can switch between the two in the editor by clicking the appropriate tab at the bottom of the screen.

First we build the request flow, and then the response flow.

Request flow

The request flow we will build will look like Figure 7-24.

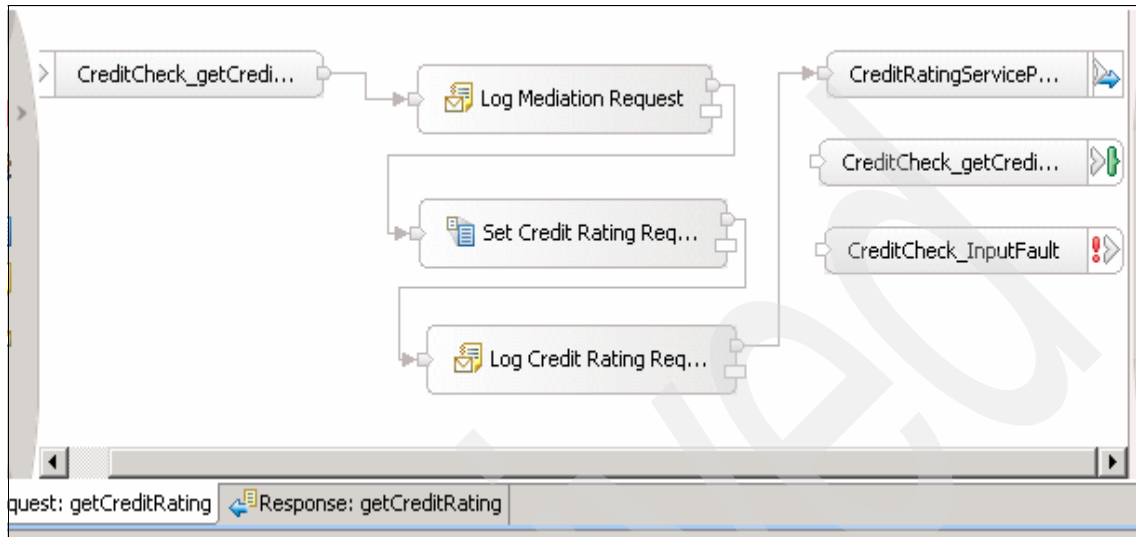


Figure 7-24 *CreditRatingMediation request flow*

In the request flow, you can see that several nodes have been automatically generated:

- ▶ The `CreditCheck_getCreditRating_Input` input node (left) for the source operation. The input node is the starting point for the request flow. It sends the message from the source operation into the request flow.
- ▶ The `CreditRatingServicePartner_getCreditRating_Callout` callout node (top right) for the target service that sends the processed message to the target operation.
- ▶ The `CreditCheck_getCreditRating_InputResponse` input response node (middle right) that returns the processed message as a response to the source operation.
- ▶ The `CreditCheck_InputFault` input fault node (bottom right). It has an input terminal for each fault message type defined in the source operation (`CreditCheck`). Any message propagated to an input fault terminal will result in a WSDL fault of the source operation.

To build the request flow we will add mediation primitives between these nodes and wire the flow together.

1. Click the icon for the Message Logger primitive, and then click in the editor pane to add the primitive between the generated nodes. The new primitive will have a default name of MessageLogger1.

Use the Properties view to change the display name to Log Mediation Request.

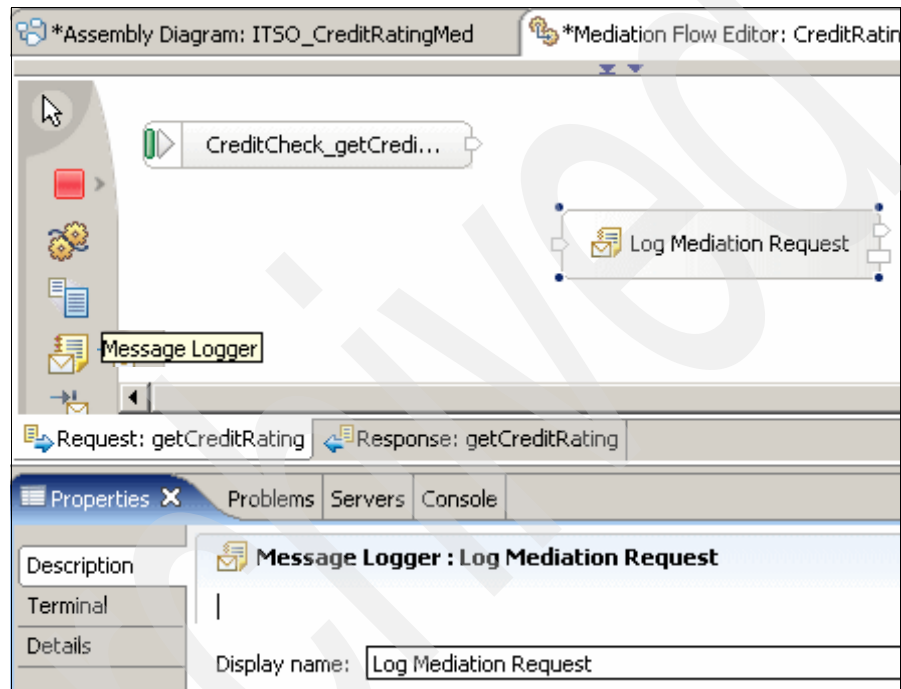


Figure 7-25 Change display name for Message Logger

2. Click the icon for the XSL Transformation primitive, and then click in the editor pane to add the primitive under the Message Logger primitive.
Change the display name to Set Credit Rating Request.

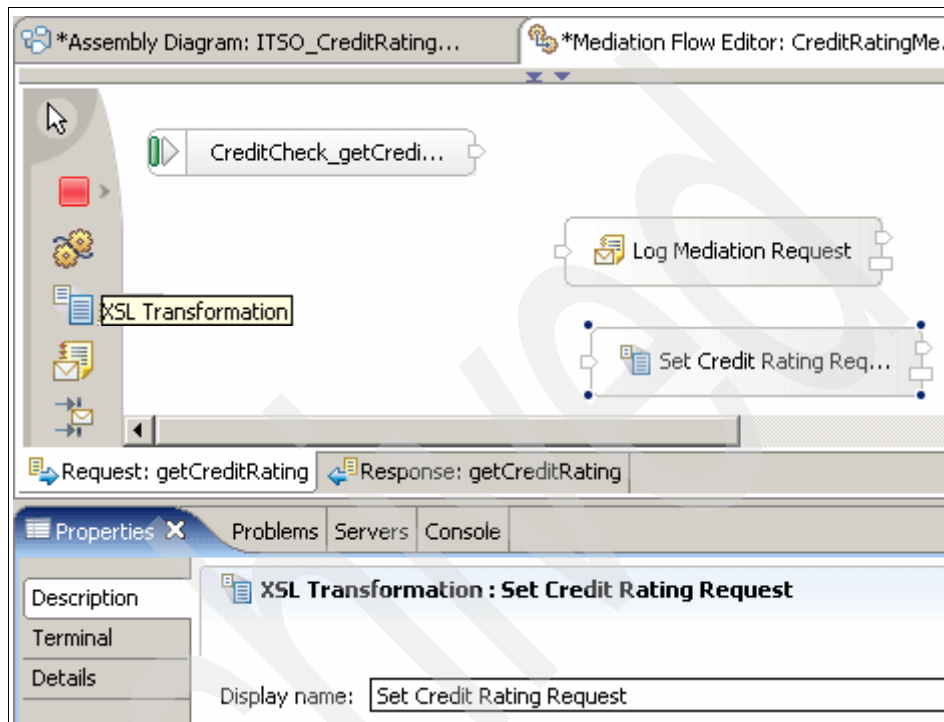


Figure 7-26 Change display name for XSL transformation

3. Add another Message Logger primitive under the XSL Transformation primitive. Change the display name to Log Credit Rating Request.
4. The next step is to wire the nodes in the flow together. Each node has one or more input and output terminals. To wire one node to another, click the out terminal of the first node and drag the wire to the in terminal of the next node. Using this method, wire the following:
 - The out terminal of the CreditCheck getCreditRating Input node to the in terminal of the Log Mediation Request node
 - The out terminal of the Log Mediation Request node to the in terminal of the Set Credit Rating Request node
 - The out terminal of the Set Credit Rating Request node to the in terminal of the Log Credit Rating Request node

- The Out terminal of the Log Credit Rating Request node to the In terminal of the CreditRatingServicePartner getCreditRating Callout node

Note: When you dropped the primitives onto the canvas, the message types of the in and out terminals were null. As you wire the input node to the Message Logger primitive and then the Message Logger to the XSL Transformation primitive, the message type of the input node's out terminal (getCreditRatingRequestMsg) is propagated to the in and out terminals of the Message Logger primitive and to the in terminal of the XSL Transformation primitive. Similarly, when you wire the out terminal of the XSL Transformation primitive to the Message Logger primitive and this one to the callout node, the message type of the callout node's in terminal (getCreditRatingRequest) is propagated to the out terminal of the XSL Transformation primitive and to the in and out terminals of the Message Logger primitive.

This becomes important later when the mapping editor is used to define the mapping for the XSL Transformation primitive.

The results of the wiring are shown in Figure 7-27. The message type has been depicted on the image as well.

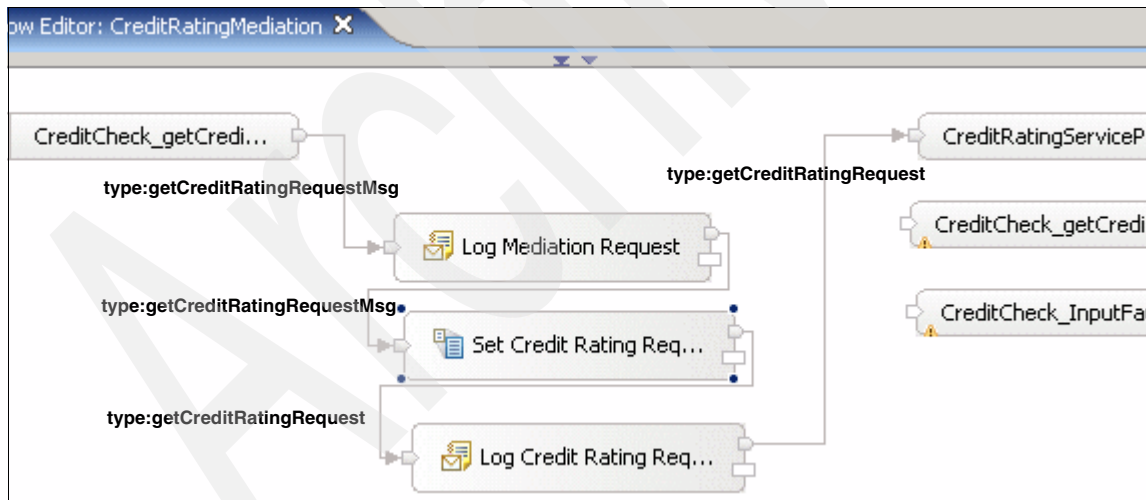


Figure 7-27 Connect the nodes in the flow

5. Now we need to define the mapping to be performed by the XSL Transformation primitive. The mapping needs to take the input message format and map the fields to the output message format. To set the properties of the Set Credit Rating Request node, do the following:
 - a. Select the node.
 - b. In the Properties view click the **Details** tab.

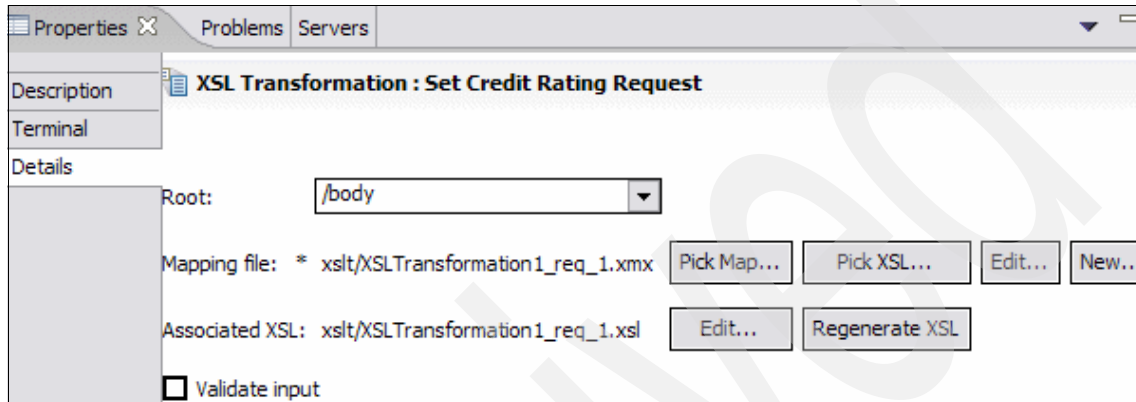


Figure 7-28 XSL transformation properties

The Root field specifies the root of the SMO message to use for the source and target message during transformation. Valid values are:

- / for the complete SMO
- /body for the body section of the SMO
- /headers for the headers of the SMO
- /context for the context of the SMO.

For this example, we want to use the message body, so we select **/body**.

- c. Click **New** to create an XML mapping. (You could choose an existing map by selecting the **Pick Map** button instead.)

- d. Since the XSL Transformation primitive is wired, the wizard knows the input and output message types to be mapped (Figure 7-29). Click **Finish** to create an XML mapping.

New XSLT Mapping


Specify Message Types
Select Input and Output Message Type

Message Root: /body

Input Message Body: getCreditRatingRequestMsg

Output Message Body: getCreditRatingRequest

Defined Contexts

Correlation Context:  No correlation context is set for this flow. Contexts can be set on the input node of the flow.


Transient Context:  No transient context is set for this flow. Contexts can be set on the input node of the flow.

Figure 7-29 New XSLT Mapping

A mapping editor will be opened.

Elements can be mapped by dragging the source element and dropping it on the target element or by selecting a source element and a target element, right-clicking, and selecting **Create Mapping**.

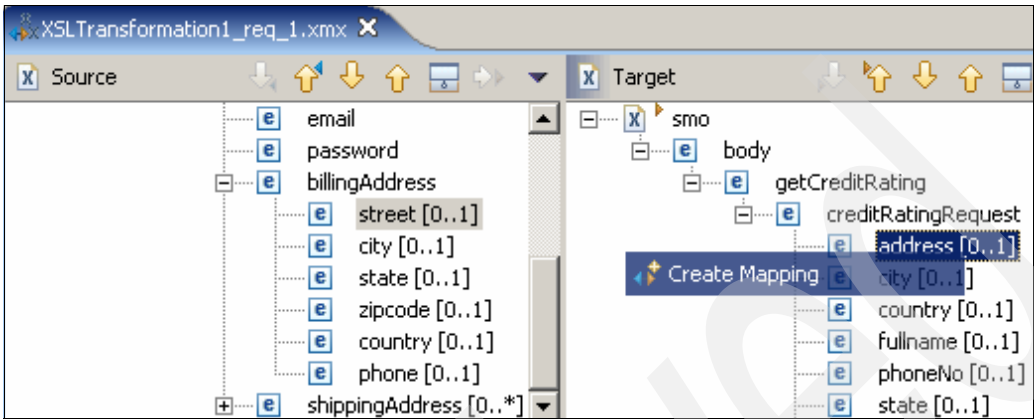


Figure 7-30 Create mapping

This will generate a new entry in the Overview view.

Overview	
Target	Source
smo	smo
body	
getCreditRating	
creditRatingRequest	
address [0..1]	street [0..1]

Figure 7-31 New mapping

e. Map the elements as shown in Table 7-1.

Table 7-1 XML mapping

CreditCheckRequest	CreditRatingRequest
billingAddress/street	address
billingAddress/city	city
billingAddress/country	country

¹ To map firstName and lastName to fullName, click both source fields using the Ctrl key, and then drag and drop them onto fullName.

CreditCheckRequest	CreditRatingRequest
firstName ¹	fullName ¹
lastName ¹	fullName ¹
billingAddress/phone	phoneNo
billingAddress/state	state
billingAddress/zipcode	zip

¹To map firstName and lastName to fullName, click both source fields using the Ctrl key, and then drag and drop them onto fullName.

- f. Add a blank in fullName between firstName and lastName:
 - i. Select **fullName** in the Overview view. Right-click and select **Define XSLT Function** (Figure 7-32).

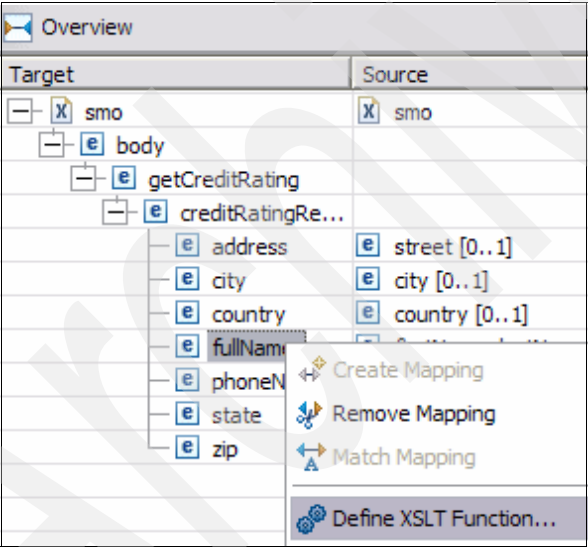


Figure 7-32 Define XSLT Function

- ii. Select **String** and click **Next**.

- iii. Select **concat** as the function (default). Then click **Add**. See Figure 7-33.

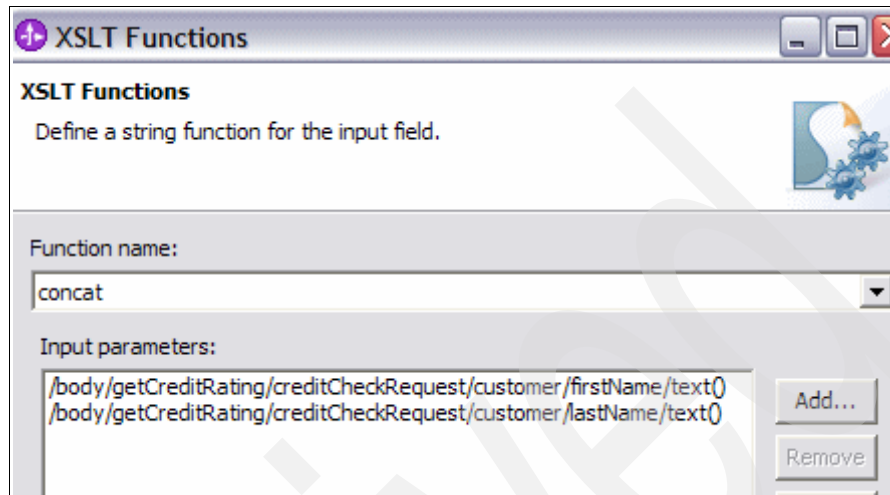


Figure 7-33 Define an XSLT Function

- iv. Enter a blank surrounded by single quotation marks (' ') as the parameter value and click **OK**.

- v. Select the quotes in the Input Parameters window and move them between FirstName and LastName using the up and down buttons, as shown in Figure 7-34.

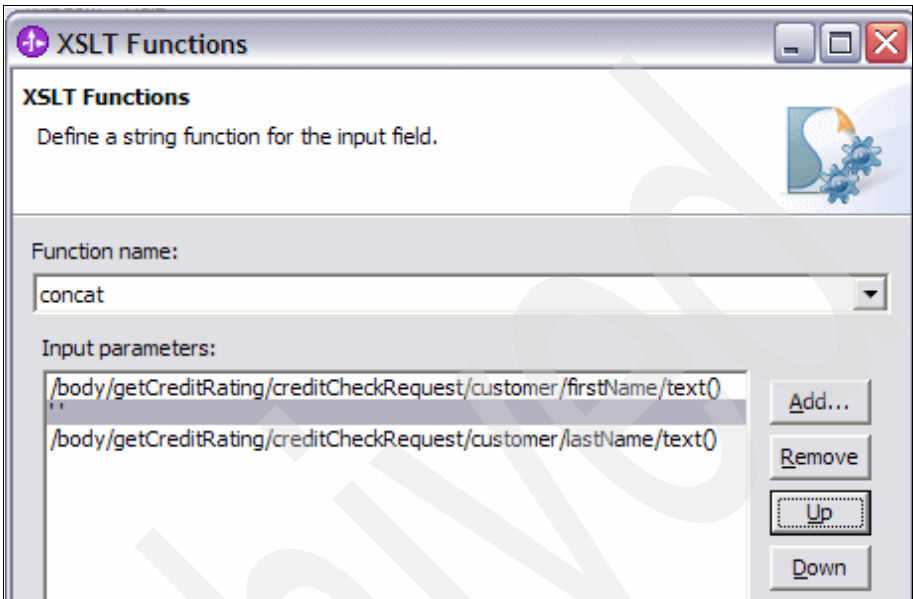


Figure 7-34 Define an XSLT function

- vi. Click **Finish**. If the function is defined correctly, you will see the concat function listed in the Applied Function column (Figure 7-35).

Target	Source	Applied Function/Grouping
smo	smo	
body		
getCreditRating		
creditRatingRequest		
address	street [0..1]	
city	city [0..1]	
country	country [0..1]	
fullName	firstName, lastName	concat
phoneNo	phone [0..1]	
state	state [0..1]	

Figure 7-35 New applied XSLT function

- g. Close and save the map.

6. In the Properties view click the **Regenerate XSL** button to generate an XSL style sheet from the XML map. Click **OK** at the prompt.

Response flow

Next we populate the response flow for both normal and fault responses. The final flow will look like Figure 7-36.

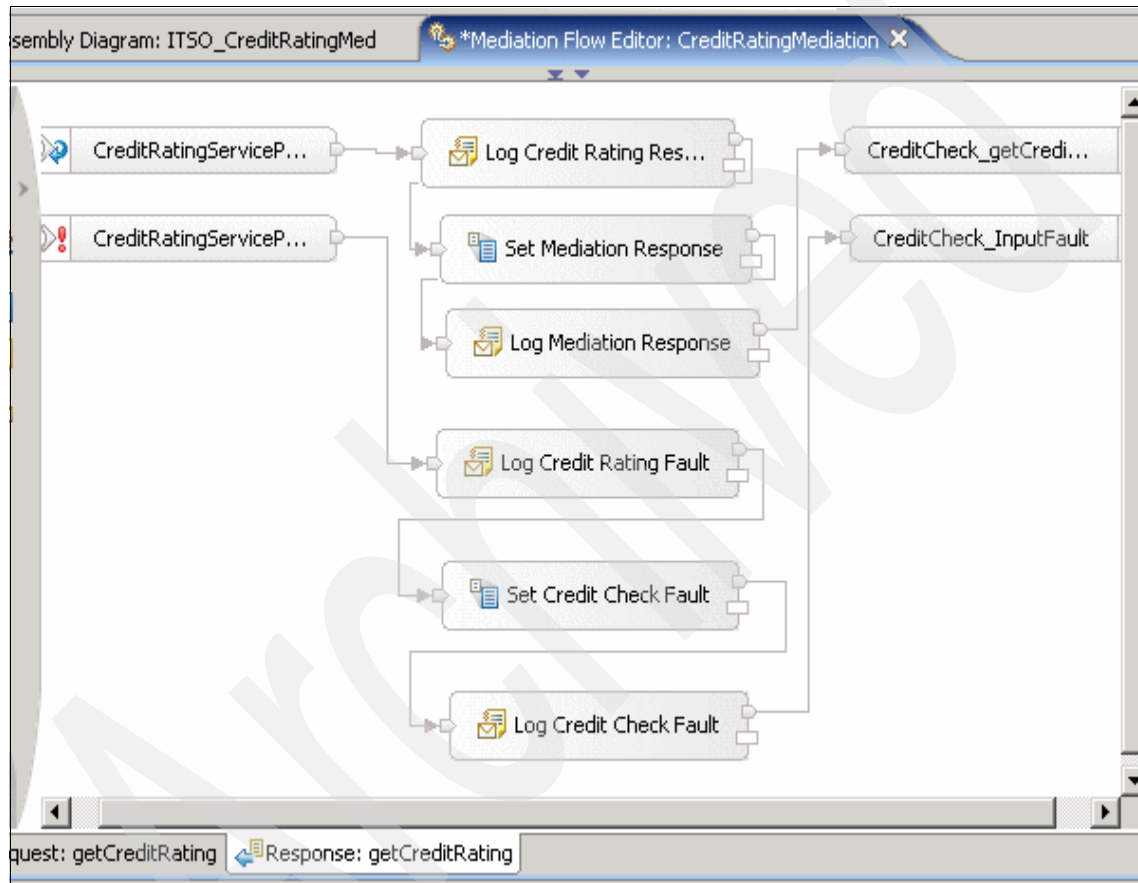


Figure 7-36 CreditRatingMediation response flow

To build the flow we will add mediation primitives between these nodes and wire the flow together.

1. In the mediation flow editor switch to the Response flow by selecting the tab at the bottom of the pane. You will see the nodes that were automatically generated:
 - The CreditRatingServicePartner_getCreditRating_CalloutResponse callout response node that receives the message from the target service.
 - The CreditRatingServicePartner_CalloutFault callout fault node containing an output terminal for each fault message type defined in the target operation. When a WSDL fault occurs, the callout fault node propagates the message to the primitive or node to which it is wired.
 - The CreditCheck_getCreditRating_InputResponse input response node that returns the processed message as a response to the source operation.
 - The CreditCheck_InputFault input fault node containing a terminal for each fault message type defined in the source operation. Any message propagated to an input fault terminal will result in a WSDL fault of the source operation.
2. Add a Message Logger primitive between the generated nodes and change the name to Log Credit Rating Response.
3. Add an XSL Transformation primitive under the Message Logger mediation primitive and change the name to Set Mediation Response.
4. Add another Message Logger primitive under the XSL Transformation primitive and change the name to Log Mediation Response.
5. Wire the nodes as follows:
 - The out terminal of the CreditRatingServicePartner getCreditRating CalloutResponse node to the in terminal of the Log Credit Rating Response node
 - The out terminal of the Log Credit Rating Response node to the in terminal of the Set Mediation Response node
 - The out terminal of the Set Mediation Response node to the in terminal of the Log Mediation Response node
 - The Out terminal of the Log Mediation Response node to the In terminal of the CreditCheck getCreditRating InputResponse node

As you wire the nodes, the message types are propagated to the next node.

The results are shown in Figure 7-37, with the message type overlaid on the image.

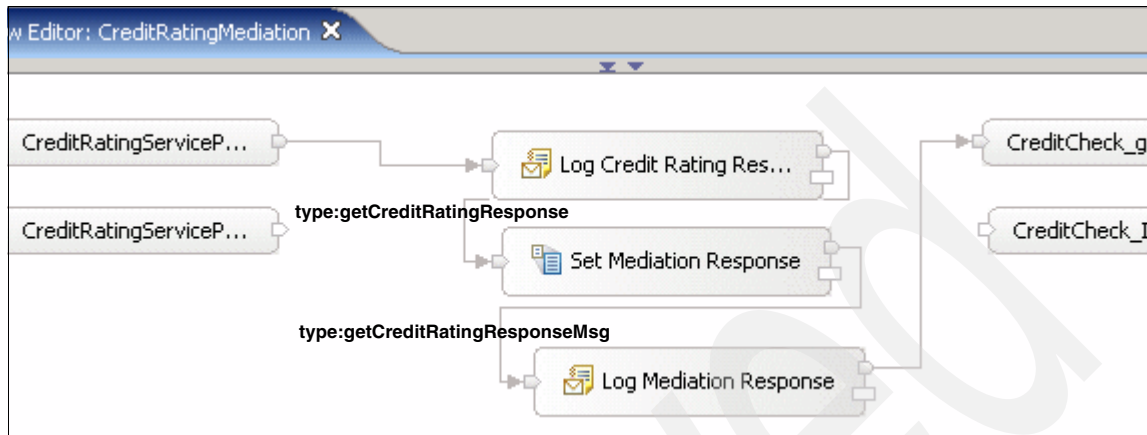


Figure 7-37 Connect the nodes in the flow

6. Create a new XML mapping for the Set Mediation Response XSL Transformation primitive with the mapping shown in Table 7-2. The final mapping should look like Figure 7-38.

Table 7-2 XML mapping

CreditRatingResponse	CreditCheckResponse
rating	creditRating

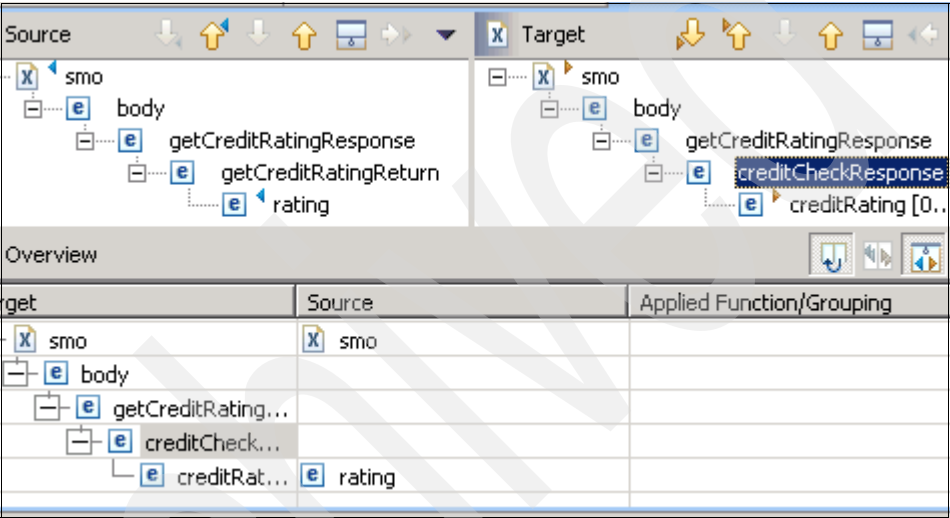


Figure 7-38 XSLT mapping

- h. Close and save the map.
7. In the Properties view click the **Regenerate XSL** button to generate an XSL style sheet from the XML map. Click **OK** at the prompt.

Handling faults

In the Response flow, you will see the starting point and endpoint nodes for the flow. Next we populate the flow with mediation primitives to handle a fault condition:

1. Add a Message Logger primitive between the input and callout node and change the display name to Log Credit Rating Fault.
2. Add an XSL Transformation primitive under the Message Logger primitive and change the display name to Set Credit Check Fault.
3. Add another Message Logger primitive under the XSL Transformation primitive and change the display name to Log Credit Check Fault.

4. Wire the following nodes:

- The out terminal of the CreditRatingServicePartner CalloutFault node to the in terminal of the Log Credit Rating Fault node
- The out terminal of the Log Credit Rating Fault node to the in terminal of the Set Credit Check Fault node
- The out terminal of the Set Credit Check Fault node to the in terminal of the Log Credit Check Fault node
- The Out terminal of the Log Credit Check Fault node to the In terminal of the CreditCheck InputFault node

Wiring the nodes propagates the message type to each terminal. The results are shown in Figure 7-39.

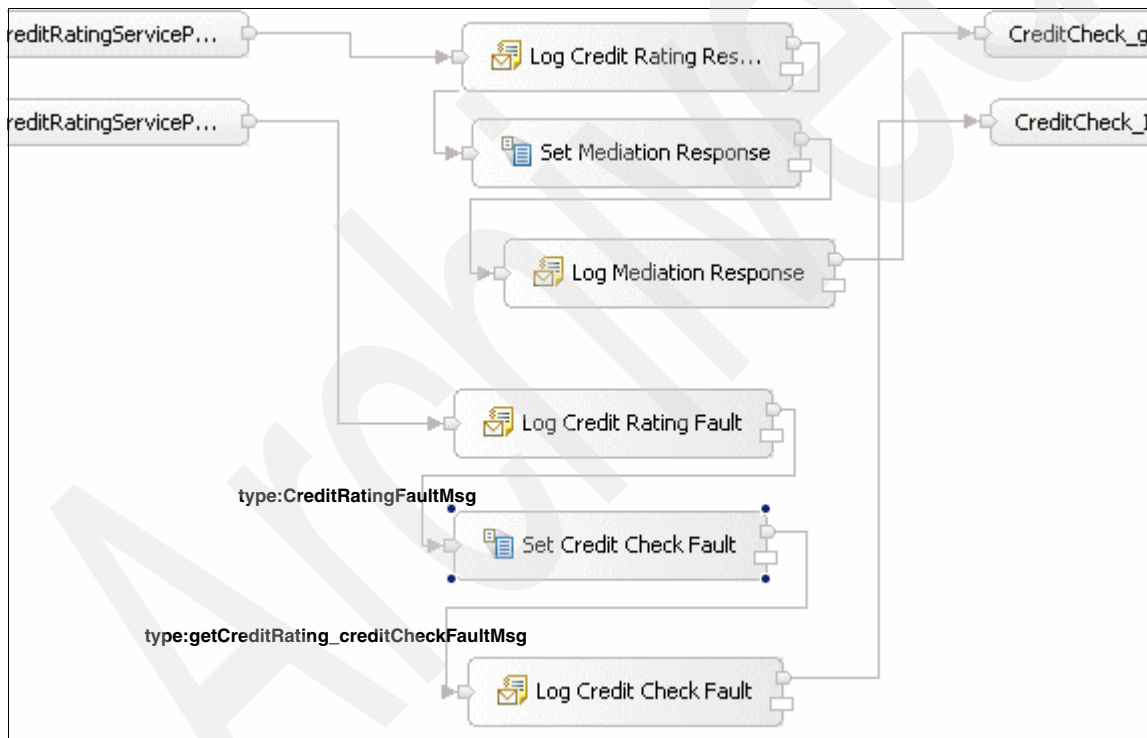


Figure 7-39 Connect the nodes in the flow

5. Create a new mapping for the Set Credit Check Fault node. Map the elements as shown in Table 7-3.

Table 7-3 XML mapping

CreditRatingFault	CreditCheckFault
errorMessage	errorMessage
fullName	billingAddress/name

The final mapping should look like Figure 7-40.

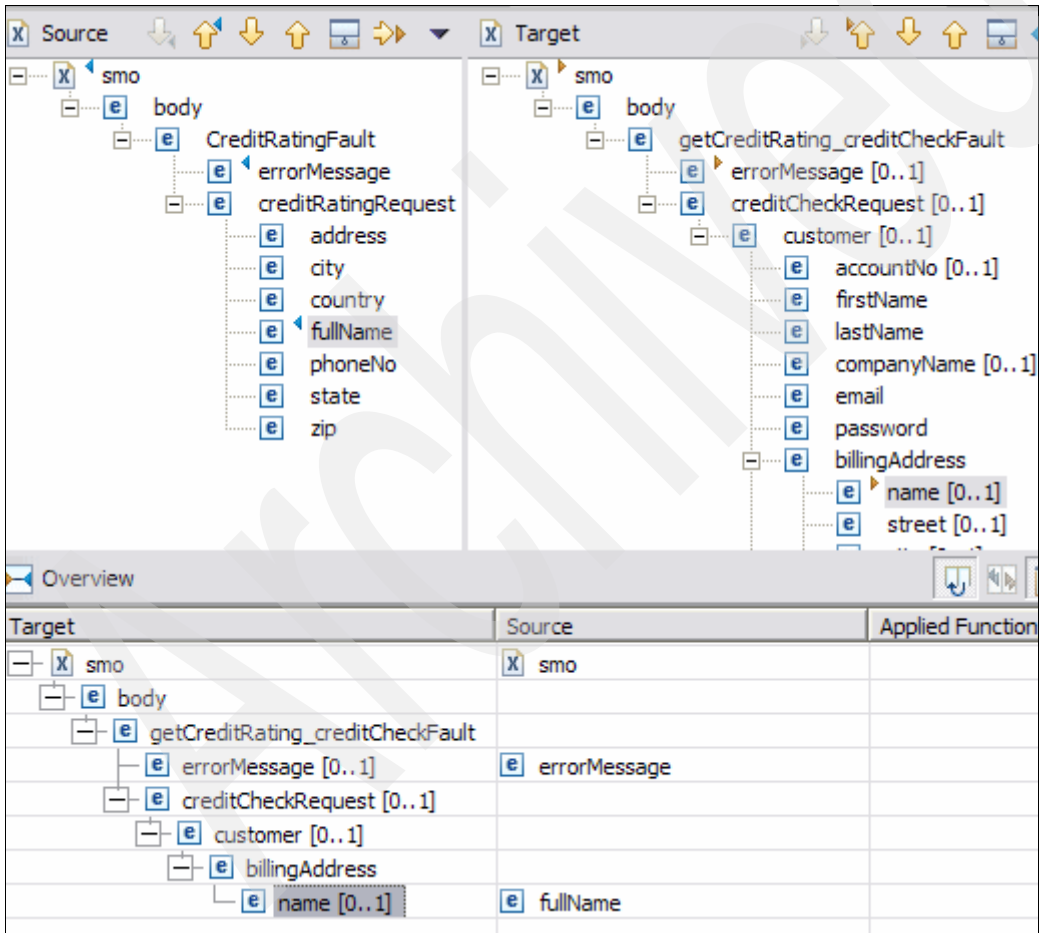


Figure 7-40 XSLT mapping

- i. Close and save the map.

6. In the Properties view click the **Regenerate XSL** button to generate an XSL style sheet from the XML map. Click **OK** at the prompt.
7. Close and save the mediation flow.

7.3.7 Test the mediation

To test the mediation, WebSphere Integration Developer provides the integration test client. This is a useful tool for testing and debugging integration modules. For information about using it, refer to the WebSphere Integration Developer help.

Runtime requirements

To test the mediation you will need to add the following applications to the server and start them:

- ▶ ITSO_CreditRatingService: This is the Credit Rating Service called by the import.
- ▶ (Optional) MessageLogApp.

To add an application to the server, do the following:

1. Start the WebSphere ESB Server test environment.
2. In the Servers view, select **WebSphere ESB Server v6.0**, right-click the server, and select **Add and remove projects**.
3. Select each application that you want to add to the server in the Available projects and click **Add**.
4. Click **Finish**.

Test the flow

To test our mediation flow:

1. Rebuild all projects (select **Project** → **Build All**).
2. Add the mediation application, ITSO_CreditRatingMedApp, to the server.
3. Launch the test client for the ITSO_CreditRatingMed module.

You can test the module by doing one of the following:

- Select the module in the Business Integration view, right-click, and select **Test** → **Test Module**.
- Open the module assembly. Right-click in the open space of the module assembly diagram and select **Test Module**.
- Open the module assembly. Select the export component, right-click, and choose **Test Component**.

4. In the Component field, select **CreditRatingMediationExport**.
5. Set the values in the test client. Fill in some test data for the request. See Figure 7-41.
 - To see a bronze rating, enter a last name less than five characters.
 - To see a silver rating, enter a last name of five or six characters.
 - To see a gold rating, enter a last name of seven or eight characters.
 - Any last name nine characters or greater in length will throw a fault.

Detailed Properties

Configuration: Default Module Test

Module: ITSO_CreditRatingMed

Component: CreditRatingMediationExport

Interface: CreditCheck

Operation: getCreditRating

Initial request parameters

Name	Type	Value
[-] creditCheckR...	CreditCheckRe...	
[-] customer	Customer	
accountNo	string	12345
firstName	string	Janet
lastName	string	Smythe
compan...	string	myco
email	string	Janet@mymail...
password	string	password
[-] billingAd...	BillingAddress	
name	string	Janet Smythe
street	string	123 Main

Data Pool Continue

Figure 7-41 Test client values for ITSO_CreditRatingMed

- i. Check the test client Events pane to verify that everything ran successfully (Figure 7-42).

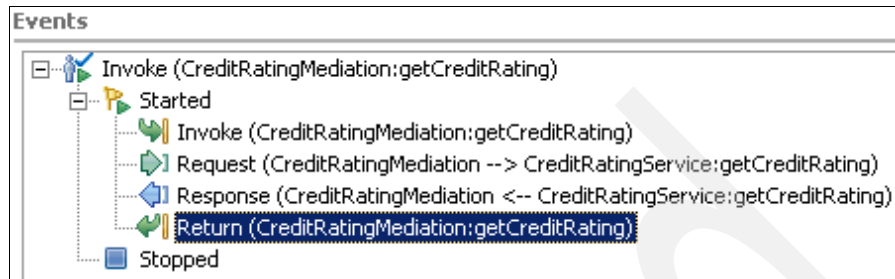


Figure 7-42 Test client events generated during the test

- ii. Look at the messages that were logged inside the mediation using the Message Logger primitives.

Open the sample MessageLogApp application using the following URL:

<http://localhost:9080/MessageLogWeb/faces/MessageLogView.jsp>

You can see the log messages in descending order of when they were inserted into the database.

In Figure 7-43 you can see the messages logged by the Login Mediation Request, Log Credit Rating Request, Log Credit Rating Response, and the Log Mediation Response Message Logger primitives in the flow.

ESB Mediation Message Log				
Refresh		Clear Log		
	TIMESTAMP	MESSAGE ID	MEDIATION	MODULE
	Mar 24, 2006 12:56:17 AM MST	24f4402b-0a01-0000-0080-c845f688df82	Log Mediation Response	ITSO_CreditRatingMed
	Mar 24, 2006 12:56:17 AM MST	24f4402b-0a01-0000-0080-c845f688df82	Log Credit Rating Response	ITSO_CreditRatingMed
	Mar 24, 2006 12:56:17 AM MST	ecf2402b-0a01-0000-0080-c845f688df82	Log Credit Rating Request	ITSO_CreditRatingMed
	Mar 24, 2006 12:56:17 AM MST	ecf2402b-0a01-0000-0080-c845f688df82	Log Mediation Request	ITSO_CreditRatingMed
< < Page 1 of 1 > > 1				

Figure 7-43 Message Logger messages

- iii. Click anywhere in one of the rows of the table displayed to see the contents of the message that was logged. For example, the message for the Log Mediation Response entry shows the result for the credit check returning the user's corresponding rate.

```
<?xml version="1.0" encoding="UTF-8"?>
<smo:smo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:check="http://ITSOMartLib/CreditCheck"
xmlns:smo="http://www.ibm.com/websphere/sibx/smo/v6.0.1">
  <context/>
  <headers>
    <SMOHeader>
      <MessageUUID>24f4402b-0a01-0000-0080-c845f688df82</M
      <Version>
        <Version>6</Version>
        <Release>0</Release>
        <Modification>1</Modification>
      </Version>
      <MessageType>Reply</MessageType>
    </SMOHeader>
  </headers>
  <body xsi:type="check:getCreditRatingResponseMsg">
    <getCreditRatingResponse>
      <creditCheckResponse>
        <creditRating>GOLD</creditRating>
      </creditCheckResponse>
    </getCreditRatingResponse>
  </body>
</smo:smo>
```

Figure 7-44 Log Mediation Response log message

- j. To test the flow for a fail condition, enter a surname that has nine or more characters.
- i. Check the test client Events pane to verify that it failed (Figure 7-45).

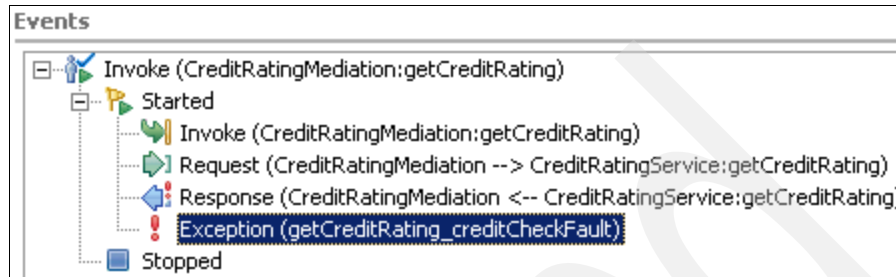


Figure 7-45 Test Client - events - failure

- ii. Look at the messages logged by the Message Logger primitives.

In Figure 7-46 you can see the messages logged by the Login Mediation Request, Log Credit Rating Request, Log Credit Rating Fault, and the Log Credit Check Fault Message Logger primitives in the flow.

TIMESTAMP	MESSAGE ID	MEDIATION	MODULE
Mar 24, 2006 1:25:46 AM MST	00ef5b2b-0a01-0000-0080-c845f688df82	Log Credit Check Fault	ITSO_CreditRatingMed
Mar 24, 2006 1:25:46 AM MST	00ef5b2b-0a01-0000-0080-c845f688df82	Log Credit Rating Fault	ITSO_CreditRatingMed
Mar 24, 2006 1:25:45 AM MST	f7ed5b2b-0a01-0000-0080-c845f688df82	Log Credit Rating Request	ITSO_CreditRatingMed
Mar 24, 2006 1:25:45 AM MST	f7ed5b2b-0a01-0000-0080-c845f688df82	Log Mediation Request	ITSO_CreditRatingMed

Figure 7-46 Message Logger messages

- iii. Click anywhere in one of the rows of the table displayed to see the contents of the message that was logged. For example, the message for the **Log Credit Check Fault** entry shows the result for the credit check returning the user's corresponding rate.

```
<?xml version="1.0" encoding="UTF-8"?>
<body xsi:type="check:getCreditRating_creditCheckFaultMsg"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:check="http
<getCreditRating_creditCheckFault>
  <errorMessage>No credit rating information available for this perso
  <creditCheckRequest>
    <customer>
      <firstName></firstName>
      <lastName></lastName>
      <email></email>
      <password></password>
      <billingAddress>
        <name>Luciana CalcagnoC</name>
      </billingAddress>
      <shippingAddress>
        <name>Luciana CalcagnoC</name>
      </shippingAddress>
    </customer>
  </creditCheckRequest>
</getCreditRating_creditCheckFault>
</body>
```

Figure 7-47 Log Credit Check Fault log message

7.4 Developing the Credit Score mediation

This section discusses how to build the Credit Score mediation described in 7.1.3, “Get Credit Rating scenario stage 2” on page 266.

The credit check service we used in the Credit Rating mediation returned a value of gold, silver, or bronze. This value is obtained by invoking the Credit Rating Service. This service will no longer be available, and in its place another service, the Credit Score Service, will return a score that corresponds with the rates we used before. The relation between the new numerical values and the old text must be made within a mediation. To do this, we build a new mediation called the Credit Score mediation and alter the Credit Rating mediation to call the new mediation instead of the Credit Rating Service. The new mediation will call the

new Credit Score Service and translate the numerical values to the proper text values expected by the rest of the solution.

The new mediation will use a Database Lookup primitive to access a table that correlates the new scores to the ratings.

7.4.1 Mediation development steps

Figure 7-48 shows the mediation module contents we are going to build.

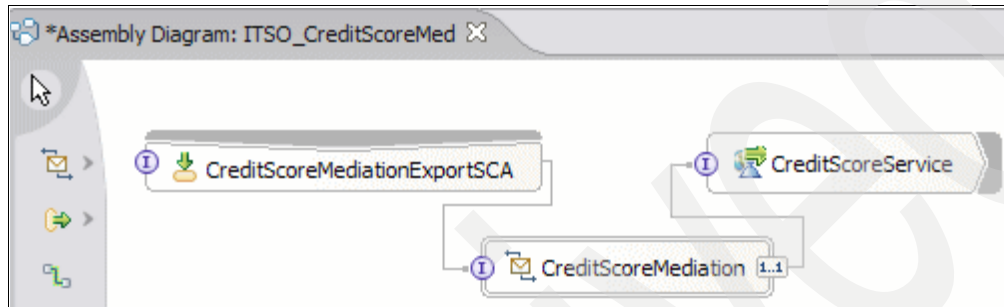


Figure 7-48 ITSO_CreditScoreMediation

The steps required to build the mediation in WebSphere ESB are the following:

1. Create the mediation module.
2. Define the business objects.
3. Define the interface to the Credit Score Service Web service.
4. Add the components to the module assembly.
5. Build the mediation flow.
6. Test the mediation.

7.4.2 Create the mediation module

The first step is to create a mediation module. As we did with the Credit Rating mediation, we will be using some of the business objects and interfaces that are in the ITSOMartLib library we created earlier. We will be adding it as a dependency while creating the mediation module.

1. Create a mediation module by right-clicking in the Business Integration view and selecting **New** → **Mediation module**.
 - a. Enter `ITSO_CreditScoreMed` for the name of the mediation module.
 - b. Select **WebSphere ESB Server v6.0** for the target runtime.
 - c. Check the **Create mediation flow component box** and click **Next**.
 - d. Add ITSOMartLib as a dependency.
 - e. Click **Finish**.

Note: This module is named CSMed in the sample zip file.

Next we add the ITSOMartUtils Java project to the dependent Java projects. We need to do this to have the project deployed as a utility JAR file at runtime. We will be using this Java project in one of the following sections:

1. Open the ITSO_CreditScoreMed Dependencies by double-clicking the **ITSO_CreditScoreMed** mediation module.
2. Expand the Java section to see the Java dependencies.



Figure 7-49 Java dependencies

3. Click **Add** on the Java dependencies and select **ITSOMartUtils** in the next window, as shown in Figure 7-50.

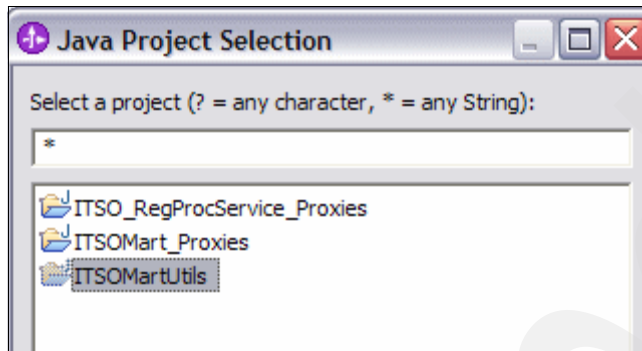


Figure 7-50 Java Project Selection

4. Click **OK**. You will see the ITSOMartUtils project added in the Java dependencies.

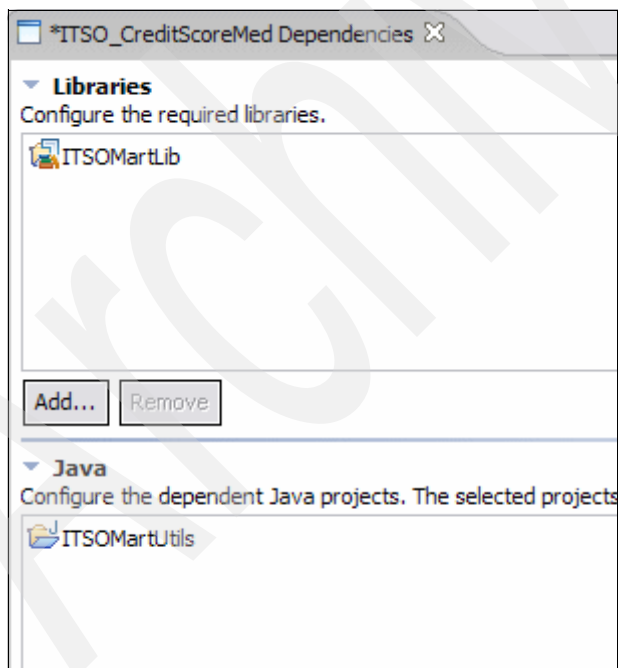


Figure 7-51 Added Java dependency

5. Save the changes and close the ITSOMartUtils Dependencies.

7.4.3 Define the business objects

Since the data carried in the SMO body is the operation defined by the interface specification and the inputs/outputs/faults specified in the message parts set in the business object definition, we need to be able to save the data returned by the Web service in order to access it from the Database Lookup primitive. The transient context of the SMO maintains application data across mediation primitives in one direction. In order to do so, we need to specify a business object for the Transient context. The object we use is the CreditScoreMediationContext.

1. Create the CreditScoreMediationContext business object in ITSOMartLib. Add one string attribute called creditRating. The results should look like Figure 7-52.

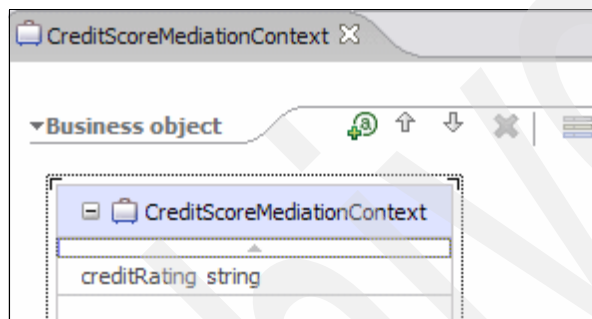


Figure 7-52 Create the CreditScoreMediationContext business object

2. Save and close the business object.

7.4.4 Define the interface to the Credit Score Service Web service

The mediation will use the ITSO_CreditScoreService Web service. The service will provide a numerical credit score for the customer. We will copy the WSDL file from this existing Web service to ITSOMartLib, where it will be available to the mediation module. When we do this, WebSphere Integration Developer will automatically build the interface and the data types required.

To copy the WSDL file and create the interface:

1. Right-click in the Business Integration view and select **Show Files**.

2. This will open a different view, where all the available resources can be seen. Select the **ITSO_CreditScoreService** project and navigate to the **CreditScoreService.wsdl**.

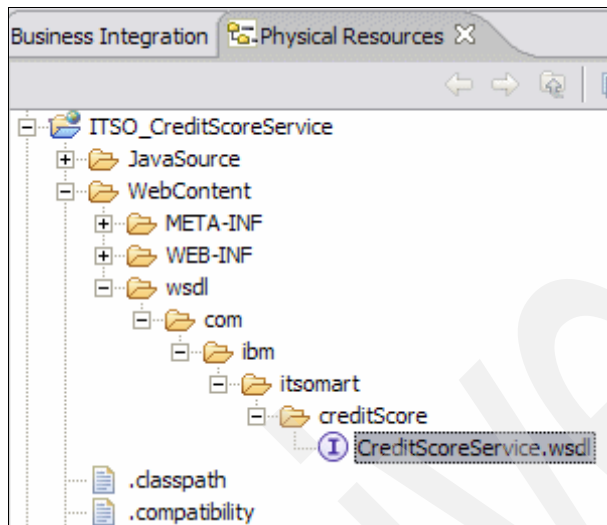


Figure 7-53 Navigate to CreditScoreService

3. Right-click **CreditScoreService.wsdl** and select **Copy**.
4. Select the **ITSOMartLib**, right-click, and select **Paste**.

This will make the service available to the ITSO_CreditScoreMed mediation module. If we change to the Business Integration view, we will see that the CreditScoreService interface and Web service port have been added to ITSOMartLib.

7.4.5 Add the components to the module assembly

Next we populate the mediation module with the necessary export and import (the mediation flow component was added while creating the mediation module). The final module assembly will look like Figure 7-54.



Figure 7-54 ITSO_CreditScoreMed

First we add the import that invokes the Credit Score Service. Then we add the export to be used to invoke the module. Then we wire the components together.

1. Open the module assembly for the mediation.
2. Select the mediation flow component called **Mediation1** and use the Properties view to change the name to **CreditScoreMediation**.
3. Select the **CreditScoreService** Web service port from ITSOMartLib and drag and drop it onto the assembly diagram to the right of the mediation flow component.
 - When you drop the service onto the assembly diagram, you will be asked to choose which type of component you want to create. Select **Import with Web Service Binding**.
 - Select the new Import component and in the Properties view and change the display name to **CreditScoreService**.
4. Right-click the **CreditScoreMediation** component and select **Add → Interface**. Select **CreditRatingService**.
5. Right-click the **CreditScoreMediation** component and select **Export → SCA Binding**. This export type will allow you to call the ITSO_CreditScoreMed module from another mediation module. The export component will be added for you and will already be wired to the CreditScoreMediation component.
6. Click the Wire icon and then click the **CreditScoreMediation** component. Drag the wire to the interface on CreditScoreService. You will be prompted and told that by adding the wire a matching reference will be created on the source node. Click **OK**.

7. Click the arrow at the top left of the palette to get out of the wiring mode.

Leave the assembly diagram open. We will use it in the next step to generate the implementation for the mediation flow component, `CreditScoreMediation`.

7.4.6 Build the mediation flow

Next we generate and refine the implementation for the mediation flow component. The implementation is the mediation flow.

1. Right-click **CreditScoreMediation** and select **Generate Implementation** to generate the mediation flow.

Click **OK** in the next window. The Mediation Flow editor will open with the new flow.

2. Click the **getCreditRating** operation under CreditRatingService (you will see an input node in the Mediation flow pane) and drag it to the getCreditScore operation under CreditScoreServicePartner.

This will create an operation connection between the two and generate the starting point and endpoint nodes in the flow. See Figure 7-55.

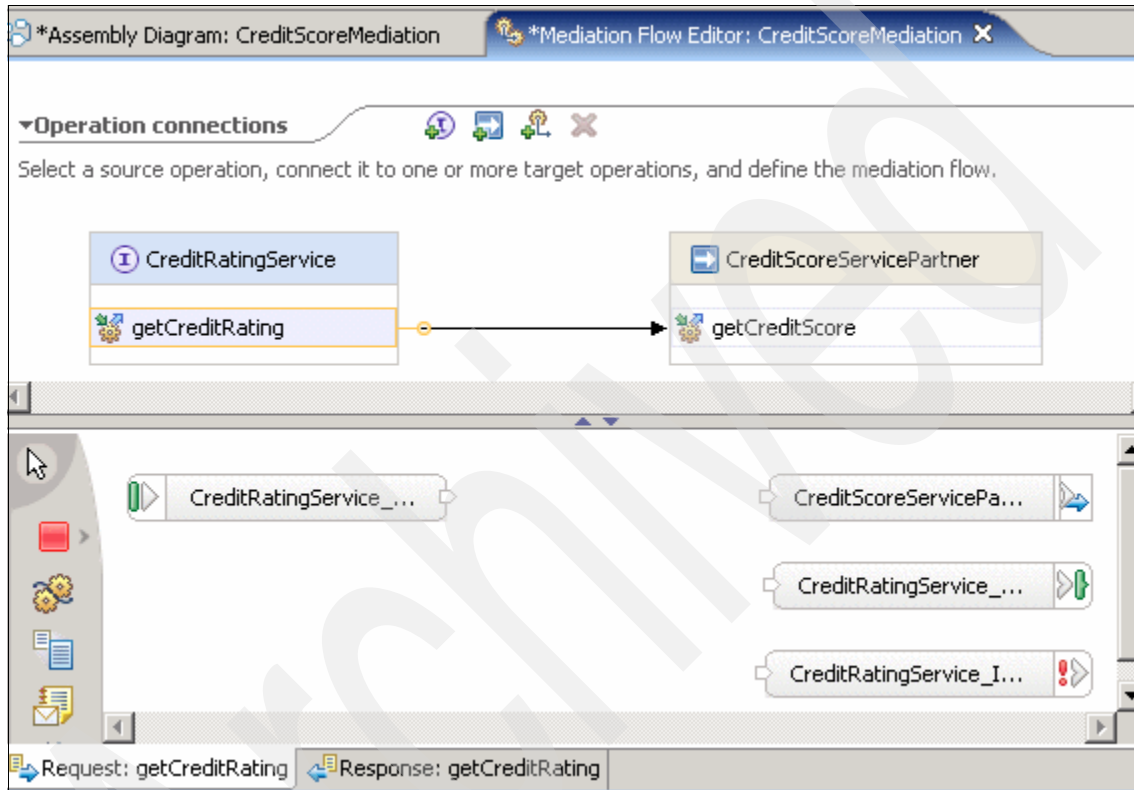


Figure 7-55 Mediation flow editor

Note that there are two flows, a request flow labeled Request: getCreditRating and a response flow labeled Response: getCreditRating. This is because the getCreditRating operation on the CreditRatingService interface is a request/response operation. You can switch between the two flows in the editor by clicking the appropriate tab at the bottom of the screen. First we will build the request flow, and then the response flow.

Request flow

In the request flow, you will see the starting point and endpoint nodes for the flow that were automatically generated. To the left you will find icons representing mediation primitives. To build the flow:

1. Add a Message Logger primitive. Change the display name to Log Credit Rating Request.
2. Add an XSL Transformation primitive. Change the display name to Set Credit Score Request.
3. Add a Message Logger primitive. Change the display name to Log Credit Score Request.
4. Wire the nodes in the flow so that it will look like Figure 7-56.

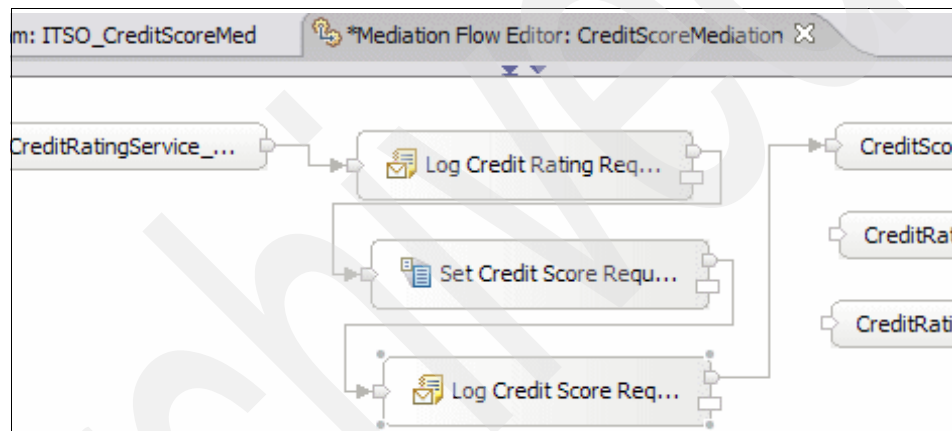


Figure 7-56 Connect the nodes in the flow

The response flow will use the transient context to store the credit score from the response message. Using the transient context makes the credit score available to the response flow.

5. Specify the Transient context business object (Figure 7-57):
 - a. Click the **CreditRatingService_getCreditRating_Input** node.
 - b. Click the **Details** tab in the Properties view.
 - c. On the Transient Context line click **Browse**.
 - d. Select the **CreditScoreMediationContext** business object.
 - e. Click **OK**.

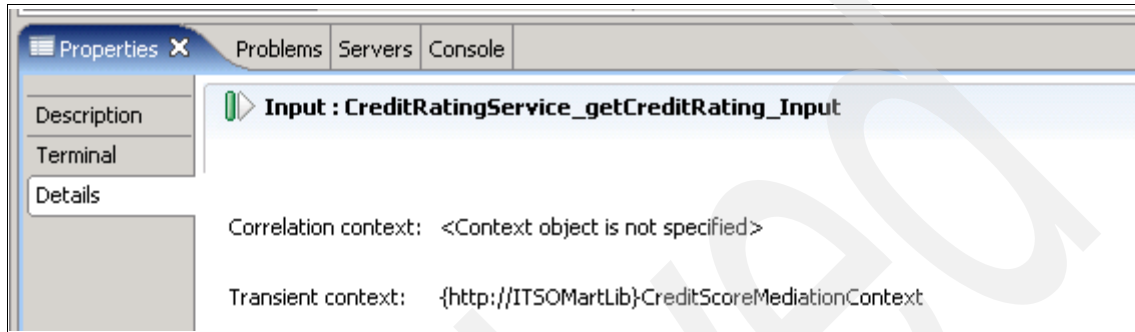


Figure 7-57 Transient context

6. Create a new XML mapping for the Set Credit Score Request node. Map the elements as shown in Table 7-4.

Table 7-4 XML mapping

CreditRatingService	CreditScoreService
phoneNo	phone
fullName	firstName
fullName	lastName

- a. The attributes firstName and lastName will be obtained from the fullName by parsing it:
 - i. Select **firstName** in the Overview view. Right-click and select **Define XSLT Function**.
 - ii. Select **Custom Java Bean** and click **Next**.
 - iii. For Java project select **ITSOMartUtils**.
 - iv. For Java bean select **MediationUtils**.
 - v. For Method select **parseFirstName**.
 - vi. For Input parameters select **fullName**.

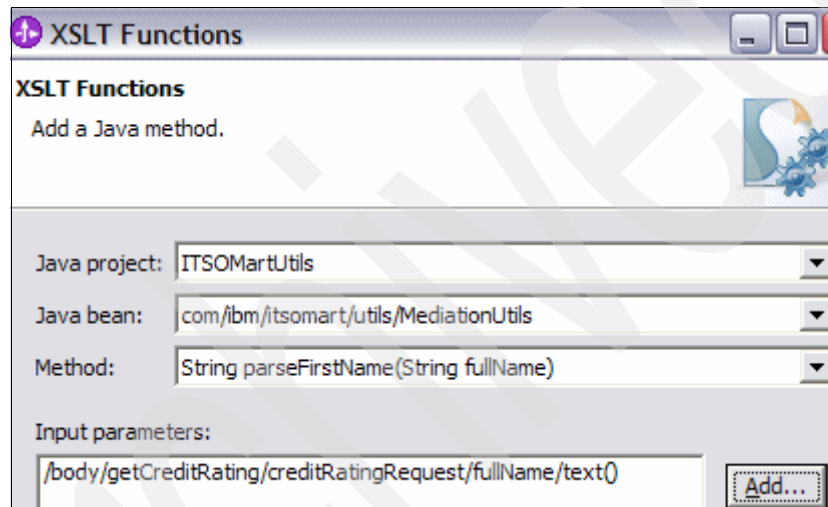


Figure 7-58 Define an XSLT Function

- vii. Click **Finish**.

- b. Repeat the procedure for lastName, selecting the **parseLastName** method instead.

If the functions are defined correctly, you will see the functions listed in the Applied Function column (Figure 7-59).

Target	Source	Applied Function/Grouping
[-] X smo	X smo	
[-] e body		
[-] e getCreditScore		
[-] e firstName	e fullName	⚙️ parseFirstName
[-] e lastName	e fullName	⚙️ parseLastName
[-] e phone	e phoneNo	

Figure 7-59 New applied XSLT Functions

- c. Close and save the map.
7. In the Properties view click the **Regenerate XSL** button to generate an XSL style sheet from the XML map.

Response flow

The Response flow is a little more complex than the ones we have seen so far. The final response flow looks like Figure 7-60.

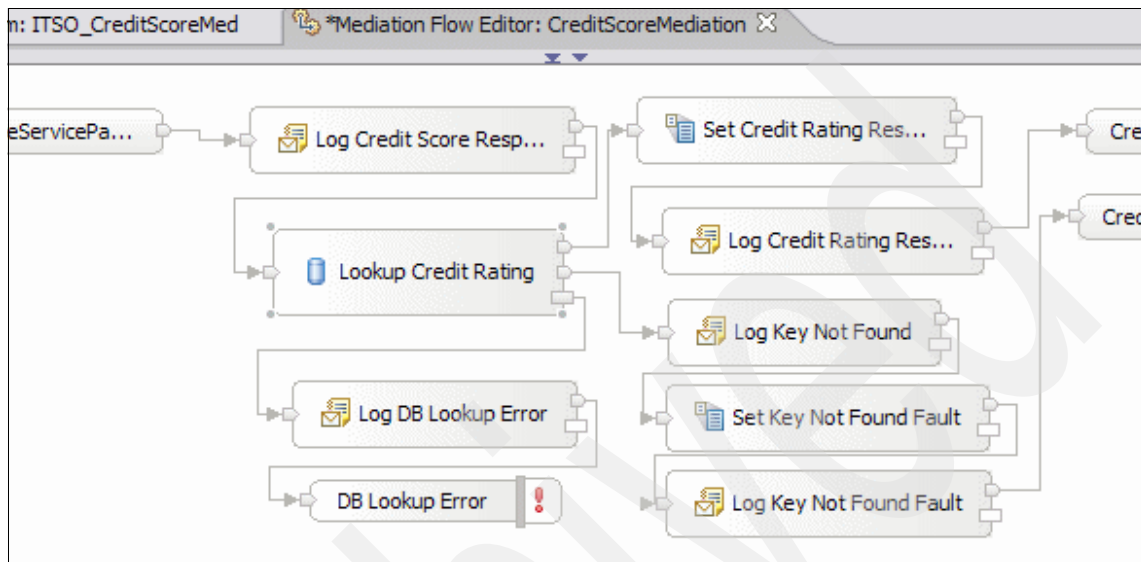


Figure 7-60 Response flow

To populate the response flow:

1. In the mediation flow editor switch to the response flow by selecting the tab at the bottom of the pane. You will see the starting point and endpoint nodes for the flow. To the left you will find icons representing mediation primitives.
2. Add a Message Logger primitive. Change the display name to Log Credit Score Response.

3. Add a Database Lookup primitive (Figure 7-61). Change the display name to Lookup Credit Rating.

Note that the Database Lookup primitive has an additional output terminal for *key not found*.

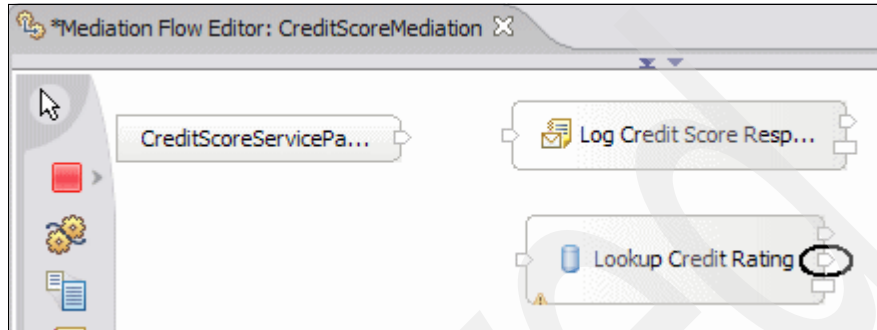


Figure 7-61 Change display name

4. Add an XSL Transformation primitive. Change the display name to Set Credit Rating Response.
5. Add a Message Logger primitive. Change the display name to Log Credit Rating Response.
6. Connect the nodes in the flow so that it looks like Figure 7-62.

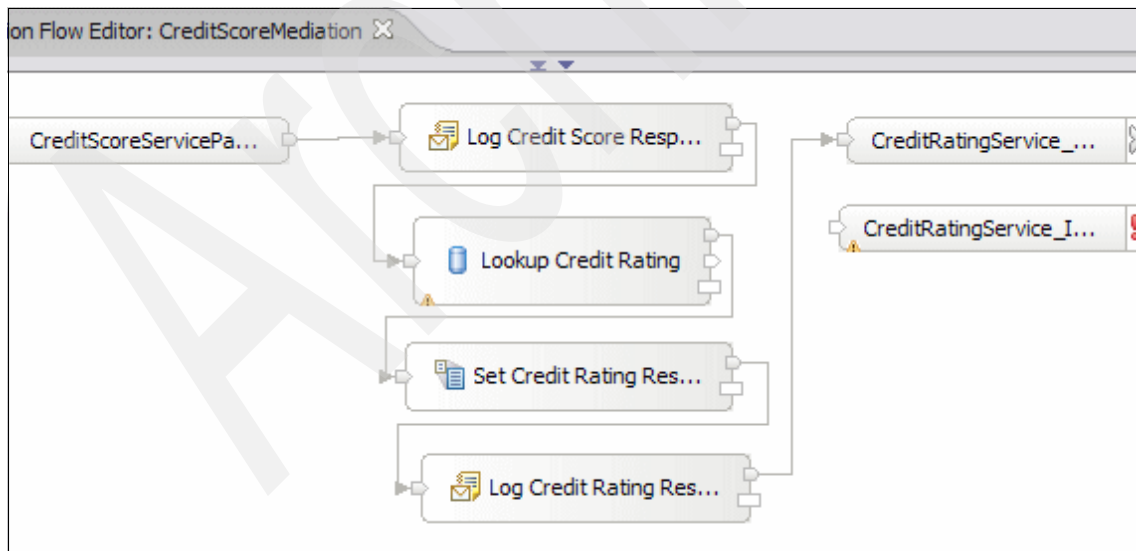


Figure 7-62 Connect the nodes in the response flow

7. To set the properties of the Database Lookup primitive do the following:
 - a. Select the Lookup Credit Rating node.
 - b. In the Properties view click the **Details** tab.
 - c. Enter the following settings:
 - i. Data source name: jdbc/ITSOMartDataSource
 - ii. Table name: ITSOMART.CREDIT_SCORE_XREF
 - iii. Key column name: SCORE
 - iv. Click the **Custom XPath** button and navigate to and select **getCreditScoreReturn:int**. See Figure 7-63. The resulting XPath location field should look like the following:
 /body/getCreditScoreResponse/getCreditScoreReturn

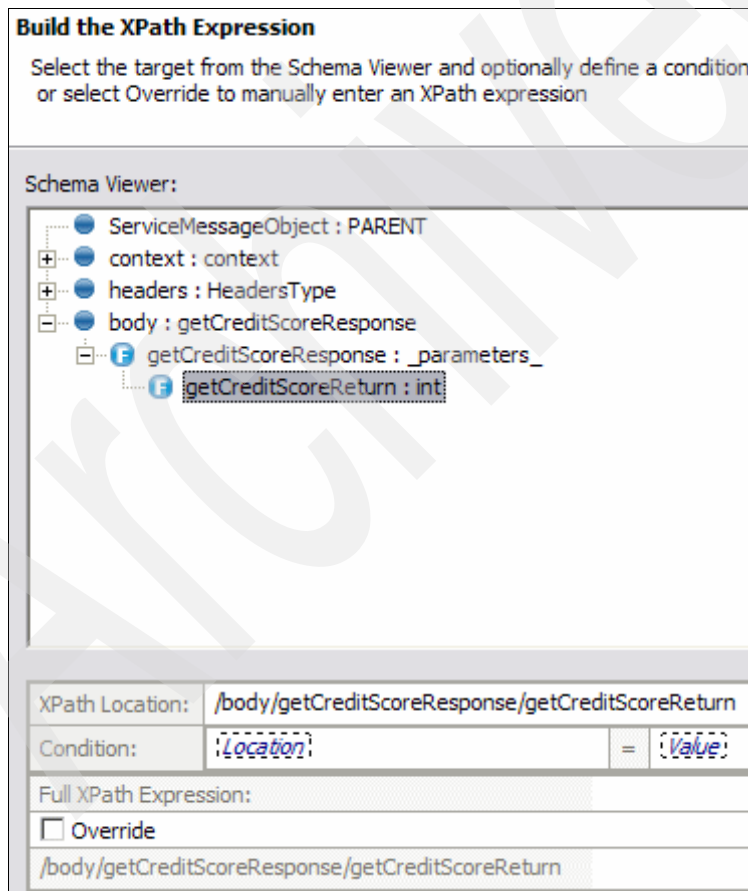


Figure 7-63 Key path

- v. Click **OK**.
- d. In the Data elements section, click **Add** and enter the following:
- Value column name: RATING
 - Message value type: java.lang.String
 - Message element:
/context/transient/creditRating
- To enter this, select **Custom XPath**, as shown in Figure 7-64.

Build the XPath Expression

Select the target from the Schema Viewer and optionally define a condition or select Override to manually enter an XPath expression

Schema Viewer:

- ServiceMessageObject : PARENT
 - context : context
 - correlation : anyType
 - transient : CreditScoreMediationContext
 - creditRating : string
 - failInfo : FailInfoType
 - headers : HeadersType
 - body : getCreditScoreResponse

XPath Location: /context/transient/creditRating

Condition: =

Full XPath Expression:

☐ Override

/context/transient/creditRating

Figure 7-64 Message element XPath

The properties for the Database Lookup primitive should look as shown in Figure 7-65.

Properties Problems Servers Console

Description

Terminal

Details

Database Lookup : Lookup Credit Rating

Data source name: * jdbc/ITSOMartDataSource

Table name: * ITSOMART.CREDIT_SCORE_XREF

Key column name: * SCORE

Key path: * /body/getCreditScoreResponse/getCreditScoreReturn Custom XPath

☐ Validate input

Data elements:

Value	column name	Message value type	Message element
	RATING	java.lang.String	/context/transient/creditRating

Add... Edit... Remove

Up Down

Figure 7-65 Database Lookup mediation primitive properties

8. To set the properties of the XSL Transformation primitive do the following:
 - a. Select the Set Credit Rating Response node.
 - b. In the Properties view click the **Details** tab.

- c. Change the Root field of the SMO message to a slash (/) so the complete SMO is used (Figure 7-66).

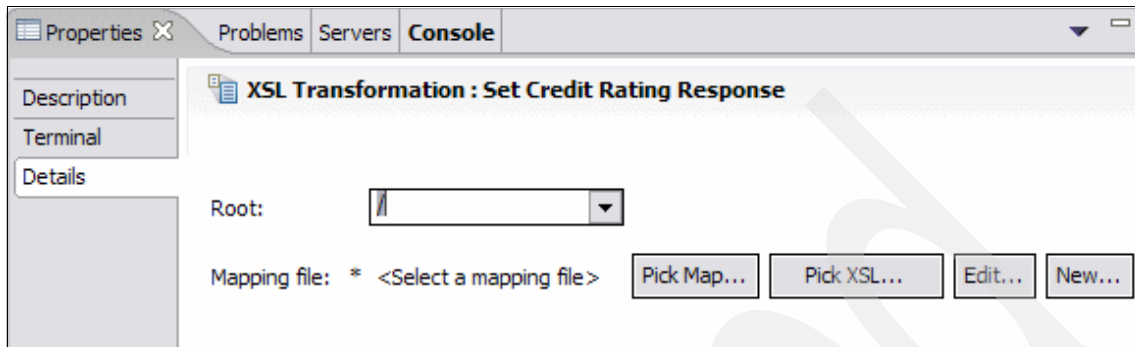


Figure 7-66 Creating a new mapping

- d. Click **New** and take the default input and output message types specified. Click **Finish** to launch the XML mapping editor.
- e. In the mapping editor, do the following:
 - i. Expand the source and target **tns:smo** tree.
 - ii. Select **context** in both the source and target panels, then right-click **context** in the source panel and select **Match Mapping**.
 - iii. Select **headers** in both the source and target panels, then right-click **headers** in the source panel and select **Match Mapping**.
 - iv. In the source panel select **context/transient/creditRating**, and drag and drop it onto the target element **body/getCreditRatingResponse/getCreditRatingReturn/rating**.

The final mapping should look like Figure 7-67.

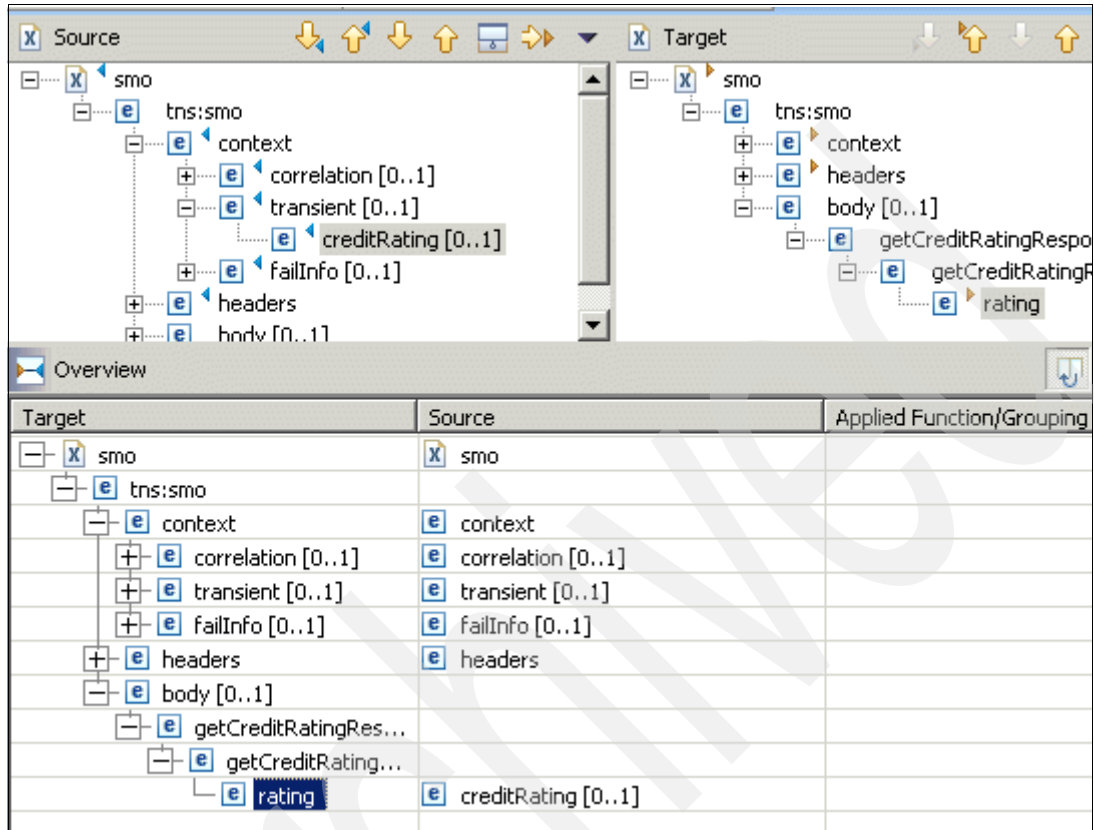


Figure 7-67 XSLT Mapping

- v. Close and save the map.
- f. In the Properties view click the **Regenerate XSL** button to generate an XSL style sheet from the XML map. Click **OK**.

Database Lookup primitive key not found terminal

The key not found terminal propagates the original message to the output terminal if the key is not found in the database. We now add primitives to handle this condition and wire them to this terminal:

1. Add a Message Logger primitive. Change the display name to Log Key Not Found.
2. Add an XSL Transformation primitive. Change the display name to Set Key Not Found Fault.

3. Add a Message Logger primitive. Change the display name to Log Key Not Found Fault.
4. Wire the nodes in the flow so the mediation flow looks like Figure 7-68. Note that we are wiring these nodes (in the order we created them) from the KeyNotFound terminal of the Database Lookup primitive, and ending with the CreditRatingService_InputFault callout node.

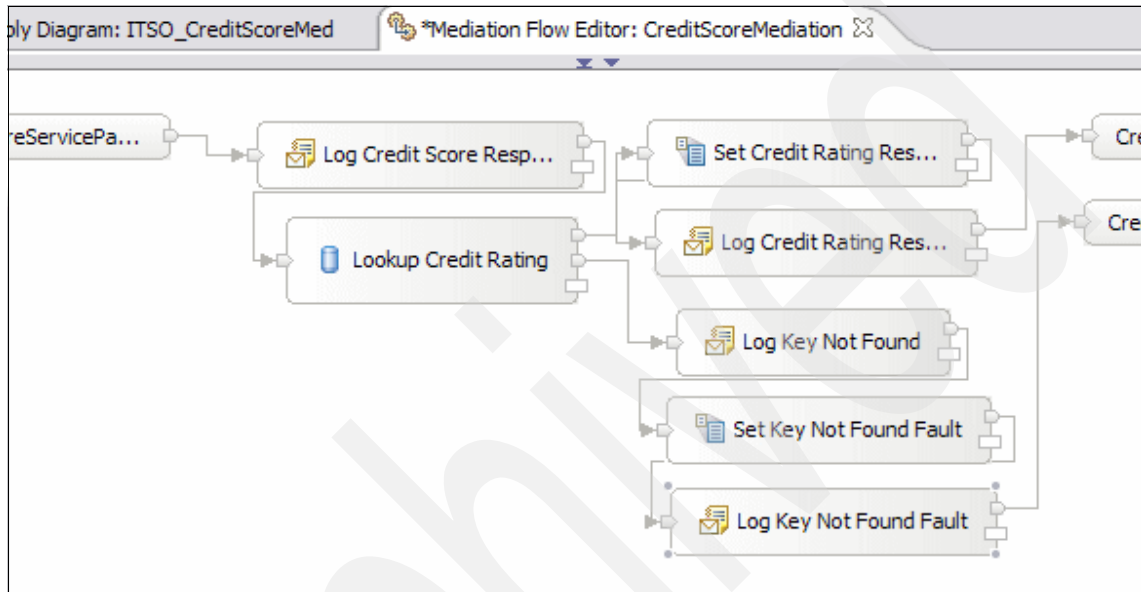


Figure 7-68 Connect the nodes in the response flow

5. Create a new XML mapping for the Set Key Not Found Fault node.
 - a. In the mapping editor, right-click **body/CreditRatingFault/errorMessage** on the target panel and select **Define XSLT Function**.
 - i. Select **string** for the function.
 - ii. Select **string** for the function name and click **Add**.

iii. Enter the error message shown in Figure 7-69. Click **OK**.

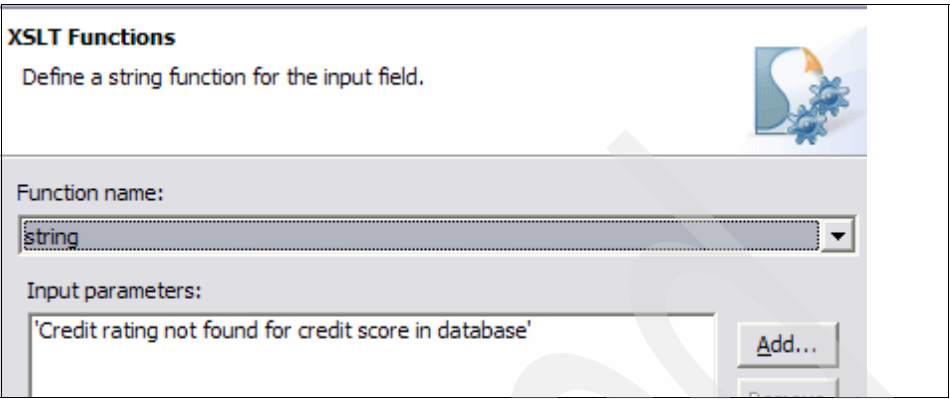


Figure 7-69 New XSLT message

iv. Click **Finish** to close the XSLT Function wizard.

The final mapping should look like Figure 7-70.

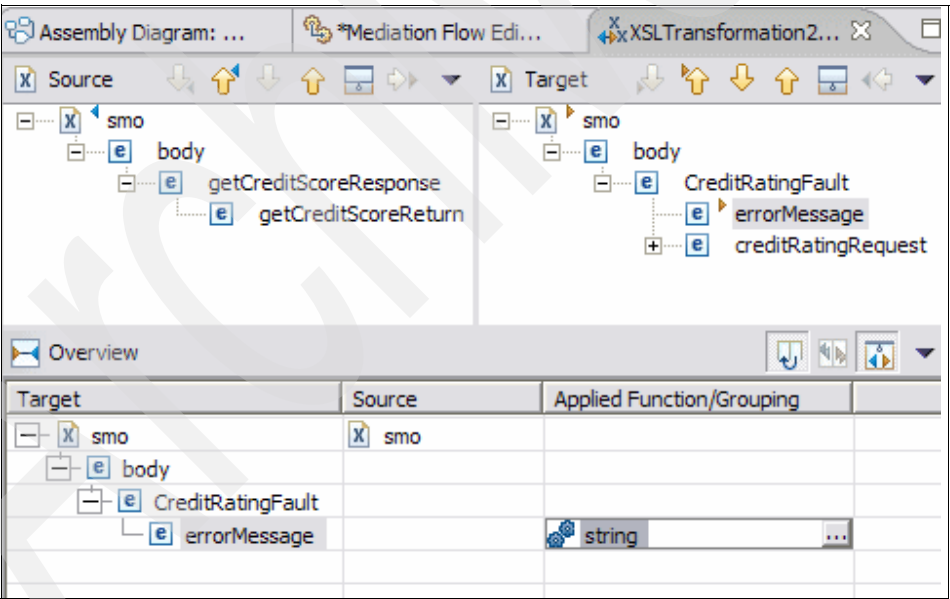


Figure 7-70 XSLT mapping

- b. Close and save the map.
- 6. Regenerate the XSL.

Database Lookup primitive Fail terminal

We now add primitives for the fail terminal:

1. Add a Message Logger primitive. Change the display name to Log DB Lookup Error.
2. Add a Fail primitive (Figure 7-71). Change the display name to DB Lookup Error.

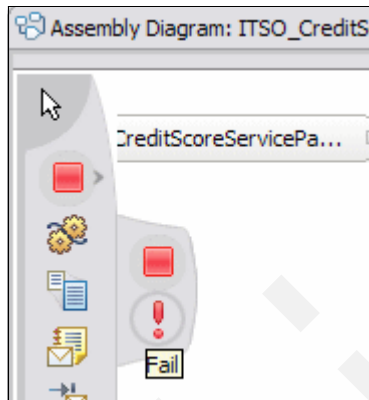


Figure 7-71 Show Fail mediation primitive

3. Connect the nodes in the flow so that it will look like Figure 7-72. Start with the fail terminal on the Lookup Credit Rating primitive and end with the in terminal of the DB Lookup Error primitive.

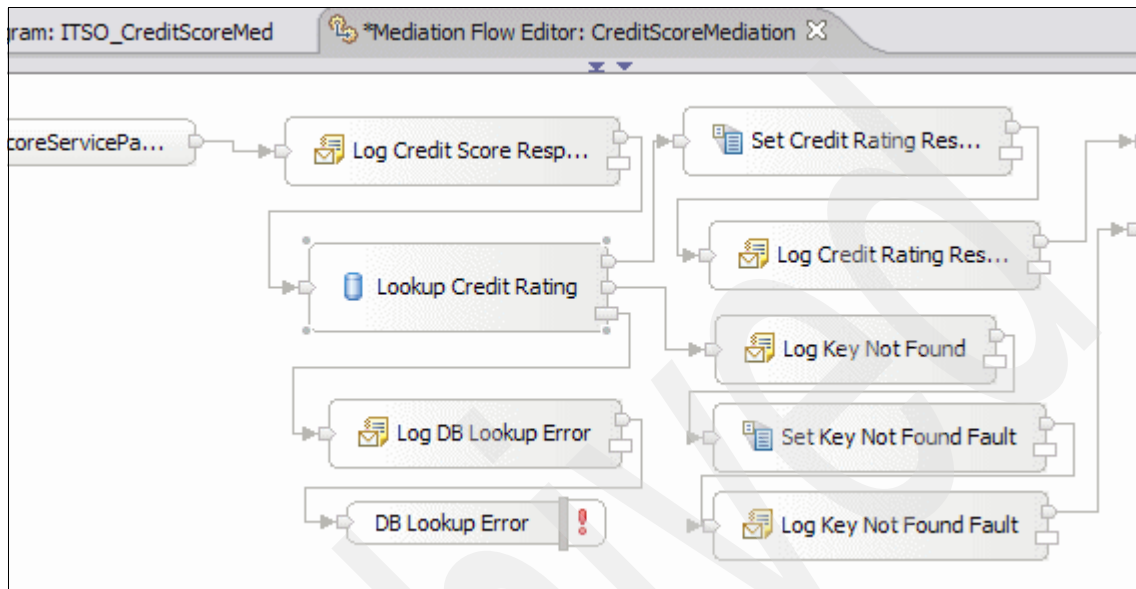


Figure 7-72 Connect the nodes in the flow

4. To set the properties of the Fail primitive, do the following:
 - a. Select the **DB Lookup Error** node.
 - b. In the Properties view click the **Details** tab.
 - c. Set the error message to Unexpected error occurred looking up credit rating in the database (Figure 7-73).

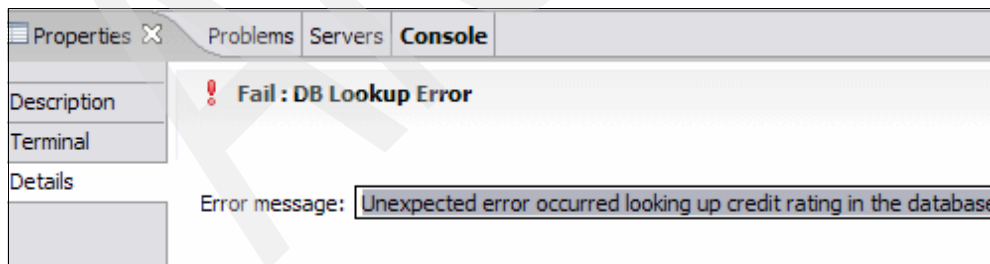


Figure 7-73 Fail error message

5. Save and close the Mediation Flow Editor.

Invoke the new mediation module

Now that we have a mediation to invoke the new service and translate the credit scores, we want to invoke it from the original mediation. In order to keep the original mediation flow intact, we work with a copy.

The steps required to complete the new mediation are:

1. Make a copy of the ITSO_CreditRatingMed mediation module.
2. Invoke the Credit Score mediation from the copy of the Credit Rating mediation.

The new assembly module looks like Figure 7-74.

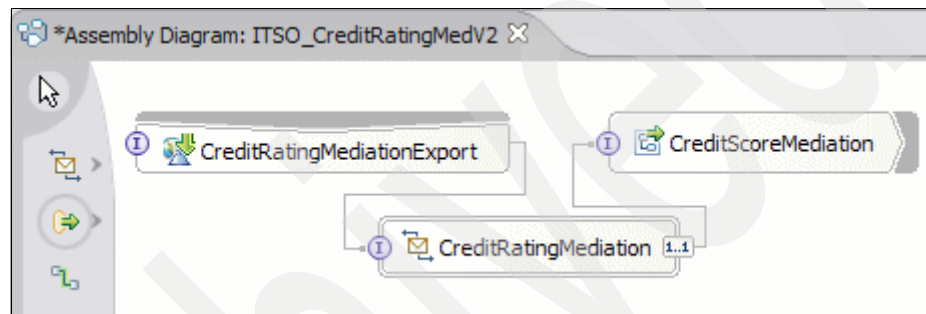


Figure 7-74 ITSO_CreditRatingMedV2 assembly module

Make a copy of the ITSO_CreditRatingMed mediation module

To make a copy of the ITSO_CreditRatingMed:

1. Right-click in the Business Integration view and select **Show Files** to open the Physical Resources view.
2. Right-click **ITSO_CreditRatingMed** and select **Copy**.
3. Right-click in the blank area of the Physical Resources view and select **Paste**.
A window will open indicating that a copy of the project is made. Change the Project name to ITSO_CreditRatingMedV2.
4. Click **OK**.
5. The mediation module will be copied and an error will appear. This is because when you copy the module, it does not rename all the references.

Note: The same problem occurs when renaming a mediation module.

6. Select **ITSO_CreditRatingMedV2** and navigate to the `sca.module`. Right-click and select **Open with** → **Text Editor** (Figure 7-75).

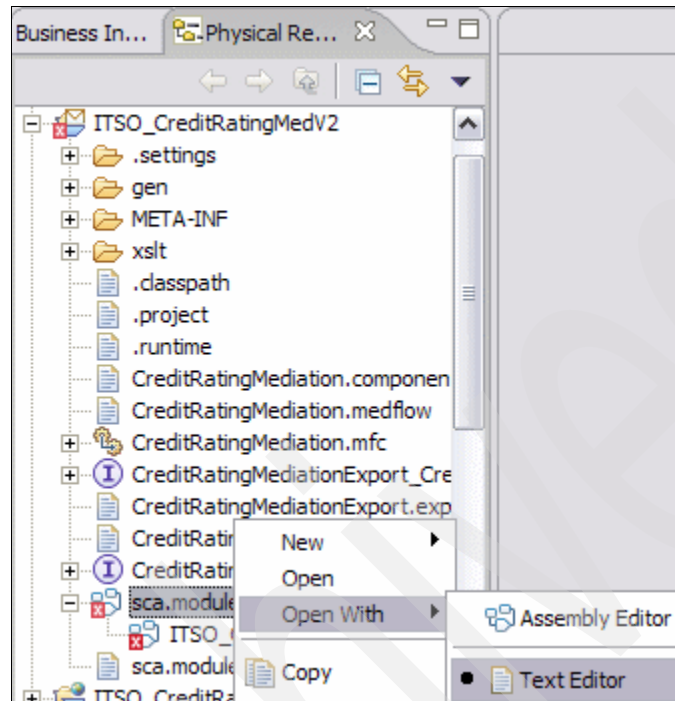


Figure 7-75 Open `sca.module`

7. The `sca.module` will open in an XML format. In the `scdl:module` element look for the `name` attribute. Change it to `ITSO_CreditRatingMedV2` (Figure 7-76).

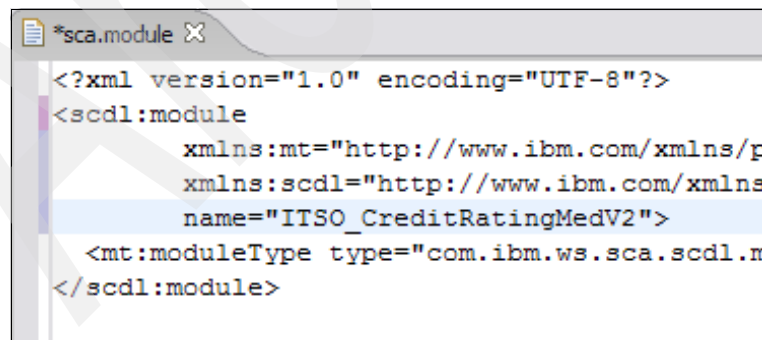


Figure 7-76 `sca.module`

8. Save and close the `sca.module`.

Tip: We recommended that after doing this you do the following:

1. Open the assembly diagram and delete the CreditRatingMediationExport component.
2. Delete the CreditRatingMediationExport_CreditCheckHttpPort Web service port.
3. Add the Export component back to the diagram (and create the new WSDL file). To do this, right-click the mediation component and select **Export** → **Web service binding**.
4. Close the project. To do this from the Physical Resources view, right-click the project, and select **Close project** from the context menu.
5. Reopen the project and rebuild it.

This will force a regeneration of all the J2EE integration modules and eliminate any errors that are still around.

Invoke the Credit Score mediation from the Credit Rating mediation

Next we need to alter the copy of the Credit Rating mediation to import the Credit Score mediation:

1. Navigate to the module assembly of the ITSO_CreditRatingMedV2 in the Business Integration view. Open the module assembly by double-clicking it.

Note: It may happen that when trying to open the module assembly it opens it in XML format. To open the Assembly Editor right-click the module assembly and select **Open With** → **Assembly Editor**.

2. Select the **CreditRatingService** node and press the Delete key, or right-click and select **Delete** to remove the node from the assembly editor.

3. Navigate to the CreditScoreMediationExportSCA (Figure 7-77).

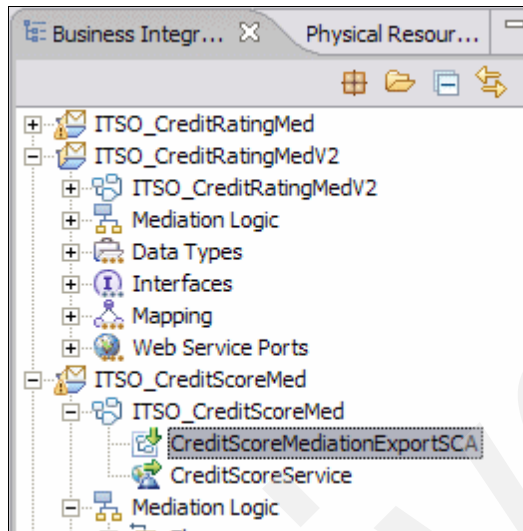


Figure 7-77 CreditScoreMediationExportSCA

4. Select the CreditScoreMediationExportSCA and drag and drop it on to the ITSO_CreditRatingMedV2 assembly diagram to the right of the mediation flow component.

When you drop the component onto the assembly diagram, you will be asked to choose which type of component you want to create. Select **Import with SCA binding**.

5. An import component will appear on the assembly diagram. Move the new import to the right of the mediation flow component.
6. Select the new import component and in the Properties view and change the display name to CreditScoreMediation.

This allows you to call the ITSO_CreditScoreMed module from the new mediation module.

7. Wire the components as shown in Figure 7-78.

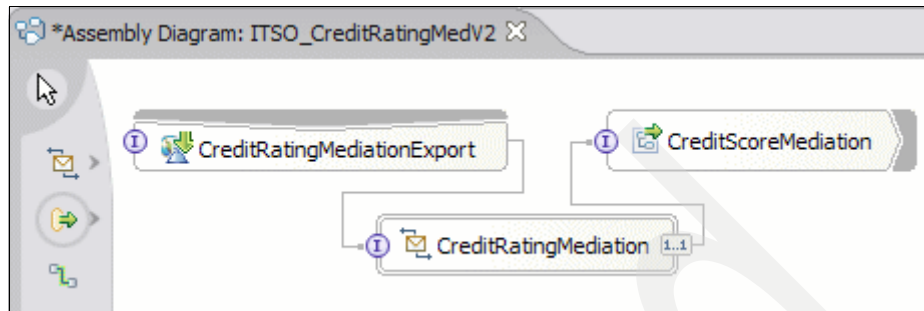


Figure 7-78 Wire the components.

8. Save and close the assembly diagram.

7.4.7 Test the mediation

The new mediation is ready to test with the integration test client.

Runtime requirements

For the test you will need the following:

- ▶ ITSO_CreditScoreService will need to be installed and running on the application server.

Note: ITSO_CreditScoreService is zipped with the sample using the name CSSvc.

- ▶ The ITSOMart database will need to be built and available.

The ITSOMart database holds the values used to correlate numbers returned from ITSO_CreditScoreService to the values gold, silver, or bronze. In our tests we used both DB2 and Cloudscape™.

Instructions for creating both types of databases can be found in “Create the database and configure the JDBC data source” on page 597.

So you can get a feel for what the database looks like, we have included the DB2 SQL statements in Example 7-1.

Example 7-1 SQL statements for database

```
CREATE SCHEMA ITSOMART;  
  
CREATE TABLE ITSOMART.CREDIT_SCORE_XREF  
  (SCORE INTEGER NOT NULL,  
   RATING VARCHAR(10) NOT NULL);  
  
ALTER TABLE ITSOMART.CREDIT_SCORE_XREF  
  ADD CONSTRAINT CREDIT_SCORE_PK PRIMARY KEY (SCORE);  
  
COMMENT ON COLUMN ITSOMART.CREDIT_SCORE_XREF.SCORE IS 'credit score';  
  
COMMENT ON COLUMN ITSOMART.CREDIT_SCORE_XREF.RATING IS 'ITSOMart credit rating';  
  
INSERT INTO ITSOMART.CREDIT_SCORE_XREF (SCORE, RATING) VALUES (100,'BRONZE');  
INSERT INTO ITSOMART.CREDIT_SCORE_XREF (SCORE, RATING) VALUES (200,'SILVER');  
INSERT INTO ITSOMART.CREDIT_SCORE_XREF (SCORE, RATING) VALUES (300,'GOLD');
```

In our scenario, we unzipped the Cloudscape database into the `<profile_name>/databases` directory (Figure 7-79).

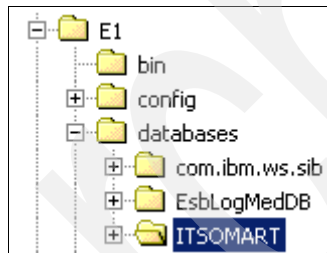


Figure 7-79 ITSOMART Cloudscape database location

- ▶ The data source for the ITSOMart database needs to be defined to the application server. The WebSphere ESB application server has a JDBC driver for Cloudscape predefined at the server level. We defined the data source using this JDBC driver:

- JDBC driver (server scope): Cloudscape JDBC Provider (XA)
- Name: ITSOMART
- JNDI name: jdbc/ITSOMartDataSource

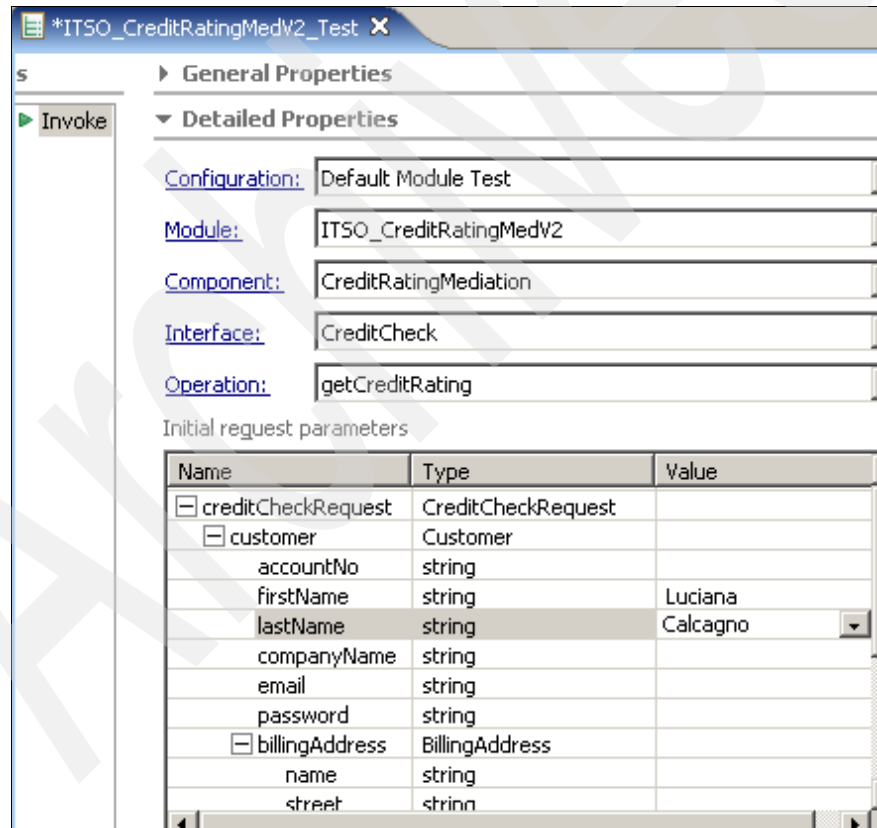
This has to match what you specified in the data source name field of the Database Lookup primitive (page 324).

– Database name: c:\e1\databases\ITSOMART

Test the flow

To use the tool to test our mediation flow:

1. Rebuild all projects (select **Project** → **Build All**).
2. Start the WebSphere ESB Server test environment.
3. Add the mediation application, ITSO_CreditRatingMedV2App, to the server.
4. Launch the test client for the ITSO_CreditRatingMedV2 module.
5. Set the values in the test client. Fill in some test data for the request. The only values actually required for this mediation to work properly are the first name and the last name (Figure 7-80).
 - a. For a success flow, enter a surname that has less than nine characters.



Name	Type	Value
<input type="checkbox"/> creditCheckRequest	CreditCheckRequest	
<input type="checkbox"/> customer	Customer	
accountNo	string	
firstName	string	Luciana
lastName	string	Calcagno
companyName	string	
email	string	
password	string	
<input type="checkbox"/> billingAddress	BillingAddress	
name	string	
street	string	

Figure 7-80 Test client - ITSO_CreditRatingMedV2

You may be prompted to select the deployment location. If so, select **WebSphere ESB Server v6.0**.

Check the test client Events pane to verify that everything ran successfully (Figure 7-81).

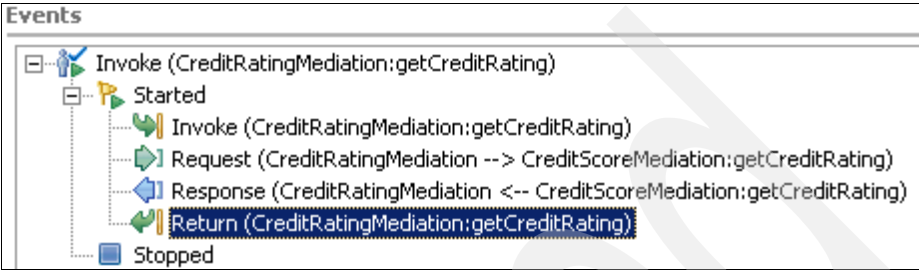




Figure 7-81 Test client - Events

In the detail properties we can see the return parameters (Figure 7-82).


General Properties


Detailed Properties

Module:
[ITSO_CreditRatingMedV2](#)

Component:
[CreditRatingMediation](#)

Interface:
[CreditCheck](#)

Operation:
[getCreditRating](#)

Return parameters:

Name	Type	Value
<input type="checkbox"/> creditCheckResponse	CreditCheckResponse	
creditRating	String	GOLD

Figure 7-82 Return parameters

- v. Look at the messages logged by the Message Logger primitives (Figure 7-83).

Open the sample MessageLogApp application using the following URL:

`http://localhost:9080/MessageLogWeb/faces/MessageLogView.jsp`

ESB Mediation Message Log				
<div>Refresh</div> <div>Clear Log</div>				
	TIMESTAMP	MESSAGE ID	MEDIATION	MODULE
	Mar 24, 2006 12:58:58 PM MST	9f96d62d-0a01-0000-0080-c845f688df82	Log Mediation Response	ITSO_CreditRatingMedV2
	Mar 24, 2006 12:58:58 PM MST	9f96d62d-0a01-0000-0080-c845f688df82	Log Credit Rating Response	ITSO_CreditRatingMedV2
	Mar 24, 2006	6096d62d-0a01-	Log Credit	

Figure 7-83 Message Logger messages

- b. For a key not found flow, enter a last name that has nine or more characters.
 - i. Check the test client Events pane to verify that it failed.
 - ii. Look at the messages that were logged by the Message Logger primitives.

7.5 Calling the service from the application

The Register Customer process application calls the Credit Rating mediation via a Web service proxy that makes a SOAP/HTTP request. This Web service proxy is generated based on the WSDL file that describes the mediation. This is the WSDL file generated when you create the export for the mediation.

For example, to generate a Web service proxy that calls the Credit Score mediation (the second version), use the WSDL file `CreditRatingMediationExport_CreditCheckHttp_Service.wsdl`.

Your application will also need access to any WSDL files referenced in this file, and the XSD files for the data objects used. The best way to pick up the files you need is to build the mediation module and go through the test process for the mediation. Then copy the files found under the wsdl folder for the mediation EJB to the application EJB.

You can find these files using the Physical Resources view under <mediationEJB>/ejbModule/wsdl.

The Register Customer process application, ITSO_RegProcServiceApp, uses a Java utility project to hold the proxies for the services it calls. This utility project is called ITSO_RegProcService_Proxies.

The steps required to generate the proxy are:

1. Copy or import the required files to the utility project.
2. Generate the Web service client proxy using the Web Service Client wizard. Use the service WSDL file as input.
3. Add code to the application to call the service via the proxy.

7.5.1 Import or copy the WSDL files

You will need to obtain a copy of the WSDL files for the service and put them in your workspace. In our example, we have the EAR files for the Web services in our workspace and thus have the WSDL available.

Using the J2EE perspective, we have created a folder called `wsdl` in the `ITSO_RegProcService_Proxies` project to hold a copy of the files. You can see in Figure 7-84 the files we copied from the EJB project to the utility project.

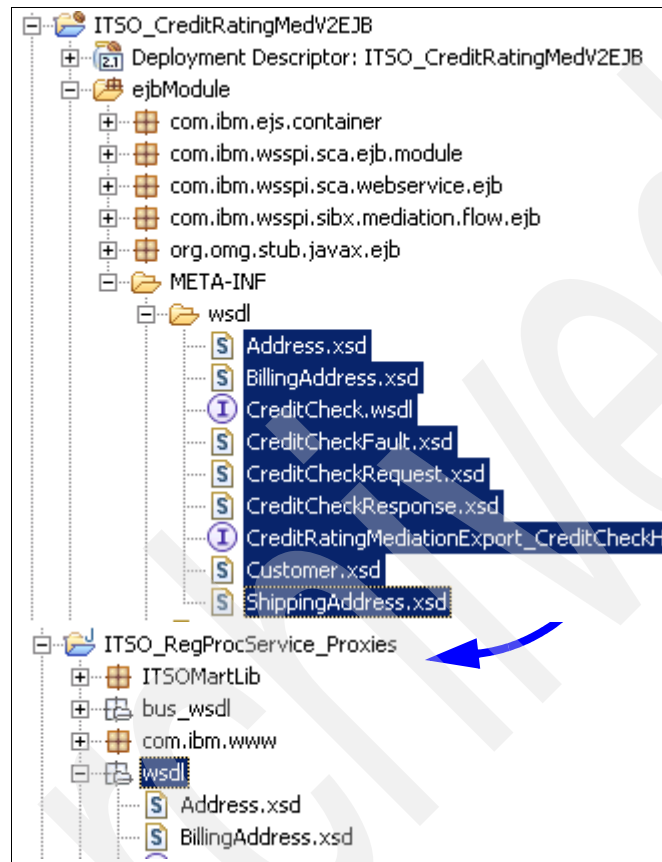


Figure 7-84 WSDL files copied to utility project

7.5.2 Generate the Web service client proxy

You are now ready to generate the Web service client proxy. Enable the Web Service Developer capability for your workspace (see 6.3.3, “Configure Web services workspace preferences” on page 226).

1. Switch to the J2EE perspective, and in the Project Explorer view, navigate to the WSDL files you saved.

2. Right-click the service file and select **Web Services** → **Generate Client** (Figure 7-85).

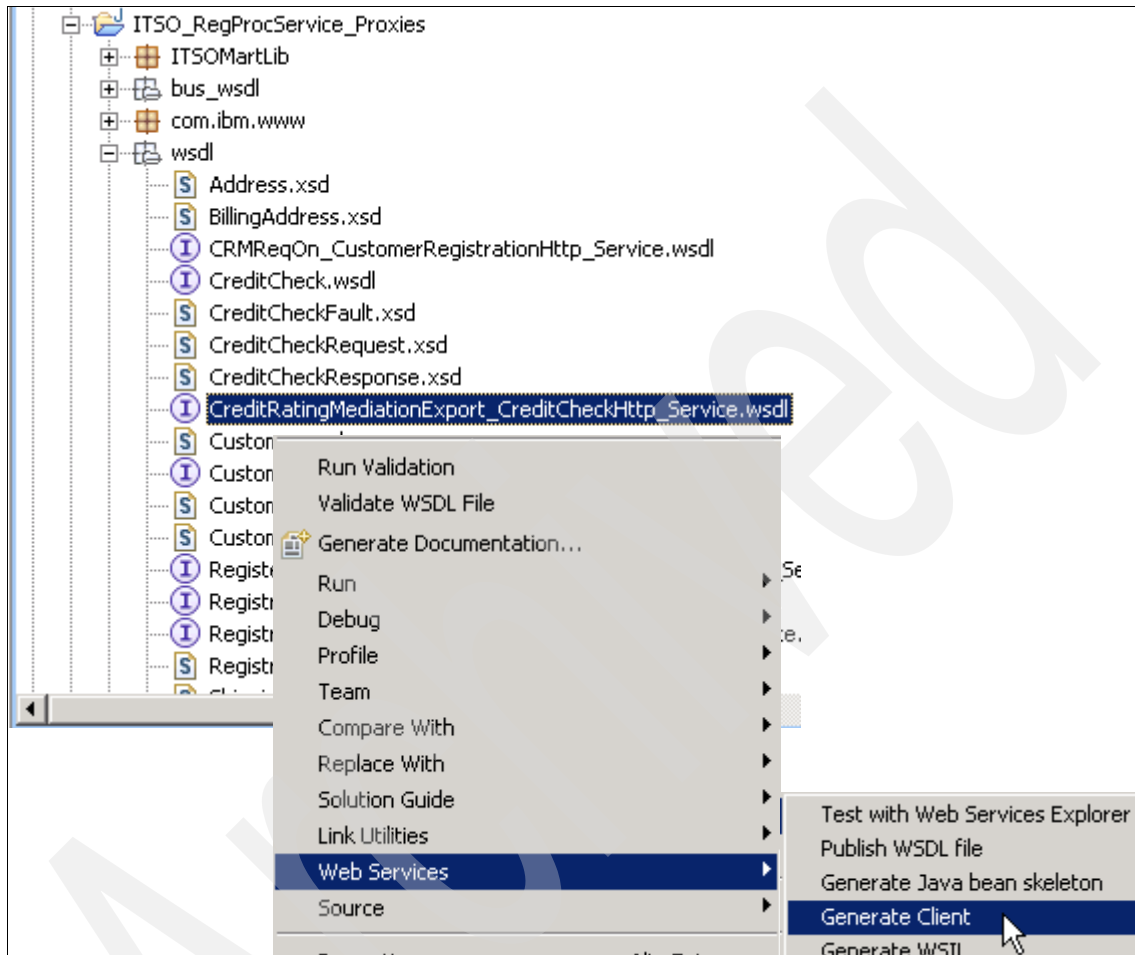


Figure 7-85 Start the Web Service Client wizard

This will start the Web Service Client wizard.

- a. Select **Java proxy** for the client proxy type and click **Next**.
- b. The WSDL file that you selected should already be filled in. Click **Next**.
- c. Select the following values to indicate how the client proxy will be generated (Figure 7-86 on page 345):
 - Web service runtime: **IBM WebSphere**.
 - Server: **WebSphere ESB Server v6.0**.

- J2EE version: You can leave **N/A** as the value here because the proxy is getting generated into a Java project, not a J2EE project.
- Client type: **Java**.
- Client project: **ITSO_RegProcService_Proxies**.

See Figure 7-86.

Web Service Client

Client Environment Configuration

Choose from the list of supported runtime protocols and servers for the client environment, or use default settings.

Client-Side Environment Selection:

Web service runtime: IBM WebSphere

Server: WebSphere ESB Server v6.0

J2EE version: N/A

Edit...

Client type: Java

Client project: ITSO_RegProcService_Proxies

EAR project:

Figure 7-86 Web Service Client: Client Environment Configuration

- d. Click **Finish** to generate the Web service client proxy. Acknowledge any warning messages you receive during the Web service client generation.

7.5.3 Update the application to call the service using the proxy

The following code in `ITSO_RegProcServiceApp` calls the service using the proxy (Example 7-2).

Example 7-2 Invoking the credit check service

```
try {
    CreditCheckRequest creditCheckRequest = new CreditCheckRequest();
    creditCheckRequest.setCustomer(customer);

    CreditCheckProxy creditCheckProxy = new CreditCheckProxy();
    System.out.println("ITSOMart RegistrationProcessorService.registerCustomer >>
invoking CreditCheck Service (soap/http) w/ endpoint: " + creditCheckProxy.getEndpoint());
    CreditCheckResponse creditCheckResponse =
creditCheckProxy.getCreditRating(creditCheckRequest);

    creditRating = creditCheckResponse.getCreditRating();
    System.out.println("ITSOMart RegistrationProcessorService.registerCustomer >>
credit rating: " + creditRating);
} catch (Exception creditCheckException) {
    registrationProcessStatus = FAILURE;
    registrationStatusDetails = "CreditCheck Service exception: " +
creditCheckException;
}
```

Building the CRM mediation

This chapter shows an example of using the capabilities of WebSphere ESB to provide routing between a service requester and one of multiple service providers based on the request.

The mediation built for this example illustrates the following:

- ▶ Connectivity to EIS systems using WebSphere Adapters
- ▶ Request/response operation
- ▶ XML transformation of messages
- ▶ Message routing
- ▶ Using the mediation as an endpoint
- ▶ Message logging
- ▶ Building a custom mediation

This chapter includes the following topics:

- ▶ Scenario overview
- ▶ Developing a mediation to update a CRM system
- ▶ Calling the service from the application

If you are not familiar with the process of building mediations, review Chapter 6, “Assemble with WebSphere Integration Developer” on page 211, and Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263.

8.1 Scenario overview

This chapter focuses on the CRM Registration scenario of the ITSOMart solution.

Note: The samples included in this chapter can be downloaded from the Web. See Appendix A, “Sample application install summary” on page 589, for instructions on downloading and importing the sample projects.

8.1.1 Business scenario

The first step in the Register Customer process is to evaluate the customer credit rating. The rating is used to determine how to proceed with the registration. This step of the process was built in Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263.

We now look at the next step in the process. This step evaluates the credit rating and determines which action should be taken with the request. The possibilities are to deny registration, send the registration to a queue for manual handling, or register the customer.

Based on the credit rating, the following actions are taken:

- ▶ Gold
ITSOMart uses a Siebel system to manage its Customer Relationship Management (CRM) data. If the credit rating is gold, the customer data is sent to the Siebel system for registration.
- ▶ Silver
The customer registration request is sent to an internal department for manual handling. Currently these requests are stored in a file on the server. Other than logging the transaction, no further processing by the Register Customer process is required.
- ▶ Bronze
The customer registration request is denied. Other than logging the transaction, no further processing by the Register Customer process is required.

The activity diagram for the CRM Registration scenario of the ITSOMart solution can be seen in Figure 8-1. The activity diagram for the entire process can be seen in Figure 5-15 on page 143.

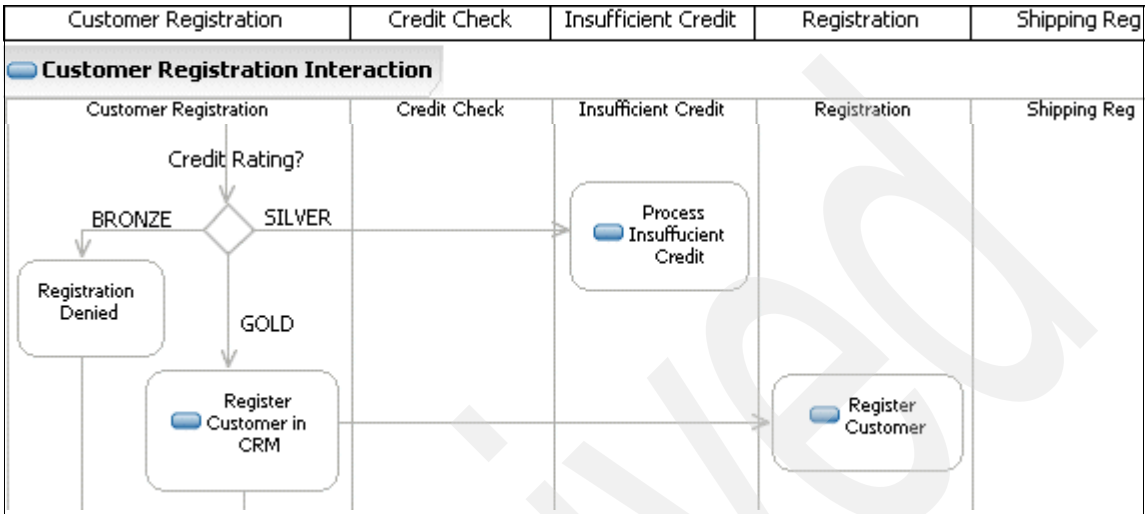


Figure 8-1 CRM Registration scenario

8.1.2 CRM mediation

The call from the Register Customer process will call the CRM Registration Service via the ESB. Since only gold level customers are automatically registered, a mediation in the ESB will be used to determine the proper destination for the request and to transform it to the appropriate format.

Figure 8-2 shows the activity diagram for the mediation.

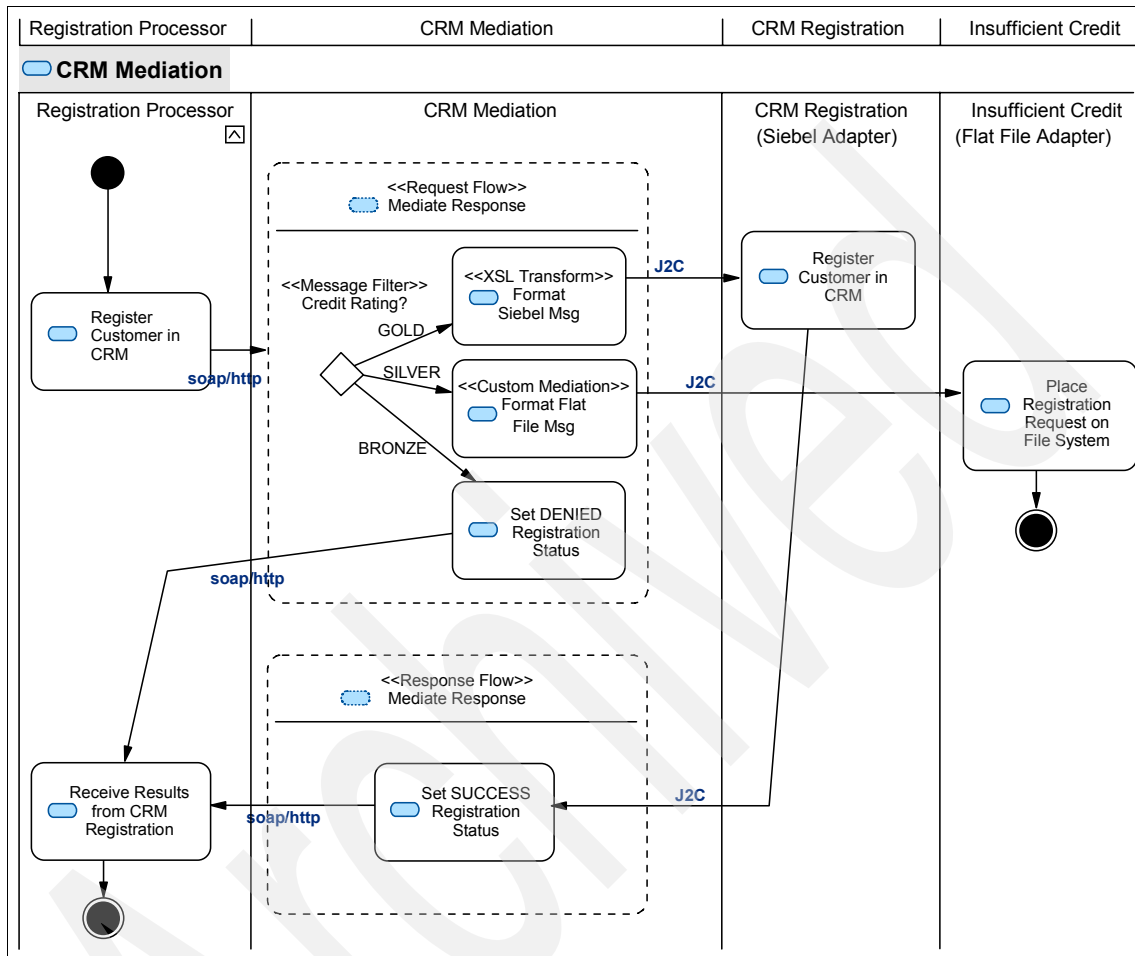


Figure 8-2 CRM mediation activity diagram

The primary purpose of the mediation is to route the customer registration request to the appropriate service. The mediation is invoked using SOAP/HTTP. However, two of the services used in the mediation are called using J2C adapters. The first is the Siebel system, which is accessed using the IBM WebSphere Adapter for Siebel Business Applications. The second is a flat file on the server, which is accessed using IBM WebSphere Adapter for Flat Files.

The interface for the mediation defines the input and output for the mediation. The input will consist of customer registration data and the customer credit rating.

The response consists of a string (SUCCESS, DENIED) that describes the registration status.

8.2 Developing a mediation to update a CRM system

This example builds a mediation to update a CRM system using two WebSphere JCA Adapters, one for Siebel Business Applications and a second one to interact with flat files on the file system. The decision of whether to update the Siebel System or to write a file to the file system is made by a Message Filter primitive based on the credit rating of the customer. Once the Message Filter primitive selects a path, we use a combination of XSLT and Custom primitives to transform the input request to the format required by the destination system. This mediation will also use a Message Logger primitive to log the messages at different points in the flow.

8.2.1 Mediation development steps

Figure 8-3 shows the mediation module contents we are going to build.

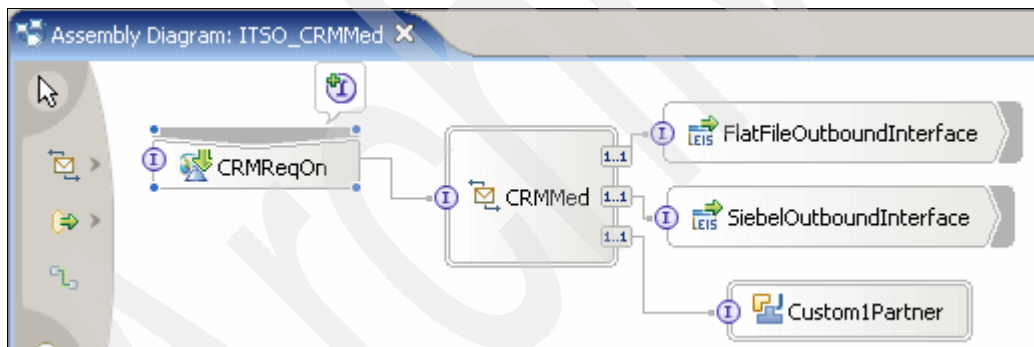


Figure 8-3 ITSO_CRMMed mediation components

The steps required to build the mediation in WebSphere Integration Developer having a target deployment in the WebSphere ESB Test Environment are:

1. Create the mediation module.
2. Create an interface for each EIS system.
3. Define the interface for the mediation.
4. Add the components to the module assembly.
5. Build the mediation flow.

Note: If you are not familiar with using WebSphere Integration Developer to develop mediations, review Chapter 6, “Assemble with WebSphere Integration Developer” on page 211, and Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263, before proceeding.

To build and test this scenario you must have the following projects in your workspace:

- ▶ **ITSOMartLib:** This project was built in Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263.
- ▶ **MessageLogApp:** This project contains a J2EE application that can be used to display the contents of the logs populated by the Message Logger mediation primitives.

You will also need to have access to the following:

- ▶ IBM WebSphere Adapter for Flat Files
- ▶ IBM WebSphere Adapter for Siebel Business Applications
- ▶ A running Siebel system

8.2.2 Create the mediation module

The first step is to create a mediation module. You can optionally create a library first to hold resources, such as business objects or interfaces, that you may use in other modules. If you create a library, you can add it as a dependency while you create the mediation module. In this example, the ITSOMartLib library will be used (see 7.2.1, “Create a library” on page 268).

To create the new mediation module for this sample, do the following:

1. While in the Business Integration view, select **File** → **New** → **Other** → **Mediation module**.
2. Name the mediation module `ITSO_CRMMed` and be sure to check the box that allows the wizard to create the mediation flow component. Click **Next**.
3. Check the box by the ITSOMartLib library to include it as a dependency library and click **Finish**.

8.2.3 Create an interface for each EIS system

This mediation will use the IBM WebSphere Adapter for Siebel Business Applications to access a Siebel system and IBM WebSphere Adapter for Flat Files.

When using a WebSphere Adapter in a mediation, you deploy the adapter as part of the application. The RAR files should be imported into the WebSphere Integration Developer workspace and included in the mediation application. (You do not install stand-alone adapters in the WebSphere Application Server.)

WebSphere Integration Developer provides an enterprise service discovery wizard that can use the adapters to query the EIS systems to discover services available, then use that information to build interfaces and import/export components to be used in the mediations.

Guidance for using the enterprise service discovery for each WebSphere Adapter can be found in the product documentation for the adapter. You can find this documentation at:

<http://www-306.ibm.com/software/integration/wbiadapters/library/infocenter/doc/index.html>

IBM WebSphere Adapter for Siebel Business Applications

This section shows how to import the WebSphere Adapter for Siebel into the workspace. This assumes that you have installed the adapter product (see “Installing WebSphere Adapters” on page 667).

1. Start WebSphere Integration Developer. Select **File** → **Import** → **RAR file** and click **Next**.
 - a. In the Connector file field, browse to the location of the installed adapter RAR file. Normally it is `<install_root>/adapter/<adapter_name>/deploy`. Click **Open**.

The Connector project value and EAR project name will be automatically filled in for you.

- b. Check the **Add module to an EAR project** and click **Finish**.

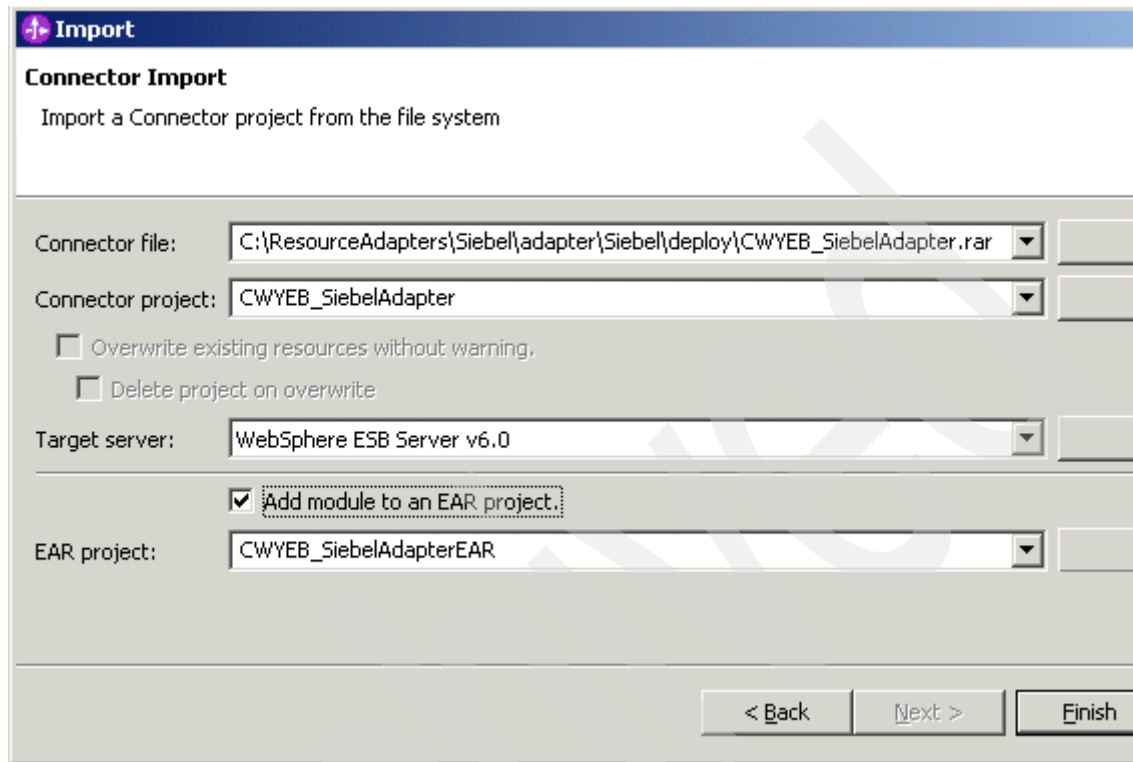


Figure 8-4 Browse the location of the RAR file

- c. Click **Yes** to allow the switch to the J2EE perspective.
- The next step is to copy the adapter-specific dependent JAR files to the connector module folder in the workspace and add them as an internal dependency so that when we create the EAR file for the project, this will also become part of it.

For our sample, we used Siebel Version 7.7 and thus required Siebel.jar and SiebelJI_enu.jar to be copied from the Siebel server for use as the dependent JAR files.

Right-click the connector project, **CWYEB_SiebelAdapter**, select **Properties** → **Java Build Path** → **Libraries** → **Add External Jars**, and add these two JAR files as external library files.
 - Switch back to the Business Integration perspective.

Create an import for the target Siebel server

You can use the enterprise service discovery wizard to create imports and exports for the Siebel system. The wizard uses the resource adapter to connect to the Siebel system. You can then query the Siebel system for metadata that describes the business service objects available. The result will be services generated for use in the mediation.

In this example we create an import for the Siebel system for use in our mediation. To do this:

1. Right-click in the white space of the Business Integration view, select **New** → **Other** → **Business Integration** → **Enterprise Service Discovery**, and click **Next**.
2. Select **IBM WebSphere Adapter for Siebel Business Applications** (Figure 8-5) and click **Next**.

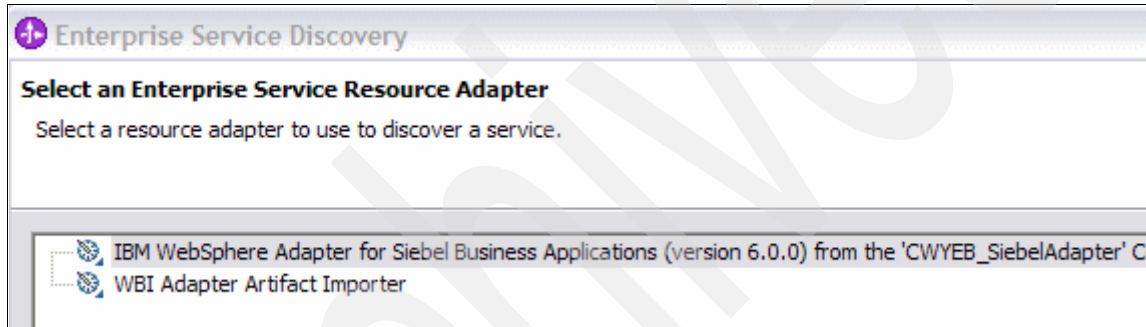


Figure 8-5 Select an enterprise service resource adapter

3. Fill the connection details of the server you wish to connect to. You will need to supply the connection information, user name, password, and the language code to connect to the Siebel application (Figure 8-6). Check check with your Siebel administrator for these values.

Figure 8-6 Configure settings for the enterprise service discovery

Click **Next**.

Tip: If you have problems connecting to the Siebel system, check the following possible reasons:

1. The dependent JAR files have not been added in the classpath.
2. The connect string you specified is wrong.
3. The Siebel application instance is not started on the server. Try pinging the machine so that you can make sure that you are able to connect to the machine. If the connect string uses a host name instead of an IP address, ensure that you can resolve the host name to an IP address.

4. Click **Edit Query** to enter the filter for the business objects you would like to retrieve. If you do not enter a filter the service will return all the business objects and it may take a long time, depending on the network traffic and the infrastructure.

Enterprise Service Discovery

Find and Discover Enterprise Services

Use "Edit Query" to create a query and press "Run Query" to discover matching objects on the enterprise system.

Query:

Business Service Filter=null

Edit Query...

Run Query

Objects discovered by query:

>> Add

Filter...

Figure 8-7 Edit query

5. Enter the name of the business service you want to retrieve. You can enter values using a wild card as well. For example, to see all the business services starting with A, enter A *.
Enter EAI Siebel Adapter as the filter value and click **OK** (Figure 8-8).

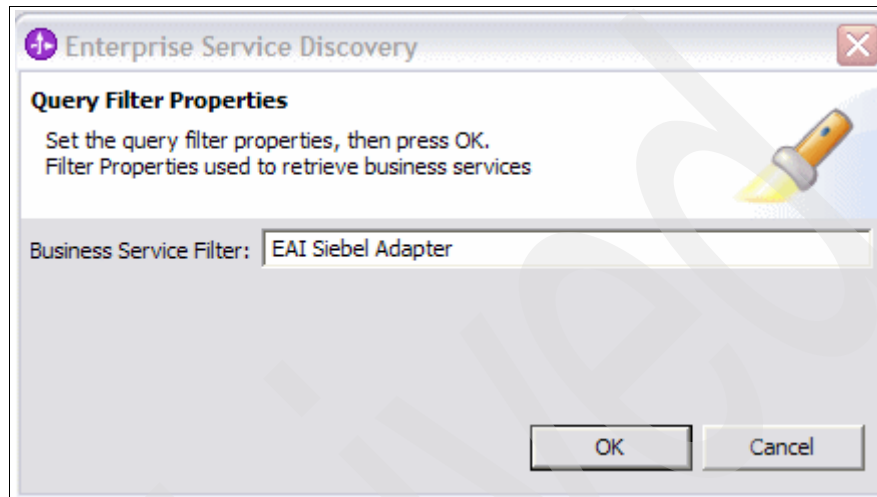


Figure 8-8 Business service filter

6. Click **Run Query** (Figure 8-7 on page 357). The enterprise service discovery will return the business services matching your filter and the supported operations it finds that match in the Siebel application.
- Click the plus sign (+) to get the list of supported operations.

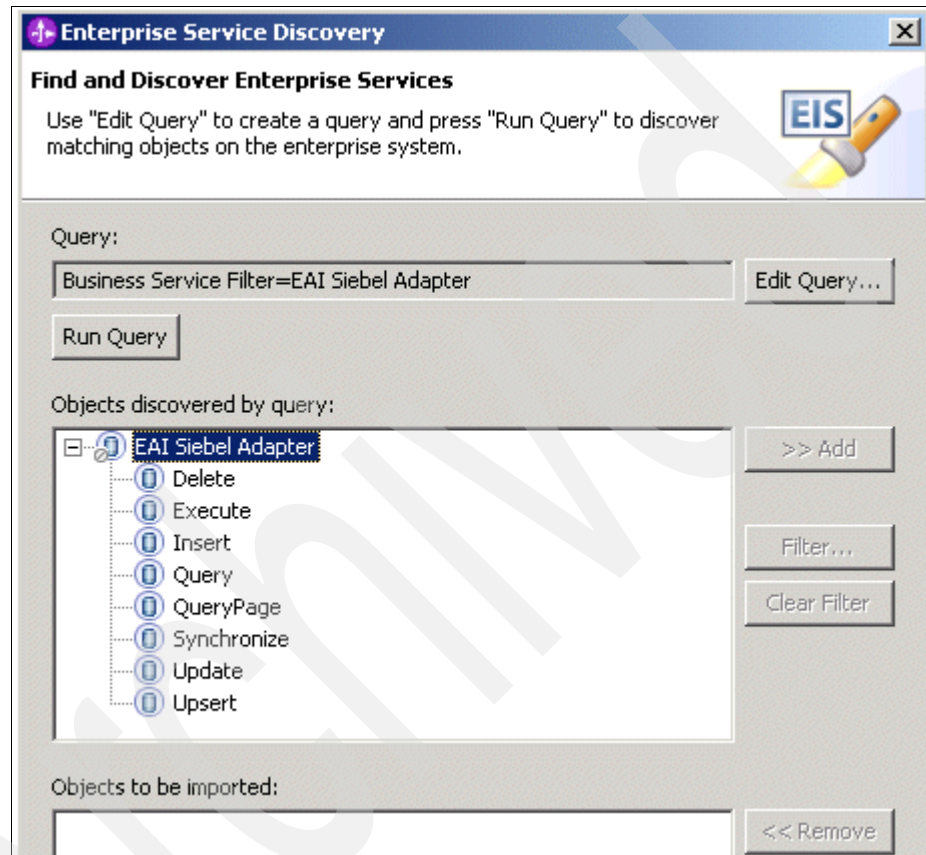


Figure 8-9 Query result

7. Select the operations you want to import. Operations are the activities you can perform on a business service. For example, to create an EAI Siebel Adapter object in the in Siebel application you can select **Insert**, to update a record select **Update**, and so on (Figure 8-10).

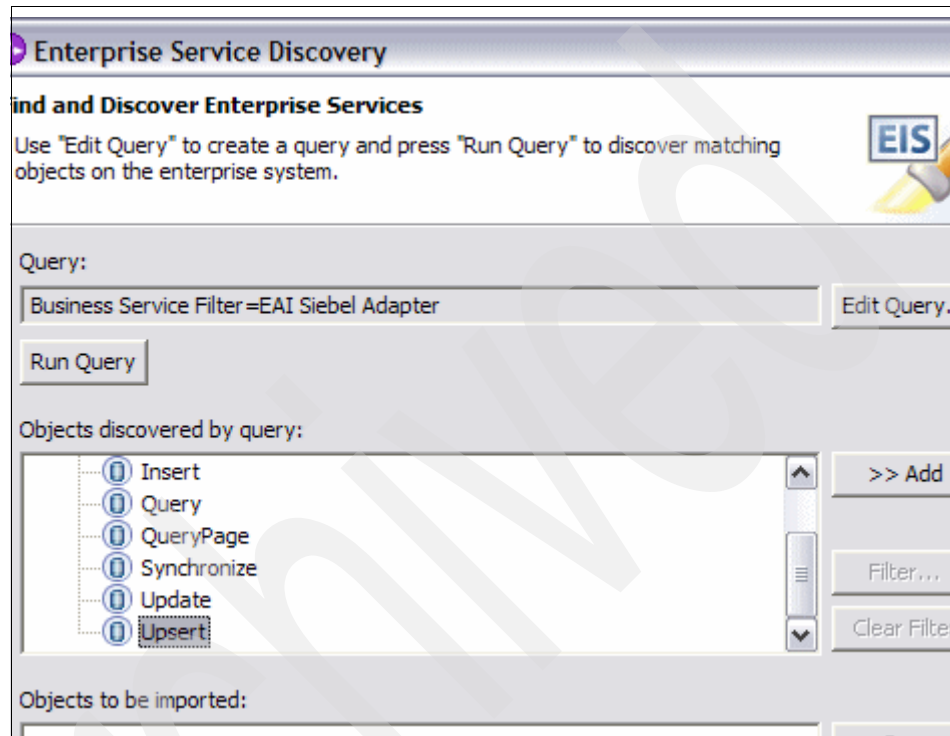


Figure 8-10 Select operations

- a. Select each operation you want to include and click **>>Add**. For the ITSOMart sample, you will only need the Upsert operation.

- b. Select **Account (PRM ANI)** from the drop-down menu in the SiebelMessage field (Figure 8-11). There is no need to give a value for the Event Method field since we are going to do an outbound operation (an import) here. Click **OK**.

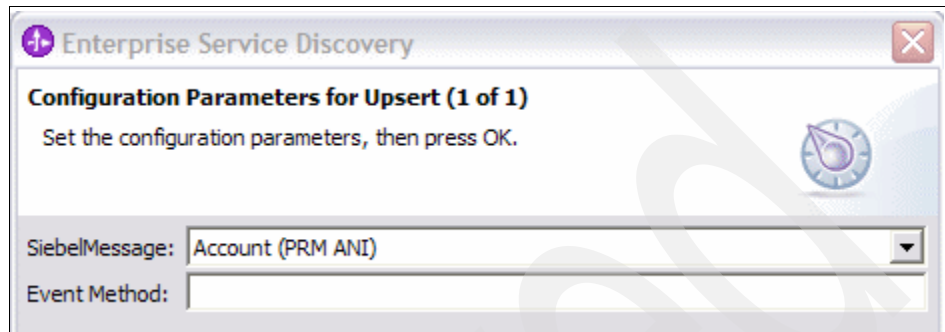


Figure 8-11 Select Siebel Message

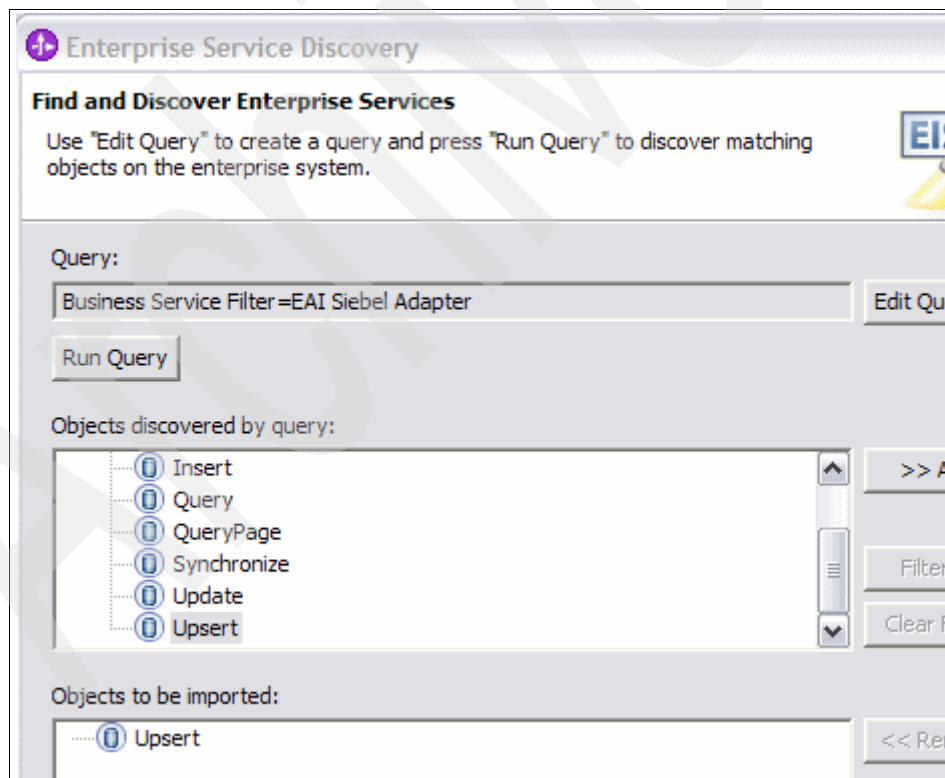
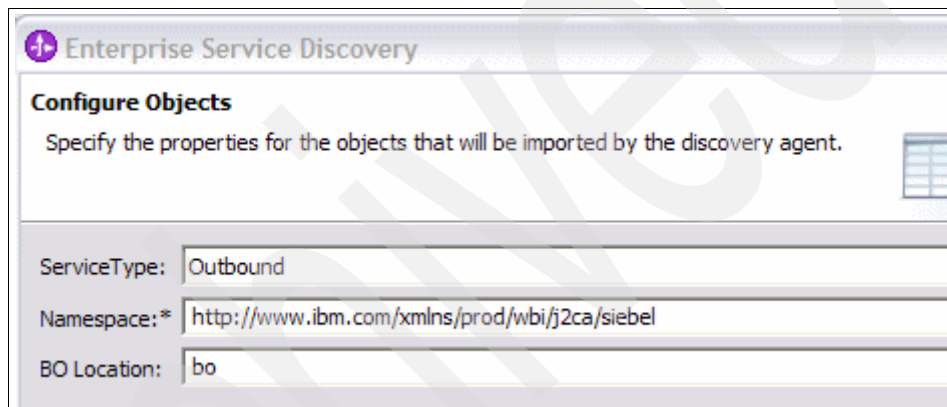


Figure 8-12 Adding objects

- c. Repeat this step for each operation. When done adding the operations, click **Next**.
8. Next configure the business objects based on the configuration of Siebel objects. To configure the object properties, enter the following (Figure 8-13):
 - Service type: Outbound
This will cause an import to be created.
 - Namespace: `http://www.ibm.com/xmlns/prod/wbi/j2ca/siebel`
 - BO location: `bo`
This is the folder in the mediation module where imported objects (.xsd files) will be stored.



The screenshot shows a window titled "Enterprise Service Discovery" with a sub-header "Configure Objects". Below the sub-header is a descriptive text: "Specify the properties for the objects that will be imported by the discovery agent." The main area contains three input fields: "ServiceType:" with the value "Outbound", "Namespace: *" with the value "http://www.ibm.com/xmlns/prod/wbi/j2ca/siebel", and "BO Location:" with the value "bo".

Figure 8-13 Configure service type, namespace, and BO location

Click **Next**.

9. In the next panel:
 - a. Select **ITSO_CRMMed** as the module.
 - b. Click the **Use discovered connection properties** radio button.

- c. Enter `<your_server_node>/CRM` as the J2C authentication data entry (Figure 8-14). This will point to the J2C authentication data defined in the application server that will contain the user ID and password needed to log in to Siebel. We will define this entry to the WebSphere runtime later.

The screenshot shows the 'Enterprise Service Discovery' window with the 'Generate Artifacts' tab selected. The window title is 'Enterprise Service Discovery'. Below the title bar, there's a section 'Generate Artifacts' with a floppy disk icon and the text 'Specify the properties for the artifacts that will be generated in your workspace.' The main area contains several input fields and buttons: 'Module:' with a dropdown menu showing 'ITSO_CRMMed' and a 'New...' button; 'Namespace:' with a text field containing 'http://ITSO_CRMMed/SiebelOutboundInterface' and a 'Browse...' button; a checked checkbox 'Use Default Namespace'; 'Folder:' with an empty text field and a 'Browse...' button; 'Name:' with a text field containing '* SiebelOutboundInterface'; 'Description:' with an empty text field; and an 'Edit operation names...' button. Below these fields, there's a checked checkbox 'Deploy connector with module' and a section 'Specify the connection properties which will be used to connect to the Enterprise Information System at run' with two radio buttons: 'Use connection properties specified on server' (unselected) and 'Use discovered connection properties' (selected). At the bottom, there's a 'J2C Authentication Data Entry:' label and a text field containing 'esbNode/CRM'.

Figure 8-14 Specify J2C authentication data entry

Scrolling further down on this panel, you will find settings that allow you to configure adapter-specific properties, such as log file name, log file size, trace file name, trace file size, and so on.

Click **Finish**.

Now you have successfully generated the artifacts needed to use the adapter in the CRM mediation.

IBM WebSphere Adapter for Flat Files

The next step is to install and import the IBM WebSphere Adapter for Flat Files.

1. Install the IBM WebSphere Adapter for Flat Files. The installation is similar to the installation for the IBM WebSphere Adapter for Siebel Business Applications (see “Installing WebSphere Adapters” on page 667).
2. Import the RAR files into the workspace. The procedure to this is similar to that outlined in “Create an interface for each EIS system” on page 352. In the case of the Flat File adapter you do not need to import any extra dependency JAR files. The RAR file name is CWYFF_FlatFile.rar.

Create an import for the flat file applications

This step uses the Enterprise Service Discovery wizard to create the imports associated with each Partner Application Adapter into the mediation module.

1. Right-click in the white space of the Business Integration view, select **New** → **Other** → **Business Integration** → **Enterprise Service Discovery**, and click **Next**.
2. Select the Flat File adapter and click **Next**.

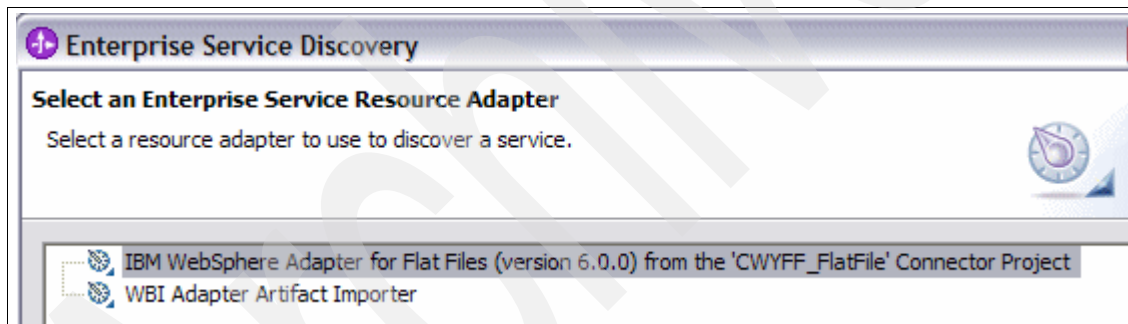
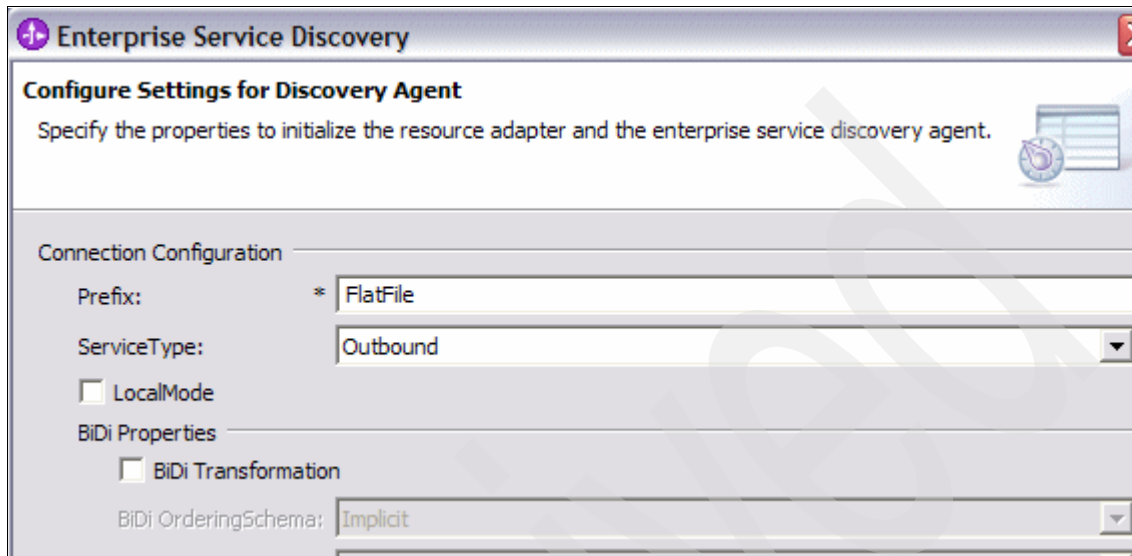


Figure 8-15 Enterprise Service Discovery

3. Keep the default values for the discovery agent settings (Figure 8-16). Click **Next**.



The image shows a Windows-style dialog box titled "Enterprise Service Discovery". Inside, there is a section titled "Configure Settings for Discovery Agent" with a subtitle "Specify the properties to initialize the resource adapter and the enterprise service discovery agent." and a small icon of a document with a magnifying glass. Below this, there are two expandable sections. The first, "Connection Configuration", is expanded and contains a "Prefix:" label followed by a text box containing "FlatFile" with an asterisk to its left, a "ServiceType:" label followed by a dropdown menu showing "Outbound", a checkbox for "LocalMode" which is unchecked, and a "BiDi Properties" section. The "BiDi Properties" section is also expanded and contains a checkbox for "BiDi Transformation" which is unchecked, and a "BiDi OrderingSchema:" label followed by a dropdown menu showing "Implicit".

Figure 8-16 Configure settings

4. Click **Run Query** to get a list of the objects. Select the Outbound object and click **Add** (Figure 8-17).

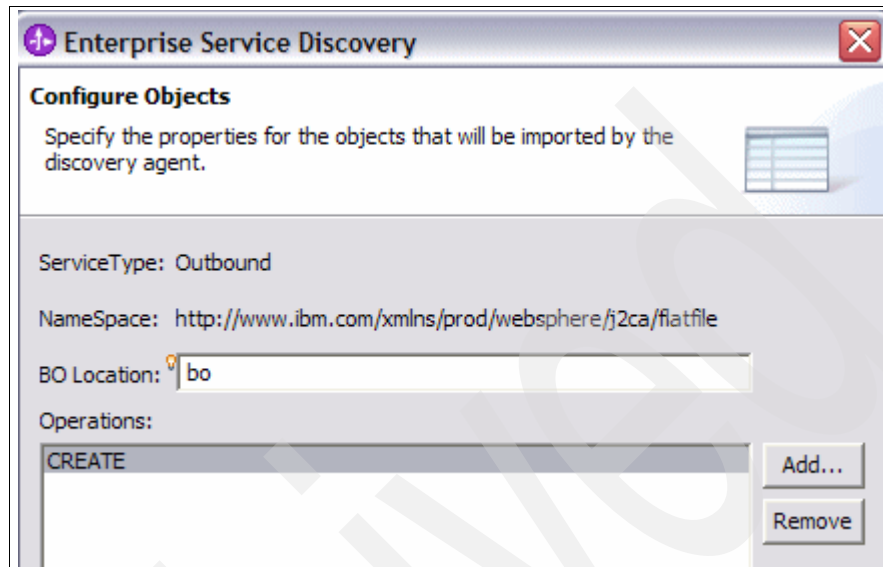


Figure 8-17 Enterprise service discovery - flat file

Click **Next**.

5. To configure the objects:
 - a. Remove all operations but CREATE.
 - b. Select **CREATE** and click **Add**.
 - c. Select **CREATE** from the pop-up window.
 - d. Click **OK**.

- e. Enter bo as the BO location (Figure 8-18). This is the folder in the mediation module where imported objects (.xsd files) will be stored.



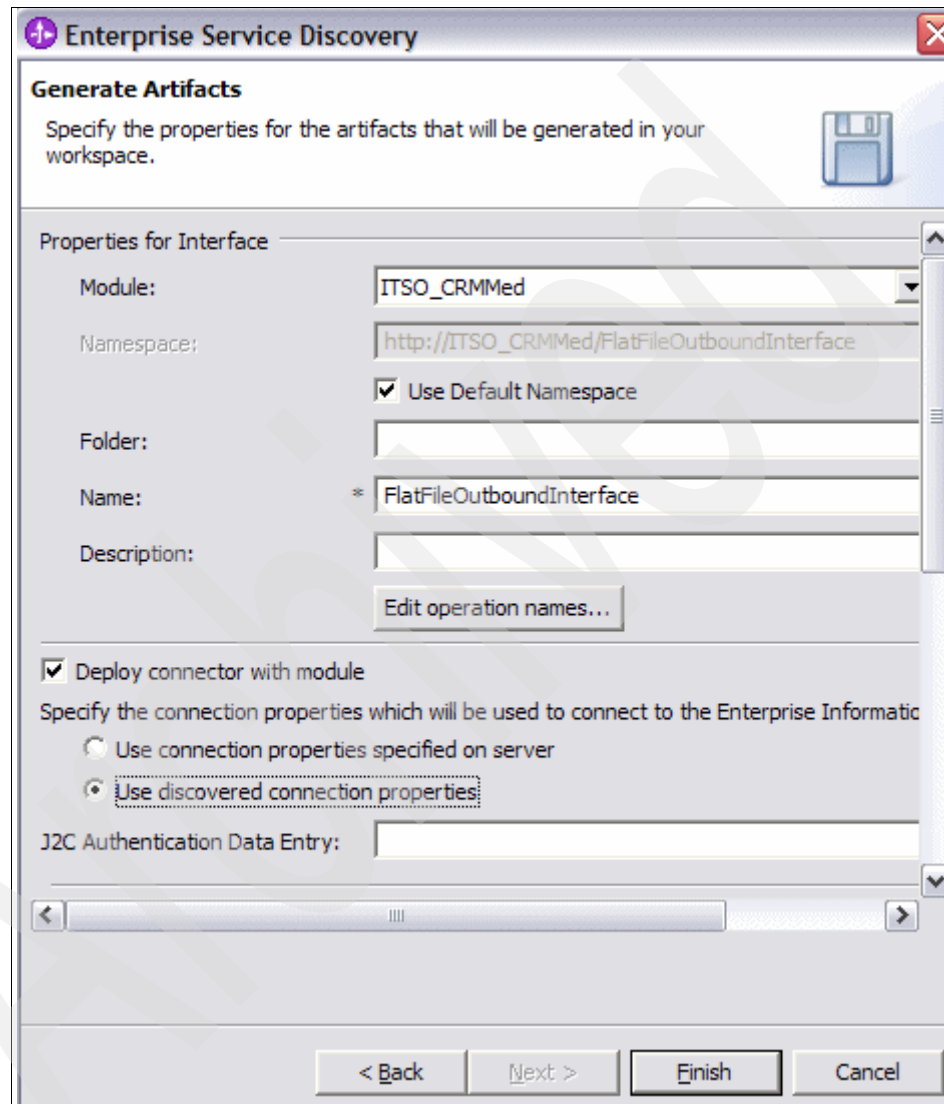
The image shows a Windows-style dialog box titled "Enterprise Service Discovery" with a close button (X) in the top right corner. Below the title bar is a section titled "Configure Objects" with a subtitle "Specify the properties for the objects that will be imported by the discovery agent." and a small icon of a document with a grid. The main area of the dialog contains the following fields and controls:

- ServiceType:** Outbound
- Namespace:** http://www.ibm.com/xmlns/prod/websphere/j2ca/flatfile
- BO Location:** A text input field containing the text "bo".
- Operations:** A list box containing the text "CREATE".
- Buttons:** "Add..." and "Remove" buttons are located to the right of the "Operations" list box.

Figure 8-18 Select operations

Click **Next**.

6. Select **ITSO_CRMMed** as the module, and select **Use discovered connection properties**. Take the defaults for the rest of the fields (Figure 8-19).



The image shows a Windows-style dialog box titled "Enterprise Service Discovery". Inside, there's a section "Generate Artifacts" with a floppy disk icon and the text "Specify the properties for the artifacts that will be generated in your workspace." Below this is a "Properties for Interface" section. It contains several fields: "Module:" with a dropdown menu showing "ITSO_CRMMed"; "Namespace:" with a text box containing "http://ITSO_CRMMed/FlatFileOutboundInterface" and a checked checkbox "Use Default Namespace"; "Folder:" with an empty text box; "Name:" with a text box containing "* FlatFileOutboundInterface"; and "Description:" with an empty text box. There is an "Edit operation names..." button below the description field. Below the "Properties for Interface" section is a checked checkbox "Deploy connector with module". Underneath it is the text "Specify the connection properties which will be used to connect to the Enterprise Information System". There are two radio buttons: "Use connection properties specified on server" (unselected) and "Use discovered connection properties" (selected). Below the radio buttons is a text box labeled "J2C Authentication Data Entry:". At the bottom of the dialog are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Figure 8-19 Generate Artifacts - flat file

Click **Finish**.

After running the enterprise service discovery wizard, you will see the new interfaces in ITSO_CRMMed (Figure 8-20).

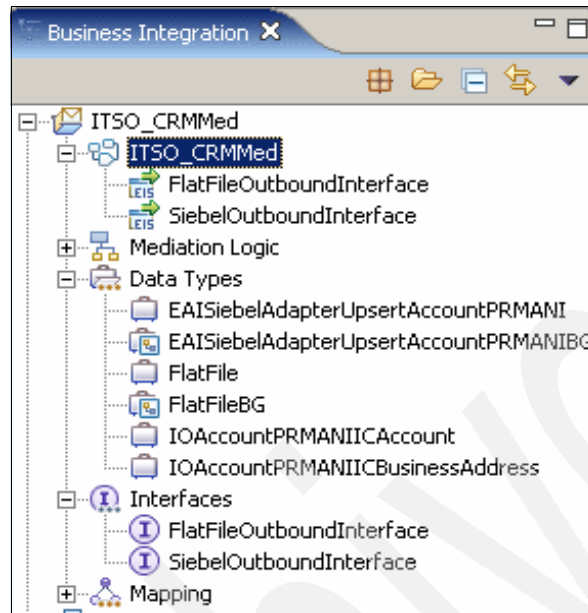


Figure 8-20 Mediation module contents at this point

8.2.4 Define the interface for the mediation

The next step in building the mediation is to create an interface that can be used to call the mediation.

Define the business objects

Each operation in the interface defines the data that can be passed in the form of inputs to and outputs from the component when the operation is invoked. This step creates the business objects that represent the data as it flows through the mediation.

Create two business objects:

- ▶ CustomerRegistrationRequest business object (Figure 8-21) with the following attributes:
 - customer of type Customer business object
 - creditRating of type string

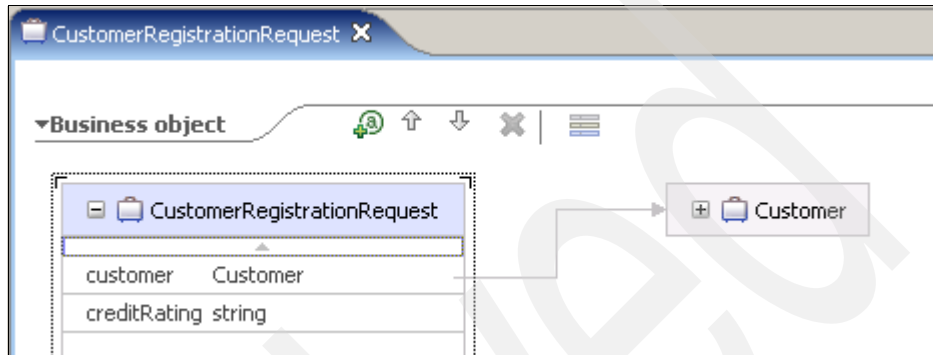


Figure 8-21 CustomerRegistrationRequest business object

- ▶ CustomerRegistrationResponse business object (Figure 8-22) with the following attributes:
 - registrationStatus of type string
 - statusDetails of type string
 - crmAccountId of type string

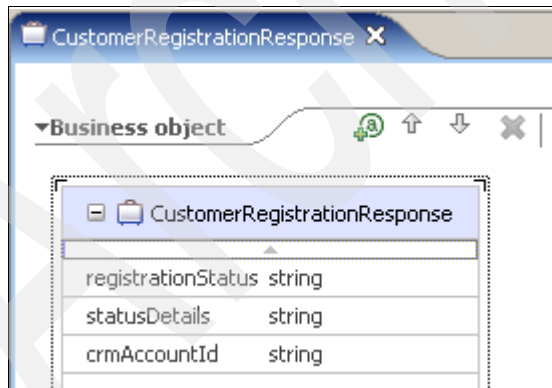


Figure 8-22 CustomerRegistrationResponse business object

Build the interface

In this scenario we define an interface that represents the functions provided by the mediation. It includes one operation, registerCustomer. The input to the

interface will be defined by the CustomerRegistrationRequest business object, while the output will be defined by the CustomerRegistrationResponse object.

The interface will look like Figure 8-23.

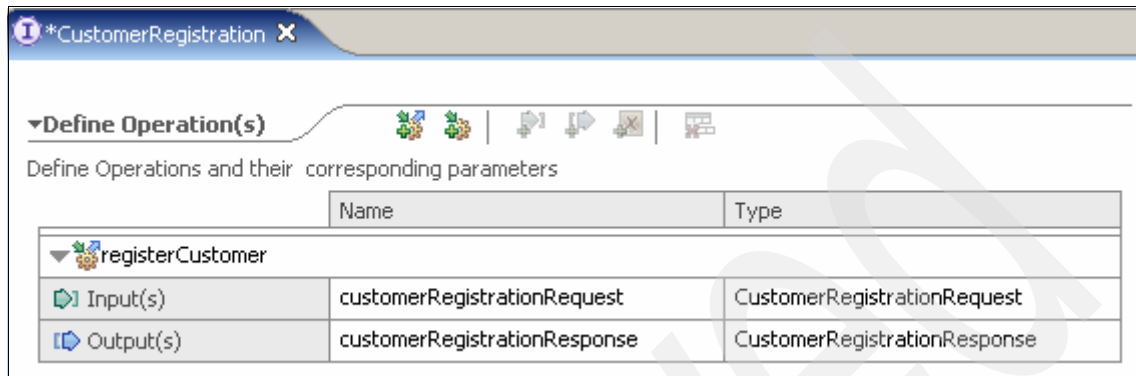


Figure 8-23 registerCustomer Interface is complete

To build the interface:

1. Create a new interface in ITSOMartLib. Name it CustomerRegistration.
2. Since our example is expecting a response to be returned, add a new Request Response operation. A new operation will be added to the interface with the default name of Operation1. Change this name to registerCustomer by typing over it.
3. Click the registerCustomer operation and click the Add Input icon. A new Input entry will appear on the canvas.
 - Enter customerRegistrationRequest in the Name field.
 - Select the **CustomerRegistrationRequest** business object as the type.
4. Click the **registerCustomer** operation and click the Add Output icon. A new Output entry will appear on the canvas.
 - Enter customerRegistrationResponse in the Name field.
 - Select the **CustomerRegistrationResponse** business object as the type.
5. Save and close the interface.

8.2.5 Add the components to the module assembly

Now that we have the interfaces required for the mediation, let us complete the module assembly. The module assembly we will build is shown in Figure 8-24.

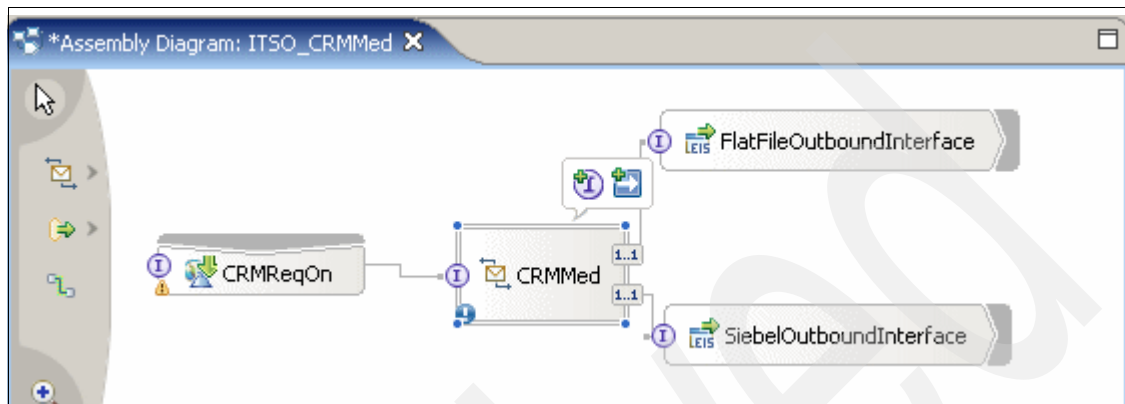


Figure 8-24 ITSO_CRMMed assembly diagram

To begin, switch to the Business Integration perspective and do the following:

1. Double-click the module assembly in the Business Integration view to open the assembly diagram. You will see the new interfaces for the Siebel and flat file systems and the mediation flow component. Note that both of the components for the adapters are for outbound operations (imports).

2. What is missing is the export that is used to call the mediation. To create an export, expand the contents of the ITSOMartLib library and expand the Interfaces folder.

Select the **CustomerRegistration** interface and drag and drop it in onto the assembly diagram. This will start a wizard.

- a. In the Component Creation window, select **Export with Web Service Binding**, and click **OK**.
- b. In the Binding File Generation prompt window, select **Yes**. We want the tool to automatically generate the wsdl file associated with the export.
- c. On the Select Transport window, select **soap/http** and click **OK**.

The export will be generated and placed in the assembly diagram.

3. Change the name of the export from Export1 to CRMReqOn.
4. Change the name of the mediation flow component from the default, Mediation1, to CRMMed.

5. Next wire the following components:
 - CRMReqOn to CRMMed
 - CRMMed to FlatFileOutboundInterface
 - CRMMed to SiebelOutboundInterface
6. Save the assembly diagram but do not close it. We will be using it in the next step.
7. Rebuild the project. After rebuilding the project, the only remaining warning at this point should be that shown in Figure 8-25. We will take care of that in the next step.

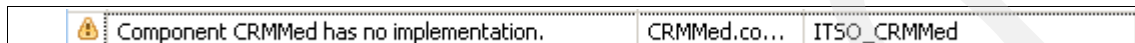


Figure 8-25 We must implement the Mediation Flow component *CRMMed*

8.2.6 Build the mediation flow

Next we generate the mediation flow component implementation (the mediation flow) and populate it with mediation primitives that will perform the mediation functions.

1. In the assembly diagram, double-click the **CRMMed** mediation flow component to start defining the mediation flow.
 - a. Click **Yes** in the Open window prompt. We want to implement the CRMMed component.
 - b. In the Generate Implementation window, create a new folder and name it impl. Click **OK**.
2. In the Operation connections window, you will see the interface and the two references used in this flow.

Click the **registerCustomer** operation under CustomerRegistration and drag it to the create operation under FlatFileOutboundInterfacePartner. This will create an operation connection between the two and generate the Callout node in the flow.

3. Create a second operation connection by clicking **registerCustomer** and dragging it to the upsertEASiebelAdapterUpsertAccountPRMANI operation.

The results should look like Figure 8-26.

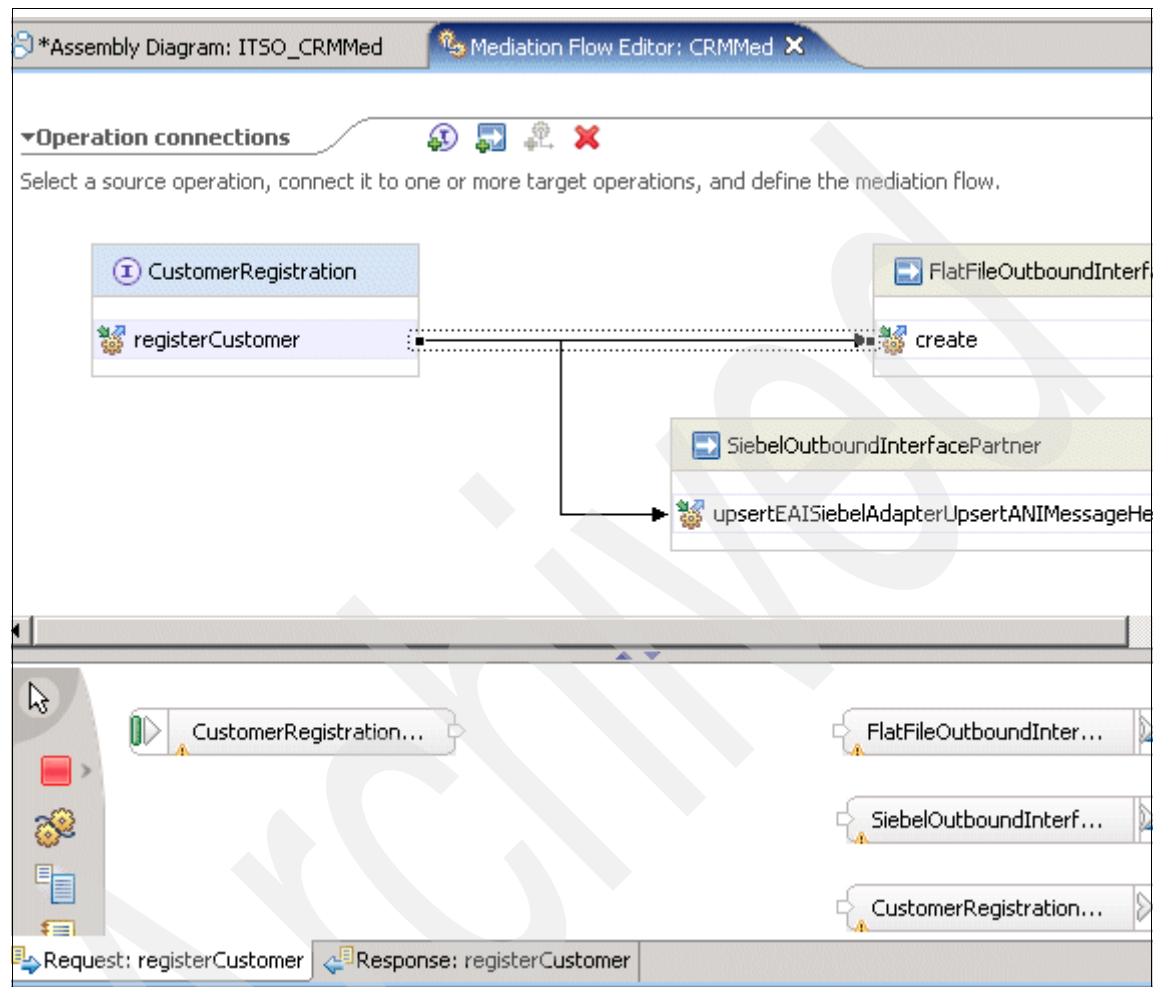


Figure 8-26 Mediation Flow Editor after the operations are wired

Note that there are two flows, a request flow labeled Request: registerCustomer and a response flow labeled Response: registerCustomer. This is because the registerCustomer operation on the CustomerRegistration interface is a request/response operation.

First we will build the request flow, and then the response flow.

Request flow

The request flow will do the following:

- ▶ Record the incoming message using a Message Logger primitive.
- ▶ Route the message based on the creditRating information provided in the Customer Registration request using a Message Filter primitive.
 - If the credit rating is gold, an XSL Transformation primitive is used to transform the message before sending it to the Siebel system. The response flow will set a status of success in the message.
 - If the credit rating is silver, an XSL Transformation primitive is used to transform the message. A Custom primitive is then used to create the file contents and set the file name and location before sending the request to the flat file adapter. The response flow will set a status of success.
 - If the credit rating is bronze, an XSL Transformation primitive is used to set the status to denied.

To build this flow, do the following:

1. Add a Message Logger primitive and change the name to **InLog**.
 - a. Wire the Customer Registration input node to the InLog Message Logger primitive. The results will look like Figure 8-27.

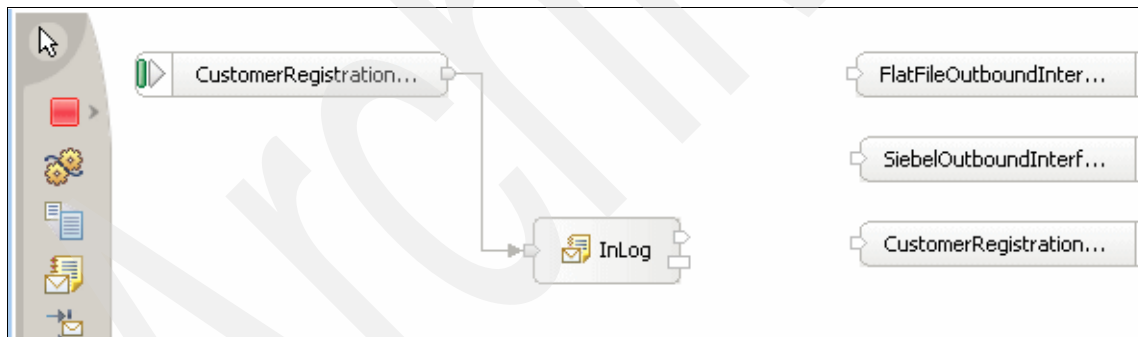


Figure 8-27 Wiring the Customer Registration input to the InLog Message Logger

- b. Select the **InLog** primitive. In the Details tab of the Properties view, change the Root field from /body to /.

We will do this for all the Message Logger primitives so that we will be able to see the entire contents of the SMO including context and headers. The Properties tab should look like Figure 8-28.

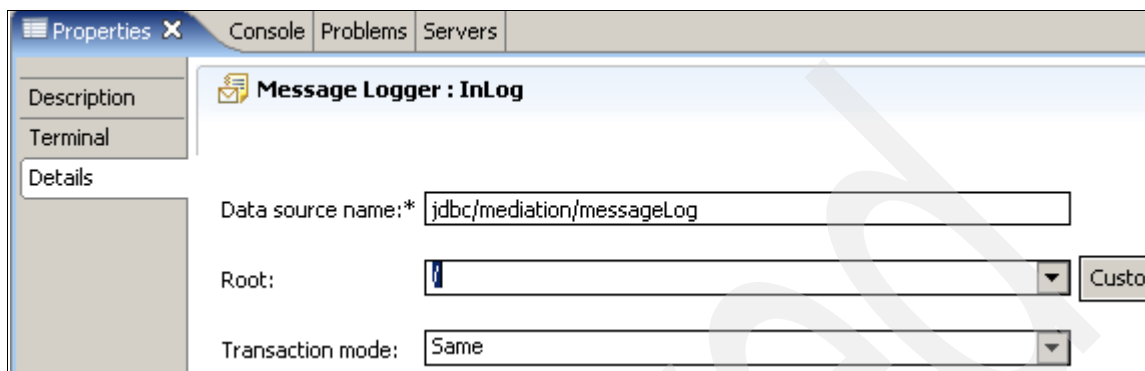


Figure 8-28 Setting the Root element to /

2. Next we add a Message Filter primitive to route the message based on the creditRating information provided in the Customer Registration request. There are three possible creditRating values that will be used to route the message: gold, silver, and bronze.

To create this filter, do the following:

- a. Add the Message Filter primitive to the mediation flow and change the name to CRMFilter.
- b. Add three terminals named Bronze, Silver, and Gold.

To add each new terminal, select the filter in the flow, right-click, and select **Add Output Terminal**.

- i. Enter the terminal name.
- ii. Select **match** for the terminal type (the only selection).
- iii. Click **OK**.

You can see the values used for the first terminal, Gold, in Figure 8-29.

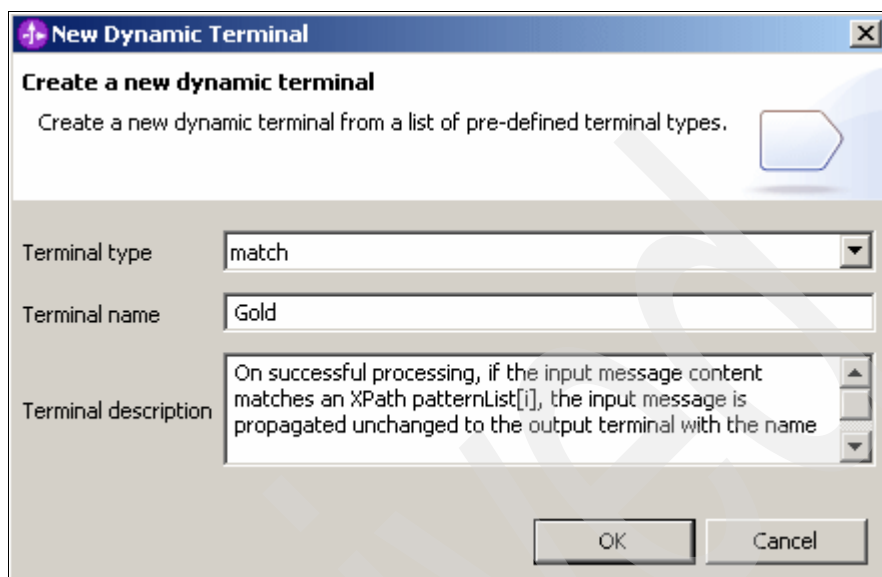


Figure 8-29 Adding an Output Terminal to the Filter

3. Wire the CustomerRegistration node to CRMFilter. Note that this is the second wire coming from CustomerRegistration (the first went to the Message Logger primitive).

We need the wire in place so the message type is set for the filter. This will populate the fields that we can choose from in the input message to use to route the message.
4. Select **CRMFilter** and switch to the Details tab of the Properties view.
 - a. Select **First** as the distribution mode.
 - b. To add the XPath associated with each terminal, click **Add** to the right of the Filters list. When the Add/Edit window comes up, click **Custom XPath**.
 - c. Expand the body of the message until you see the creditRating field, and select it.
 - d. Click in the Condition box. Select **self::node()**.
 - e. Click in the value box and enter "GOLD" (including the quotes).

The results should look like Figure 8-30.

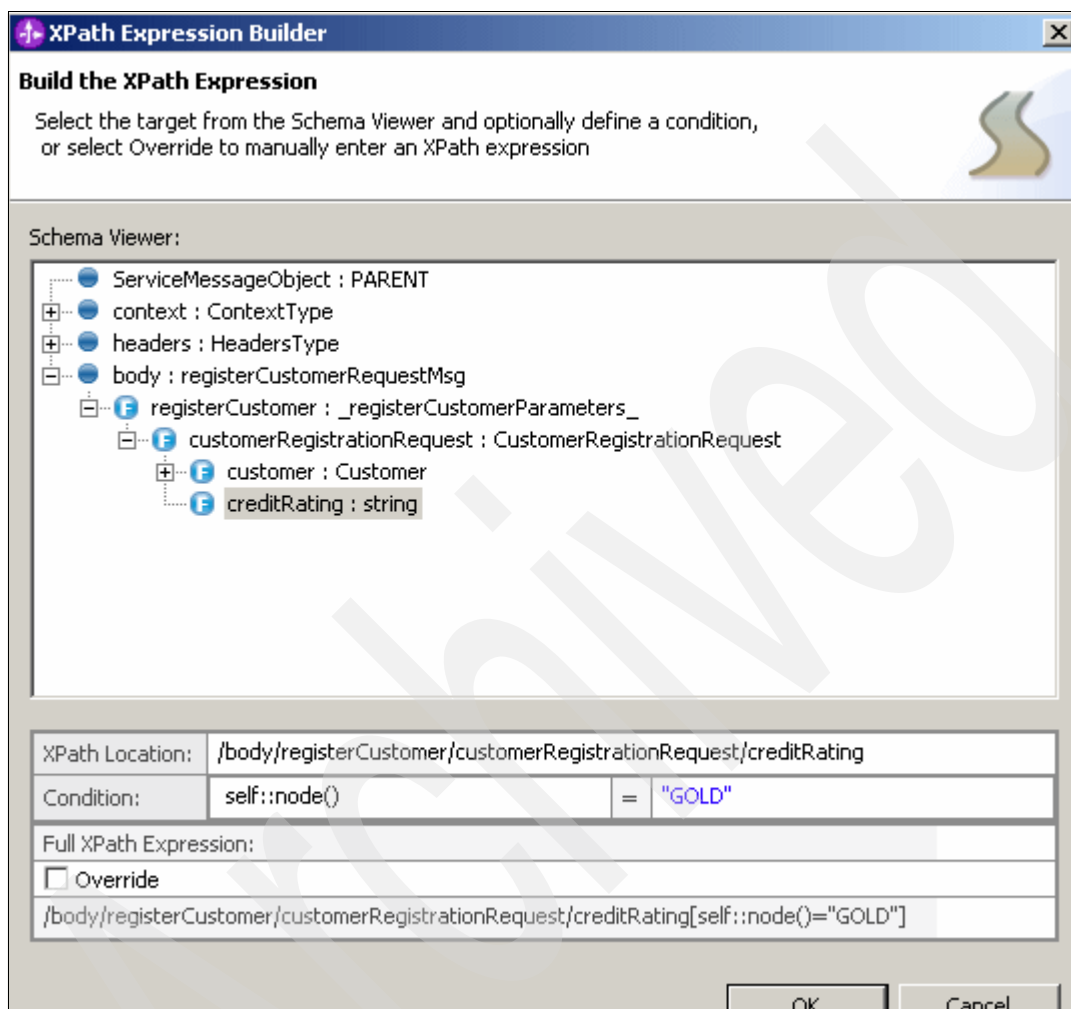


Figure 8-30 XPath Expression@ Builder window for the “GOLD” case

- f. Click **OK**. You should be back to the Add/Edit window.
 - g. Click **Finish**.
5. Repeat the same process for the Silver and Bronze terminals. Note that the names of the terminals are in mixed case, while the string values used in the XPath expressions are in uppercase.

When you are done, the Filters list should look like Figure 8-31.

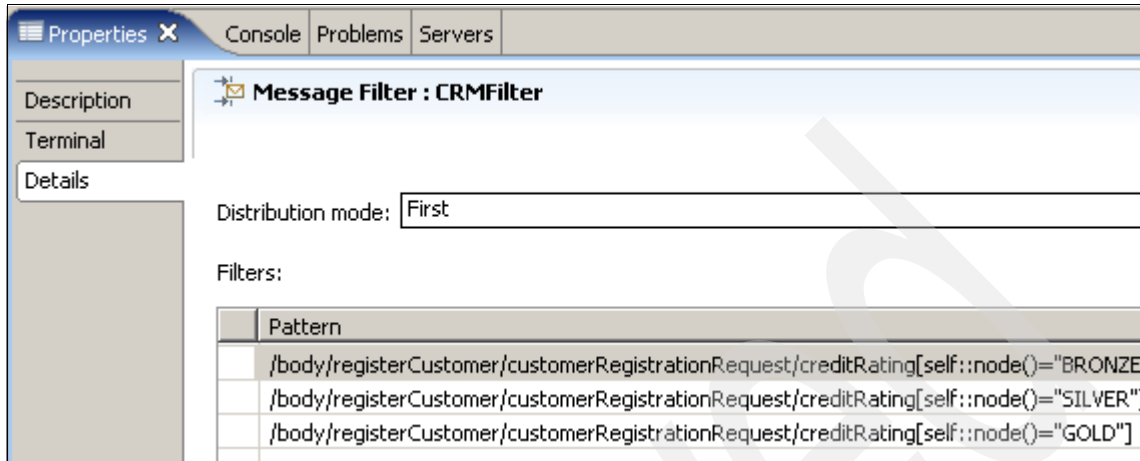


Figure 8-31 Filter Table for Filter CRMFilter

6. Save the mediation flow.

Request flow - Gold output terminal

Before we can wire the output terminals, we need to add the next primitive in each path. Let us start with the one associated with the “Gold” terminal.

1. We are going to use an XSL Transformation primitive to map the CustomerRegistration input message to the message required for the Siebel create operation.
 - a. Add an XSL Transformation primitive to the flow between the CRMFilter and the OutputBound Partner icons.
 - b. Change the name to XSL1.
 - c. Wire the “Gold” Terminal of the CRMFilter to the input terminal of the XSL1 transformation and the output terminal of XSL1 to the input terminal of the SiebelOutBoundInterfacePartner.

The mediation flow should look like Figure 8-32.

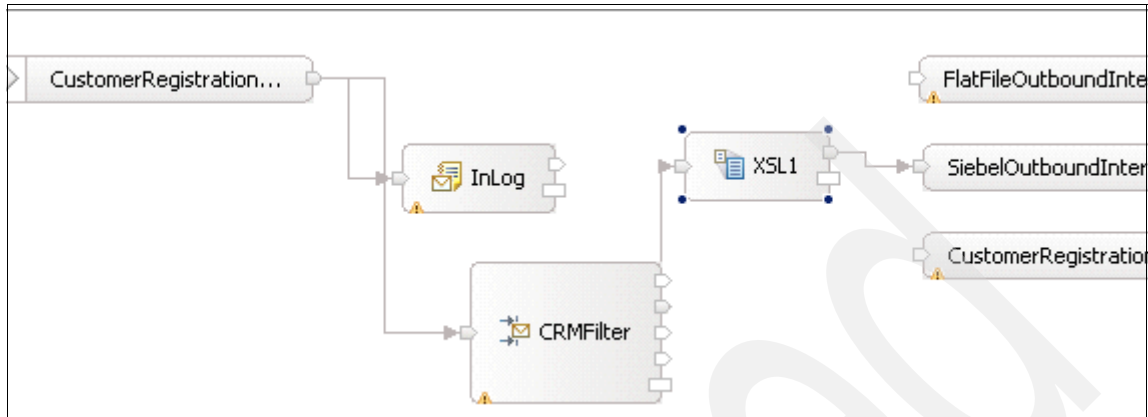


Figure 8-32 Mediation flow after wiring the XSL1 transformation to the Siebel Outbound input

- d. Save the mediation flow and assembly diagram and rebuild the project.
2. Next we create the XML mapping that defines how the XSL1 node transforms the CustomerRegistration input message to the message required for the Siebel create operation.
 - a. Select the **XSL1** node.
 - b. In the Details tab of the Properties view, click the **New** button to define the mapping. The new XSLT Mapping window will open.
 - c. Verify that the input message to the transformation is registerCustomerRequestMsg and that the output message from the transformation is upsertEASiebelAdapterUpsertAccountPRMANIRequest.
 - d. Click **Finish** to open the Mapping Editor.
 - e. Expand the body of each message until you can see all of the elements.

- f. First we map the `CurrencyCode` field in the `SiebelMessage`. We want to define an XSLT function for it, so select it, right-click, and select **Define XSLT Function**, as in Figure 8-33.

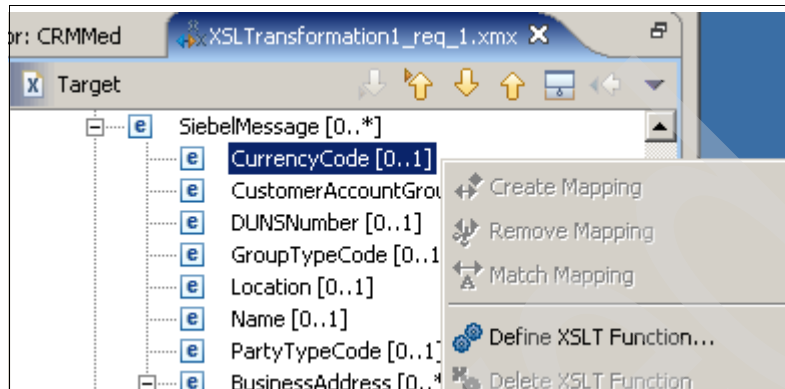


Figure 8-33 Mapping the `CurrencyCode` in the `SiebelMessage`

- g. On the XSLT Functions window, select **String** and click **Next**.
- h. In the next screen:
- Select **string** for the Function Name.
 - Click **Add** to add an input parameter.
 - Enter `'USD'`, using single quotes, and click **OK**.
 - Verify that you have the results shown in Figure 8-34 and click **Finish**.

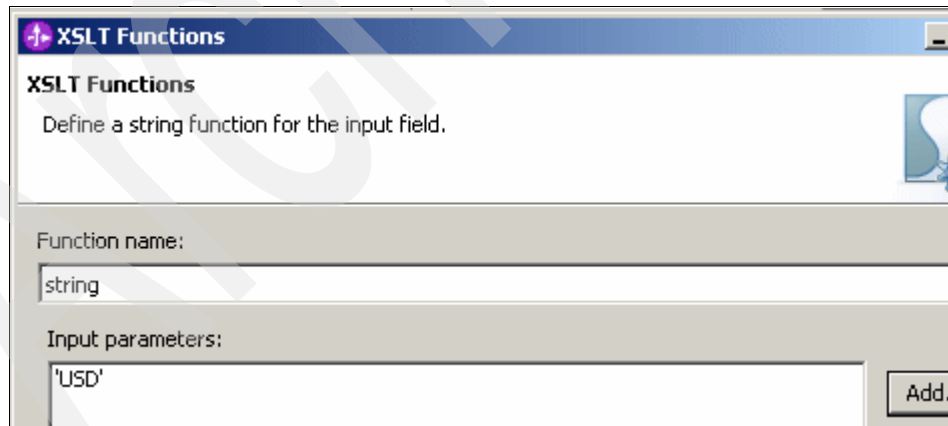


Figure 8-34 Mapping the `CurrencyCode` in the `SiebelMessage`

- i. Map the elements as shown in Table 8-1.

Table 8-1 XML mapping

customerRegistrationRequest	SiebelMessage
email	BusinessAddress/EmailAddress
billingAddress/name	BusinessAddress/AddressName
billingAddress/street	BusinessAddress/StreeAddress
billingAddress/city	BusinessAddress/City
billingAddress/city	Location
billingAddress/state	BusinessAddress/State
billingAddress/zipcode	BusinessAddress/PostalCode
billingAddress/country	BusinessAddress/Country
billingAddress/phone	BusinessAddress/PhoneNumber
firstName ¹	Name ¹
lastName ¹	Name ¹
¹ To map firstName and lastName to Name, click both source fields using the Ctrl key, then drag and drop them onto Name.	

- j. Add a blank in Name between firstName and lastName using the same technique illustrated in the CreditRatingMediation flow (page 294). The overview of the final map should look like Figure 8-35 on page 383.

Overview		
Target	Source	Applied Function/Grouping
[-] e body		
[-] e upsertEASiebelAdapterU...		
[-] e tns_1:upsertEASiebel...		
[-] e EASiebelAdapterU...		
[-] e SiebelMessage [...]		
[-] e CurrencyCod...		string
[-] e Location [0..1]	e city [0..1]	
[-] e Name [0..1]		concat
[-] e BusinessAdd...		
[-] e AddressN...	e name [0..1]	
[-] e City [0..1]	e city [0..1]	
[-] e County [...]	e country [0..1]	
[-] e EmailAddr...	e email	
[-] e PhoneNu...	e phone [0..1]	
[-] e PostalCo...	e zipcode [0..1]	
[-] e State [0..1]	e state [0..1]	
[-] e StreetAd...	e street [0..1]	

Figure 8-35 Overview of the map between the SiebelMessage and the customerRegistrationRequest

- k. Save and close the map.
3. Regenerate the XSL.
4. Save the mediation flow and the assembly diagram.

Response flow - Gold output terminal

At this point we have completed the request flow for the Gold terminal, that directs the message to the Siebel system. We are now going to do the response flow for it. The final flow will look like Figure 8-36.

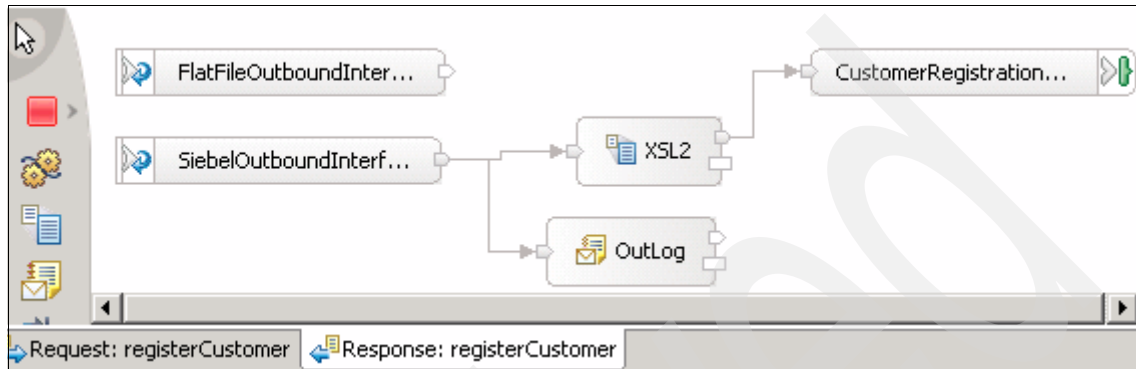


Figure 8-36 CRMMed response flow

The steps are:

1. Select the **Response flow** tab.
2. Add an XSL Transformation primitive between the SiebelOutboundInterfacePartner and the CustomerRegistration_registerCustomer_InputReponse.
 - a. Name it XSL2.
 - b. Wire the output terminal of the SiebelOutboundInterfacePartner to the input terminal of XSL2.
 - c. Wire the output terminal of XSL2 to the input terminal of CustomerRegistration_registerCustomer_InputReponse.
3. XSL2 will be implemented using a mapping that transforms the output message provided by the Siebel upsert operation to the Customer Registration Output message.
 - a. Create a new XML mapping for XSL2. Verify that the input message to the transformation is upsertEASiebelAdapterUpsertAccountPRMANIResponse and that the output message from the transformation is registerCustomerResponseMsg.
 - b. Map the statusDetails field in registerCustomerResponse to the constant 'SUCCESS'.
 - i. Right-click **statusDetails** and select **Define XSLT Function**.
 - ii. Select **String** and click **Next**.

In the next screen:

- i. Select **string** for the Function Name.
 - ii. Click **Add** to add an input parameter.
 - iii. Enter ‘SUCCESS’, using single quotes, and click **OK**.
 - iv. Click **Finish**.
- c. Map the elements as shown in Table 8-1 on page 382. To create the mapping, select the element under customerRegistrationRequest and drag it to the proper element in SiebelMessage.

Table 8-2 XML mapping

customerRegistrationRequest	SiebelMessage
PrimaryRowId	registrationStatus
PrimaryRowId	crmAccountId

The final map (in the Overview view) should look like Figure 8-37.

Target	Source	Applied Function/Grouping
smo	smo	
body		
registerCustomerResponse		
customerRegistrationR...		
registrationStatus [...]	PrimaryRowId [0..1]	
statusDetails [0..1]		string
crmAccountId [0..1]	PrimaryRowId [0..1]	

Figure 8-37 Final map overview for XSL2

- d. Save and close the map.
- 4. Regenerate the XSL.
- 5. Add a Message Logger primitive named OutLog and wire it to the Siebel outbound interface.
- 6. Save the mediation flow and assembly diagram. Rebuild the project.

Test the flow for Gold

At this point you should be able to test the CRMMed mediation and access the Siebel System.

Runtime requirements

The WebSphere Siebel adapter is automatically packaged as an EAR file in WebSphere Integration Developer.

1. In the Servers view, select the server, right-click, and use the **Add and remove projects** menu option to add CWYEB_SiebelAdapterEAR to the server. This will start the server, install the application, and start the application.
2. Create the J2C authentication data entry. The application server needs to be configured with the J2C authentication data entry specified in Figure 8-14 on page 363. Using the WebSphere administrative console, configure the CRM entry to contain the user ID and password to connect to the Siebel system.
 - a. Log in to the administrative console.
 - b. Click **Security** → **Global security**.
 - c. Expand **JAAS configuration** on the right panel and click **J2C Authentication data**.
 - d. Click **New** to create a new authentication alias called CRM with credentials that will allow you to log in into the Siebel system and click **OK**.
 - e. Save the changes.

Note that the new alias will be prefixed with the node name. For example, if your node is esbNode, the reference to the alias when you use the enterprise service discovery should be esbNode/CRM.

If you discover that the alias name used in the enterprise service discovery wizard does not match the alias defined in the application server, you can change it in the mediation module by doing the following:

- a. Open the module assembly.
 - b. Select the outbound interface for the Siebel adapter.
 - c. Select the **Bindings** tab in the Properties view.
 - d. Click the **Connection** tab and expand the **Authentication Properties**. You will see the alias in the J2C Authentication Data Entry field.
3. Update the JVM™ classpath for the server with the location of the dependent JAR files for the adapter.
 - a. In the WebSphere administrative console navigate to **Application Servers** → **Server1** → **Process Definition** → **Java Virtual Machine**.
 - b. Add the following to the classpath:
c:\cp\SiebelJI_enu.jar
c:\cp\Siebel.jar
 - c. Click **OK** and save the changes.

- d. Restart the server.
4. (Optional) For viewing the message logger entries, add MessageLogApp to the server (see “MessageLogApp application” on page 595).

Test the module

To do this:

1. Switch to the **Server** tab and start the WebSphere ESB Server v6.0.
2. Add ITSO_CRMMedApp to the server.
3. In the Business Integration view select the **ITSO_CRMMed** mediation module, right-click, and select **Test** → **Test Module**.

4. This will open the unit test window. Enter values for the input. Be sure to specify **GOLD** in the creditRating parameter and click **Continue** (Figure 8-38).

Initial request parameters

Name	Type	Value
<input type="checkbox"/> customerRe...	CustomerRegi...	
<input type="checkbox"/> customer	Customer	
accou...	string	1234
firstNa...	string	Gabriel
lastName	string	Shad
compa...	string	IBM
email	string	gshad@hotm...
passw...	string	gshad
<input type="checkbox"/> billing...	BillingAddress	
name	string	home
street	string	4265 Home Rd
city	string	Raleigh
state	string	NC
zipc...	string	27713
cou...	string	US
phone	string	(900)-898-7777
shippi...	ShippingAddr...	<null>
creditRating	string	GOLD

Figure 8-38 Testing the Mediation Module accessing the Siebel System

5. Select the **WebSphere ESB Server v6.0** as the Deployment Location when prompted and click **Finish**.

Attention: When starting the application in the WebSphere Integration Developer test environment, you may see a sequence of eight stack frames in the SystemOut.log, starting with the following:

```
[3/24/06 10:00:52:844 EST] 00000061 ConnectionFac E    J2CA0009E: An
exception occurred while trying to instantiate the
ManagedConnectionFactory class
com.ibm.j2ca.siebel.SiebelManagedConnectionFactory used by resource
TestSiebel/intf/SiebelOutboundInterface_CF :
java.lang.ClassNotFoundException:
com.ibm.j2ca.siebel.SiebelManagedConnectionFactory
    at
com.ibm.ws.classloader.CompoundClassLoader.findClass(CompoundClassLoader.
java(Compiled Code))
```

The module will still work successfully. This exception only seems to appear in the test environment. It does not appear in a WebSphere ESB stand-alone server.

After the mediation executes you should see output similar to that in Figure 8-39.

General Properties

Detailed Properties

Module: [ITSO_CRMMed](#)

Component: [CRMMed](#)

Interface: [CustomerRegistration](#)

Operation: [registerCustomer](#)

Return parameters:

Name	Type	Value
customerRegis...	CustomerRegist...	
registration...	String	1-YVW
statusDetails	String	SUCCESS
crmAccountId	String	1-YVW

Figure 8-39 Output message coming from the Siebel System

6. To make sure the mediation worked properly, you can check the following:
 - a. Look for a value in the registrationStatus and crmAccountId fields. These are returned by the Siebel server.
 - b. Log on to the Siebel system to make sure the account has been added.

- c. Check the messages logged by the Message Logger primitives. The input message logged by the InLog mediation primitive should look similar to Figure 8-40. If you use the MessageLogApp application shipped with the sample, you can do this without stopping the server.

```
<?xml version="1.0" encoding="UTF-8"?>
<smo:smo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:registration="http://ITSOMartLib/CustomerRegistration"
xmlns:smo="http://www.ibm.com/websphere/sibx/smo/v6.0.1">
  <context/>
  <headers>
    <SMOHeader>
      <MessageUUID>bf0600e1-0901-0000-0080-e0137fa86a21</MessageUUID>
      <Version>
        <Version>6</Version>
        <Release>0</Release>
        <Modification>1</Modification>
      </Version>
      <MessageType>Request</MessageType>
    </SMOHeader>
  </headers>
  <body xsi:type="registration:registerCustomerRequestMsg">
    <registerCustomer>
      <customerRegistrationRequest>
        <customer>
          <accountNo>1234</accountNo>
          <firstName>Gabriel</firstName>
          <lastName>Shad</lastName>
          <companyName>IBM</companyName>
          <email>gshad@hotmail.com</email>
          <password>gshad</password>
          <billingAddress>
            <name>home</name>
            <street>4265 Home Rd</street>
            <city>Raleigh</city>
            <state>NC</state>
            <zipcode>27713</zipcode>
            <country>US</country>
            <phone>(900)-898-7777</phone>
          </billingAddress>
        </customer>
        <creditRating>GOLD</creditRating>
      </customerRegistrationRequest>
    </registerCustomer>
  </body>
</smo:smo>
```

Figure 8-40 registerCustomerRequest message as logged by the InLog primitive

The output message that came from the Siebel System that is logged by the OutLog mediation primitive looks like Figure 8-41.

```
<?xml version="1.0" encoding="UTF-8"?>
<smo:smo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:interface="http://ITSO_CRMMed/intf/SiebelOutboundInterface"
xmlns:smo="http://www.ibm.com/websphere/sibx/smo/v6.0.1">
  <context/>
  <headers>
    <SMOHeader>
      <MessageUUID>82c529e1-0901-0000-0080-fabeb75ae853</MessageUUID>
      <Version>
        <Version>6</Version>
        <Release>0</Release>
        <Modification>1</Modification>
      </Version>
      <MessageType>Reply</MessageType>
    </SMOHeader>
  </headers>
  <body xsi:type="interface:upsertEASiebelAdapterUpsertAccountPRMANIResponse">
    <upsertEASiebelAdapterUpsertAccountPRMANIResponse>
      <interface:upsertEASiebelAdapterUpsertAccountPRMANIOutput>
        <changeSummary/>
        <properties/>
        <EASiebelAdapterUpsertAccountPRMANI>
          <ErrorCode>0x0</ErrorCode>
          <ErrorContextIntComp></ErrorContextIntComp>
          <ErrorContextSearchSpec></ErrorContextSearchSpec>
          <ErrorSymbol></ErrorSymbol>
          <OMErrorCode></OMErrorCode>
          <OMErrorSymbol></OMErrorSymbol>
          <PrimaryRowId>1-YVV</PrimaryRowId>
          <SiebelMessage>
            <CurrencyCode>USD</CurrencyCode>
            <Location>Raleigh</Location>
            <Name>Gabriel Shad</Name>
            <BusinessAddress/>
          </SiebelMessage>
        </EASiebelAdapterUpsertAccountPRMANI>
      </interface:upsertEASiebelAdapterUpsertAccountPRMANIOutput>
    </upsertEASiebelAdapterUpsertAccountPRMANIResponse>
  </body>
</smo:smo>
```

Figure 8-41 Response message from Siebel as logged by the Outlog primitive

Request flow - bronze output terminal

At this point the request flow looks like Figure 8-32 on page 380. The flow for the gold credit rating is complete. Now we are going to continue wiring the output terminals of CRMFilter to add the flow needed to handle a bronze credit rating. This flow will also serve as the default flow and be connected to the default terminal. In both cases, the message processing will be handled within the mediation and will not need to use any imports.

In the event that the mediation flow is passed a value of **bronze**, or an invalid value for the credit rating, the mediation will put “DENIED” in the statusDetail field of the CustomerRegistrationResponse Business Object associated with the CustomerRegistration_registerCustomer_InputResponse partner.

To build the mediation, do the following:

1. Add an XSL Transformation primitive between CRMFilter and the output bound partner icons:
 - a. Change the name to XSL3.
 - b. Wire the bronze terminal of CRMFilter to the input terminal of XSL3.
 - c. Wire the default terminal of CRMFilter to the input terminal of XSL3.
 - d. Wire the output terminal of XSL3 to the input terminal of the CustomerRegistration_registerCustomer_InputResponse partner.

The results are shown in Figure 8-42.

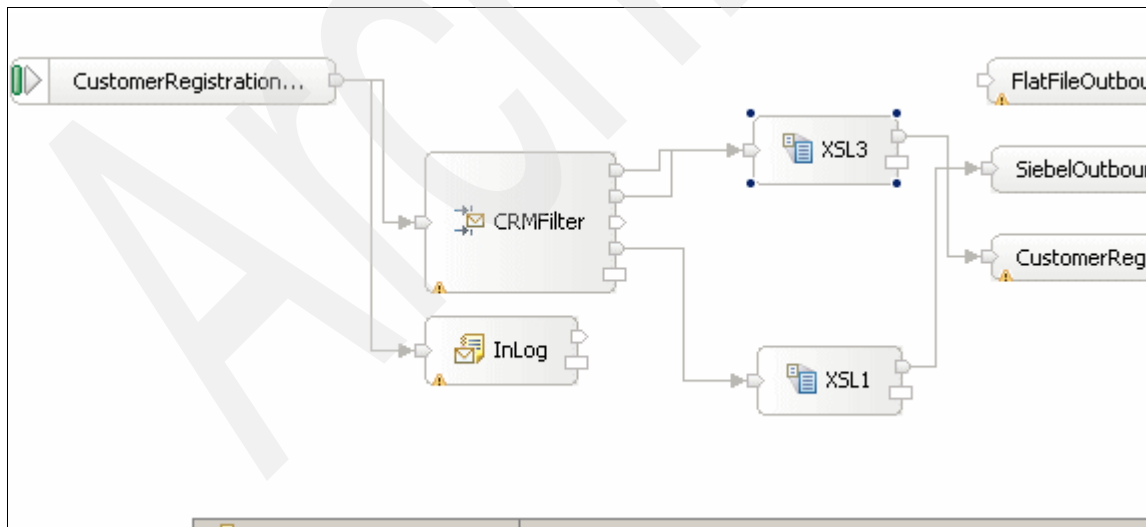


Figure 8-42 The Mediation Flow Editor after adding the XSL3 Transformation

2. Next we create the XML mapping that defines how the XSL3 node transforms the CustomerRegistration input message to a direct response to the requestor.
 - a. Select the XSL3 primitive.
 - b. In the Details tab of the Properties view, click **New** to create a new mapping.
 - c. The new XSLT Mapping window pops up. Verify that the input message is registerCustomerRequestMsg and that the output message is registerCustomerResponseMsg, and click **Finish**. This opens the Mapping Editor.
 - d. Map the statusDetails field in registerCustomerResponse to the constant 'DENIED' (using single quotes). This is done using the same procedure used in to map the statusDetails field to 'SUCCESS' in the Gold response flow (see page 384).
 - e. Map firstName and lastName to the registrationStatus field. Concatenate firstName and lastName using the same procedure used in defining the Name field (page 382).

The Overview view of the final mapping will look like Figure 8-43.

Target	Source	Applied Function/Group
smo	smo	
body		
registerCustomerResponse		
customerRegistrationResponse		
registrationStatus [0..1]		concat
statusDetails [0..1]		string

Figure 8-43 Overview of the XSL3 Transformation

- f. Save and close the map.
 - g. With XSL3 still selected, go to the Details tab of the Properties view and click **Regenerate XSL**. Click **OK** when the regeneration process completes.
3. Save the mediation flow and the assembly diagram. Rebuild the project.

At this point the mediation flow should look like Figure 8-44.

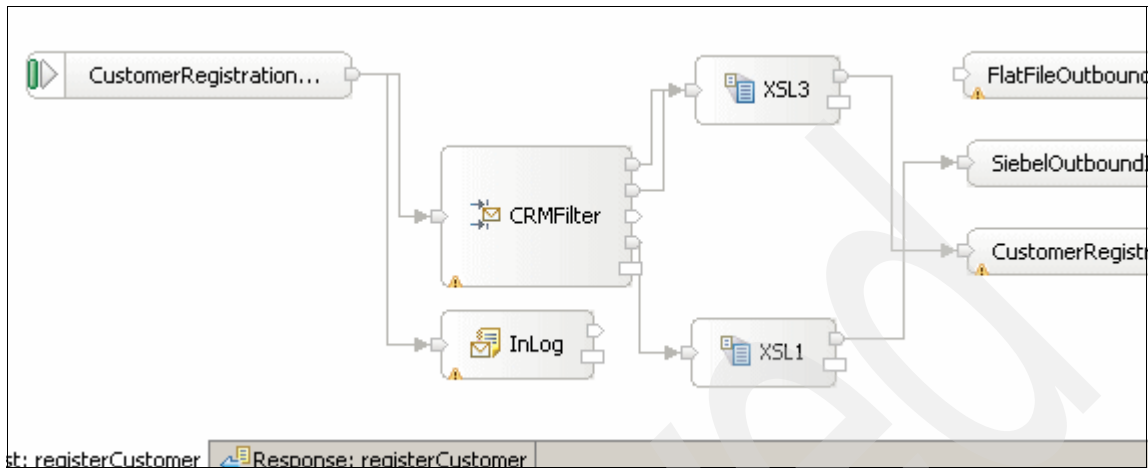


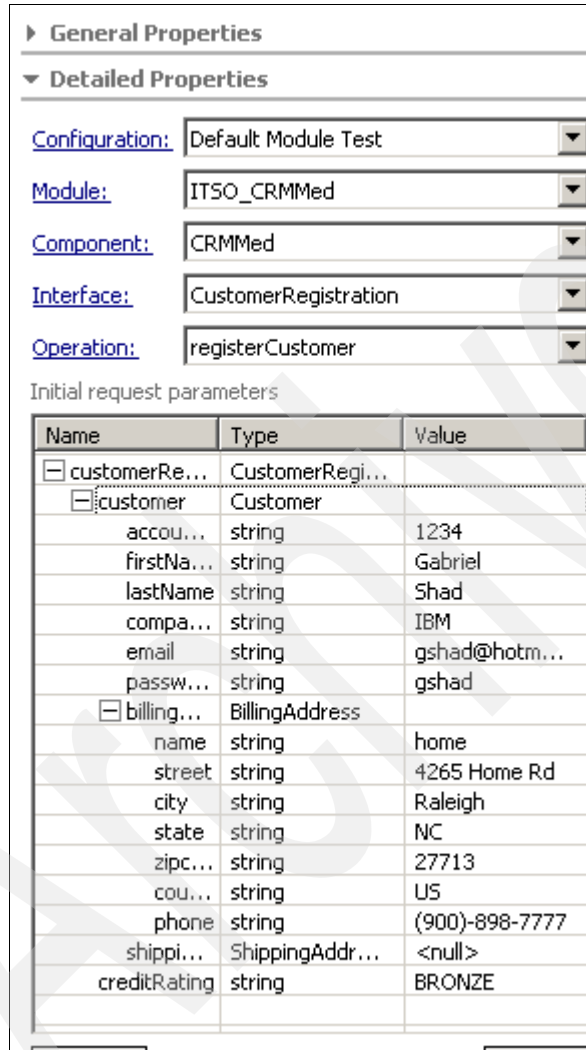
Figure 8-44 Mediation Flow Editor before adding the Custom Mediation for the “Silver” path

Test the flow for Bronze

At this point you should be able to test the Bronze path of the CRMMed mediation.

1. To test, switch to the Servers tab and start the WebSphere ESB Server v6.0.
2. In the Business Integration Explorer select the ITSO_CRMMed mediation module. Right-click and select **Test** → **Test Module**. This will open the unit test window for the mediation.

3. Enter values for the input and click **Continue** when done (see Figure 8-45). Entering the firstName, lastName, and creditRating (BRONZE) fields is sufficient. The mediation will automatically return a “DENIED” status when it sees the creditRating is BRONZE.



► General Properties

▼ Detailed Properties

Configuration: Default Module Test

Module: ITSO_CRMMed

Component: CRMMed

Interface: CustomerRegistration

Operation: registerCustomer

Initial request parameters

Name	Type	Value
customerRe...	CustomerRegi...	
customer	Customer	
accou...	string	1234
firstNa...	string	Gabriel
lastName	string	Shad
compa...	string	IBM
email	string	gshad@hotm...
passw...	string	gshad
billing...	BillingAddress	
name	string	home
street	string	4265 Home Rd
city	string	Raleigh
state	string	NC
zipc...	string	27713
cou...	string	US
phone	string	(900)-898-7777
shippi...	ShippingAddr...	<null>
creditRating	string	BRONZE

Figure 8-45 Testing the BRONZE path

4. Select **WebSphere ESB Server v6.0** as the Deployment Location when prompted and click **Finish**.

After execution you should see an output message that will look like Figure 8-46.

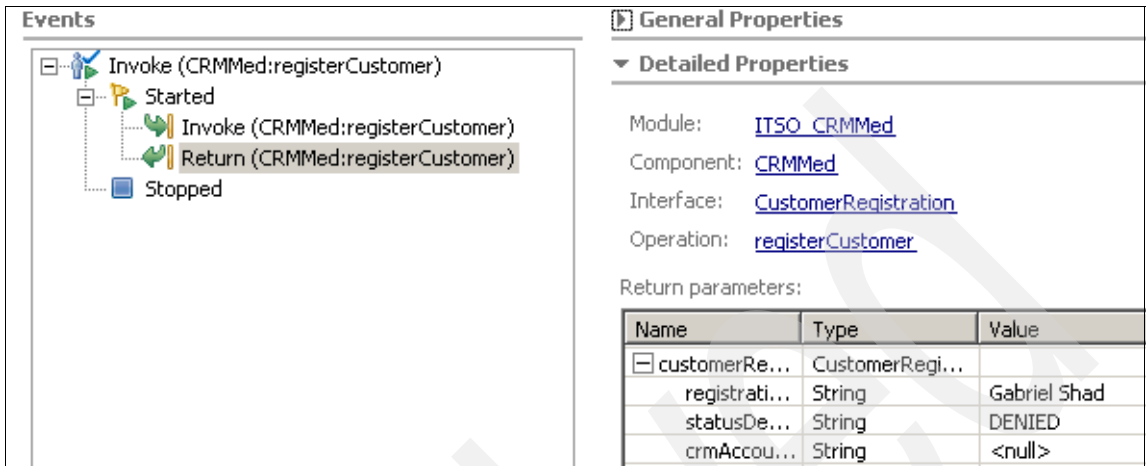


Figure 8-46 Reply coming from the CRMMed Meditation Module for the BRONZE path

Request flow - silver output terminal

Finally, we wire the last output terminal of CRMFilter to handle a credit rating of silver. In this case we want to send a request to the flat file adapter to create a file with a subset of the CustomerRegistrationRequestMsg data. That file can be used later for further offline or asynchronous processing. This last leg of the mediation flow will illustrate the use of a Custom mediation primitive.

1. Add a Custom primitive between CRMFilter and the OutputBound Partner icons.
 - a. Change the name to Custom1.
 - b. Wire the silver terminal of CRMFilter to the input terminal of Custom1.
 - c. Wire the output terminal of Custom1 to the input terminal of the FlatFileOutboundInterfacePartner_create_Callout partner.

The mediation flow should now look like Figure 8-47.

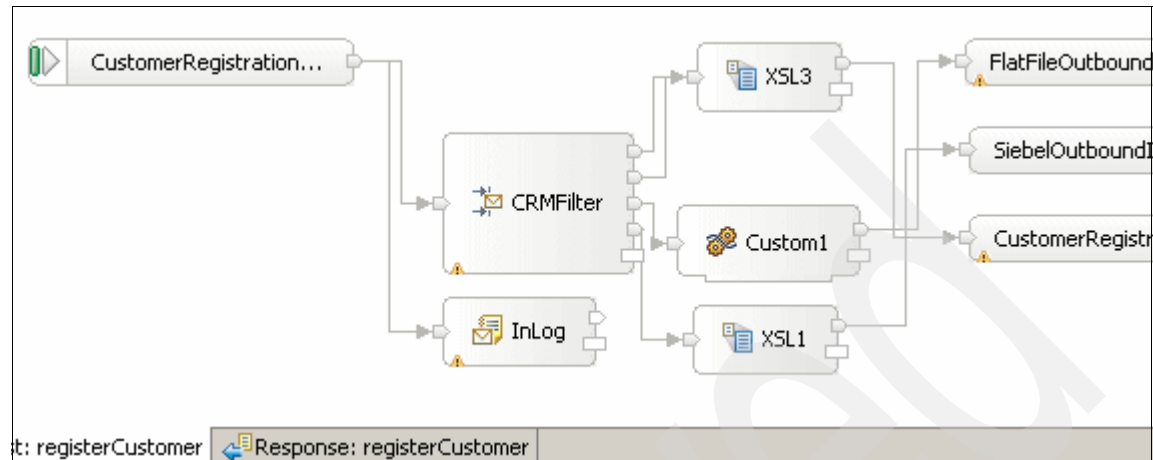


Figure 8-47 Mediation flow after adding the Custom mediation for the silver path

2. Custom primitives are used to do functions not covered by the other primitive types. It calls a Java SCA component, which you create or provide. The target SCA component must exist within the same mediation module as the Custom mediation primitive.

To implement Custom1, first select it in the mediation flow. In the Details tab of the Properties view you will see the following error:

“Service operation: cannot be empty”

- a. Click the **Define** button to start the Define Custom Mediation wizard.

- b. We do not have an existing interface for this primitive so leave the radio button “Create a new interface with implementation” checked and click **Next** (Figure 8-48).

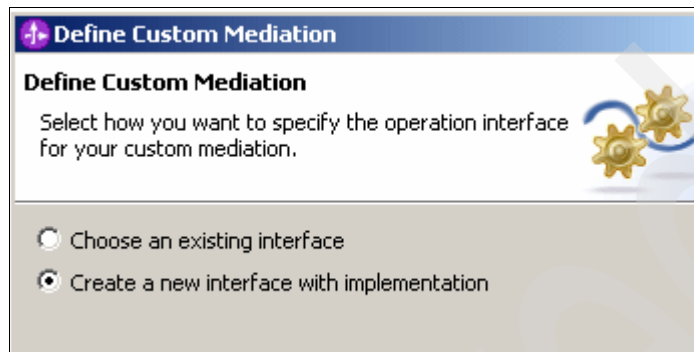


Figure 8-48 Define Custom Mediation wizard

- c. We are going to use the defaults for the message types. This includes specifying /body as the message root.

The body of the registerCustomerRequestMsg becomes the input to Custom1 and the body associated with createRequest becomes the return value of Custom1.

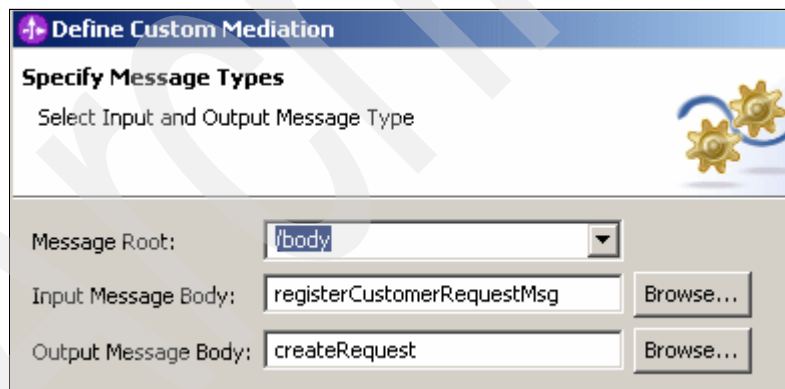
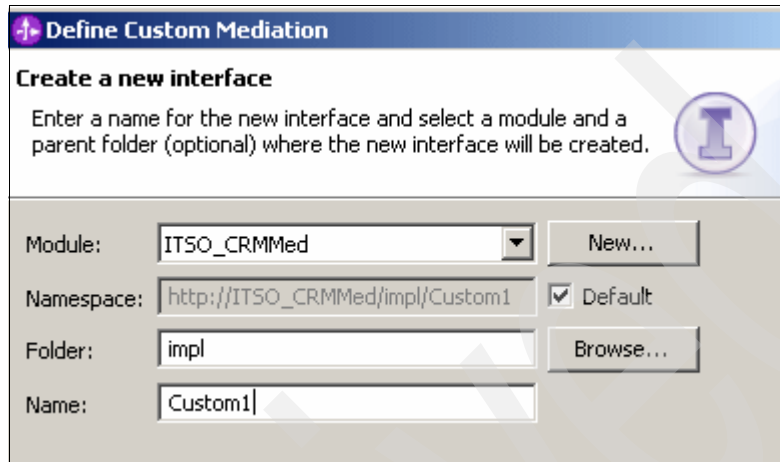


Figure 8-49 Specify message types

Click **Next**.

- d. The next panel (Figure 8-50) allows you to specify settings to create the new interface for the Custom primitive.

Enter `impl` for the folder name and change the Name to `Custom1`.



The image shows a dialog box titled "Define Custom Mediation". It has a sub-header "Create a new interface" and a descriptive text: "Enter a name for the new interface and select a module and a parent folder (optional) where the new interface will be created." To the right of the text is a purple circular icon with a white letter 'I'. Below the text are four input fields and two buttons. The "Module:" field is a dropdown menu with "ITSO_CRMMed" selected and a "New..." button to its right. The "Namespace:" field contains the text "http://ITSO_CRMMed/impl/Custom1" and has a checked "Default" checkbox to its right. The "Folder:" field contains the text "impl" and has a "Browse..." button to its right. The "Name:" field contains the text "Custom1".

Module:	ITSO_CRMMed	New...
Namespace:	http://ITSO_CRMMed/impl/Custom1	<input checked="" type="checkbox"/> Default
Folder:	impl	Browse...
Name:	Custom1	

Figure 8-50 Create a new interface

Click **Next**.

- e. The next panel helps you generate the implementation for the Custom primitive. We want the wizard to generate a default implementation, so use the default selection and click **Finish**.

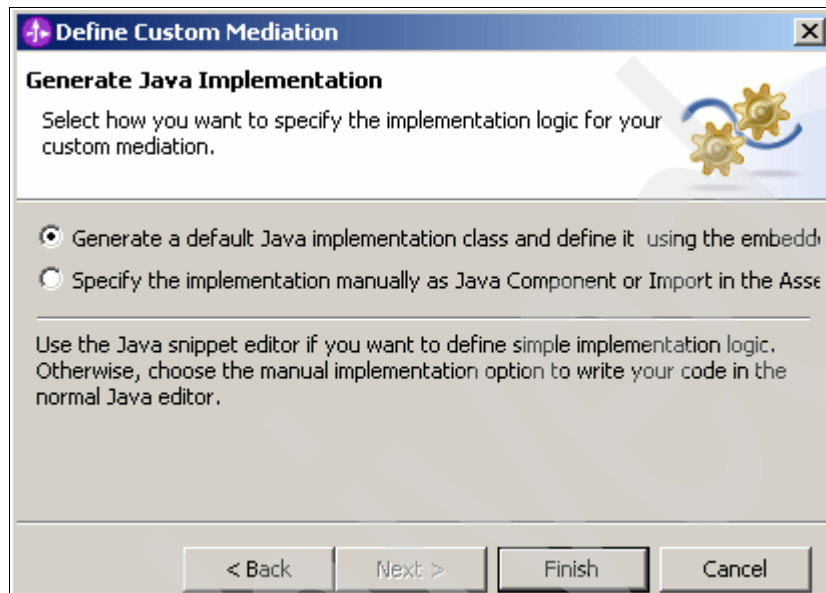


Figure 8-51 Generate a Java implementation

The Operation Connections view of the mediation flow should look like Figure 8-52.

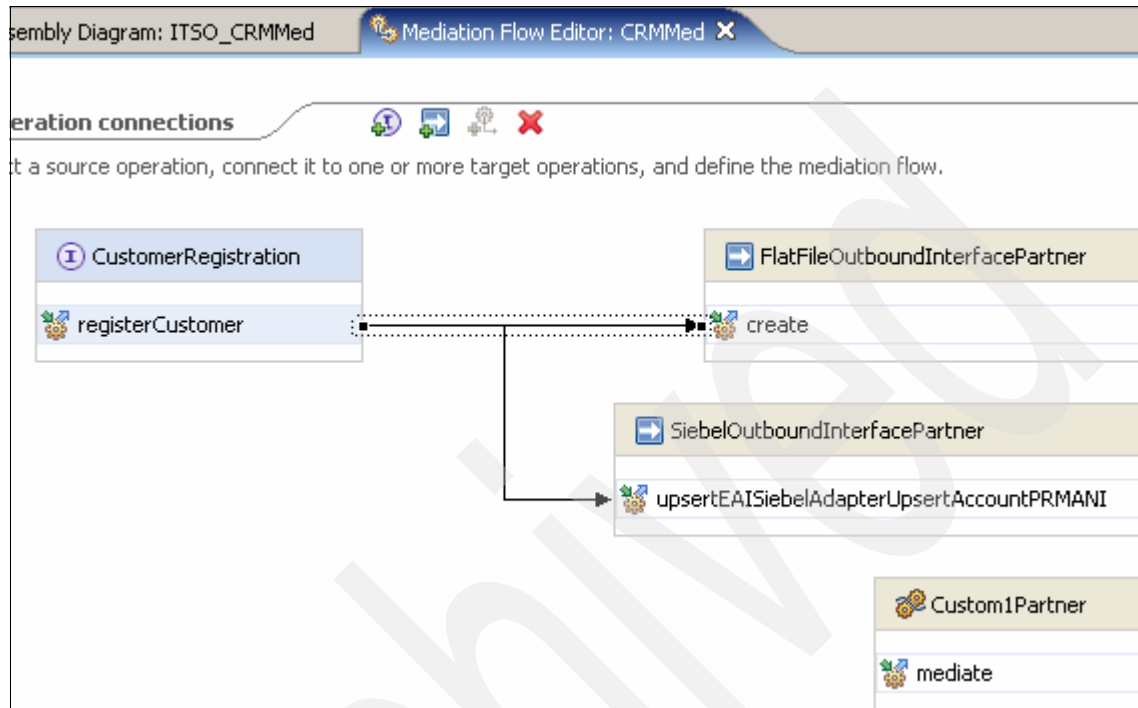


Figure 8-52 Adding a Custom mediation primitive

3. Save the mediation flow and assembly editor. Rebuild the project. You may see the error shown in Figure 8-53.

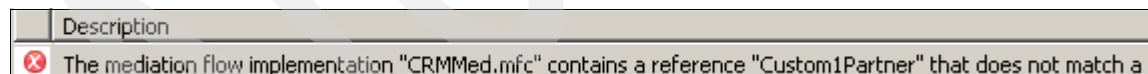


Figure 8-53 Possible error at this point

To eliminate this message we need to merge the implementation into the assembly diagram.

- a. In the assembly diagram, right-click the **CRMMed** mediation component and select **Merge Implementation** from the context menu.

The warning window shown in Figure 8-54 comes up.

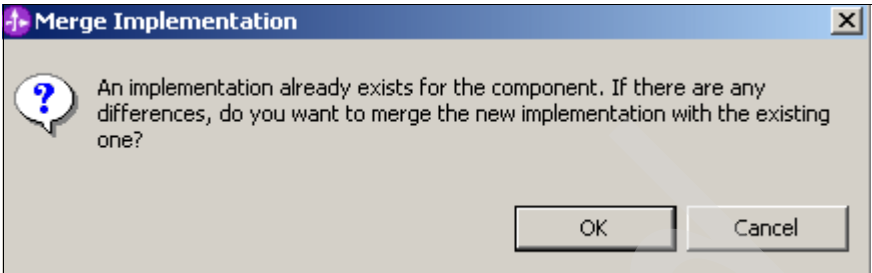


Figure 8-54 Merge Implementation warning

- b. Click **OK**.
- c. Make sure the Create Java Component check box for CustomPartner1 is checked and click **OK**.

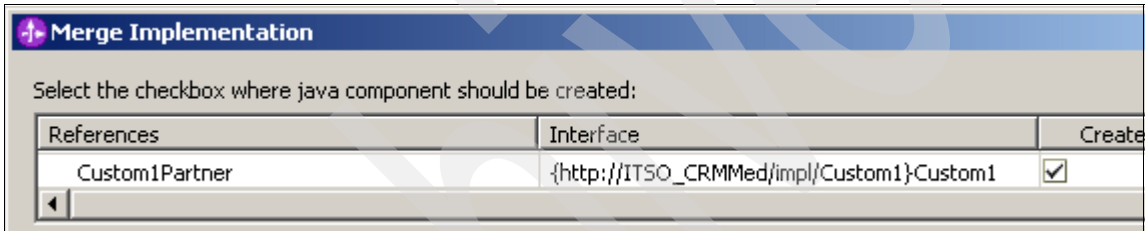


Figure 8-55 Merge Implementation - selecting the interface

- d. Save the assembly diagram and build the project. Any errors should go away. The assembly diagram should look like Figure 8-56.

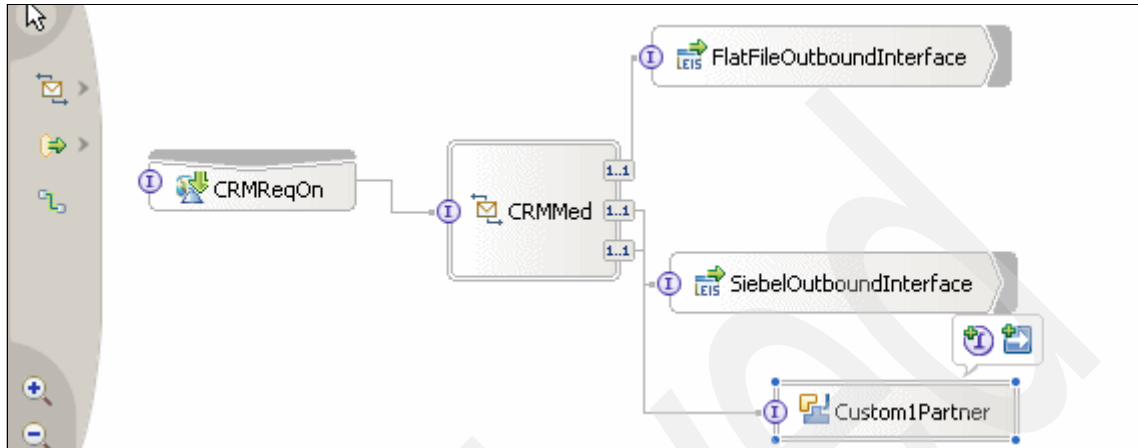


Figure 8-56 Assembly diagram after the new Custom interface is merged

4. We will now use the Visual Programming Editor to enter the Java code associated with Custom1. Select **Custom1** in the mediation flow (the bottom window) and in the Properties tab, select the **Implementation** tab.

5. Copy and paste the code shown in Figure 8-57 inside the implementation box, replacing the return null; statement that the tool placed there for us.

```
//Random x = new Random();
//int i = x.nextInt();
int i = 456;
//
// Obtain the data from the customerRegistrationRequest message
//
String fname =
a_type.getString("customerRegistrationRequest/customer/firstName");
String lname =
a_type.getString("customerRegistrationRequest/customer/lastName");
String city =
a_type.getString("customerRegistrationRequest/customer/billingAddress/city");

String contents = "Dear: " + fname + " " + lname + " " + city ;
com.ibm.websphere.bo.BOFactory factory =
(com.ibm.websphere.bo.BOFactory)
new com.ibm.websphere.sca.ServiceManager().locateService("com/ibm/websphere/bo/BOFactory");
//
// Create all the DataObject required to build the output DataObject
//
DataObject createOperation =
factory.createByElement("http://ITS0_CRMMed/FlatFileOutboundInterface", "create");
DataObject createInput = createOperation.createDataObject("createInput");

DataObject flatFile =
factory.create("http://www.ibm.com/xmlns/prod/websphere/j2ca/flatfile/flatfile",
"FlatFile");
//
// Make sure the following directory does exist
//
flatFile.setString("directoryPath", "C:\\\\FF");
flatFile.setString("fileName", fname + "_" + lname + "_" + i + ".txt");
flatFile.setBytes("inputBytes", contents.getBytes());
createInput.setDataObject("FlatFile", flatFile);

return createOperation;
```

Figure 8-57 Custom primitive code

The input interface to the Custom node is the customerRegistrationRequest input of the CustomerRegistration interface. This input type is the CustomerRegistrationRequest data object. This code takes the firstName, lastName, and city fields from this object as input. The output interface of the Custom node is createInput in FlatFileOutboundInterface.

Tip: It is difficult to build the implementation code without being able to see the SMOs used for input and output. To view these, we added a temporary XSL Transformation primitive. We wired the new primitive the same as the Custom node, with the input wired to the Silver terminal on the Filter and the output terminal to the FlatFileOutboundInterface. Then we used the XML mapper to view the input and output message structure. When we were done, we simply deleted the temporary XSL Transformation primitive.

Figure 8-58 shows the SMO associated with customerRegistrationRequest. From this, we built the XPath statements used in the getString calls above.

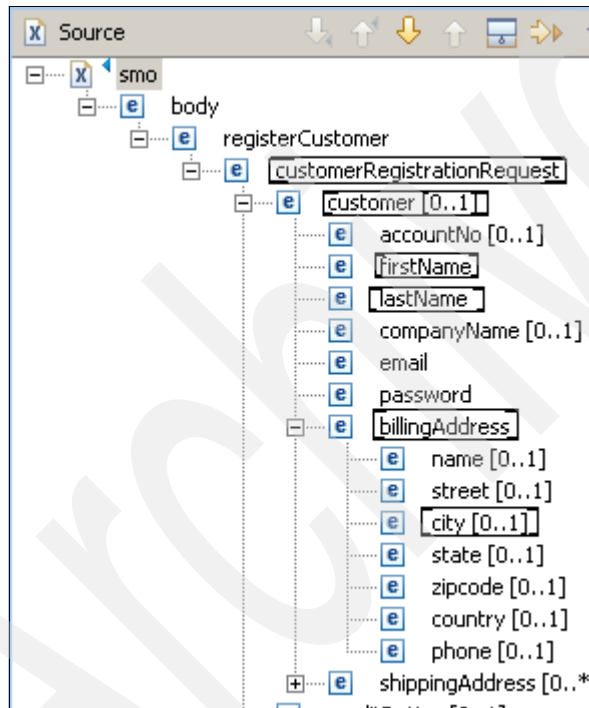


Figure 8-58 Input parameter

A list of all possible XPath strings available from the input parameter can be seen in Figure 8-59.

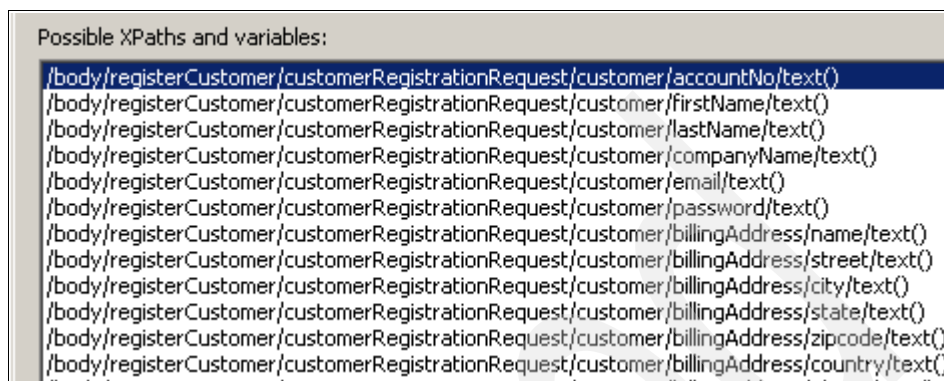


Figure 8-59 Possible XPath strings and variables

Figure 8-60 shows the output SMO.

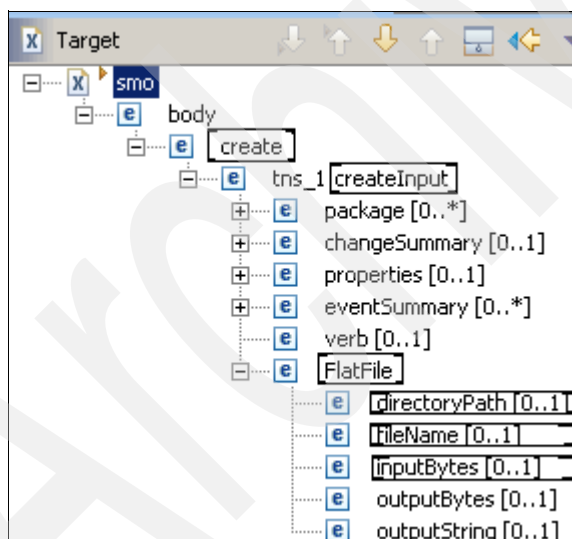


Figure 8-60 Output parameter

The contents of the Implementation page should look like Figure 8-61.

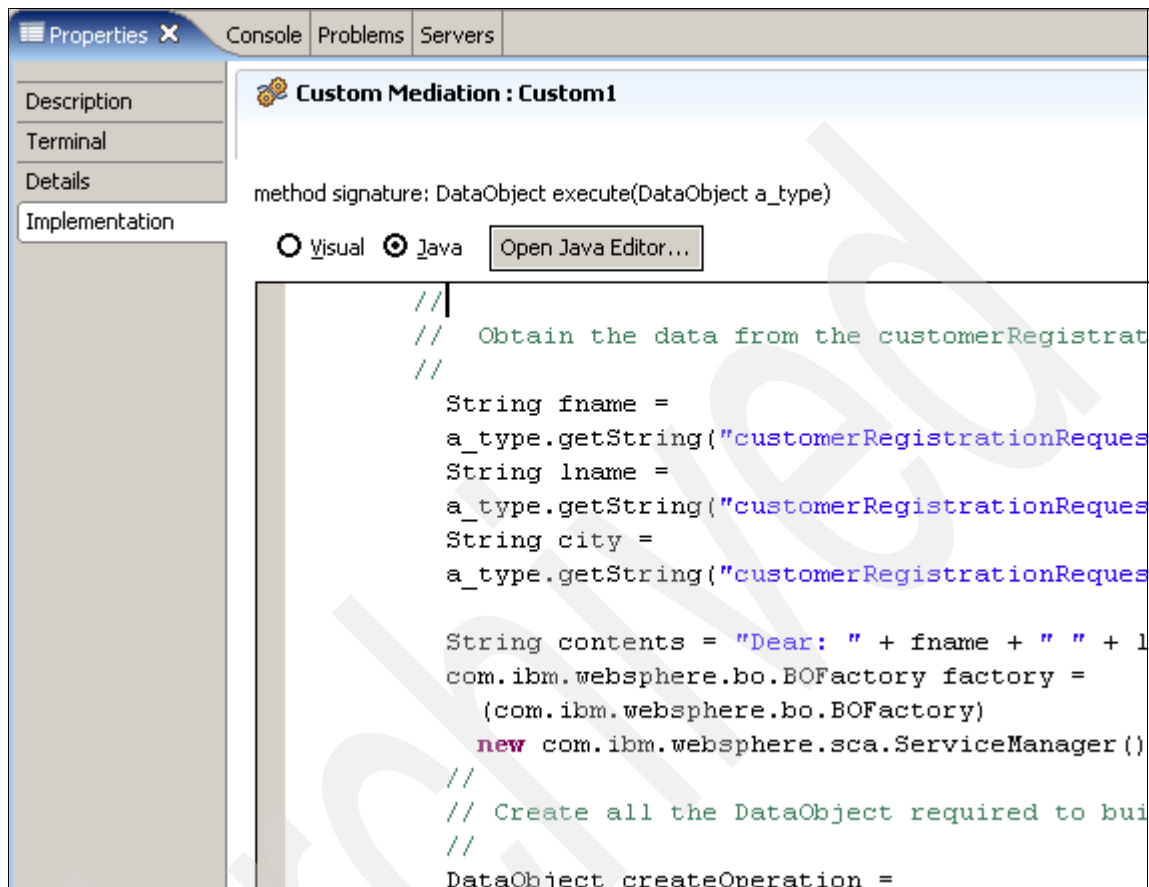


Figure 8-61 Implementation page for the Custom primitive

You can see the three ways to implement the custom mediation: via the visual editor, via a Java snippet (the default view, as above), and via a full Java editor.

6. We want to generate random names so that we minimize the risk of overwriting previously generated files. This requires a full-blown Java Editor.
Before going ahead with the Java work, save the contents of the Implementation box.
7. Click the **Open Java Editor** button.

8. Click **OK** at the following warning (Figure 8-62).

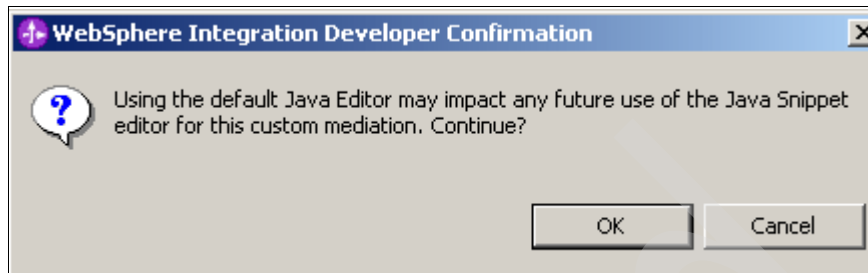


Figure 8-62 Warning when switching from the snippet editor to the full-blown Java Editor

9. The Custom1PartnerCustomLogic.java file will open for editing. Locate the code associated with the following method (Figure 8-63).

```
public DataObject execute(DataObject a_type)
```

Figure 8-63 Execute Custom mediation primitive method

You should recognize the contents of this method as the code (Figure 8-57 on page 404) you entered into the Implementation box.

- a. Uncomment the following lines (Figure 8-64) (they should be right at the top of the execute method).

```
//Random x = new Random();  
//int i = x.nextInt();
```

Figure 8-64 Adding the a random number to the file name to provide file name uniqueness

- b. Comment out the provisional line we inserted to provide an integer variable of name i (Figure 8-65).

```
int i = 456;
```

Figure 8-65 Removing the temporary variable i

- c. Press Ctrl+S inside the Java editor to save your code.

10. You will see an error by the line where the variable `x` is created. To fix this, add the import required by the `Random` type:
- Right-click inside the Java editor and select **Source** → **Organize Imports**, as shown in Figure 8-66.

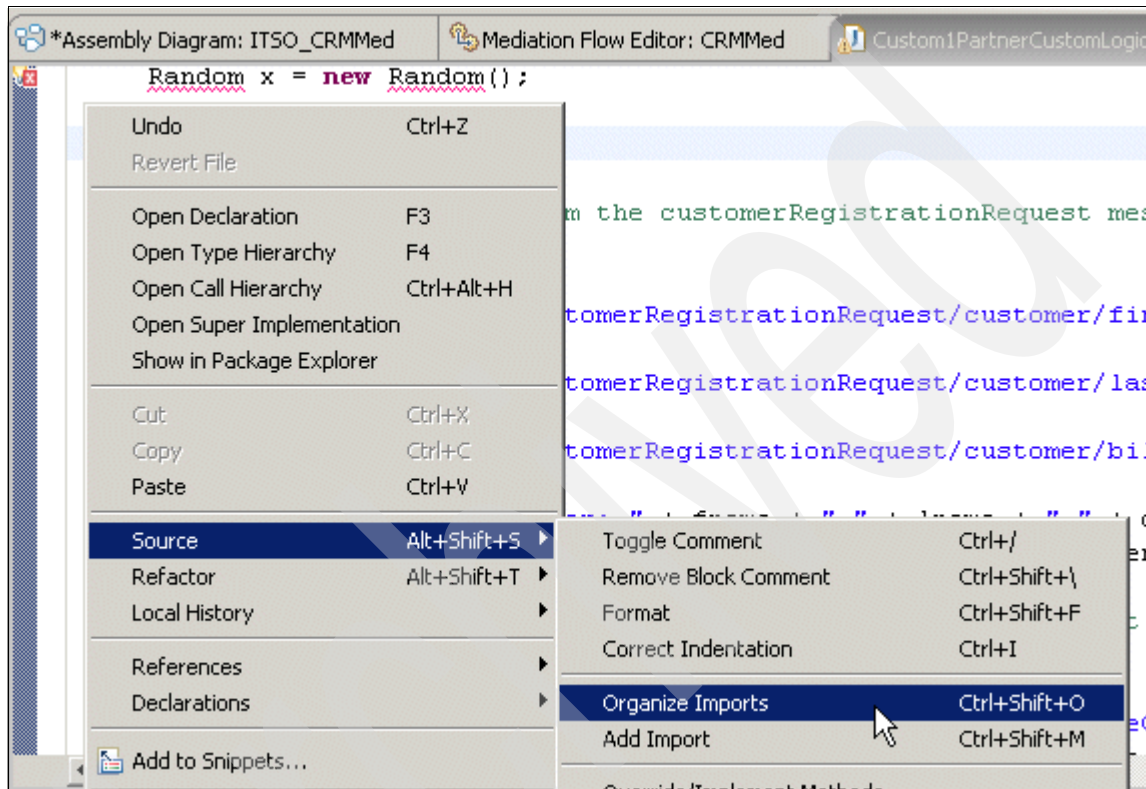


Figure 8-66 Adding the import required by the use of the `Random` type

- Press `Ctrl+S` inside the Java editor again to save your code. This should remove the error. Close the Java editor window. We are done with the code.
11. Save the mediation flow and assembly diagram. Rebuild the project.

Response flow - silver

At this point we have completed the request flow for the silver terminal path. We are now going to do the response flow.

1. At the bottom of the Mediation Flow Editor select the **Response: registerCustomer** tab. It should look like Figure 8-67.

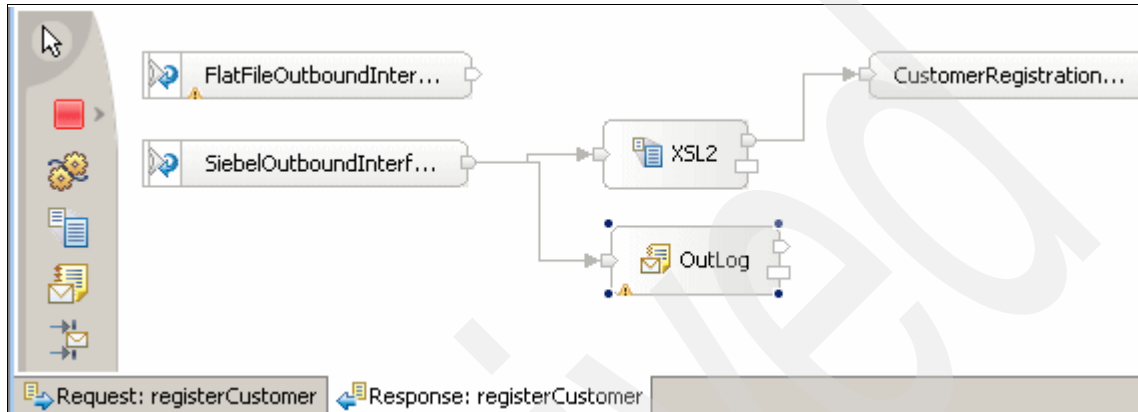


Figure 8-67 Response flow at this point

2. Add an XSL Transformation primitive between FlatFileOutboundInterfacePartner and CustomerRegistration_registerCustomer_InputResponse.
 - a. Rename it to XSL4.
 - b. Wire the output terminal of FlatFileOutboundInterfacePartner to the input terminal of XSL4.
 - c. Wire the output terminal of XSL4 to the input terminal of CustomerRegistration_registerCustomer_InputResponse.

The Response flow should look like Figure 8-68.

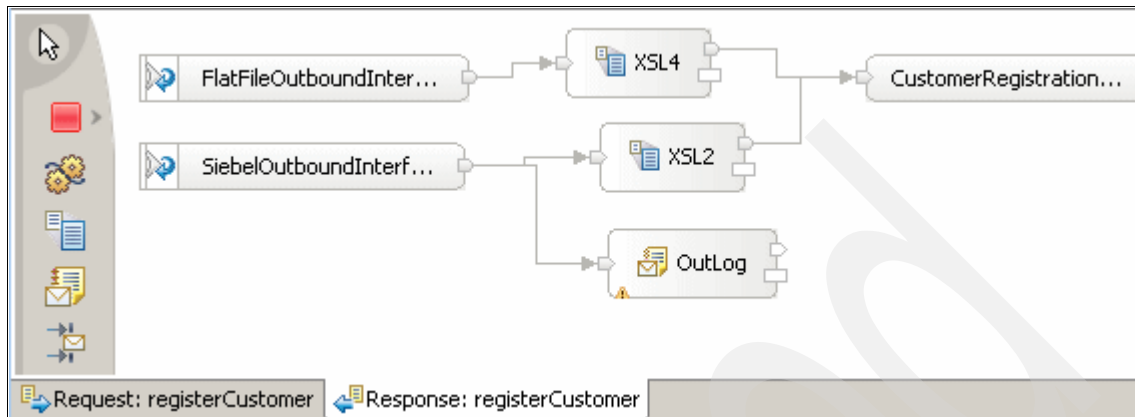


Figure 8-68 Response flow after adding XSL4

3. Create a new XML mapping that will take the output message provided by the create operation of the flat file adapter and map it to the Customer Registration Output message.
 - a. Verify that the input message to the transformation is createResponse and that the output message from the transformation is registerCustomerResponseMsg.

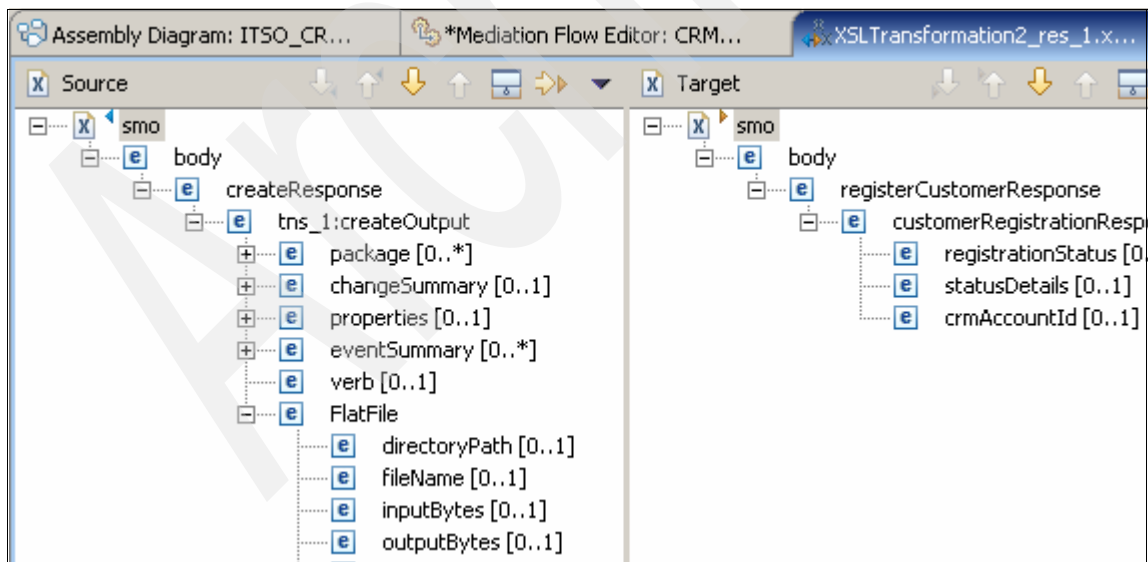


Figure 8-69 Building the map for XSL4

- b. Map the statusDetails field in the registerCustomerResponse to the constant 'SUCCESS' (using single quotes). This will require you to define an XSLT function (we did this XSL2 on page 384).
- c. Map fileName to registrationStatus.

The Overview view should look like Figure 8-70.

Target	Source	Applied Function
smo	smo	
body		
registerCustomerResponse		
customerRegistrationResponse		
registrationStatus [0..1]	fileName [0..1]	
statusDetails [0..1]		string

Figure 8-70 Overview of the final map for XSL4

- d. Save and close the map.
4. Regenerate the XSL.
5. Save the mediation flow and assembly diagram. Rebuild the project.

Test the flow for silver

At this point you should be able to test the CRMMed mediation for the Silver credit rating.

Runtime requirements: The custom Java code assumes the following:

- ▶ C:\FF directory exists. This directory is specified in the implementation code for the Custom primitive.
- ▶ Install and start CWYFF_FlatFileEAR. This is the adapter EAR file. It is packaged automatically by WebSphere Integration Developer.
- ▶ For viewing the message logger entries, install MessageLogApp (see “MessageLogApp application” on page 595).

1. Switch to the Server tab and start the WebSphere ESB Server v6.0.
2. In the Business Integration Explorer right-click the **ITSO_CRMMed** mediation module and select **Test** → **Test Module**.

This will open the unit test window for the mediation module.

3. Enter values for the input and click **Continue** (Figure 8-71). Minimum input values are firstName, lastName, billingAddress/city, and creditRating SILVER.

General Properties

Detailed Properties

Configuration: Default Module Test

Module: ITSO_CRMMed

Component: CRMMed

Interface: CustomerRegistration

Operation: registerCustomer

Initial request parameters

Name	Type	Value
<input type="checkbox"/> customerRe...	CustomerRegi...	
<input type="checkbox"/> customer	Customer	
accou...	string	1234
firstNa...	string	Gabriel
lastName	string	Shad
compa...	string	IBM
email	string	gshad@hotm...
passw...	string	gshad
<input type="checkbox"/> billing...	BillingAddress	
name	string	home
street	string	4265 Home Rd
city	string	Raleigh
state	string	NC
zipc...	string	27713
cou...	string	US
phone	string	(900)-898-7777
shippi...	ShippingAddr...	<null>
creditRating	string	SILVER

Figure 8-71 Testing the SILVER path using the flat file adapter

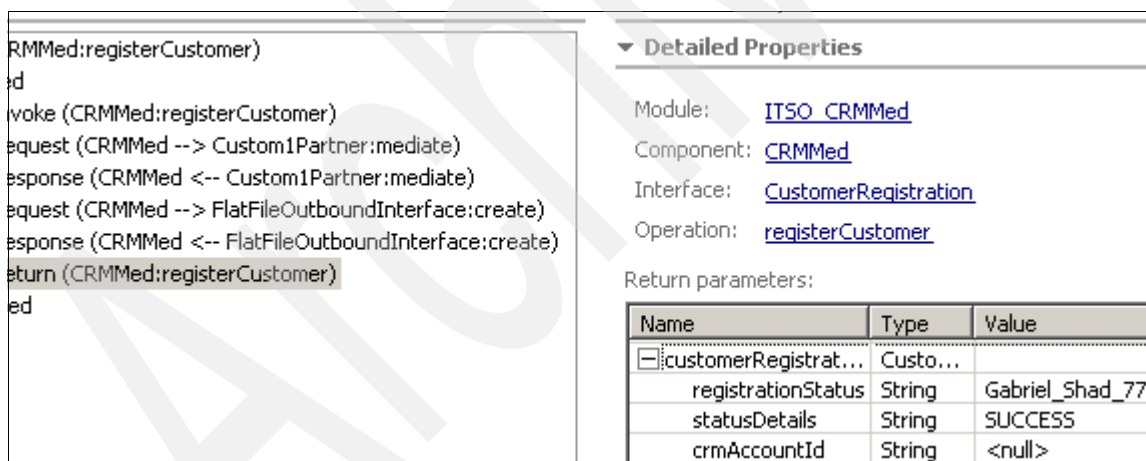
4. Select **WebSphere ESB Server v6.0** as the Deployment Location when prompted and click **Finish**.

Attention: When starting the application in the WebSphere Integration Developer test environment, you may see a sequence of stack frames in the SystemOut.log, starting with the following:

```
[3/24/06 9:46:46:031 EST] 00000069 ConnectionFac E    J2CA0009E: An
exception occurred while trying to instantiate the
ManagedConnectionFactory class
com.ibm.j2ca.flatfile.FlatFileManagedConnectionFactory used by resource
ITSO_CRMMed/intf/FlatFileOutboundInterface_CF :
java.lang.ClassNotFoundException:
com.ibm.j2ca.flatfile.FlatFileManagedConnectionFactory
    at
com.ibm.ws.classloader.CompoundClassLoader.findClass(CompoundClassLoader.
java(Compiled Code))
```

The module will still work successfully. This exception only seems to appear in the test environment. It does not appear in a WebSphere ESB stand-alone server.

After execution you should see an output message that will look like Figure 8-72.



The screenshot displays the test results for the CRMMed module. The left pane shows the test execution log, and the right pane shows the detailed properties of the response.

Test Execution Log:

```
CRMMed:registerCustomer()
    id
    invoke (CRMMed:registerCustomer)
    request (CRMMed --> Custom1Partner:mediate)
    response (CRMMed <-- Custom1Partner:mediate)
    request (CRMMed --> FlatFileOutboundInterface:create)
    response (CRMMed <-- FlatFileOutboundInterface:create)
    return (CRMMed:registerCustomer)
    ed
```

Detailed Properties:

Module: [ITSO_CRMMed](#)
Component: [CRMMed](#)
Interface: [CustomerRegistration](#)
Operation: [registerCustomer](#)

Return parameters:

Name	Type	Value
customerRegistrat...	Custo...	
registrationStatus	String	Gabriel_Shad_77
statusDetails	String	SUCCESS
crmAccountId	String	<null>

Figure 8-72 CRMMed test response for the SILVER path

You should also have a new file created in the directory specified in the custom node implementation code. In our example, this is C:\FF.

8.3 Calling the service from the application

The Register Customer process application calls the CRM mediation via a Web service proxy that makes a SOAP/HTTP request. This Web service proxy is generated based on the WSDL file that describes the mediation. This is the WSDL file generated when you create the export for the mediation, or if you are using the service integration bus to manage Web services, it is the WSDL file for the inbound service that defines the mediation.

To generate a Web service proxy that calls the CRM mediation, use the WSDL file `CRMReqOn_CustomerRegistrationHttp_Service.wsdl`.

Your application will also need access to any WSDL files referenced in this file, and the XSD files for the data objects used. The best way to pick up the files you need is to first go through the test process for the mediation. Then copy the files found under the `wsdl` folder for the mediation EJB to the application EJB.

You can find these files using the Physical Resources view under `<mediationEJB>/ejbModule/wsdl`.

The Register Customer process application, `ITSO_RegProcServiceApp`, uses a Java utility project to hold the proxies for the services it calls. This utility project is called `ITSO_RegProcService_Proxies`.

The steps required to generate the proxy are:

1. Copy or import the required files to the utility project.
2. Generate the Web service client proxy using the Web Service Client wizard. Use the service WSDL file as input.
3. Add code to the application to call the service via the proxy.

You can see an example of this process in 7.5, “Calling the service from the application” on page 341.

The following code (Example 8-1) in ITSO_RegProcServiceApp calls the service using the proxy.

Example 8-1 Invoking the CRM mediation

```
if (registrationProcessStatus == SUCCESS) { //nothing went wrong with the CreditCheck Service

    // register customer in CRM system
    try {
        System.out.println("ITSOmart RegistrationProcessorService.getCreditRating >> invoking
CustomerRegistration Service (soap/http)");
        // check for registration denied, if denied, set registrationProcessStatus = DENIED

        CustomerRegistrationRequest customerRegistrationRequest = new
CustomerRegistrationRequest();
        customerRegistrationRequest.setCreditRating(creditRating);
        customerRegistrationRequest.setCustomer(customer);
        CustomerRegistrationProxy customerRegistrationProxy = new CustomerRegistrationProxy();
        CustomerRegistrationResponse customerRegistrationResponse =
customerRegistrationProxy.registerCustomer(customerRegistrationRequest);
        String accountId =customerRegistrationResponse.getCrmAccountId();
        customer.setAccountNo(accountId);
        String crmStatus = customerRegistrationResponse.getRegistrationStatus();
        if (crmStatus.equals("SUCCESS")) registrationProcessStatus = SUCCESS;
        else if (crmStatus.equals("DENIED")) registrationProcessStatus = DENIED;

        //customerRegistrationResponse.getStatusDetails();
    } catch (Exception customerRegistrationException) {
        registrationProcessStatus = FAILURE;
        registrationStatusDetails = "Failure occurred during registration process.
CustomerRegistration Service exception: " + customerRegistrationException;
    }
}
```

8.4 For more information

For more information about the WebSphere Adapters see the following:

- ▶ WebSphere Adapter home page
<http://www-306.ibm.com/software/integration/wbiadapters/>
- ▶ WebSphere Adapter product documentation
<http://www-306.ibm.com/software/integration/wbiadapters/library/infocenter/doc/index.html>

Building the Register Shipping mediation

This chapter focuses on the Register Shipping scenario of the ITSOMart solution.

The sample in this chapter illustrates:

- ▶ SOAP/JMS transport
- ▶ Using a mediation to simulate a service for testing
- ▶ Working with arrays in Filter and XSL Transformation primitives
- ▶ Using a Stop primitive to stop a mediation flow

This chapter includes the following topics:

- ▶ Scenario overview
- ▶ Creating the Register Shipping Service emulator
- ▶ Developing the Register Shipping mediation
- ▶ Testing the mediation
- ▶ Calling the service from the application
- ▶ Considerations for handling arrays, decomposition

9.1 Scenario overview

The Register Shipping mediation takes an incoming request and breaks it down into multiple, simpler messages for handling by the mediation and target service. The idea allows service requestors to use a relatively complex format for service requests while the mediation within the ESB breaks down the request into separate, smaller requests to send to the target services. This gives the ability to provide a batch interface to a service when no such interface exists and allows requestors using different message formats to continue using services even if the service interface changes or if the service provider changes.

Note: In this instance, using a mediation to perform this simple function makes sense. However, more complex actions may need to be incorporated into the application itself. The decision about whether to do this in the mediation is an architectural decision that should be made after considering the topics covered in 9.6, “Considerations for handling arrays, decomposition” on page 447.

Note: The samples included in this chapter can be downloaded from the Web. See Appendix A, “Sample application install summary” on page 589, for instructions on downloading and importing the sample projects.

9.1.1 Business scenario

The first step in the Register Customer process is to evaluate the customer credit rating. The rating is used to determine how to proceed with the registration. This step of the process was built in Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263.

The next step evaluates the credit rating and determines what action should be taken with the request. The possibilities are to deny registration, send the registration to a queue for manual handling, or register the customer. This step of the process was built in Chapter 8, “Building the CRM mediation” on page 347.

In the event that a customer has a gold credit rating, the previous step registered the customer in the CRM. The address registered is the customer billing address.

The Register Shipping scenario will supplement that by registering the shipping addresses for the customer. The Register Shipping Service can only take one address as input, but the ITSOMart solution will allow up to two addresses to be entered. A mediation will determine whether two addresses have been entered and send them to the Register Shipping Service one at a time.

The activity diagram for the Register Shipping scenario of the ITSOMart solution can be seen in Figure 9-1. The activity diagram for the entire process can be seen in Figure 5-15 on page 143.

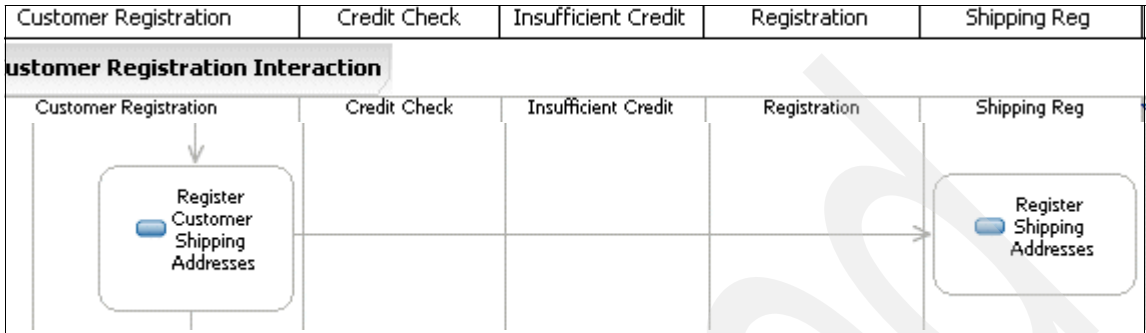


Figure 9-1 Register Shipping scenario

9.1.2 Register Shipping mediation

The call from the Register Customer process will call the Register Shipping Service via the ESB. Since the customer can enter up to two shipping addresses, a mediation in the ESB will determine the number of addresses entered and register each one.

Figure 9-2 shows the activity diagram for the mediation.

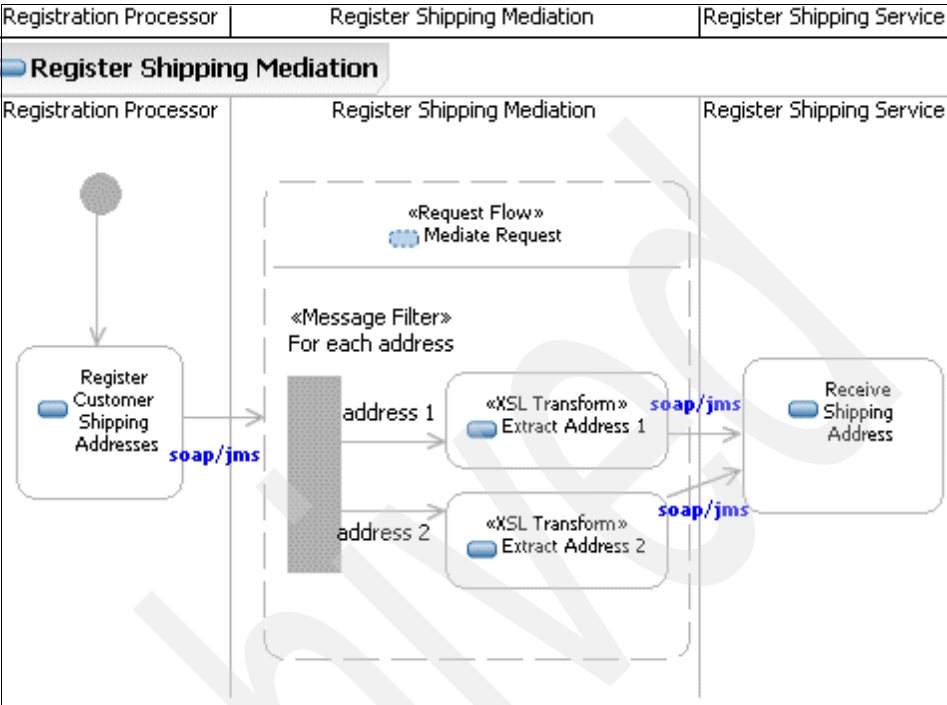


Figure 9-2 Register Shipping mediation activity diagram

The primary purpose of the mediation is to parse the input array of shipping addresses, then send each to the Register Shipping Service in the format required by the service. The mediation is invoked using SOAP/JMS. Figure 9-3 shows an overview of the Register Shipping mediation.

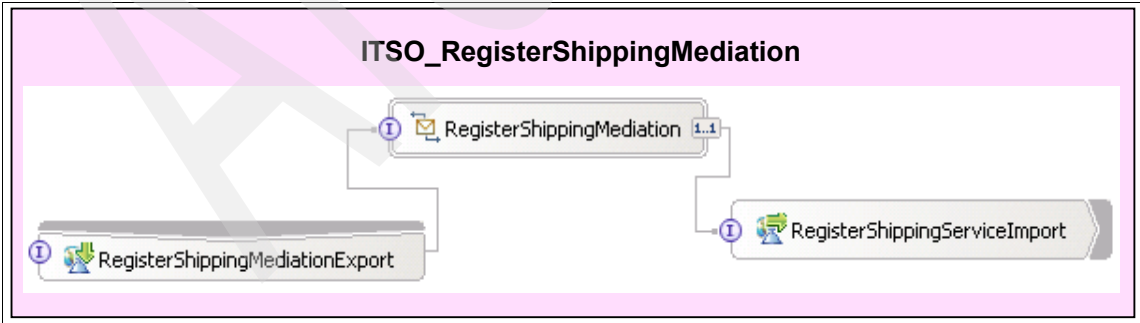


Figure 9-3 Register Shipping scenario

Note: For testing purposes, this example uses a second mediation to emulate the service that will eventually provide the registration service. This is to illustrate a technique that we found to be useful for a quick test. WebSphere ESB is not intended to host services.

9.2 Creating the Register Shipping Service emulator

Unlike the other mediations in this book, we do not only create the mediation using WebSphere Integration Developer, we also create a simple service using a mediation module. This approach can be useful to quickly create a service using SCA and expose that service to a variety of clients.

The following steps must be completed to create the shipping registry service:

1. Define the RegisterShippingService interface.
2. Create the Java component that implements the service.
3. Implement the mediation flow for the service.
4. Create the SOAP/JMS export binding.

Preparation

This mediation example will use the library ITSOMartLib (see 7.2.1, “Create a library” on page 268). This library contains business objects common to all the sample scenarios in this book. If you are working in a new workspace, you will need to import ITSOMartLib into your workspace or recreate it and the business objects it contains.

This chapter assumes that you have been through the earlier samples and are familiar with the steps required to build and edit mediations, business objects, and interfaces. If you are not familiar with these steps, review Chapter 6, “Assemble with WebSphere Integration Developer” on page 211, and Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263.

Important: If you are going to follow along with the instructions and build these samples, and if you are using a Windows operating system, shorten the mediation module names to something much shorter than what we used here to avoid the encountering the Windows limit on path names longer than 256 characters.

9.2.1 Define the RegisterShippingService interface

The following steps demonstrate how to define the RegisterShippingService interface.

1. Create an interface named RegisterShippingService in the ITSOMartLib project.
2. Add a one-way operation to the interface named sendShippingInfo.
3. Add an input parameter to the operation named inputSendShippingInfo and select **New** as the parameter type.
4. In the New Business Object window, enter RegisterShippingServiceRequest in the Name field and click **Finish**.
5. Now edit the RegisterShippingServiceRequest business object you just created and add the fields listed in Table 9-1.

Table 9-1 RegisterShippingServiceRequest Business object

Field	Type
accountNo	string
address	Address

The RegisterShippingServiceRequest business object should look like Figure 9-4.

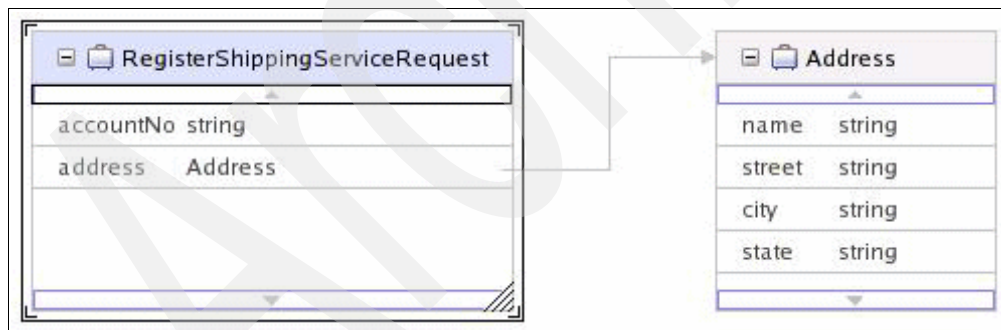


Figure 9-4 RegisterShippingServiceRequest business object

The interface should look like Figure 9-5.

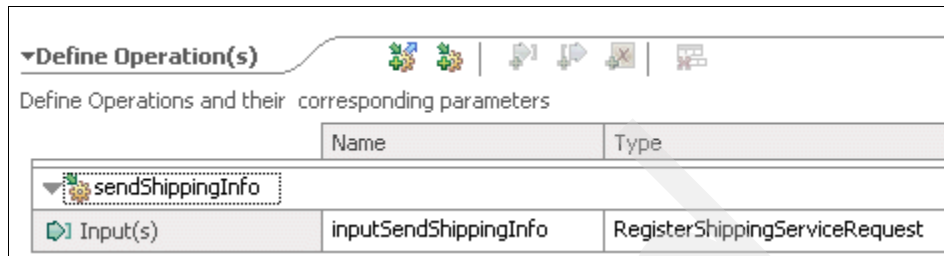


Figure 9-5 RegisterShippingService interface

6. Save the business object and interface.

9.2.2 Create the Java component that implements the service

Unlike the other target service implementations used in the scenario, the RegisterShippingService will be implemented as a mediation module that resides within the service bus. This approach allows developers to quickly create a service and provide a simple implementation without requiring additional resources or development tools. Because the service is hosted as an SCA module within WebSphere ESB, the service may be quickly tested and used by other SCA modules using a number of different export bindings without re-generating the service. This allows the developer to focus on the basic service implementation without worrying about generating Web service client proxies, or testing with network bindings.

Note: This approach is very useful for testing and debugging the mediation without requiring additional services or resources. However, for production, Web services should be implemented using traditional means (not as mediation modules).

Now that the RegisterShippingService interface has been defined, you must create the mediation module that will implement the service. Do this using the following steps:

1. Create a new mediation module:
 - a. Enter ITS0_ShippingRegistrationService for the name.
 - b. Select **WebSphere ESB Server v6.0** as the target runtime.
 - c. Select **Create mediation flow component**.
 - d. Add **ITSOMartLib** as a required library.

Note: This module is named SRSvc in the sample zip file.

2. Open the assembly diagram for the new mediation module you have created.
3. Drag a new Java component onto the diagram. To do this, click the mediation flow icon. You will then see both the mediation flow icon and a Java component icon (Figure 9-6).

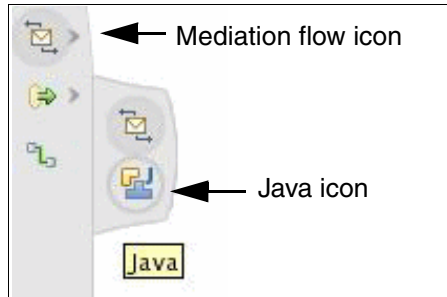


Figure 9-6 Selecting the Java component

4. Drag and drop the RegisterShippingService interface to the new Java component. Be sure to drop this onto the component and not as a new component in the assembly. You will see the interface icon added to the component.
5. Right-click the component and select **Generate Implementation**.
Create the component into a new package named `com.ibm.itso.shippingregistrationservice`.
6. Implement the `sendShippingInfo` method with code from Example 9-1.

Example 9-1 Sample service implementation

```
public void sendShippingInfo(DataObject inputSendShippingInfo) {
    System.out.println("ShippingAddressServiceComponentImpl.registerShippingAddress");
    String accountNo = inputSendShippingInfo.getString("accountNo");
    System.out.println("registerShippingAddress accountNo: " + accountNo);
    DataObject address = inputSendShippingInfo.getDataObject("address");
    if (address != null) {
        String name = address.getString("name");
        String street = address.getString("street");
        String city = address.getString("city");
        String state = address.getString("state");
        String zipcode = address.getString("zipcode");
        String country = address.getString("country");
        String phone = address.getString("phone");
        System.out.println("registerShippingAddress address: [" +
            name + ", " +
            street + ", " +
```

```

        city + ", " +
        state + ", " +
        zipcode + ", " +
        country + ", " +
        phone + "]" );
    }
    else {
        System.out.println("No address provided!");
    }
}

```

Tip: If you attempt to generate an implementation before adding the RegisterShippingService interface, the generated class will not have the sendShippingInfo method.

To fix this, add the interface to the component and select **Regenerate Implementation**. Be aware that doing this will erase any changes you have already made to the implementation class.

At times we found that WebSphere Integration Developer did not add the new method when regenerating the implementation. In this case, simply remove the component, drop a new Java component onto the assembly diagram, and add the interface before generating the implementation.

7. If you see errors that indicate that the class does not import the correct classes, right-click in the Java editor window and select **Source** → **Organize Imports** from the Source menu.
8. Close the Java editor.
9. Save the assembly diagram (but leave it open).

You have now created the service implementation. The assembly diagram should look like Figure 9-7.



Figure 9-7 The assembly diagram after completing the Java component implementation

Note: The example code does not perform any concrete actions. Rather, its only purpose is to demonstrate the shipping registration service function so that when the mediation is implemented, you can see the service working.

9.2.3 Implement the mediation flow for the service

If the service were being implemented in a business module to be hosted by WebSphere Process Server, you could add an export with whatever binding you like and wire that export directly to the Java component. However, WebSphere ESB does not support business modules, only mediation modules.

In order to properly create a mediation module, the module must contain a mediation flow component. Otherwise, WebSphere ESB will return error messages when trying to deploy the module. Because you are trying to use this module to create a service implementation, the mediation flow does not need to do anything. However, the mediation module allows a message logger to be attached to the incoming request flow for debugging purposes so the example will demonstrate this.

To implement the mediation flow, complete the following steps:

1. Rename the mediation flow component (Mediation1) to `ShippingRegistrationService`.
2. Wire the mediation flow component to the Java component you created earlier and click **OK** if you are asked about creating a matching reference.
3. Add an export component to the assembly diagram and rename the export to `ShippingRegistrationServiceExport`.
4. Wire the export component to the mediation flow component:
 - a. Select **OK** to the Add Wire message.
 - b. Select the `RegisterShippingService` interface.

Your assembly diagram should look like Figure 9-8.

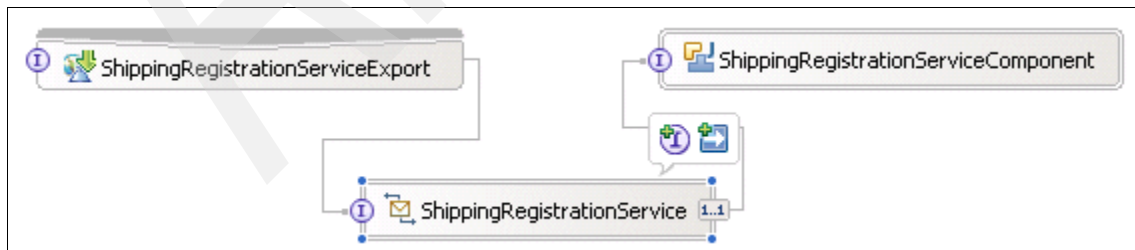


Figure 9-8 Service mediation - ITSO_ShippingRegistrationService

5. Right-click the **ShippingRegistrationService** mediation flow component and select **Generate Implementation**. Select the default folder. The new mediation flow will open.
6. In the Mediation Flow editor, wire the RegisterShippingService to the RegisterShippingServicePartner, as shown in Figure 9-9. You should now see the starting and endpoint nodes in the window below (Figure 9-9). In this case there is one input node and one callout node.

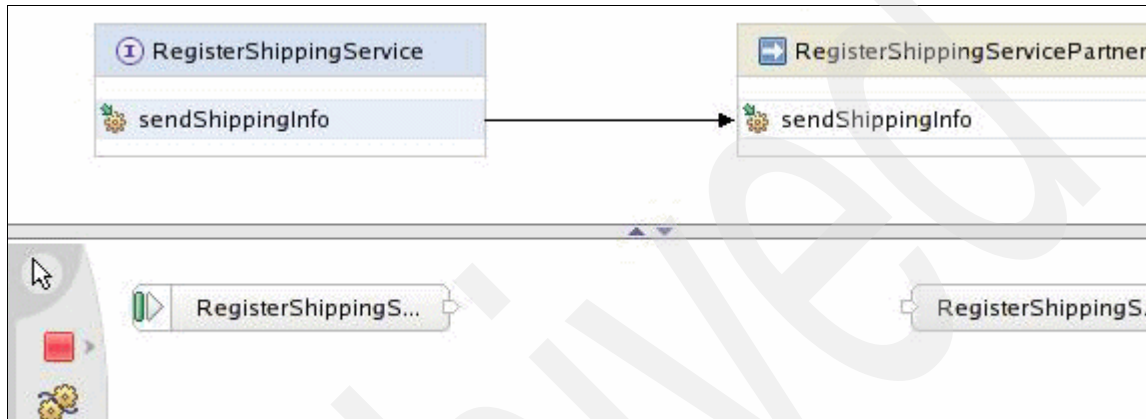


Figure 9-9 Mediation flow

7. Add a Message Logger primitive to the mediation flow and rename it RegisterShippingServiceRequestLogger.
8. Wire the mediation flow as shown in Figure 9-10.

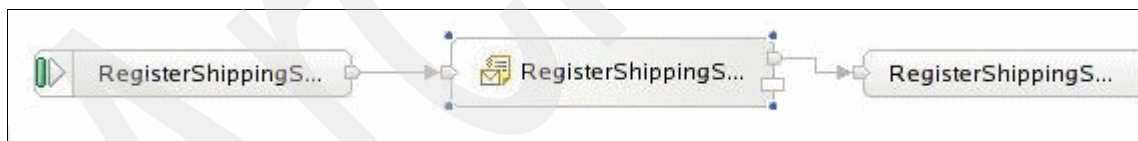


Figure 9-10 Mediation flow with Message Logger

9. Save and close the mediation flow.
10. Save the assembly diagram but leave it open.

You have now completed the mediation flow for the service.

9.2.4 Create the SOAP/JMS export binding

With the service implementation and the mediation flow completed, you will now need to generate a binding for the export. In this example you will select

SOAP/JMS (though realistically almost any binding would suffice). To test alternate bindings, you could create additional exports on the assembly diagram or regenerate the existing export binding. This allows rapid testing of different client access protocols without regenerating the service implementation.

1. In the module assembly, right-click the **ShippingRegistrationServiceExport** component and select **Generate Binding** → **Web Service Binding**.
 - a. Select **Yes** when asked about automatic wsdl file generation.
 - b. Select **soap/jms** as the transport.
2. Save and close the assembly diagram. Rebuild the project.

You have now created a SOAP/JMS binding for your service implementation. The export module should look like Figure 9-11 (note the change in the icon).

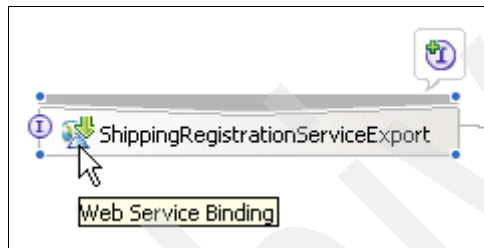


Figure 9-11 The SOAP/JMS export

Additionally, you should now see a Web Service Ports folder and a port for the Web service under the mediation module in the Business Integration view, as shown in Figure 9-12.



Figure 9-12 Web Service Ports

The service implementation is now complete and available. You will now create the mediation module to decompose the array of shipping addresses into individual requests.

9.3 Developing the Register Shipping mediation

Now that you have created the shipping registration service, you will need to create the mediation that transforms the Customer object used throughout the process into the ShippingRegistrationServiceRequest message object.

9.3.1 Mediation development steps

Figure 9-13 shows the mediation module contents we are going to build.

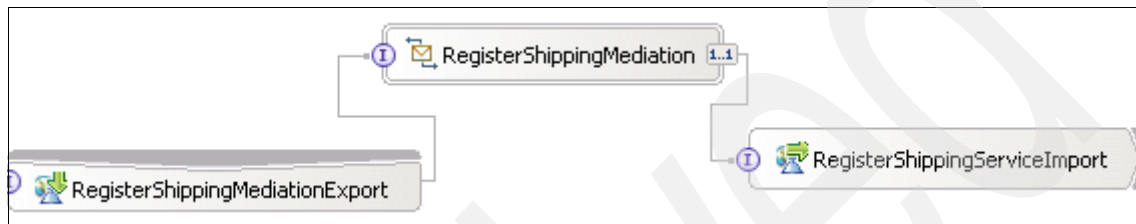


Figure 9-13 Register Shipping mediation assembly diagram

Building this mediation involves the following steps:

1. Define the ShippingRegistration interface.
2. Create the mediation module.
3. Implement the mediation flow.
4. Export the module as a Web service.

9.3.2 Define the ShippingRegistration interface

The following steps demonstrate how to define the ShippingRegistration interface that will be used by the registration process. When you are finished, the interface should look like Figure 9-15 on page 430. The request message object definition should look like Figure 9-14 on page 430.

1. Create an interface named ShippingRegistration in the ITSOMartLib project.
2. Add a one-way operation to the interface named registerCustomerShipping.
3. Add an input parameter to the operation named shippingRegistrationRequest and select **New** as the parameter type.

In the New Business Object window, enter ShippingRegistrationRequest in the Name field and click **Finish**.

4. Edit the new ShippingRegistrationRequest business object and add a single field named customer of type Customer (Figure 9-14).

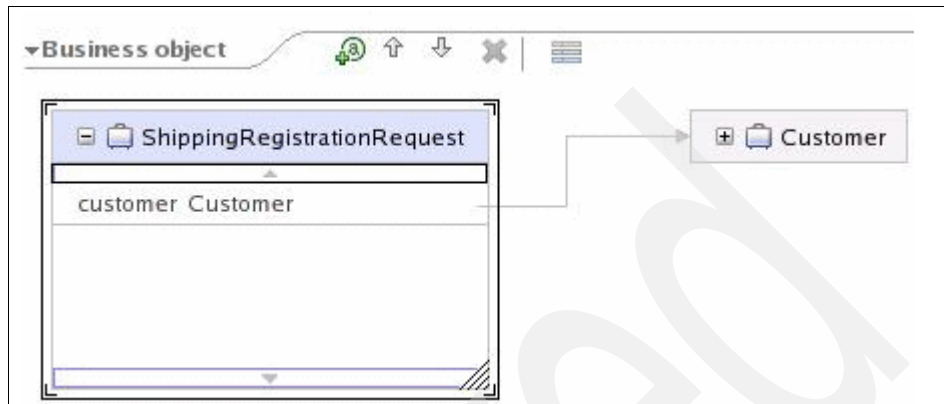


Figure 9-14 ShippingRegistrationRequest business object

5. Save and close the business object.
6. You have now created the ShippingRegistration interface. It should look like Figure 9-15.

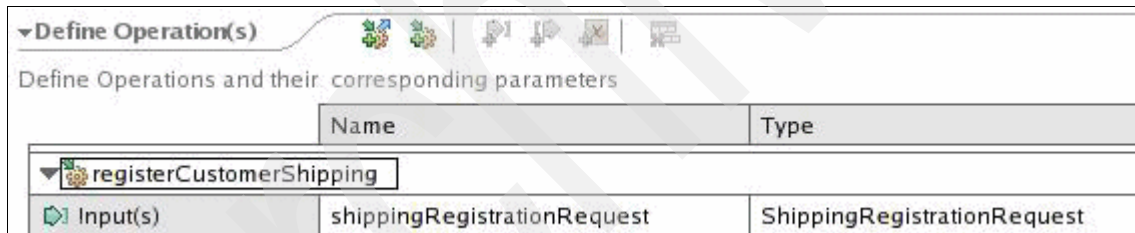


Figure 9-15 ShippingRegistration mediation interface

Save and close the new interface.

9.3.3 Create the mediation module

Now that you have created the mediation interface, you will need to create the mediation module that performs the actual message mediation. Do this using the following steps:

1. Create a new mediation module:
 - a. Enter ITS0_RegisterShippingMediation for the name.
 - b. Select **WebSphere ESB Server v6.0** as the target runtime.
 - c. Select **Create mediation flow component**.

- d. Select **ITSOMartLib** as a required library.

Note: This module is in the sample zip file as RSMed.

2. Open the assembly diagram for the new mediation module.
3. You will now need to import the WSDL file for the SOAP/JMS Web service you created earlier. To do this, switch to the Physical Resources view.

Tip: To switch to the Physical Resources view, right-click in the Business Integration view and select **Show Files**.

- a. Find and select the WSDL file under the ITS0_ShippingRegistrationService project (Figure 9-16).

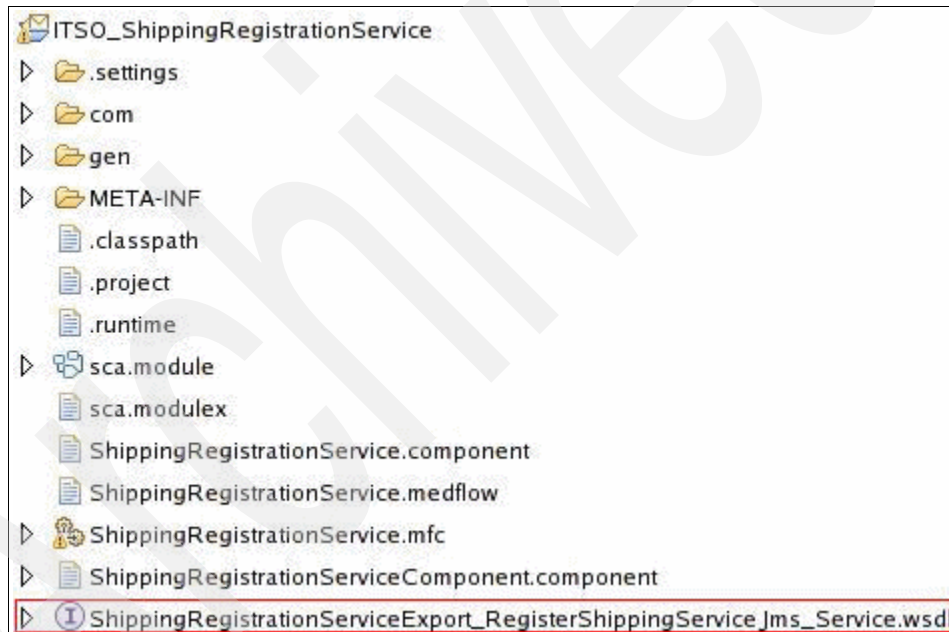


Figure 9-16 Physical resources

- b. Right-click and select **Copy** from the context menu.
- c. Right-click the **ITS0_RegisterShippingMediation** module and select **Paste**.

This copies the WSDL file from the service implementation module into the mediation module.

4. Once done, the Business Integration view for your mediation module should show a Web Service Ports item for the Web service. Figure 9-17 shows the imported Web service.



Figure 9-17 Imported Web service ports

5. To use the Web service, select the Web service port and drop it onto the assembly diagram.
Select **Import with Web Service Binding** from the Component Creation pop-up and click **OK**.
6. Rename the new import component to RegisterShippingServiceImport.
7. Drop an export component onto the assembly diagram and rename it to RegisterShippingMediationExport.
8. Add the ShippingRegistration interface to the export component.
9. Rename the mediation flow component (Mediation1) to RegisterShippingMediation.
10. Create a wire from the export component to the mediation flow component.
Click **OK** in the Add Wire message dialog.
11. Create a wire from the mediation flow component to the import component.
Click **OK** in the Add Wire message dialog.

The assembly diagram should look like Figure 9-18.

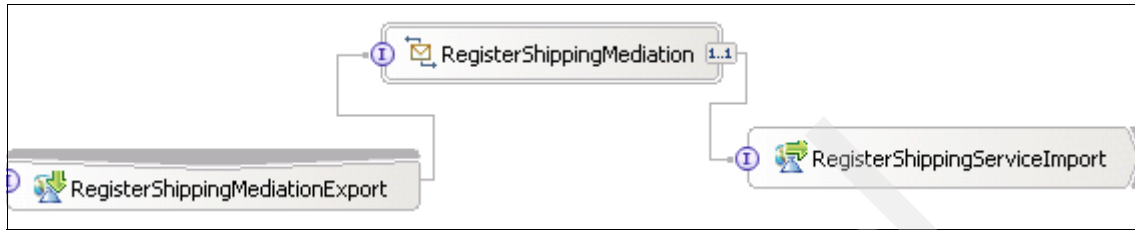


Figure 9-18 ITSO_RegisterShippingMediation assembly diagram

12. Save the assembly diagram.

9.3.4 Implement the mediation flow

The mediation flow is responsible for splitting the incoming Customer business object into a series of messages that conform to the RegisterShippingService interface. To do this you will need to use the Message Filter primitive to decompose the message and the XSL Transformation primitive to extract the message content for the downstream service.

Create the mediation flow

To create the mediation flow, do the following:

1. In the module assembly, right-click the mediation flow component and select **Generate Implementation** from the menu. Select the default location. This will open the new mediation flow.

2. In the Operation Connections view, draw a wire between the ShippingRegistration and the RegisterShippingServicePartner. You should now see the input and callout nodes in the mediation flow view.

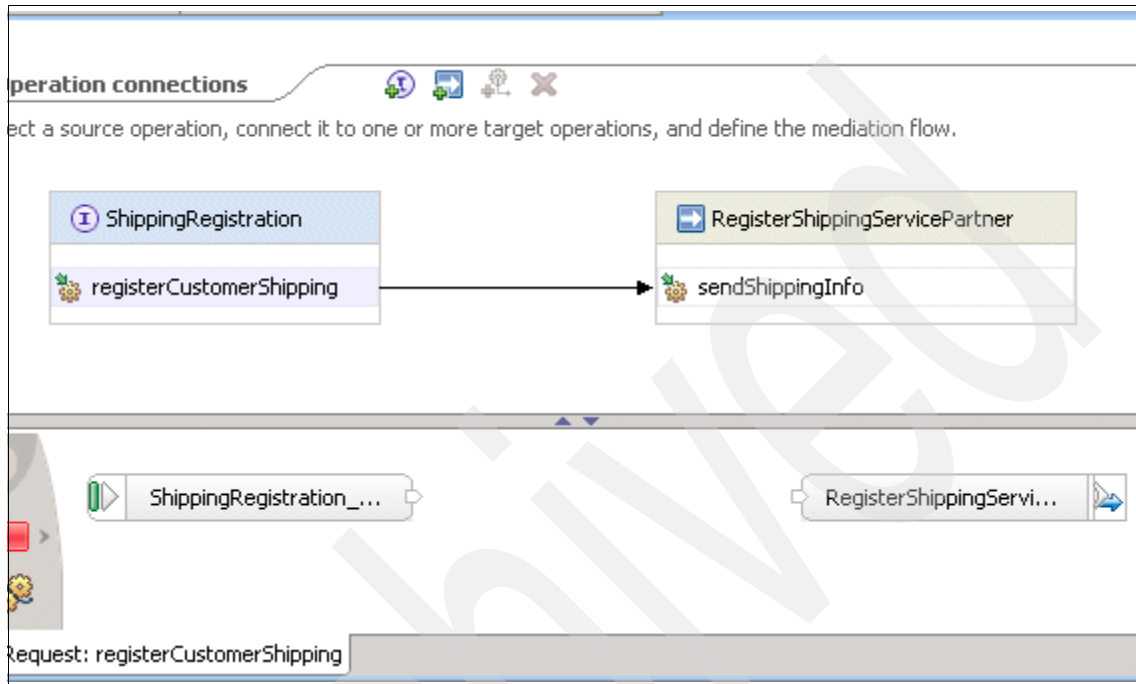


Figure 9-19 Mediation flow

Add the Message Filter primitive for routing

The first primitive in the mediation flow is a Message Filter primitive. The filter determines the number of shipping addresses in the incoming Customer business object, separates the addresses in the array, and routes each request to the shipping service.

In the mediation flow:

1. Drop a Message Filter primitive onto the mediation flow diagram. Connect the ShippingRegistration_registerCustomShipping_Input to the message filter input terminal. Change the name to FilterAddresses.
2. Right-click the filter and select **Add Output Terminal**.
 - a. Change the terminal name to Address1.
 - b. Click **OK**.

3. Add a second output terminal:
 - a. Change the terminal name to Address2.
 - b. Click **OK**.
4. Select the filter, and in the Details tab of the Properties view:
 - a. Change the distribution mode to All.
 - b. Click **Add** next to the filters.
 - i. Select **Address1** as the terminal name.
 - ii. Click **Custom XPath**.
 - iii. Expand the nodes under the body node in the XPath Expression Builder until you reach the shippingAddress[] node (the [] indicates that this is an array). When you click the shippingAddress[] node, a pop-up will appear asking for the index for the repeating element. Enter a 1 into this pop-up to indicate the first element in the array and click **OK**.
 - iv. Click **Finish** in the Add/Edit window.

The XPath expression should match Example 9-2 and the terminal name should be Address1.

Example 9-2 XPath expression for terminal Address1

```
/body/registerCustomerShipping/shippingRegistrationRequest/customer/shippingAddress[1]
```

- c. Click **Add** next to the filters and select **Address2** as the terminal name.
- Rather than using the XPath Expression Builder, enter the XPath expression, as show in Example 9-3.

Example 9-3 XPath expression for terminal Address2

```
/body/registerCustomerShipping/shippingRegistrationRequest/customer/shippingAddress[2]
```

5. When you are finished, the details for the filter should appear as shown in Figure 9-20.

Distribution mode: <input type="text" value="All"/>		
Filters:		
Pattern	Terminal name	
/body/registerCustomerShipping/shippingRegistrationRequest/customer/shippingAddress[2]	Address2	
/body/registerCustomerShipping/shippingRegistrationRequest/customer/shippingAddress[1]	Address1	

Figure 9-20 Details of the filter

6. Save the mediation flow and assembly diagram.

Add an XSL Transformation primitive

The message containing the address must be transformed into a format as appropriate for the downstream shipping registration service. You will use XSL Transformation primitives to achieve this.

1. Drop two XSL Transformation primitives onto the mediation flow. Rename them XSLAddress1 and XSLAddress2, respectively.
2. Wire the output from the filter's Address1 terminal to XSLAddress1 and Address2 to XSLAddress2.
3. Connect the output terminals from both XSLAddress1 and XSLAddress2 to the RegisterShippingServicePartner.
4. In the Details tab of the Properties view for XSLAddress1:
 - a. Click **New** to create a new XML mapping.

The New XSLT Mapping window should automatically populate registerCustomerShippingRequestMsg as the input message body and sendShippingInfoRequestMsg as the output message body.
 - b. Click **Finish**.
 - c. Expand the nodes in the source and target frames.

- d. Map the customer[0..1] array to inputSendShippingInfo. To do this, drag customer[0..1] to inputSendShippingInfo (Figure 9-21).

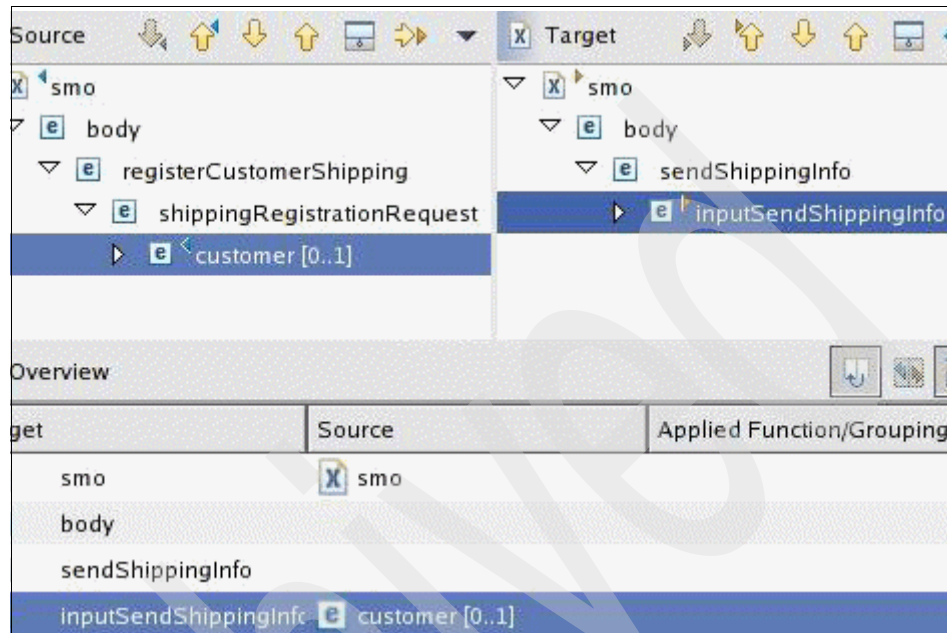


Figure 9-21 XSL Mapping

- e. Map accountNo to accountNo.

To do this, select **accountNo** in the source pane, then select **accountNo** in the target pane. With both highlighted, right-click **accountNo** in the target pane and select **Match Mapping** from the menu.

- f. Map shippingAddress to address.

To do this, select **shippingAddress** in the source pane and then select **address** in the target pane. With both highlighted, right-click **address** in the target pane and select **Match Mapping** from the menu.

- g. Save and close the XSLT Mapping window.

5. Click **Regenerate XSL** to generate an XSL style sheet for the mapping you have just created. Click **OK**.
6. You will now need to customize the XSL style sheet manually to select only the appropriate shipping address from the array. Since this is the XSL for XSLAddress1, you will need to modify the XSL to only match the first shipping address element. To do this, click **Edit** next to the XSL.

7. Replace the XSL that reads:

```
<xsl:apply-templates select="shippingAddress"/>
```

with:

```
<xsl:apply-templates select="shippingAddress[1]"/>
```

And replace the XSL that reads:

```
<xsl:template match="shippingAddress">
```

with:

```
<xsl:template match="shippingAddress[1]">
```

Note: What you are doing here is ensuring that the XSL only matches the first shippingAddress element in the incoming message.

Save and close the XSL file.

Example 9-4 shows the relevant listing for XSLAddress1.

Example 9-4 XSL for XSLAddress1

```
</xsl:template>
```

```
<!-- Composed element template -->
```

```
<xsl:template match="customer">
```

```
  <inputSendShippingInfo>
```

```
    <accountNo>
```

```
      <xsl:value-of select="accountNo/text()"/>
```

```
    </accountNo>
```

```
    <xsl:apply-templates select="shippingAddress[1]"/>
```

```
  </inputSendShippingInfo>
```

```
</xsl:template>
```

```
<!-- Rename transformation template -->
```

```
<xsl:template match="shippingAddress[1]">
```

```
  <address>
```

```
    <xsl:apply-templates  
select="*|@*|comment()|processing-instruction()|text()"/>
```

```
  </address>
```

```
</xsl:template>
```

```
<!-- Identity transformation template -->
```

8. Repeat the previous steps to create an identical mapping for XSLAddress2. However, when modifying the XSL, replace shippingAddress with shippingAddress[2].

Example 9-5 shows the relevant list for XSLAddress2.

Example 9-5 XSL for XSLAddress2

```
</xsl:template>

<!-- Composed element template -->
<xsl:template match="customer">
  <inputSendShippingInfo>
    <accountNo>
      <xsl:value-of select="accountNo/text()"/>
    </accountNo>
    <xsl:apply-templates select="shippingAddress[2]"/>
  </inputSendShippingInfo>
</xsl:template>

<!-- Rename transformation template -->
<xsl:template match="shippingAddress[2]">
  <address>
    <xsl:apply-templates
select="*|@*|comment()|processing-instruction()|text()"/>
  </address>
</xsl:template>

<!-- Identity transformation template -->
```

9. Save the mediation flow.

The mediation flow should look like Figure 9-22.

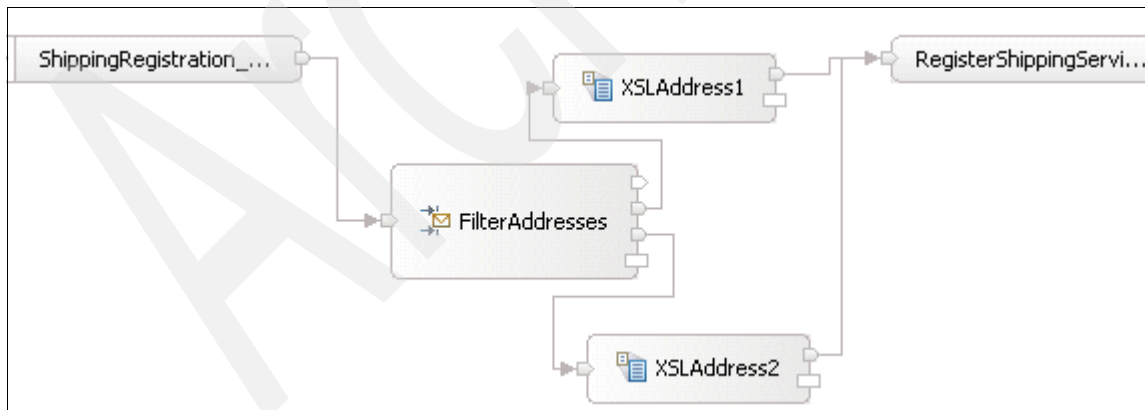


Figure 9-22 RegisterShippingMediation mediation flow

Add a Stop primitive to stop the flow if no address is provided

Although you have completed all of the basic functionality required for the mediation flow, there are a few other things that would make the mediation flow more usable. For example, what if the incoming customer message had no shipping addresses at all? Currently, the registration user interface does not permit this situation, but other services that use the mediation might not have similar checks. Also, it would be nice to log the separate addresses that were output by the XSL primitive for testing.

To log requests that contain no shipping addresses do the following.

1. Add a Message Logger primitive to the mediation flow:
 - a. Change its name to NoAddress.
 - b. Wire the default output terminal of the message filter to the NoAddress logger. The default output terminal is the top output terminal shown on the primitive and has the name default.
2. Add a Stop primitive to the mediation flow. Connect the output terminal from the NoAddress Message Logger to the Stop primitive. This prevents further processing of the message. The completed mediation flow appears in Figure 9-23.

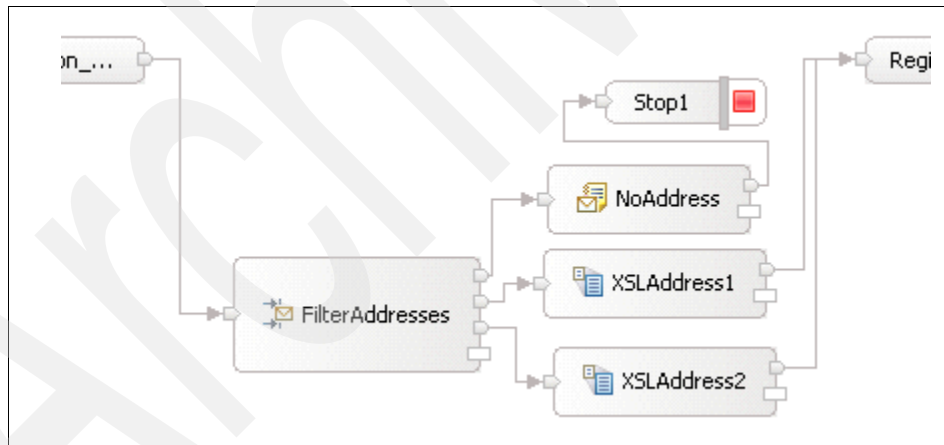


Figure 9-23 Mediation flow

3. Save the mediation flow.

Add message logging for debugging

For initial debugging, it may be useful to insert Message Logger primitives to the output terminals of XSLAddress1 and XSLAddress2 to ensure that the appropriate shipping address is being extracted by the XSL transform. To do this:

1. Add two Message Logger primitives to the mediation flow. Name them Address1Log and Address2Log, respectively.
2. Remove the two wires from the output of XSLAddress1 and XSLAddress2.
3. Wire the output of XSLAddress1 to Address1Log.
4. Wire the output from Address1Log to the RegisterShippingServicePartner.
5. Wire the output of XSLAddress2 to Address2Log.
6. Wire the output from Address2Log to the RegisterShippingServicePartner.

The mediation flow appears in Figure 9-24.

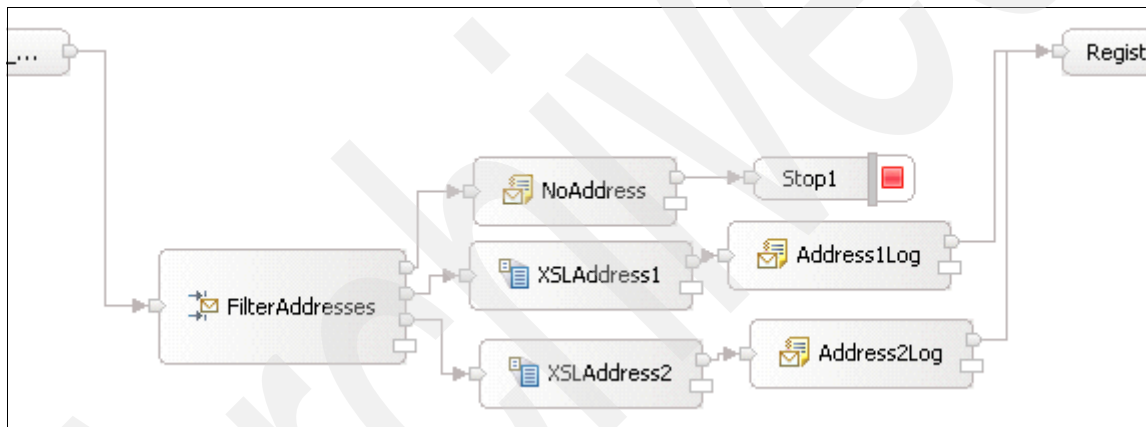


Figure 9-24 Mediation flow with logging

7. Save the mediation flow.

You have now completed the implementation of the mediation flow.

9.3.5 Export the module as a Web service

You will now need to generate an export binding on the `RegisterShippingMediationExport` so that the mediation may be invoked by the registration processor. This is done in the same way as with the shipping registration service using the following steps:

1. In the assembly diagram, right-click the **RegisterShippingMediationExport** and select **Generate Binding** → **Web Service Binding**.
 - a. Select **Yes** when asked about automatic wsdl file generation.
 - b. Select **soap/jms** as the transport.
2. Save the assembly diagram and rebuild the project.

You have now created a SOAP/JMS binding for your mediation module. Additionally, you should now see a Web Service Ports item in the mediation module and a port for the Web service.

Note: Because there was originally a port for the WSDL you copied from the service implementation, you should now see two Web service ports in the mediation module.

You have now completed all of the steps necessary for the shipping registration mediation and the service implementation.

9.4 Testing the mediation

Now that you have built the mediations, it is time to test them. As a best practice, try to test the modules with the smallest number of dependencies first to ensure that the module is working before moving on to test the multiple modules together. In this case, you will first test the shipping registration service and then test the mediation.

Runtime configuration

Because the mediation uses a service that also resides in the bus, no services need to be installed.

For viewing the log entries made by the Message Logger primitives, install the `MessageLogApp` sample application. (See Appendix A, “Sample application install summary” on page 589.) You can open the application to view the log entries using the following URL:

<http://localhost:9080/MessageLogWeb/faces/MessageLogView.jsp>

Because you are using SOAP/JMS as a transport, JMS objects and bus destinations will be required. However, these are created for you automatically when you deploy the mediation.

9.4.1 Testing the RegisterShippingService emulator

First, test the service that handles the shipping address registration:

1. Start the WebSphere ESB Server test environment.
2. Add the ITSO_ShippingRegistrationService application to the server.
3. Open the assembly diagram for the ITSO_ShippingRegistrationService mediation. Right-click in the assembly diagram and select **Test Module** to launch the test client.
4. Set the values in the test client:
 - a. The test client allows you to set details for the test, including the component to test. The default component is the mediation flow component. Change this to select the ShippingRegistrationServiceExport component so that you are testing the complete module, from the export through the mediation and all the way to the Java component.
 - b. Fill in some test data for the request. The service does not require any test data at all, but it is a good idea to fill in complete test data so that you can see the service working properly. Sample test data can be found in Figure 9-25.

Request parameters		
Name	Type	Value
<input type="checkbox"/> inputSendShipping	RegisterShippingService	
accountNo	String	2112
<input type="checkbox"/> address	Address	
name	String	John Doe
street	String	123 Fake St
city	String	Faketown
state	String	CA
zipcode	String	90101
country	String	US

Figure 9-25 Sample data for testing

5. Check the test client events pane to make sure that requests were generated for the export, through the mediation, and ultimately to the Java component. Also check the output logs for messages, as shown in Example 9-6.

Example 9-6 Sample content from output log

```
... ShippingAddressServiceComponentImpl.registerShippingAddress
... registerShippingAddress accountNo: 2112
... registerShippingAddress address: [John Doe, 123 Fake St,
Faketown, CA, 90101, US, 555.555.1212]
```

6. You can also use the MessageLogApp sample application to examine the database entries in the Message Logger database. You should see a log entry for the RegisterShippingServiceLogger. Clicking that entry, you should see the contents of the logged message, as in Example 9-7.

Example 9-7 Sample message logger content

```
<sendShippingInfo>
  <inputSendShippingInfo>
    <accountNo>2112</accountNo>
    <address>
      <name>John Doe</name>
      <street>123 Fake St</street>
      <city>Faketown</city>
      <state>CA</state>
      <zipcode>90101</zipcode>
      <country>US</country>
      <phone>555.555.1212</phone>
    </address>
  </inputSendShippingInfo>
</sendShippingInfo>
```

9.4.2 Test the Register Shipping mediation

Next, test the mediation that calls the service. This will test the application end-to-end:

1. Start the WebSphere ESB Server test environment.
2. Add the mediation application for RegisterShippingService and RegisterShippingMediation to the server. Note that all JMS configuration required for the export using soap/jms transport is done automatically when you deploy.
3. Launch the test client for the RegisterShippingMediation module.
4. Set the values in the test client.

- a. Make sure to select the RegisterShippingMediationExport component so that you are testing the complete module, from the export through the mediation, and all imports that then call the service module.
- b. Fill in some test data for the request. The service does not require any test data at all, but it is a good idea to fill in complete test data so that you can see the service working properly.

Tip: In order to test the shippingAddress array elements, right-click **shippingAddress[]** and select **Add Element** from the menu.

5. Check the test client events pane to make sure that requests were generated for the export, through the mediation and the service import, and ultimately through the service export to the Java component. Also check the output logs for messages. You should see a separate output message for each of the shippingAddress[] elements you added.
6. You can also use the MessageLogApp sample application to examine the database entries in the Message Logger database. You should see log entries for the Address1Logger, Address2Logger, or NoAddressLogger and RegisterShippingServiceLogger based on the number of shipping addresses you entered.
7. Try testing the service with no shipping addresses, one shipping address, and then two shipping addresses to ensure that all of the cases in the mediation flow work properly.

9.5 Calling the service from the application

The Register Customer process application calls the Register Shipping mediation via a Web service proxy that makes a SOAP/JMS request. This Web service proxy is generated based on the WSDL file that describes the mediation. This is the WSDL file generated when you create the export for the mediation, or if you are using the service integration bus to manage Web services, it is the WSDL file for the inbound service that defines the mediation.

To generate a Web service proxy that calls the Register Shipping mediation, use the RegisterShippingMediationExport_ShippingRegistrationJms_Service.wsdl WSDL file.

Your application must also have access to any WSDL files referenced in this file, and the XSD files for the data objects used. The best way to pick up the files you need is to first go through the test process for the mediation. Then copy the files found under the wsdl folder for the mediation EJB to the application EJB.

You can find these files using the Physical Resources view under <mediationEJB>/ejbModule/wsdl.

The Register Customer process application, ITSO_RegProcServiceApp, uses a Java utility project to hold the proxies for the services it calls. This utility project is called ITSO_RegProcService_Proxies.

The steps required to generate the proxy are:

1. Copy or import the required files to the utility project.
2. Generate the Web service client proxy using the Web Service Client wizard. Use the service WSDL file as input.
3. Add code to the application to call the service via the proxy.

You can see an example of this process in 7.5, “Calling the service from the application” on page 341.

The following code in ITSO_RegProcServiceApp calls the service using the proxy (Example 9-8).

Example 9-8 Invoking the Register Shipping mediation

```
if (registrationProcessStatus == SUCCESS) { // if nothing went wrong with the
CustomerRegistration Service and the customer was not denied registration

    // register shipping addresses
    try {
        System.out.println("ITSOmart RegistrationProcessorService.getCreditRating
>> invoking ShippingRegistration Service (soap/jms)");
        ShippingRegistrationProxy shippingRegistrationProxy = new
ShippingRegistrationProxy();
        ShippingRegistrationRequest shippingRegistrationRequest = new
ShippingRegistrationRequest();
        shippingRegistrationRequest.setCustomer(customer);

        shippingRegistrationProxy.registerCustomerShipping(shippingRegistrationRequest);
    } catch (Exception shippingRegistrationException) {
        registrationProcessStatus = FAILURE;
        registrationStatusDetails = "Failure occurred during registration process.
ShippingRegistration Service exception: " + shippingRegistrationException;
    }
}
```

9.6 Considerations for handling arrays, decomposition

Although this scenario successfully uses a filter to handle an array in the input data, you should note that there are aspects of this implementation that could be considered limitations to building a more advanced mediation.

If you want to implement a true decomposition/recomposition scenario where the mediation can handle a message with an unknown number of address sub-elements (0..* cardinality) and gather a set of responses from the target services, WebSphere ESB may not be the proper choice to implement your ESB.

9.6.1 Handling an unknown number of input request elements

The first aspect of the scenario to consider is the fairly limiting assumption that there can be at most two shipping addresses in the incoming message.

The Message Filter primitive requires an output terminal for each outbound request invocation. That is, a single terminal may only result in a single call to the partner service. There is no way to dynamically create terminals on the Message Filter primitive at runtime.

In addition, the XPath expression used to determine whether to fire a terminal was based on a specific occurrence of the shippingAddress sub-element. For example, the Address1 terminal XPath ends with shippingAddress[1], which only fires if there is at least one shippingAddress sub-element. The Address2 terminal XPath ends with shippingAddress[2], which only fires if there are at least two shippingAddress sub-elements.

Without knowing the maximum number of shippingAddress sub-elements, there is no way to determine how many output terminals should fire. These restrictions mean that there is a non-trivial amount of work that must be done to handle each shippingAddress within the incoming message.

While none of the mediation primitives include a looping mechanism that would allow you to invoke a downstream service multiple times, this type of functionality is easily handled in BPEL.

One possibility is to use a Custom mediation. A custom mediation is implemented using Java code and may directly manipulate the message as it flows through the mediation. Using Java, the integration developer could decompose the message regardless of the number of shippingAddress sub-elements. However, this approach also has limitations. Like any other mediation primitive, the custom mediation may only invoke a Partner service once for each output terminal. To get around this, you could have the Java component access the service directly and invoke the service (for example, drop

messages onto a JMS queue if the service is operating over JMS or generate SOAP requests for Web service implementations). This direct approach would solve both the output terminal issue as well as the issue of recomposing replies from the service. However, doing this would defeat the componentry of the mediation flows, as the call to the downstream service is now hard coded into the mediation. Changes to the downstream service now could require significant changes to the custom mediation, which would not be desirable. Alternatively, you could create a mediation module with an SCA export wired directly to the appropriate service import. The custom mediation module performing the disaggregation then invokes the SCA export rather than calling the service directly.

9.6.2 Handling multiple responses

The other anomaly in the shipping registration scenario is that the shipping registration service `sendShippingInfo` operation is a one-way operation rather than a request-response. The reason for this is because WebSphere ESB offers no easy way to accept multiple response messages for a single request.

The separation of mediating the request and response flow also means that in the response flow, the mediation has no easy way of determining how many outbound requests were sent and thus how many responses the mediation should wait to receive. WebSphere Integration Developer allows the mediation to be created, but when the mediation module is run within WebSphere ESB, an exception stating that multiple responses were received for a single invocation, which is an error condition.

If handling multiple responses is a requirement of your solution, you should consider using WebSphere Message Broker.

Building Log Registration mediation

This chapter shows an example of using messaging to Access Services. In previous chapters, SOAP/JMS and SOAP/JMS were the transport mechanisms used. In this chapter we look at using JMS to put messages on a queue. The queues exist on the service integration bus during the testing phase. Later in Chapter 11., “Deploy with WebSphere ESB” on page 479, we show you how to integrate WebSphere MQ into the environment.

The mediation built for this example illustrates the following:

- ▶ SOAP/JMS and JMS transport
- ▶ One-way operation
- ▶ XSL transformation of the message
- ▶ Routing messages
- ▶ Message logging in the mediation
- ▶ Fault handling
- ▶ WebSphere MQ integration

This chapter includes the following topics:

- ▶ Scenario overview
- ▶ Developing the mediation
- ▶ Testing the mediation
- ▶ Calling the service from the application

10.1 Scenario overview

This chapter focuses on the Log Registration scenario of the ITSOMart solution.

Note: The samples included in this chapter can be downloaded from the Web. See Appendix A, “Sample application install summary” on page 589, for instructions on downloading and importing the sample projects.

10.1.1 Business scenario

The first step in the Register Customer process is to evaluate the customer credit rating. The rating is used to determine how to proceed with the registration. This step of the process was built in Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263.

The next step evaluates the credit rating and determines what action should be taken with the request. The possibilities are to deny registration, send the registration to a queue for manual handling, or register the customer. This step of the process was built in Chapter 8, “Building the CRM mediation” on page 347.

In the event that a customer has a gold credit rating, the previous step registered the customer in the CRM. The address registered is the customer billing address. The Register Shipping scenario supplements that by registering the shipping addresses for the customer. This step of the process was built in Chapter 9, “Building the Register Shipping mediation” on page 417.

Last, the transaction is logged into an audit system. There are three possible values for the status of the customer registration:

- ▶ **SUCCESS:** The customer has a gold or silver credit rating.
- ▶ **DENIED:** The customer has a bronze credit rating.
- ▶ **FAILURE:** An exception has occurred during normal processing. For example, an exception is thrown by the CRM mediation if the credit rating is gold and the Siebel system is not available. A FAILURE condition will also occur if the Credit Score mediation throws an exception during the database lookup for a *key not found* condition.

A separate log is kept for each status. A mediation is used to route the request to the proper log in the audit system.

The activity diagram for the Log Registration scenario of the ITSOMart solution can be seen in Figure 10-1. The activity diagram for the entire process can be seen in Figure 5-15 on page 143.

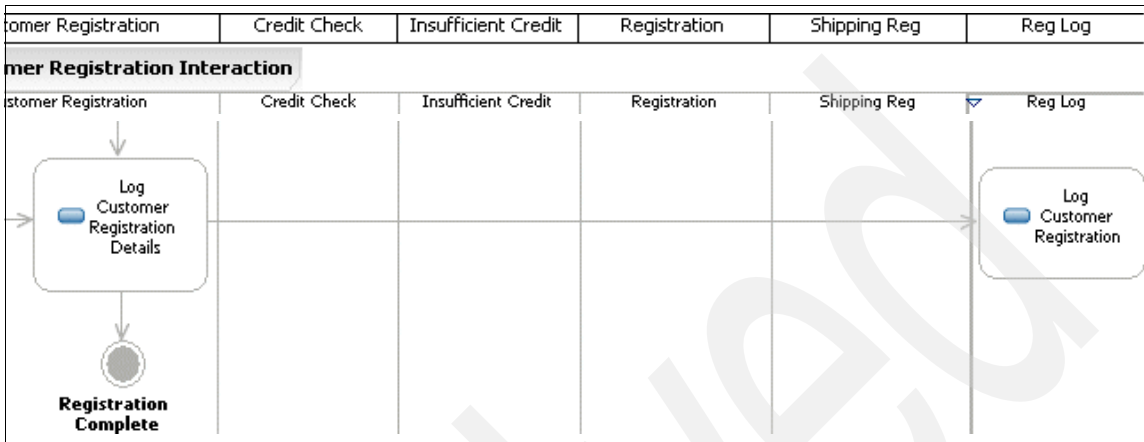


Figure 10-1 Log Registration scenario

10.1.2 Log Registration mediation

The call from the Register Customer process will call the Audit System service via the ESB. Since the log used to record the transaction depends on the status of the customer registration request, a mediation will be used to route the request to the proper log.

Figure 10-2 shows the activity diagram for the mediation.

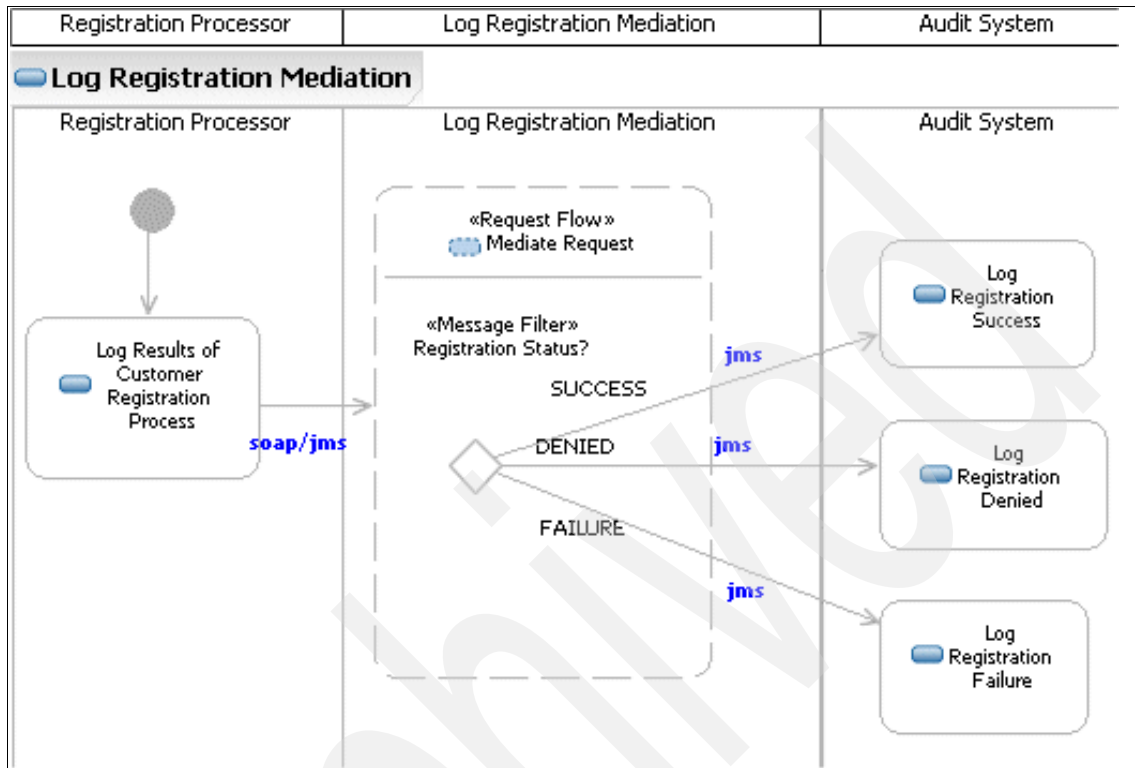


Figure 10-2 Log Registration mediation activity diagram

The mediation is invoked using SOAP/JMS. The audit system uses messaging and WebSphere MQ for transport.

10.2 Developing the mediation

This sample demonstrates how a mediation flow can be invoked as a SOAP/JMS Web service and then have the inbound message payload passed on to WebSphere MQ queues. We do this by binding import components to JMS queues and then configuring an MQLINK on the service integration bus to move messages over to queues that reside in a WebSphere MQ queue manager.

We also show how enumerations can be used in a business object to provide constraints on attribute values (a scenario in which you can use the Stop mediation primitive to dispose of invalid messages) and how to perform content-based routing to route messages to different JMS queues.

Figure 10-3 shows the mediation module contents we are going to build.

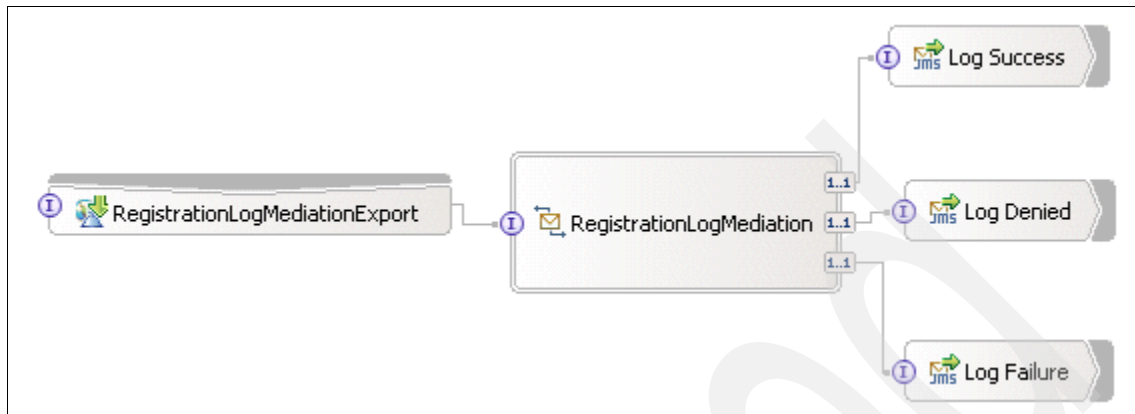


Figure 10-3 Log Registration mediation assembly diagram

The steps required to build the Log Registration mediation are:

1. Create the mediation module.
2. Create the business object.
3. Build the interface.
4. Assemble the mediation components.
5. Bind the imports to JMS.
6. Build the mediation flow.

Preparation

This mediation example uses the library ITSOMartLib (see 7.2.1, “Create a library” on page 268). This library contains business objects common to all the sample scenarios in this book. If you are working in a new workspace, you must import ITSOMartLib into your workspace or recreate it and the business objects it contains.

This chapter assumes that you have been through the earlier samples and are familiar with the steps required to build and edit mediations, business objects, and interfaces. If you are not familiar with these steps, review Chapter 6, “Assemble with WebSphere Integration Developer” on page 211, and Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263.

10.2.1 Create the mediation module

The first step is to create a mediation module. The interfaces and business objects for this mediation will be added to the Library project, ITSOMartLib. You will add this library as a dependency when you create the mediation module.

Create a mediation module by right-clicking in the Business Integration view and selecting **New** → **Mediation module**.

1. Name the mediation module ITS0_RegLogMed and be sure to check the box that allows the wizard to create the mediation flow component.
2. Select **WebSphere ESB Server** as the target runtime and click **Next**.
3. Check the box by the library ITSOMartLib and click **Finish**.

10.2.2 Create the business object

We only need one additional business object for this mediation. This is the business object that will be passed to the mediation.

1. Create a business object in ITSOMartLib:
 - a. Enter RegistrationLogRequest as the name.
 - b. Add the attributes in Table 10-1.

Table 10-1 Attributes to add

Attribute	Type
customer	Customer
creditRating	string
registrationStatus	string
registrationStatusDetails	string
registrationDate	dateTime

This business object should now look like Figure 10-4.

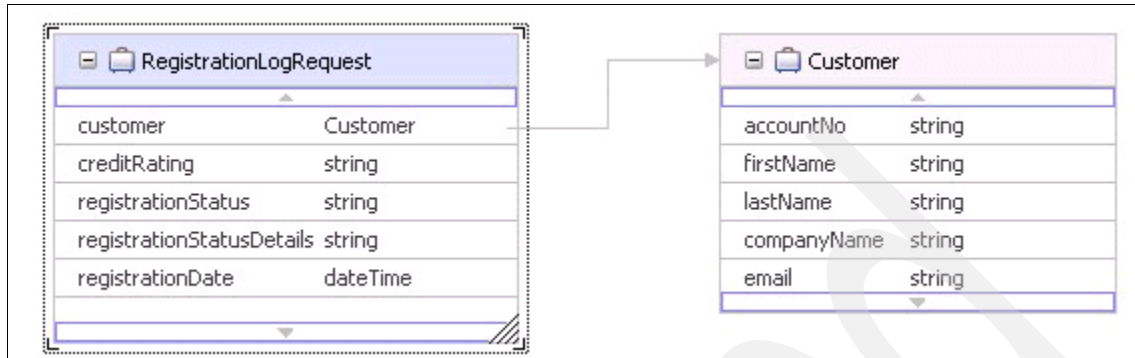


Figure 10-4 RegistrationLogRequest business object

2. Select the registrationStatus attribute and look at its properties in the Properties view.

A useful feature we want to demonstrate with this business object is the use of enumerations as a type of constraint that you can specify on the values that can be entered for this attribute.

- a. Indicate that this attribute is required by selecting the **Required** check box.
- b. Select **Only permit certain values** and then **Enumerations**.

- c. Click **Add**. A new string called *value* will appear in the window. Type SUCCESS over the name. Click **Add** two more times to enter DENIED and FAILURE.

Attribute - registrationStatus

Name:

Type:

☒ Required ☐ Array

Minimum length

Maximum length

☐ Collapse whitespace

☒ Only permit certain values

☒ Enumerations

☐ Patterns

SUCCESS
DENIED
FAILURE

Add
Edit
Remove

Figure 10-5 Enumerations

- d. Save and close the business object.

You will see how these enumerations can be used later when you build the mediation flow.

10.2.3 Build the interface

We only need one interface for this mediation. This will be the interface to the mediation as well as the interface to all of the import components with which the mediation will interact.

1. Create an interface in ITSOMartLib. Enter RegistrationLog as the name.
2. Define a one-way operation. Enter logCustomerRegistration as the name.
3. Add an input parameter:
 - a. Enter registrationLogRequest as the name.
 - b. Select RegistrationLogRequest as the type.

The RegistrationLog interface should look like Figure 10-6.

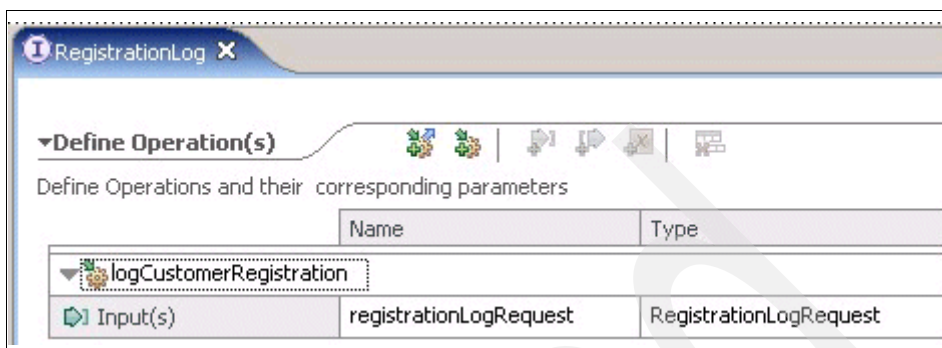


Figure 10-6 RegistrationLog interface

4. Save and close the interface.

10.2.4 Assemble the mediation components

Next we add the SCA components (export, imports, and mediation flow component) to the assembly diagram and wire them together.

1. Open the assembly diagram by double-clicking it in the Business Integration view.



Figure 10-7 ITSO_RegLogMed module assembly

2. Select the mediation flow component. In the Properties view, change the display name from Mediation1 to RegistrationLogMediation.
3. Add the RegistrationLog interface to the mediation:
 - a. Select **RegistrationLogMediation** in the assembly diagram.
 - b. Click the **Interface** pop-up.



- c. Select **RegistrationLog** from the list of interfaces.
- d. Click **OK**.

An alternative method is to drag and drop the interface onto the mediation component. Make sure you drop it on the component and not in the window as a new component.

4. Create an export for the RegistrationLogMediation with a Web service binding using SOAP/JMS for the transport.
 - a. Right-click **RegistrationLogMediation**.
 - b. Select **Export** → **Web Service Binding**.
 - c. Select **Yes** in the Binding File Generation window to have a wsdl file automatically generated.
 - d. Select **soap/jms** for the transport.
 - e. Click **OK**.

The export component called RegistrationLogMediationExport will be created.

5. Add the first of three imports to the assembly diagram with the RegistrationLog interface.
 - a. Select the RegistrationLog interface under the ITSOMartLib project, then drag and drop it onto the assembly diagram.
 - b. Select **Import with no Binding**.
 - c. Rename the import component to Log Success.
6. Repeat the process to add an import component named *Log Denied*.
7. Repeat the process to add an import component named *Log Failure*.

8. Wire all three import components to the RegistrationLogMediation. The assembly diagram should now look like Figure 10-8.

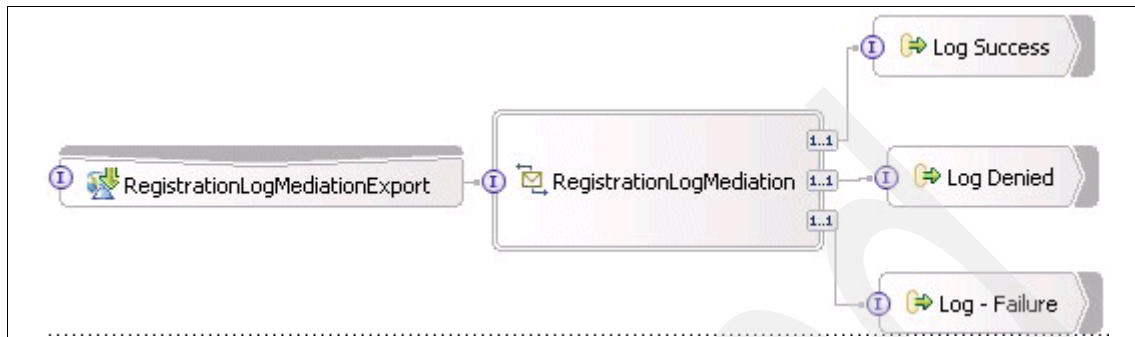


Figure 10-8 ITSO_RegLogMed assembly diagram

9. Save the assembly diagram and leave it open.

10.2.5 Bind the imports to JMS

Now we bind all of the imports to JMS, which means that when the mediation passes its message from the mediation flow to these imports, the outbound message will be serialized in XML format and sent to a JMS queue.

1. Select the **Log Success** import in the assembly diagram, right-click, and select **Generate Bindings** → **JMS Binding**.
 - a. In the JMS Binding Attributes Selection window, select **Point-to-Point** as the JMS messaging domain and **Text** for data serialization.

Note: There are predefined JMS bindings to support JMS text messages containing Business Object (BO) XML and JMS object messages containing serialized Java Business Objects. You can use JMS custom bindings to support other types of JMS message. For information about custom bindings for JMS, see the WebSphere Enterprise Service Bus 6.0.1 Information Center article titled *JMS Custom Bindings* at:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/topic/com.ibm.websphere.wesb.doc/concepts/cwesb_jmscustombindings.html

- b. Click **OK**.
2. In the Properties view for the Log Success import:
 - a. Select the **Binding** tab.

- b. In the JMS Import Binding tab, enter `jms/ITSOMart/LogQCF` as the JNDI lookup name.

This is the queue connection factory that the JMS binding will use to establish a connection to the underlying JMS provider, which in this case is the default JMS provider of WebSphere Application Server.

The screenshot shows the 'Properties' window for a JMS Import Binding. The 'Description' tab is active, displaying 'Import: Log Success (JMS Binding)'. The 'Details' tab is also visible, showing the 'JMS Import Binding' configuration. The 'Binding' section is expanded, revealing the following fields:

JMS Import Binding	
Adapter Type:	WebSphere Default Messaging Provider version 0.3
Connection	
JNDI Lookup Name:	<code>jms/ITSOMart/LogQCF</code>
Managed Connection Factory:	<code>com.ibm.ws.sib.api.jmsra.impl.JmsJcaManagedConnectionFactory</code>
JMS Destinations	
Send Destination Type:	<code>javax.jms.Queue</code>
Data Type:	<code>com.ibm.websphere.sca.jms.data.impl.JMSDataBindingImplXML</code>
Binding Description:	

Figure 10-9 Binding: JMS Import Binding

- c. Select the **JMS Destinations** tab, expand **Send Destination Properties**, and enter `jms/ITSOMart/LogSuccessQ` as the JNDI lookup name.

The screenshot shows the 'Properties' window for 'Import: Log Success (JMS Binding)'. The 'JMS Destinations' tab is selected, and the 'Send Destination Properties' section is expanded. The 'JNDI Lookup Name' is set to 'jms/ITSOMart/LogSuccessQ' and the 'Type' is 'javax.jms.Queue'. The 'Destination Properties' section includes 'Bus Name', 'Delivery Mode' (set to 'Application'), 'Time To Live', and 'Priority' (set to 'Default'). The 'Advanced Messaging' section includes 'Read Ahead' (set to 'AsConnection'). The 'Queue' section includes 'Queue Name' (set to 'ITSO_RegLogMed.LogSuccess_SEND_D').

Figure 10-10 Binding: JMS destinations

3. Generate a JMS binding for the other two imports, the Log Denied and Log Failure imports, by repeating the steps above. Use the values in Table 10-2 for the binding properties.

Table 10-2 Values for the binding properties

Import name	JMS import binding JNDI lookup name	JMS destinations JNDI lookup name
Log Denied	jms/ITSOMart/LogQCF	jms/ITSOMart/LogDeniedQ
Log Failure	jms/ITSOMart/LogQCF	jms/ITSOMart/LogFailureQ

The complete assembly diagram with the JMS bindings should now look like Figure 10-11.

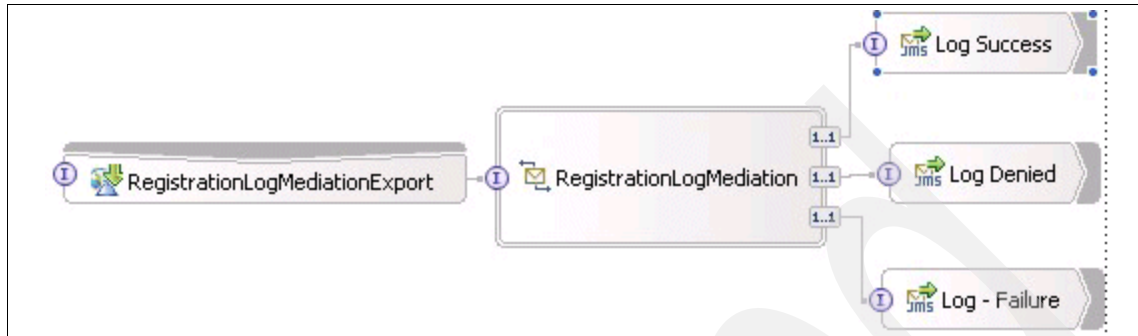


Figure 10-11 ITSO_RegLogMed completed assembly diagram

4. Save the assembly diagram.

10.2.6 Build the mediation flow

Next create and implement the mediation flow component.

1. Right-click **RegistrationLogMediation** and select **Generate Implementation**. Click **OK** in the next window and the Mediation Flow editor will open.

2. Click the **logCustomerRegistration** operation of the source RegistrationLog interface on the left and drag a connection to each of the logCustomerRegistration operations of the RegistrationLogPartner references on the right.

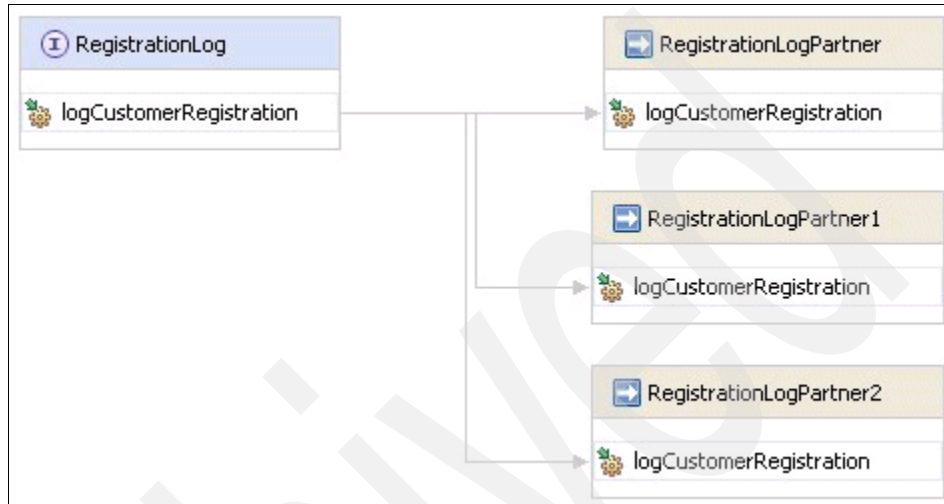


Figure 10-12 RegistrationLogMediation Operation connections

3. Select the logCustomerRegistration operation of the source RegistrationLog interface to display the flow in the bottom pane. Notice that there is only a request flow labeled Request: logCustomerRegistration and no response flow for this mediation. This is because the logCustomerRegistration operation on the RegistrationLog interface is a one-way operation.
4. Drop a Message Logger primitive onto the request flow editor.
 - Change its display name to Log Mediation Request.
 - Wire the RegistrationLog_logCustomerRegistration_Input output terminal to the Log Mediation Request input terminal.
5. Drop a Message Filter primitive onto the flow.
 - Change its display name to Route Message.
 - Wire the Log Mediation Request output terminal to the Route Message input terminal.
6. Add three terminals to the filter:
 - a. Right-click the **Message Filter primitive** and select **Add Output Terminal**.
 - b. Change the Terminal name to registrationSuccess.

- c. Repeat to add an output terminal called registrationDenied.
- d. Repeat to add an output terminal called registrationFailure.
7. In the Properties view for the Message Filter primitive, select the **Details** tab.
 - a. Keep the Distribution mode set to *First*.
 - b. Under Filters, click **Add** to add a filter pattern for the registrationSuccess terminal.
 - c. In the Terminal field, select the **registrationSuccess** terminal.
 - d. Click **Custom XPath** to specify the filter pattern.
 - e. In the XPath Expression Builder, select **body/logCustomerRegistration/registrationLogRequest/registrationStatus**.

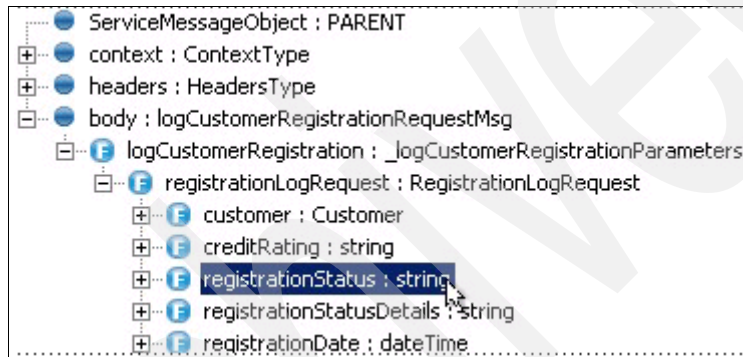


Figure 10-13 Route Message filter pattern XPath Location

- f. In the Condition field, click **Location** and select **self::node()**.

XPath Location:	/body/logCustomerRegistration/registrationLogRequest/registrationStatus		
Condition:	Location	=	{Value}
Full XPath Expression	<div> <div>+</div> <div>self::node() : string</div> </div>		

Figure 10-14 Route Message filter pattern XPath Condition

- g. Click **Value** and select **SUCCESS**.

Note that the XPath Expression Builder displays the enumerated values you defined for the registrationStatus attribute when you created the RegistrationLogRequest business object.

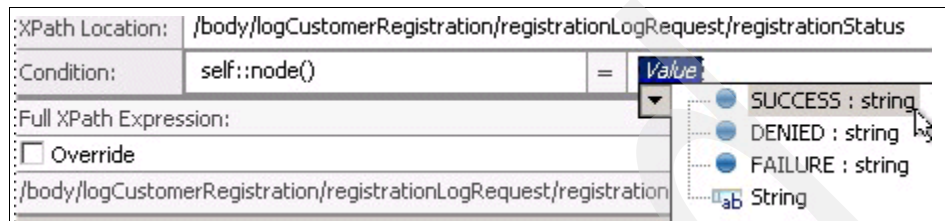


Figure 10-15 Route Message filter pattern XPath Value

- h. Click **OK** to close the XPath Expression Builder, then **Finish** to complete adding the filter pattern.
8. Add two more filter patterns, one for the registrationDenied and another one for the registrationFailure terminals. For each make sure that you select the appropriate terminal and the appropriate matching value.

Once you have added filter patterns for all three output terminals, the Filters section should look like Figure 10-16.

tern	Terminal name
dy/logCustomerRegistration/registrationLogRequest/registrationStatus[self::node()="FAILURE"]	registrationFailure
dy/logCustomerRegistration/registrationLogRequest/registrationStatus[self::node()="DENIED"]	registrationDenied
dy/logCustomerRegistration/registrationLogRequest/registrationStatus[self::node()="SUCCESS"]	registrationSuccess

Figure 10-16 Route Message filter patterns

9. Drop a Message Logger primitive onto the editor to the right of the Route Message filter.
- Change its display name to Log No Status.
 - Connect the default terminal of the message filter to the Log No Status input terminal.

The message will get routed to the default output terminal only if no other matching filter patterns are met for the other three output terminals.

10. Drop a Stop primitive onto the editor to the right of the Log No Status message logger.
- Change its display name to No Status.

- b. Connect the output terminal of the Log No Status message logger to the input terminal of the Stop primitive.

In this mediation example, the Stop primitive will handle the condition where the message that came into the mediation flow contains invalid or missing data. In this case, if the registrationStatus value is not set to one of the defined enumerated values, then the mediation flow stops and effectively disposes of that invalid message.

11. Drop another Message Logger primitive in the mediation flow to the right of the Route Message filter.
 - a. Change its display name to Log Reg Success.
 - b. Connect the registrationSuccess output terminal of the filter to the Log Reg Success input terminal.
 - c. Connect the Log Reg Success output terminal to the RegistrationLogPartner_logCustomerRegistration_Callout input terminal.
12. Repeat this process to log the messages for each of the other two message filter terminals, registrationDenied and registrationFailure. Then pass the message to the corresponding partner callout input terminal.

The completed request flow should look similar to Figure 10-17.

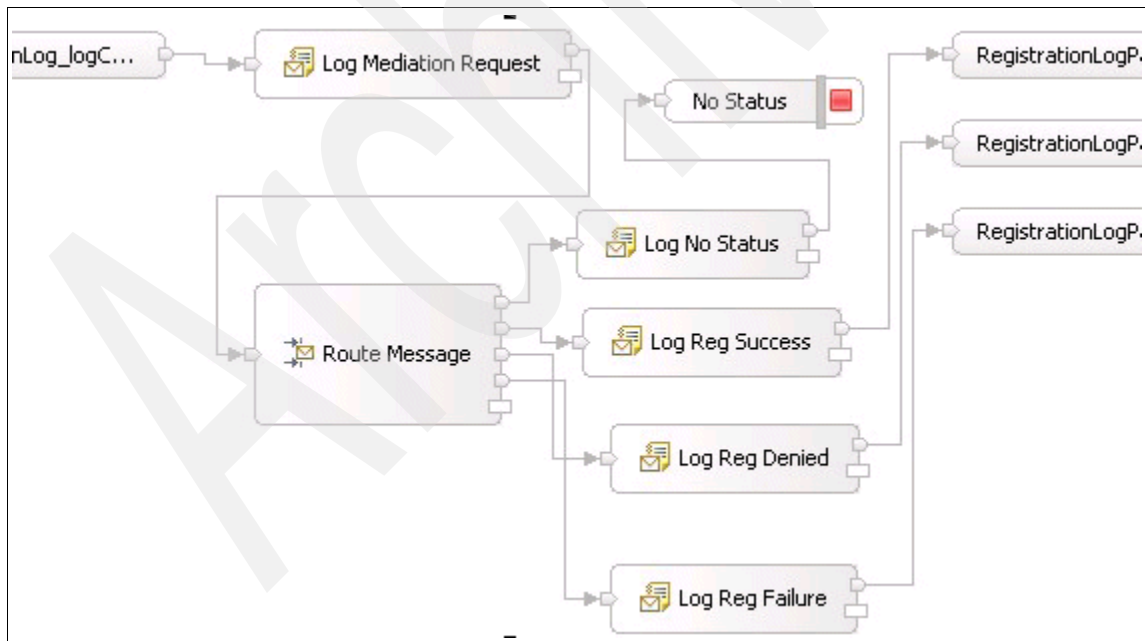


Figure 10-17 RegistrationLogMediation request flow

13. Save the mediation flow and the assembly diagram. Rebuild the project.

10.3 Testing the mediation

To test the mediation in the WebSphere Integration Developer test environment, you will first need to prepare the runtime. Then you can use the integration test client for testing.

10.3.1 Prepare the runtime

Before deploying and testing this mediation, there are definitions that need to be made on the server.

Note: For testing we use the `SCA.APPLICATION.<cell>.Bus` provided with WebSphere ESB and assume that the queues reside on the bus. Later, in Chapter 11., “Deploy with WebSphere ESB” on page 479, you will see how to create a new bus for use with the sample and connect it to WebSphere MQ.

To prepare the test environment for testing the mediation, do the following:

1. Start the WebSphere ESB server from the Servers view and log in to the administrative console.
2. Create three queue destinations on `SCA.APPLICATION.<cell>.Bus` with the following identifiers:

- `ITSOMart.LogSuccessQ`
- `ITSOMart.LogDeniedQ`
- `ITSOMart.LogFailureQ`

If you are not familiar with how to create queues on the bus, see 11.8.1, “Create a queue destination on the bus” on page 502.

3. Create a JMS queue connection factory for the default messaging provider with the following values:

- Name: `ITSOMartLogQCF`
- JNDI Name: `jms/ITSOMart/LogQCF`
- Bus name: `SCA.Application.<cell>.Bus`

If you are not familiar with how to create a queue connection factory, see 11.8.2, “Create a queue connection factory” on page 503.

4. Create three JMS queues under the default messaging JMS provider with the values shown in Table 10-3. For each definition, use the SCA.APPLICATION.<cell>.Bus bus.

Table 10-3 Registration Log Mediation JMS queues

JMS queue name	JNDI name	Queue name
ITSOMart.LogSuccessQ	jms/ITSOMart/LogSuccessQ	ITSOMart.LogSuccessQ
ITSOMart.LogDeniedQ	jms/ITSOMart/LogDeniedQ	ITSOMart.LogDeniedQ
ITSOMart.LogFailureQ	jms/ITSOMart/LogFailureQ	ITSOMart.LogFailureQ

If you are not familiar with how to create a JMS queue, see 11.8.3, “Create a JMS queue” on page 504.

10.3.2 Test the mediation

To do this:

1. Start the WebSphere ESB Server test environment.
2. Add MessageLogApp to the server. This sample application allows you to easily monitor the entries logged in the Cloudscape database by the Message Logger primitives.
3. Add the mediation application, ITSO_RegLogMedApp, to the server.
4. Launch the test client for the ITSO_RegLogMed module.
5. Set the values in the test client.
 - a. Make sure to select the RegistrationLogMediationExport component so that you are testing the complete module, from the export through the mediation and all the way to sending the messages to the queues.

- b. Fill in some test data for the request (Figure 10-18). The only value actually required for this mediation to work properly is the registrationStatus.

Select the component, interface, and operation you would like to invoke. Click Continue to run

Events Invoke

General Properties

Detailed Properties

Configuration: Default Module Test

Module: ITSO_RegLogMed

Component: RegistrationLogMediationExport

Interface: RegistrationLog

Operation: logCustomerRegistration

Initial request parameters

Name	Type	Value
<input type="checkbox"/> customer	Customer	
accountNo	string	1111
firstName	string	Ryan
lastName	string	Cox
companyName	string	IBM
email	string	ryan@ibm.com
password	string	
<input type="checkbox"/> billingAddress	BillingAddress	
name	string	Ryan
street	string	4th St
city	string	Boulder
state	string	CO
zipcode	string	80000
country	string	US
phone	string	303-333-3333
shippingAddr...	ShippingAddress []	<null>
creditRating	string	GOLD
registrationStatus	string	SUCCESS

SUCCESS
DENIED
FAILURE
<null>

Figure 10-18 Test client - RegistrationLogMediationExport

6. Check the test client Events pane to verify that everything ran successfully. You can see that the export request was made, then the mediation request, then finally one of the imports (Log Success, Log Denied, or Log Failure) was invoked.

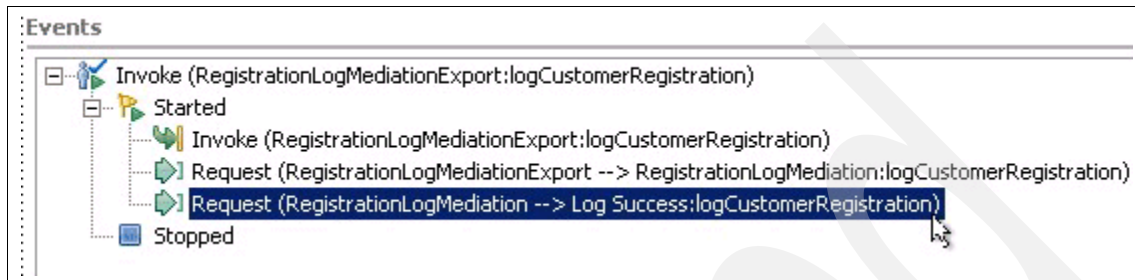


Figure 10-19 Test client - Events

7. Look at the messages that were logged inside the mediation using the Message Logger primitives.

Open the sample MessageLogApp:

<http://localhost:9080/MessageLogWeb/faces/MessageLogView.jsp>

You should see the log messages in descending order of which they were inserted into the database.

In Figure 10-20 you can see messages that were logged by the Log Mediation Request and then the Log Reg Success Message Logger primitives.

ESB Mediation Message Log				
Refresh		Clear Log		
	TIMESTAMP	MESSAGE ID	MEDIATION	MODULE
	Mar 16, 2006 3:32:57 AM MST	15809d02-0a01-0000- 0080-865904a0c467	Log Reg Success	ITSO_RegLogMed
	Mar 16, 2006 3:32:57 AM MST	15809d02-0a01-0000- 0080-865904a0c467	Log Mediation Request	ITSO_RegLogMed
Page 1 of 1 1				

Figure 10-20 Message Logger messages

8. Click anywhere in one of the rows of the table displayed to see the contents of the message that was logged. For example, the message for the Log Reg Success entry verifies that the data in the mediation flow was correctly passed.

```
<?xml version="1.0" encoding="UTF-8"?>
<body xsi:type="log:logCustomerRegistrationRequestMsg" xmlns:log="http://ITSOMartLib/RegistrationLog"
instance="http://ITSOMartLib/RegistrationLog">
  <logCustomerRegistration>
    <registrationLogRequest>
      <customer>
        <accountNo>1111</accountNo>
        <firstName>Ryan</firstName>
        <lastName>Cox</lastName>
        <companyName>IBM</companyName>
        <email>ryan@ibm.com</email>
        <password></password>
        <billingAddress>
          <name>Ryan</name>
          <street>4th St</street>
          <city>Boulder</city>
          <state>CO</state>
          <zipcode>80000</zipcode>
          <country>US</country>
          <phone>303-333-3333</phone>
        </billingAddress>
      </customer>
      <creditRating>GOLD</creditRating>
      <registrationStatus>SUCCESS</registrationStatus>
      <registrationStatusDetails>Customer registered</registrationStatusDetails>
      <registrationDate>2002-01-01T18:00:00.02</registrationDate>
    </registrationLogRequest>
  </logCustomerRegistration>
</body>
```

Figure 10-21 Log Reg Success log message

9. And, finally, verify that the message was placed on the queue correctly by the JMS import component.

Viewing messages on the bus

If you configured the JMS queues with the local bus queue destinations, then the you should be able to see the message on the service integration bus queue.

You can use the queue browser portion of MessageLogApp to see the message, or you can use the WebSphere administrative console.

Using the sample queue browser

To use MessageLogApp:

1. Open the application with the following URL:
`http://localhost:9080/MessageLogWeb/faces/QueueBrowser.jsp`
2. Enter the value for the JMS queue connection factory: `jms/ITSMart/LogQCF`
3. Enter the JMS queue JNDI name: `jms/ITSMart/LogSuccessQ`

All messages on the queue you specified are displayed. Scroll down to the last message in the window to see the last message placed on that queue.

Figure 10-22 shows the registration SUCCESS message placed on ITSOMart.LogSuccessQ. It contains the contents of the original message payload that was sent to the mediation and the JMS-specific header attributes.

Queue Browser

JMS Connection Factoryjms/ITSOMart/LogQCF

JMS Queuejms/ITSOMart/LogSuccessQ

Message:
JMSMessageId: ID:650fefefc2eace052a7d1e82110a134f00000000000000
JMSDeliveryMode: 2
JMSDestination: queue://ITSOMart.LogSuccessQ?busName=ITSOMart:
JMSReplyTo: null
JMSTimestamp: 1142505177828
JMSType: null
property - name: IsBusinessException, type: java.lang.Boolean
property - name: JMS_IBM_System_MessageID, type: java.lang.St:
03CC490E7229352F_14500002
property - name: JMS_IBM_MsgType, type: java.lang.Integer, va
property - name: JMSXAppID, type: java.lang.String, value: Se
property - name: JMSXDeliveryCount, type: java.lang.Integer, ·
property - name: TargetFunctionName, type: java.lang.String, ·
message type: TextMessage
body: <?xml version="1.0" encoding="UTF-8"?>
<request:RegistrationLogRequest xmlns:request="http://ITSOMartLib/Regi:
 <customer>
 <accountNo>1111</accountNo>
 <firstName>Ryan</firstName>
 <lastName>Cox</lastName>
 <companyName>IBM</compar.yName>

Figure 10-22 Message on the bus queue

Using the administrative console

You can also find messages on a queue in the bus using the administrative console:

1. Select **Service integration bus** → **Buses**.
2. Click the bus name.
3. In the Topology section, click **Messaging engines**.
4. Click the messaging engine name for your server.

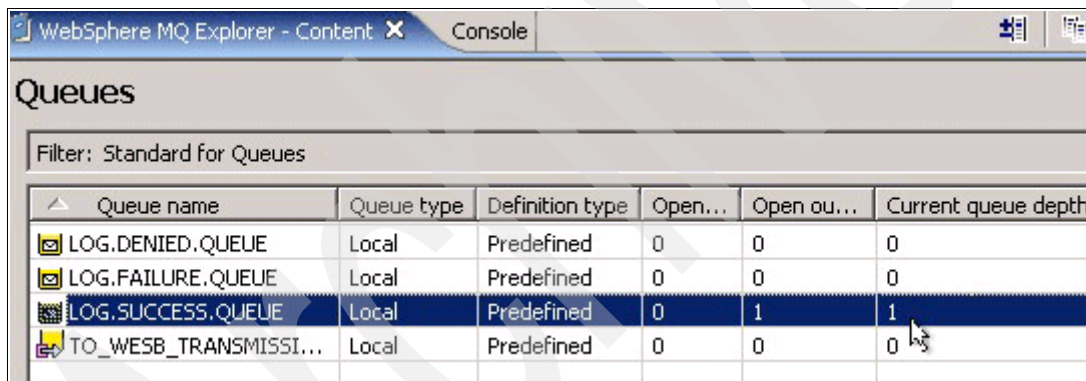
5. In the Message points section, click **Queue points**.
6. There is a queue point for each queue. Click the queue point name.
7. Select the **Runtime** tab.
8. Click **Messages**.

Viewing messages in WebSphere MQ

For testing we used queues on the bus. In Chapter 11., “Deploy with WebSphere ESB” on page 479, you can find instructions on using an MQ Link from the bus to WebSphere MQ to send messages to queues on WebSphere MQ. The queues defined on the bus are alias queues versus local queues. The alias queues contain references to queues on WebSphere MQ.

If you have this configuration, you will need to view the messages on the WebSphere MQ queues. To view the messages:

1. Open the WebSphere MQ Explorer and look at the messages on LOG.SUCCESS.QUEUE (Figure 10-23).



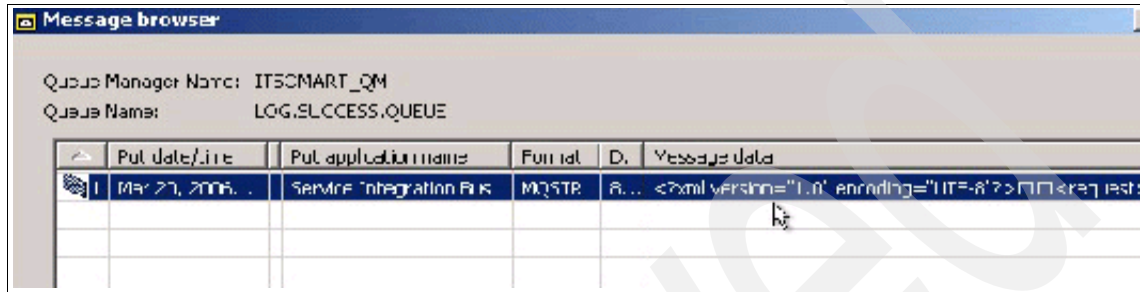
The screenshot shows the 'Queues' section of the WebSphere MQ Explorer. A filter 'Standard for Queues' is applied. A table lists four queues: LOG.DENIED.QUEUE, LOG.FAILURE.QUEUE, LOG.SUCCESS.QUEUE, and TO_WESB_TRANSMISSI... The LOG.SUCCESS.QUEUE row is selected, showing a current queue depth of 1. A mouse cursor is pointing at the '1' in the 'Current queue depth' column.

Queue name	Queue type	Definition type	Open...	Open ou...	Current queue depth
LOG.DENIED.QUEUE	Local	Predefined	0	0	0
LOG.FAILURE.QUEUE	Local	Predefined	0	0	0
LOG.SUCCESS.QUEUE	Local	Predefined	0	1	1
TO_WESB_TRANSMISSI...	Local	Predefined	0	0	0

Figure 10-23 Message on WebSphere MQ queue

2. Select the **LOG.SUCCESS.QUEUE**, right-click, and select **Browse Messages** to see the XML message that was sent from the Registration Log Mediation (Figure 10-24).

Notice that the Put application name shows that the message was sent from the service integration bus, and the data is the XML representation of the RegistrationLogRequest business object that was passed to the JMS import.



The screenshot shows a 'Message browser' window with the following details:

- Queue Manager Name: ITSMART_QM
- Queue Name: LOG.SUCCESS.QUEUE

	Put date/time	Put application name	Fun id	D.	Message data
1	May 27, 2006...	Service Integration Bus	MQSTR	8...	<?xml version="1.0" encoding="UTF-8"?><RegistrationLogRequest>

Figure 10-24 Message data on WebSphere MQ queue

10.4 Calling the service from the application

The Register Customer process application calls the Log Registration mediation via a Web service proxy that makes a SOAP/JMS request. This Web service proxy is generated based on the WSDL file that describes the mediation. This is the WSDL file generated when you create the export for the mediation, or if you are using the service integration bus to manage Web services, it is the WSDL file for the inbound service that defines the mediation.

To generate a Web service proxy that calls the Log Registration mediation, use the WSDL file

RegistrationLogMediationExport_RegistrationLogJms_Service.wsdl.

Your application also needs access to any WSDL files referenced in this file and the XSD files for the data objects used. The best way to pick up the files you need is to first go through the test process for the mediation. Then copy the files found under the wsdl folder for the mediation EJB to the application EJB.

You can find these files using the Physical Resources view under <mediationEJB>/ejbModule/wsdl.

The Register Customer process application, ITSO_RegProcServiceApp, uses a Java utility project to hold the proxies for the services it calls. This utility project is called ITSO_RegProcService_Proxies.

The steps required to generate the proxy are:

1. Copy or import the required files to the utility project.
2. Generate the Web service client proxy using the Web Service Client wizard. Use the service WSDL file as input.
3. Add code to the application to call the service via the proxy.

You can see an example of this process in 7.5, “Calling the service from the application” on page 341.

The following code in ITSO_RegProcServiceApp calls the service using the proxy (Example 10-1).

Example 10-1 Invoking the Register Shipping mediation

```
try {  
    // log customer registration details  
    RegistrationLogRequest registrationLogRequest = new RegistrationLogRequest();  
    registrationLogRequest.setCustomer(customer);  
    if (creditRating != null) registrationLogRequest.setCreditRating(creditRating);  
  
    // registration status  
    RegistrationStatus registrationStatus =  
        (registrationProcessStatus == FAILURE) ? RegistrationStatus.FAILURE  
        : (registrationProcessStatus == DENIED) ? RegistrationStatus.DENIED  
        : RegistrationStatus.SUCCESS;  
  
    if (registrationStatusDetails == null) {  
        registrationStatusDetails =  
            (registrationProcessStatus == FAILURE) ? "Failure occurred during  
registration process"  
            : (registrationProcessStatus == DENIED) ? "Customer denied registration"  
            : "Customer registered";  
    }  
  
    registrationLogRequest.setRegistrationStatus(registrationStatus);  
    registrationLogRequest.setRegistrationStatusDetails(registrationStatusDetails);  
    System.out.println("ITSOMart RegistrationProcessorService.registerCustomer >>  
registrationStatus: " + registrationStatus.getValue() + ", registrationStatusDetails: " +  
registrationStatusDetails);  
  
    // registration date  
    GregorianCalendar registrationDate = new GregorianCalendar();  
    registrationLogRequest.setRegistrationDate((Calendar)registrationDate);  
  
    RegistrationLogProxy registrationLogProxy = new RegistrationLogProxy();  
    System.out.println("ITSOMart RegistrationProcessorService.registerCustomer >>  
invoking RegistrationLog Service (soap/jms)");  
}
```

```

        registrationLogProxy.logCustomerRegistration(registrationLogRequest);

        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss a
zzzz");

        System.out.println("ITSOMart RegistrationProcessorService.registerCustomer >>
registration log submitted for customer (" + customer.getEmail() + ") at " +
dateFormat.format(registrationDate.getTime()));
    } catch (Exception registrationLogException) {
        System.out.println("ITSOMart RegistrationProcessorService.registerCustomer >>
RegistrationLog Service exception: " + registrationLogException);
    }
}
}

```

Deploy with WebSphere ESB

This chapter presents an overview of WebSphere ESB. It introduces its capabilities and discusses common topologies. It also illustrates how to integrate with a WebSphere MQ environment. To illustrate the concepts we discuss, this chapter describes the steps required to set up the ITSOMART sample, including configuring the deployment environment and deploying the mediations.

This chapter includes the following topics:

- ▶ Introduction to WebSphere ESB
- ▶ Working with profiles
- ▶ Administrative console
- ▶ Deploying mediation modules
- ▶ Creating a service integration bus
- ▶ Configuration for databases
- ▶ Configuration for adapter support
- ▶ Configuration for JMS bindings
- ▶ Connecting to WebSphere MQ
- ▶ Deploying applications
- ▶ Testing ITSOMart
- ▶ Network Deployment and clustering topologies

Note: This chapter assumes that you are familiar with the WebSphere Application Server and focuses on the enhancements for WebSphere ESB.

11.1 Introduction to WebSphere ESB

WebSphere ESB provides the capabilities of a standards-based enterprise service bus. WebSphere ESB manages the flow of messages between SCA-defined interaction endpoints and enables the quality of interaction these components request. Mediation modules within the ESB provide routing, protocol conversion, message format transformation (with and without database lookup), and logging services.

WebSphere ESB is a specialized application server built on WebSphere Application Server Network Deployment. As such, it inherits all of the qualities of service, including scalability, clustering, and failover features of a Network Deployment environment.

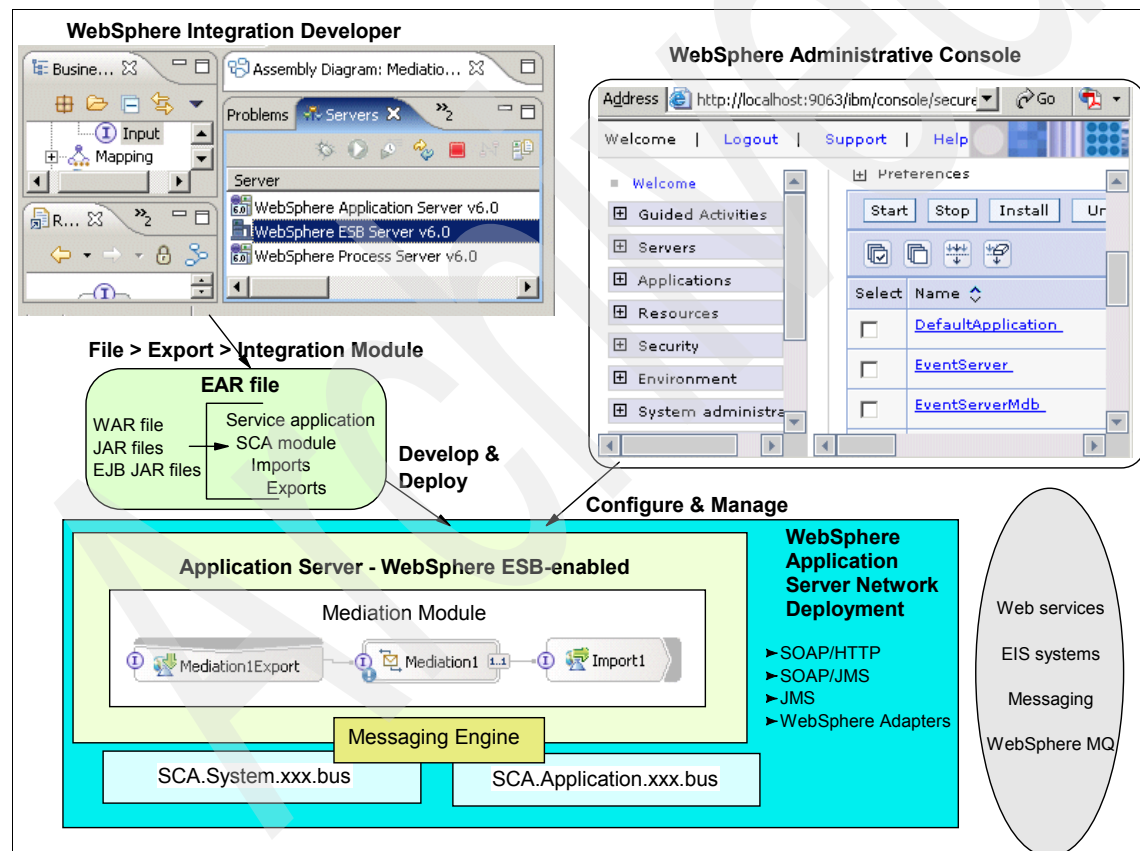


Figure 11-1 WebSphere ESB technical overview

11.1.1 Applications

The WebSphere ESB application server is an advanced application server that can run, not only traditional J2EE 1.4 applications, but also applications with SCA components.

- ▶ **Mediation modules**

Mediation modules are SCA modules that operate on messages that are in-flight between service requesters and service providers. They allow you to route messages to different service providers. They also let you transform messages: you can amend message content or form. In addition, mediation modules can provide functions such as message logging and error processing that is tailored to your requirements.

Using WebSphere Integration Developer you can develop mediation modules and package them for deployment. The underlying structure of the module is an EAR file.

- ▶ **J2EE applications**

A WebSphere ESB application server also contains the J2EE structure required to run J2EE 1.2, 1.3, and 1.4 applications. The application server is an enhanced version of the application server shipped with WebSphere Application Server Network Deployment. As such, it provides the full range of functions and features that come with that product, including configurations for scalability and high availability.

The standard and preferred development tool for WebSphere Application Server J2EE applications is Rational Application Developer. However, a set of J2EE development capabilities are also provided with other Rational and WebSphere development tools, including WebSphere Integration Developer and Rational Software Architect.

11.1.2 Administration

WebSphere ESB uses the WebSphere administrative tools available with WebSphere Application Server. Administration can be done using the browser-based administrative console, scripting, or commands. Enhancements have been made to the administrative interface to allow you to list installed SCA modules and their associated applications, view SCA module details, and modify SCA imports.

11.1.3 Service integration bus

The service integration bus provides the communication infrastructure for messaging and service-oriented applications, thus unifying this support into a common component. The service integration bus is a JMS provider that is JMS

1.1 compliant for reliable message transport and that has the capability of intermediary logic to adapt message flow intelligently in the network. It also supports the attachment of Web services requestors and providers. Service integration bus capabilities have been fully integrated within WebSphere Application Server, enabling it to take advantage of WebSphere security, administration, performance monitoring, trace capabilities, and problem determination tools.

The service integration bus is often referred to as just a *bus*. When used to host JMS applications, it is also often referred to as a *messaging bus*.

WebSphere ESB includes two buses by default:

- ▶ SCA.SYSTEM.<cell>.Bus
This bus is used as the underlying communication mechanism for asynchronous SCA invocations.
- ▶ SCA.APPLICATION.<cell>.Bus
This bus is provided and used to define queue destinations and other default messaging provider resources required for modules deployed with JMS bindings.

You can also create additional buses.

11.1.4 Web services support

WebSphere ESB also leverages advanced Web services support provided with WebSphere Application Server Network Deployment to incorporate leading edge capabilities. Those capabilities include:

- ▶ SOAP-based Web services (SOAP/HTTP, SOAP/JMS)
SOAP is a lightweight protocol for the exchange of information in a decentralized, distributed environment.
- ▶ Web Services Description Language (WSDL) 1.1
WSDL is an XML-based description language that provides a way to catalog and describe services. WSDL describes the interface of Web services (parameters and results), the binding (SOAP, EJB), and the implementation location.
- ▶ Universal Discovery Description and Integration (UDDI) 3.0
UDDI is a global platform-independent, open framework that enables businesses to discover each other, define their interaction, and share information in a global registry.

UDDI support in WebSphere Application Server V6 includes UDDI V3 APIs, some UDDI V1 and V2 APIs, UDDI V3 client for Java, and UDDI4J for compatibility with UDDI V2 registries. It also provides a UDDI V3 Registry that is integrated in the WebSphere Application Server.

► Web Services Gateway

WebSphere Application Server V6 includes a fully integrated Web Services Gateway that bridges the gap between Internet and intranet environments during Web service invocations. The administration is done directly from the WebSphere administrative console.

The primary function of the Web Services Gateway is to map an existing WSDL-defined Web service, the target service, to a new service, the gateway service, that is offered by the gateway to others. The gateway thus acts as a proxy. Each target service, whether internal or external, is available at a service integration bus destination. JAX-RPC handlers can be used to intercept and filter service messages as they pass in and out of the service integration bus.

► JAX-RPC (JSR 101)

JAX-RPC is the core programming model and bindings for developing and deploying Web services on the Java platform. It is a Java API for XML-based RPC and supports JavaBeans™ and enterprise beans as Web service providers.

► Enterprise Web services (JSR 109) adds EJBs and XML deployment descriptors to JSR 101.

► WS-Security

WS-Security is the specification that covers a standard set of SOAP extensions and can be used when building secure Web services to provide integrity and confidentiality. It is designed to be open to other security models including PKI, Kerberos, and SSL. WS-Security provides support for multiple security tokens, multiple signature formats, multiple trust domains, and multiple encryption technologies. It includes security token propagation, message integrity, and message confidentiality. The specification is proposed by IBM, Microsoft, and VeriSign for review and evaluation.

► IBM add-ons

In addition to the requirements of the specifications, IBM has added the following features to its Web services support:

– Custom bindings

JAX-RPC does not support all XML schema types. Custom bindings allow developers to map Java to XML and XML to Java conversions.

- Support for generic SOAP elements

In cases where you want generic mapping, this support allows you to eliminate binding and use the generic `SOAPElement` type.

- Multi-protocol support

This feature allows a stateless session EJB as the Web service provider, which provides enhanced performance without changes to the JAX-RPC client.

- Web services caching

The WebSphere Application Server provides server-side Web service caching for Web services providers running within the application server. It also provides caching for Web services clients running within a V6 application server, including the Web Services Gateway.

11.1.5 Messaging support

WebSphere ESB supports the use of the following JMS messaging providers:

- ▶ Default messaging provider

The default messaging provider integrated in WebSphere ESB is a pure Java-based JMS 1.1 implementation. The service integration bus provides the underlying message provider for the default messaging provider.

The JMS resources (JMS provider, JMS queues, and so on) are defined from the WebSphere administrative tools. The service integration bus destinations for the default messaging provider (queues, topics, and so on) are also managed using the WebSphere administrative tools.

The default messaging provider can be configured to connect to an existing WebSphere MQ Series network via an MQ link. The messaging provider looks to MQ like just another queue manager, enabling interoperability with every WebSphere MQ based application.

- ▶ WebSphere MQ

WebSphere ESB supplies a pre-configured JMS provider implementation for communicating with installations of WebSphere MQ.

- ▶ Generic JMS providers

WebSphere ESB supports the use of third-party JMS providers within its runtime environment through the use of a generic JMS provider. Defining a generic JMS provider ensures that the JMS provider classes are available on the application server classpath at runtime.

11.1.6 Client support

WebSphere ESB provides you with a comprehensive client package that allows you to extend your environment. The client package includes:

- ▶ A message service client for C/C++/.Net application, extending the JMS model for messaging to non-Java applications.
- ▶ A Web services client (JAX-RPC based) for C++, enabling user access to Web services hosted in the WebSphere environment.
- ▶ J2EE client support for WebSphere Application Server including the Web services client, EJB client, and JMS client.

11.1.7 Tivoli Access Manager

WebSphere ESB includes IBM Tivoli Access Manager, for optional use, to deliver a secure, unified, and personalized experience that will help manage growth and complexity, and it integrates with IBM Tivoli Composite Application Manager for SOA for added monitoring and management capabilities.

Managing composite applications based on service-oriented architectures presents a mix of new and traditional challenges, as services need to be treated by the management infrastructure as first-class managed objects, conforming to defined service characteristics. IBM Tivoli Composite Application Manager for SOA helps you monitor, manage, and control these service-based applications. This solution is integrated with IBM Tivoli Enterprise Portal, which enables end-to-end resource, application, transaction, and service management across your IT infrastructure.

11.1.8 Common Event Infrastructure (CEI)

Also in the infrastructure is the Common Event Infrastructure, which is the foundation for monitoring applications. IBM uses this infrastructure throughout its product portfolio, including monitoring products from Tivoli. The event definition (Common Business Event, or CBE) is being standardized through the OASIS standards body, so that other companies as well as customers can use the same infrastructure to monitor their environment.

11.2 Working with profiles

A profile describes the runtime environment for a server. The profile types are the same for WebSphere ESB as for WebSphere Application Servers. However, the profiles themselves are augmented for WebSphere ESB functions.

- ▶ Application server profiles describe a runtime environment that consists of a single application server.
- ▶ Custom profiles are empty nodes that are used to federate nodes into a Network Deployment cell. Once the node is federated, application servers can be created on it using the administrative console. Nodes that have been federated into a cell are referred to as managed nodes.
- ▶ Deployment manager profiles describe a deployment manager. There is one deployment manager per cell that manages the administration of the nodes and servers that belong to the cell. A cell can contain a mixture of WebSphere Application Server and WebSphere ESB nodes, but in this case the Deployment Manager must also be augmented with the WebSphere ESB functionality.

When you create an application server profile or a deployment manager profile, three buses are created:

- ▶ SCA.SYSTEM.<cell>.Bus, used for asynchronous SCA support
- ▶ SCA.APPLICATION.<cell>.Bus, used for default JMS bindings
- ▶ CommonEventInfrastructure_Bus, used for CEI events

The profile creation wizard can be used to create a new profile after WebSphere ESB has been installed. If you created the application server during the installation (Complete install option), you do not need to create another profile unless you decide to change the topology of your installation or augment an existing profile for WebSphere ESB. Note that in order to augment a WebSphere Application Server managed node, it must be unfederated first. After the augmentation it can be federated again.

When it comes to federating a custom ESB profile, the deployment manager must be a WebSphere ESB 6.0.1 or WebSphere Process Server 6.0.1 deployment manager node.

11.2.1 Starting the profile creation wizard

The profile creation wizard commands for WebSphere ESB can be found in the following location:

```
<wesb_install>/bin/ProfileCreator_wbi
```


The commands are:

- ▶ AIX®: **pcatAIX.bin**
- ▶ HP-UX platforms: **pcatHPUX.bin**
- ▶ Linux platforms: **pcatLinux.bin**
- ▶ Linux platforms (Power PC®): **pcatLinuxPPC.bin**
- ▶ Solaris™ platforms: **pcatSolaris.bin**
- ▶ Windows platforms: **pcatWindows.exe**

You can also start the profile creation wizard from the First Steps menu, and on Windows, you can use the Start menu and select **Start → All Programs → IBM WebSphere → Enterprise Service Bus 6.0 → Profile Creation Wizard**.

The profile creation wizard commands for WebSphere ESB in the WebSphere Integration Developer test environment can be found in the following location:

```
<wid_install>/runtimes/bi_v6/bin/ProfileCreator_wbi
```

The commands are:

- ▶ Windows: **esbpcatWindows.exe**
- ▶ Linux: **esbpcatLinux.bin**

11.3 Administrative console

Administration tasks can be performed from the WebSphere administrative console. The default URL to open the console for a running stand-alone server is:

```
http://localhost:9060/ibm/console
```

Where 9060 is the value of the WC_adminhost port for the server. If you have more than one server on the system or are running a deployment manager, this port number may be different.

If your application server is running in the WebSphere Integration Developer test environment, you can also open the console by doing the following:

1. Switch to the Business Integration or J2EE perspective and select the **Servers** view.
2. Right-click the **WebSphere ESB Server** entry and select **Start**. The Console view opens and you can see the messages from the startup of the server.
3. When you see the Open for e-business message, switch back to the Servers view. Right-click the server and select **Run administrative console**.

The administrative console provides a browser-based interface to the administration tools. You can install, uninstall, and manage applications and

resources for the application server. In a deployment manager environment, you can manage all the servers and clusters in the cell.

The administrative console for WebSphere ESB is almost identical to the administrative console for Network Deployment with a few minor changes.

On opening the administration console, the welcome screen presents a task filtering selector. The available filters are All, Application Integration, and Server and Bus (Figure 11-2). These filters can be used to reduce the complexity of the administration console by hiding functionality that is not applicable to the administrator's task.

We prefer using the All option.

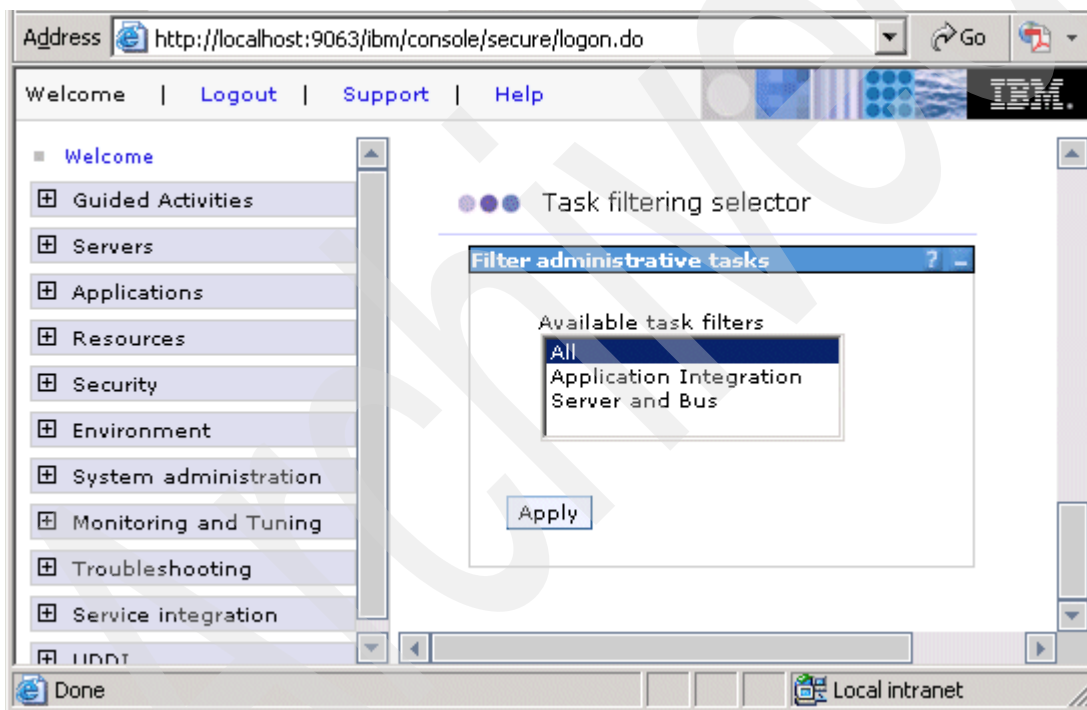


Figure 11-2 Administrative console task filtering selector

This filtering is in addition to any security constraints applicable to the user's administrative role. Once a filter has been selected, it can be modified again by returning to the Welcome screen. (You may have to scroll down to find it.)

Other new changes have to do with the SCA module support. For example, SCA modules can be displayed, started, and stopped from the console.

To navigate to an SCA module, select **Applications** → **SCA Modules** (Figure 11-3).

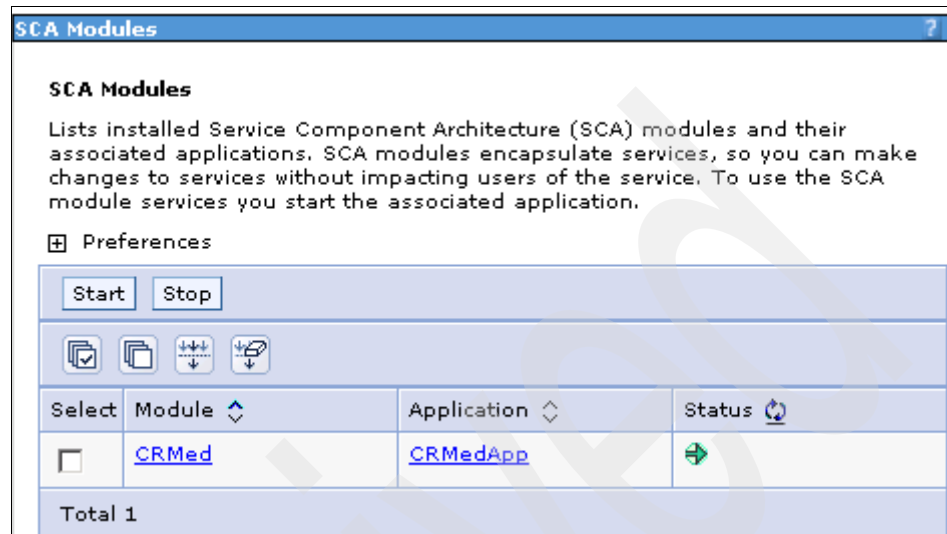


Figure 11-3 List of SCA modules

To view details of the SCA module, click the module name (Figure 11-4).

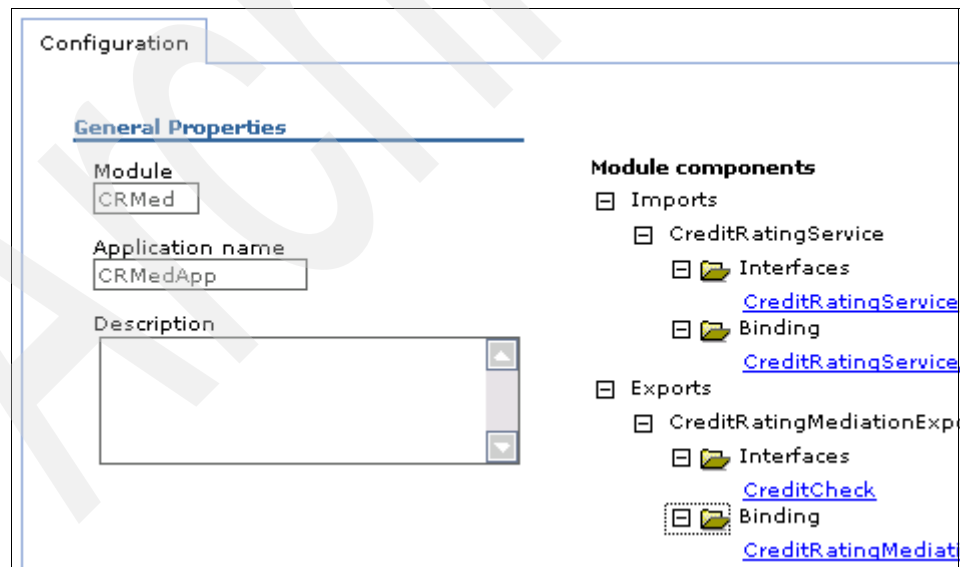


Figure 11-4 SCA module details

11.4 Deploying mediation modules

Depending on the mediation module, an administrator may have some tasks to complete before deploying the module. This will ensure a successful deploy and a working application.

The following sections on configuring the runtime for mediations and deploying mediation modules use the ITSOMart solution for illustration.

Assumptions for the sample: In the examples used here we assume the following:

- ▶ The server is a single server operating on a Windows operating system. If you are not using Windows you may need to refer to the WebSphere ESB product documentation for platform-specific information such as command locations or extensions.
- ▶ The profile name is **esb**. The home directory for the profile will be denoted as *<wesb_profile>*.
- ▶ The node name is **esbNode**. The server name is **server1** and port **9080** is used for access to the Web container.
- ▶ WebSphere ESB file locations will be denoted with *<wesb_install>*. For example, if you have WebSphere ESB installed, *<wesb_install>* will look similar to *C:\WebSphere\AppServer*. In the WebSphere Integration Developer test environment, this will be *<wid_install>\runtimes\bi_v6*.

When dealing with relatively new WebSphere features such as the service integration bus, we go into detail on how to perform these tasks. For other tasks that have remained much the same from previous versions of WebSphere such as creating JDBC data sources, we simply give you the information you need to populate those items.

11.5 Creating a service integration bus

The ITSOMart solution uses a service integration bus as the default JMS provider and for Web service destinations. When testing the mediation modules, we used a bus that is automatically included with WebSphere ESB. The name of this bus is **SCA.APPLICATION.<cell>.Bus**.

To deploy the ITSOMart solution, we have decided to create a new bus called **ITSOMartBus**. The following instructions will show you how to create this bus and add **server1** as a bus member.

From the WebSphere administrative console:

1. Select **Service integration** → **Buses**.
2. Click **New**.
3. Enter ITSOMartBus for the name and click **Apply**.
4. Click the **Bus members** link under Topology.
5. Click **Add**.
6. Click **Server** and select the server on which you will run the application. If you only have a standalone server, the defaults are correct. Click **Next**.
7. Click **Finish**.
8. Restart the server.

11.6 Configuration for databases

The Database Lookup primitive allows you to access a database from a mediation. The Credit Score mediation uses a Database Lookup primitive in the response flow to convert a numerical credit score to a text value (gold, silver, bronze). The instructions for creating the database used in this scenario can be found in “Create the DB2 database” on page 598).

The JNDI lookup name for the database is specified in the properties for the Database Lookup primitive (see Figure 7-65 on page 326). You will need an appropriate JDBC provider and a corresponding data source defined in the application server.

ITSOMart is set up to use a DB2 database or a Cloudscape database.

11.6.1 Create a J2C authentication data entry for the database

Most database products require a user ID and password for access (the default setup for Cloudscape does not). If your database requires this, create a J2C authentication entry containing the user ID and password.

1. From the administrative console select **Security** → **Global security**.
2. Expand the JAAS configuration on the right panel and click **J2C Authentication data**.
3. Click **New**.
 - a. Enter an alias name. This display name is used only for identifying the J2C entry.
 - b. Enter the user ID and password required to connect to the database.

Click **OK**.

The entries used for the DB2 system used with ITSOMart can be seen in Figure 11-5.

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
DB2 ITSOMART USER

* User ID
wsdemo

* Password

Description

Figure 11-5 J2C authentication data entry for the ITSOMART DB2 database

4. Save the configuration.

11.6.2 Create a JDBC provider

A JDBC provider defines implementation-specific classes needed to access a specific type of relational database. The WebSphere ESB application server has the JDBC provider for Cloudscape predefined at the server and cell levels. If you are using Cloudscape, you will not need to create a new JDBC provider unless you want it defined at the node scope. The ITSOMart solution will use a DB2 database.

To create a new JDBC provider, do the following:

1. From the administrative console, select **Resources** → **JDBC Providers**.
2. Select the scope and click **Apply**. We chose to create the JDBC provider at the node scope.
3. Click **New**.
 - a. Select the database type from the pull-down.
 - b. Select the provider type from the pull-down.

c. Select the implementation type and Click **Next**.

The values needed for the DB2 ITSOMart database are shown in Figure 11-6.

JDBC providers > New
Choose a type of JDBC provider to create.

Configuration

General Properties

Step 1: Select the database type
DB2

Step 2: Select the provider type
DB2 Universal JDBC Driver Provider

Step 3: Select the implementation type
Connection pool data source

Figure 11-6 JDBC driver for the ITSOMART DB2 database

Cloudscape has predefined JDBC drivers at the cell and server levels. The properties for the Cloudscape JDBC driver are shown in Figure 11-7.

JDBC providers > New
Choose a type of JDBC provider to create.

Configuration

General Properties

Step 1: Select the database type
Cloudscape

Step 2: Select the provider type
Cloudscape JDBC Provider

Step 3: Select the implementation type
Connection pool data source

Figure 11-7 JDBC driver for the ITSOMART CloudScape database

4. The next panel allows you to specify the location of the class files required to access the database. These files are supplied by the database product. The

fields are filled in with default values that use variables, making it likely that you can take these default values (as is the case for a DB2 provider).

Fill in the values or keep the defaults and click **Apply**.

Note that the Data sources option under Additional properties is now available.

WebSphere variables: Note the variables used on this page and make sure that you define them at the same scope you created the JDBC provider at. You can define the variables by selecting **Environment** → **WebSphere variables**.

In the case of DB2, you will need to define the following:

- ▶ DB2_UNIVERSAL_JDBC_DRIVER_PATH = C:\IBM\SQLLIB\java
- ▶ DB2_UNIVERSAL_JDBC_DRIVER_NATIVEPATH = C:\IBM\SQLLIB\java

The Cloudscape provider is embedded in WebSphere ESB and the variables are pre-set for you.

11.6.3 Create a data source

Data sources are related to specific JDBC providers. They provide the information about the database required to access it, specifically the JNDI lookup name and database name or location.

1. Access the list of data sources for the JDBC provider.
 - If you are continuing from the previous section and have just clicked **Apply** on the JDBC provider, you will have the Data source link available. Click this link to see the list of data sources and to begin defining the data source.
 - If you are using an existing JDBC provider, you can access the data source definitions by doing the following:
 - i. Select **Resources** → **JDBC Providers**.
 - ii. Select the scope and click **Apply**.
 - iii. Click the JDBC provider name.
 - iv. Click **Data sources** under Additional properties.

2. Click **New**. Enter the values required to identify the database. These values vary by database type. In particular, you should pay attention to the following:
 - The JNDI name must match the name you specified in the Database Lookup primitive (see Figure 7-65 on page 326).
 - If authentication is required, the component-managed authentication alias points to the alias you created in 11.6.1, “Create a J2C authentication data entry for the database” on page 491.
 - The database is correctly identified. How you do this will vary depending on the database type.

The values used for the first two of these items for the DB2 ITSOMart database are shown in Figure 11-8.

* Scope
cells:esbCell:nodes:esbNode

* Name
ITSOMart DataSource

JNDI name
jdbc/ITSOMartDataSource

☒ 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)

☐ Specify a user-defined data store helper

Enter a package-qualified data store helper class name

Component-managed authentication alias

Component-managed authentication alias
esbNode/DB2 ITSOMART USER

Figure 11-8 Data source for the ITSOMART database - part 1

At the bottom of this panel you will see the database information (Figure 11-9).



DB2 Universal data source properties

* Database name
ITSOMART

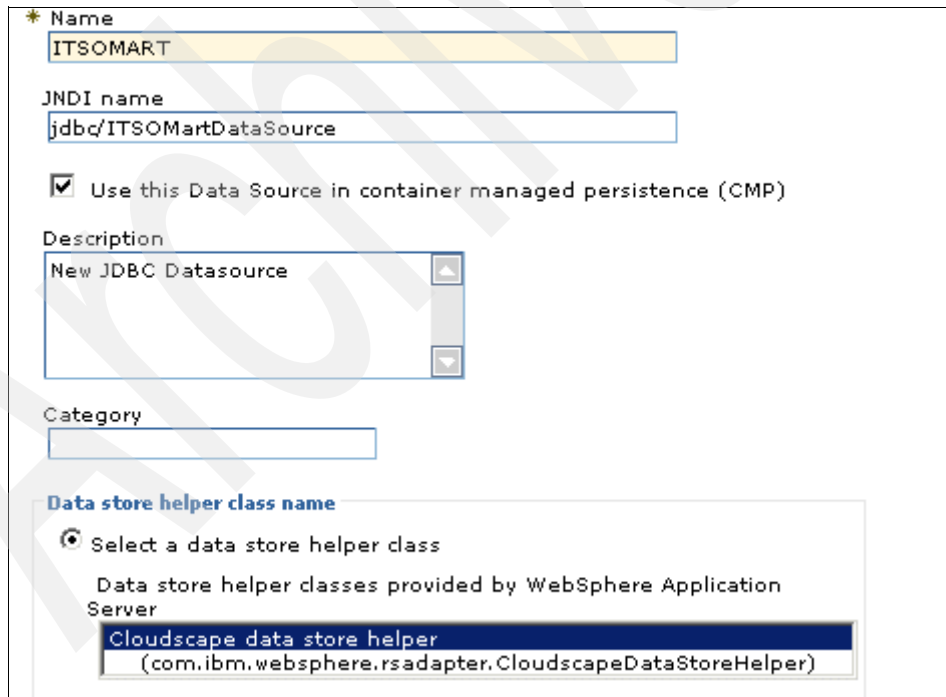
* Driver type
4

* Server name
localhost

Port number
50000

Figure 11-9 Data source for the ITSOMart database - part 2

The values used for the first two items for the Cloudscape ITSOMart database are shown in Figure 11-10.



* Name
ITSOMART

JNDI name
jdbc/ITSOMartDataSource

☒ Use this Data Source in container managed persistence (CMP)

Description
New JDBC Datasource

Category

Data store helper class name

☒ Select a data store helper class

Data store helper classes provided by WebSphere Application Server

Cloudscape data store helper
(com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper)

Figure 11-10 Data source for the Cloudscape version of ITSOMart - part 1

Note that Cloudscape does not require authentication.

The database location is specified as shown in Figure 11-11.

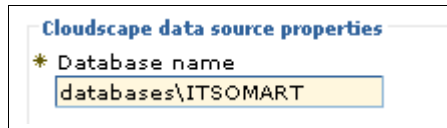


Figure 11-11 Data source for the Cloudscape version of ITSOMart - part 2

This assumes that you created the database in the location suggested in “Create the Cloudscape database” on page 598. If you created it in a different location, be sure to specify that location here. To do this simply specify the full path name. For example:

c:\mydatabases\ITSOMart

3. Click **OK**.
4. Save the configuration.
5. Test the connection to make sure the access is working. To test, navigate to the list of data sources. Check the box to the left of the data source and click **Test connection**.

11.7 Configuration for adapter support

The configuration requirements vary depending on the adapter you will be using. The ITSOMart solution uses two adapters:

- ▶ IBM WebSphere Adapter for Siebel Business Applications
- ▶ IBM WebSphere Adapter for Flat Files

These adapters were installed into the development environment. The adapter files have been packaged automatically into EAR files by WebSphere Integration Developer for deployment.

There are two runtime requirements needed for these adapters:

- ▶ Create a J2C authentication data entry for Siebel.
- ▶ Create an output folder for the flat file.

11.7.1 Create a J2C authentication data entry for Siebel

We are using J2C authentication to connect to the Siebel system. Use the following steps to create a J2C authentication in the WebSphere ESB:

1. From the administrative console select **Security** → **Global security**.

2. Expand the JAAS configuration on the right panel and click **J2C Authentication data**.
3. Click **New**.
 - a. Enter CRM as the Alias name.
 - b. Enter the user ID and password required to connect to the Siebel system.
 - c. Click **OK**.

General Properties

- * Alias: CRM
- * User ID: sadmin
- * Password: ●●●
- Description: Siebel user

Figure 11-12 Creating J2C authentication

The new alias should appear in the list (Figure 11-13). Note that the alias name now includes the node name (esbNode).

Alias	User ID
SCA Auth Alias	SCA
T42J11Node01Cell/esbNode/server1/EventAuthDataAliasCloudScape	none
T42J11Node01Cell/samples	samples
esbNode/CRM	SADMIN

Figure 11-13 Administrative console view after J2C authentication got created

Changing the J2C authentication alias

If you need to change the J2C authentication data entry that the adapter will use after the mediation has been installed, you can do so with the following:

1. Click **Enterprise Applications**.
2. Click the mediation application **ITSOCRMMed**.
3. Select **Map resource references to resources** under Additional Properties.
4. Click the radio button **Use default method** under Specify authentication method and select the authentication data entry from the menu.
5. Check the box to the left of the EJB with the reference binding to the outbound interface. For ITSOMart, this is the ITSO_CRMMedEJB module entry with the reference binding to SiebelOutboundInterface (Figure 11-14), and click **Apply**.

☒ Use default method
Select authentication data entry
esbNode/CRM

☐ Use custom login configuration
Select application login configuration
Select...

Apply

Select	Module	EJB	URI	Reference binding
<input type="checkbox"/>	ITSO_CRMMedEJB	Module	ITSO_CRMMedEJB.jar, META-INF/ejb-jar.xml	sca/resource/import/intf/FlatFileO
<input checked="" type="checkbox"/>	ITSO_CRMMedEJB	Module	ITSO_CRMMedEJB.jar, META-INF/ejb-jar.xml	sca/resource/import/intf/SiebelOu

Figure 11-14 Changing the J2C authentication entry

6. Click **OK** and save the changes.

7. Restart the application.

11.7.2 Create an output folder for the flat file

IBM WebSphere Adapter for Flat Files needs the output folder to be created to write to the file system. For ITSOMart, the folder is called C:\FF. This is defined by the Java code that implements the Custom primitive (Figure 8-57 on page 404).

11.8 Configuration for JMS bindings

WebSphere ESB supports multiple JMS providers (see 11.1.5, “Messaging support” on page 484). ITSOMart will use the default messaging provider supplied in WebSphere ESB. JMS connections that use the WebSphere Application Server default messaging provider will use the following resources:

- ▶ Destinations on the bus (queues, topic spaces, alias queues, and so on): These provide the actual transport for the default message provider. These can be defined on the SCA.APPLICATION.<cell>.Bus bus or a bus you have configured to use with the application.
- ▶ JMS resources: The default messaging provider is predefined. You will need to add definitions for JMS resources (queues or topic spaces), a queue connection factory, and possibly JMS activation specs.

SOAP/JMS: Destinations and JMS resources required by imports and exports that use SOAP/JMS transport are defined automatically when you deploy the module.

The resources required depend on whether you have an import or export, and on the operation type defined on the interface.

- ▶ JMS binding on an import

The resources needed for an import are a queue destination on the bus that represents the queue, a JMS queue connection factory, and a JMS queue that provides the JNDI name for the queue destination on the bus.

- ▶ JMS binding on an export

The resources needed for an export depend on the type of operation being used on the interface.

A one-way operation requires the same resources listed for the import. In addition, an ActivationSpec is needed to register the queue with message listener in the mediation module.

A request/response operation needs the same resources as the one-way operation, with one difference. You will need two queue destinations, one for the request and one for the response, and the corresponding JMS queues.

11.8.1 Create a queue destination on the bus

The ITSOMart sample requires several message queues. These queues can be hosted on the bus or on WebSphere MQ. If you want to use queues on the bus, use the following instructions to create them. If you are planning to use WebSphere MQ, see 11.9, “Connecting to WebSphere MQ” on page 506.

These steps describe how to create the queue destinations on the bus:

1. Open the details page for the bus by selecting **Service integration** → **Buses**. Click the bus name.
2. Under Destination resources, click **Destinations** (Figure 11-15).

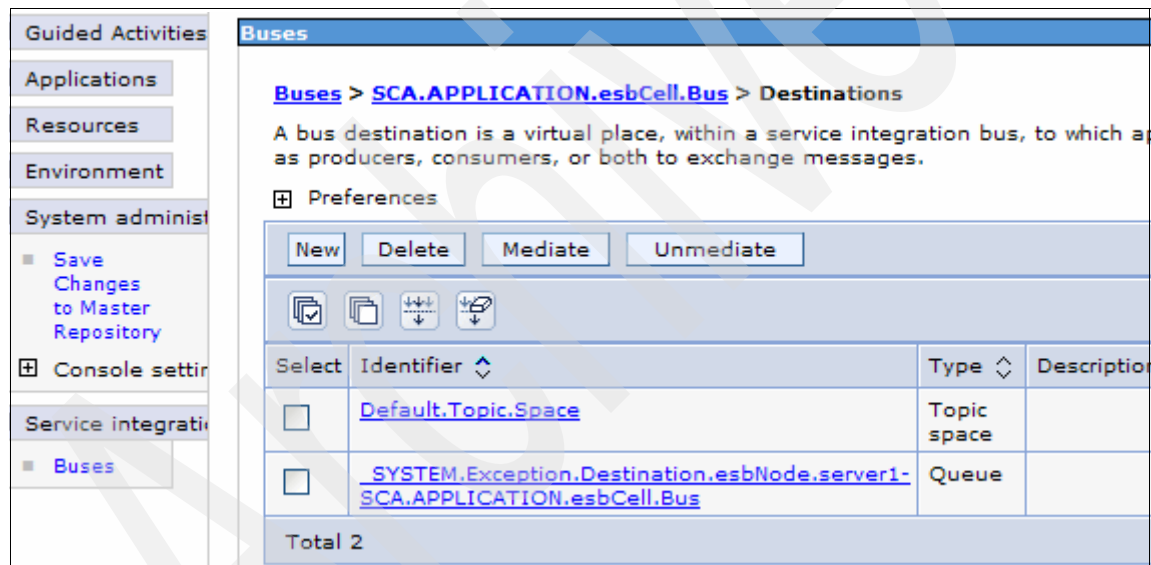


Figure 11-15 Destinations on the SCA.APPLICATION bus

3. Click **New**.
4. For the destination type, accept the default of Queue and click **Next**.
5. Enter the name for the queue and click **Next**.
6. Select the bus member. In the case of a standalone WebSphere ESB, the bus member name will be `<node_name>:server1`. Click **Next**.

7. The final page is just a summary. Click **Finish** and the destination will be created.

Use the values in Table 11-1 to create the four queues required by ITSOMartBus.

Note: If you plan to use WebSphere MQ to host the queues, do not create the queues for the Registration Log mediation. These queues will be defined as aliases and point to queues on WebSphere MQ instead (see 11.9, “Connecting to WebSphere MQ” on page 506).

Table 11-1 Service integration bus queues required for ITSOMart

Mediation or application	Queue names
Registration log mediation	<ul style="list-style-type: none">▶ ITSOMart.LogSuccessQ▶ ITSOMart.LogDeniedQ▶ ITSOMart.LogFailureQ
Registration processor service	<ul style="list-style-type: none">▶ ITSOMart.RegistrationProcessorServiceQ

11.8.2 Create a queue connection factory

A JMS queue connection factory is used to create connections to the associated JMS provider. To create a basic queue connection factory do the following:

1. From the administrative console, expand **Resources** → **JMS Providers** and click **Default messaging**.
2. Under Connection Factories click **JMS queue connection factory**.
3. Click **New**.
4. The next page allows you to specify the properties for the JMS queue connection factory. Take the defaults for everything but the following:
 - Name
Enter a display name to use for the queue connection factory.
 - JNDI Name
Enter the JNDI value of for the queue connection factory. This name must match the name specified for the JNDI lookup name on the JMS Import Binding tab when you create a JMS binding (see Figure 10-9 on page 460 for an example).
 - Bus name
Select the bus in the pull-down menu.
5. Click **OK** to create the queue connection factory.
6. Save the changes.

Use the values in Table 11-2 to create the three queue connection factories required by ITSOMart. For each queue connection factory:

1. Create the queue connection factory under the default messaging provider.
2. Specify ITSOMartBus as the service integration bus.

Table 11-2 Queue connection factories required by ITSOMart

Mediation/application	Field	Value
Registration processor service	Name	ITSOMart.RegistrationProcessorServiceReplyQCF
	JNDI Name	jms/RegistrationProcessorServiceReplyQCF
Registration processor service	Name	ITSOMart.RegistrationProcessorServiceQCF
	JNDI Name	jms/RegistrationProcessorServiceQCF
Registration log mediation	Name	ITSOMart.LogQCF
	JNDI Name	jms/ITSOMart/LogQCF

11.8.3 Create a JMS queue

Now we must define a JMS queue for the bus queue destination created earlier. This step exposes a queue destination on the bus as a JMS queue that can be accessed in JNDI.

1. From the administrative console expand **Resources** → **JMS Providers** and click **Default messaging**.
2. Under Destinations click **JMS queue**.
3. Click **New**.
4. The next page allows you to specify the values for the queue.
 - Name
Enter a display name for the queue.
 - JNDI Name
This is where the application's message reference will be bound to. The JNDI name must match the JNDI lookup name specified on the JMS Destinations tab when you create the JMS binding (see Figure 10-10 on page 461 for an example).
 - Bus name
Select the bus where the queue is defined.
 - Queue name

This field specifies the queue destination on the bus that will be used to store the messages sent to this JMS queue.

5. Click **OK**.
6. Save the changes.

Use the values in Table 11-2 on page 504 to create the three queue connection factories required by ITSOMart. For each queue connection factory:

1. Create the JMS queue under the default messaging provider.
2. Specify ITSOMartBus as the service integration bus.

Table 11-3 Registration Log Mediation JMS queues

Mediation/ application	Field	Value
Registration Log mediation	Name	ITSOMart.LogSuccessQ
	JNDI name	jms/ITSOMart/LogSuccessQ
	Queue name	ITSOMart.LogSuccessQ
Registration Log mediation	Name	ITSOMart.LogDeniedQ
	JNDI name	jms/ITSOMart/LogDeniedQ
	Queue name	ITSOMart.LogDeniedQ
Registration log mediation	Name	ITSOMart.LogFailureQ
	JNDI Name	jms/ITSOMart/LogFailureQ
	Queue name	ITSOMart.LogFailureQ
Registration Processor service	Name	ITSOMart.RegistrationProcessorServiceQ
	JNDI name	jms/RegistrationProcessorServiceQ
	Queue name	ITSOMart.RegistrationProcessorServiceQ

11.8.4 Creating a JMS activation specification

Message-driven beans act as listeners for incoming asynchronous messages. A JMS activation specification is associated with one or more message-driven beans and provides the configuration necessary for them to receive messages.

The Register Customer process J2EE application uses a message-driven bean to listen for incoming messages. The transport used for these messages is SOAP/JMS. You will need to define the corresponding JMS activation specification.

1. From the administrative console expand **Resources** → **JMS Providers** and click **Default messaging**.
2. Under Activation Specifications click **JMS activation specification**.
3. Click **New**.
4. The next page allows you to specify the values for the activation specification.

Most of the values can keep their default values. Described below are the ones of most interest.

- Name

An administrative name used for locating the JMS activation specification. Enter a value of `ITSOMart.RegistrationProcessorServiceActivationSpec`.

- JNDI name

This is where the application's message-driven bean will be bound to for message delivery. Enter a value of `jms/RegistrationProcessorServiceActivationSpec`.

- Destination type

The type of the JMS destination that will be used to deliver messages to the message-driven bean. Accept the default of `Queue`.

- Destination JNDI name

The location in JNDI of the JMS destination that should be used to receive messages from. Enter a value of `jms/RegistrationProcessorServiceQ`.

- Bus name

The name of the bus the JMS destination will receive messages from. This is not required, but for consistency select the value of **ITSOMartBus**.

5. Click **OK**.
6. Save the changes.

11.9 Connecting to WebSphere MQ

The Registration Log mediation in Chapter 10, “Building Log Registration mediation” on page 449, can be tested by creating the queues on the bus. However, in a production environment, it is much more likely that you will use WebSphere MQ to transport messages. You can integrate the bus and WebSphere MQ, allowing you to send messages to a WebSphere MQ queue.

In this section we show how to connect WebSphere ESB to WebSphere MQ to allow the flow of messages from one network to the other.

WebSphere ESB, WebSphere Application Server, and the service integration bus: The messaging infrastructure of WebSphere Application Server V6 is implemented in the service integration bus (referred to as the bus). WebSphere ESB, built on WebSphere Application Server, also uses the service integration bus as its messaging infrastructure.

For this reason, the process used to connect WebSphere ESB to WebSphere MQ and WebSphere Application Server to WebSphere MQ are the same.

The sample we show here is simply to illustrate the mechanics of connecting a bus to a WebSphere MQ environment. Before making any decisions, you should refer to the product documentation for planning assistance in designing a service integration topology.

The following configuration is required to connect the two systems.

In WebSphere ESB:

- ▶ A service integration bus.
- ▶ A foreign bus definition for WebSphere MQ.
- ▶ An MQ link that defines the specific queue manager, the listener port, and the sender channel name for the link to WebSphere MQ. The queue manager name for the bus is also specified here.

In WebSphere MQ:

- ▶ A queue manager in WebSphere MQ.
- ▶ A transmission queue
- ▶ A sender channel that defines the host and listener port for the bus. It uses a transmission queue.
- ▶ A receiver channel

The following queues are defined to support the Registration Log Mediation sample that we will use to illustrate integration with WebSphere MQ.

In the WebSphere MQ queue manager:

- ▶ LOG.SUCCESS.QUEUE
- ▶ LOG.DENIED.QUEUE
- ▶ LOG.FAILURE.QUEUE

In the bus:

- ▶ ITSOMart.LogSuccessQ: This queue is an alias of the LOG.SUCCESS.QUEUE in the WebSphere MQ queue manager.
- ▶ ITSOMart.LogDeniedQ: This queue is an alias of the LOG.DENIED.QUEUE in the WebSphere MQ queue manager.
- ▶ ITSOMart.LogFailureQ: This queue is an alias of the LOG.FAILURE.QUEUE in the WebSphere MQ queue manager.

The following sections show you how to configure WebSphere MQ and WebSphere ESB to connect the two and to create the definitions required by ITSOMart.

11.9.1 Configure WebSphere MQ

This section gives an overview of the elements required in WebSphere MQ to receive and deliver messages. It covers the basics of creating queue managers and queues, and connecting one queue manager to another using channels.

The steps required to create and test the connection are:

1. Create a queue manager.
2. Create a transmission queue.
3. Create a sender channel.
4. Create a receiver channel.
5. Create local queues.

The next sections illustrate how this is done for our example. For testing purposes, we used one machine and one installation of WebSphere MQ.

Create a queue manager

Start by creating a new queue manager in WebSphere MQ using the WebSphere MQ Explorer:

1. Open the WebSphere MQ Explorer.
2. Right-click **Queue Managers** and select **New** → **Queue Manager**.
 - a. Type ITSOMART_QM in the Queue Manager Name field and click **Next**.
 - b. Take the default for the log values and select **Next**.
 - c. Deselect **Auto Start Queue Manager** and select **Next**.
 - d. Enter the listener port number. The default is 1414. If you already have a queue manager running on the system you will need to select a different port number.
 - e. Click **Finish**.

You should now see the started queue manager in the WebSphere MQ Explorer (Figure 11-16).

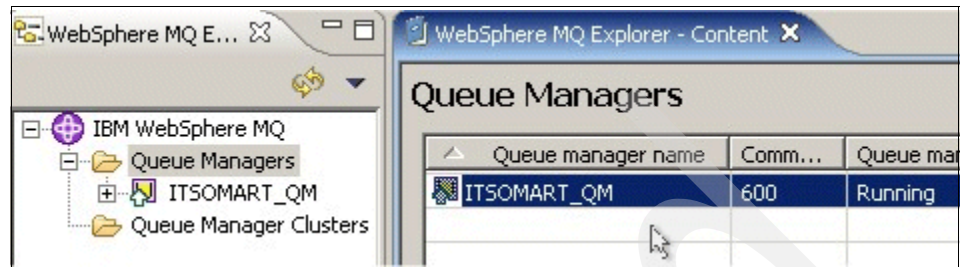
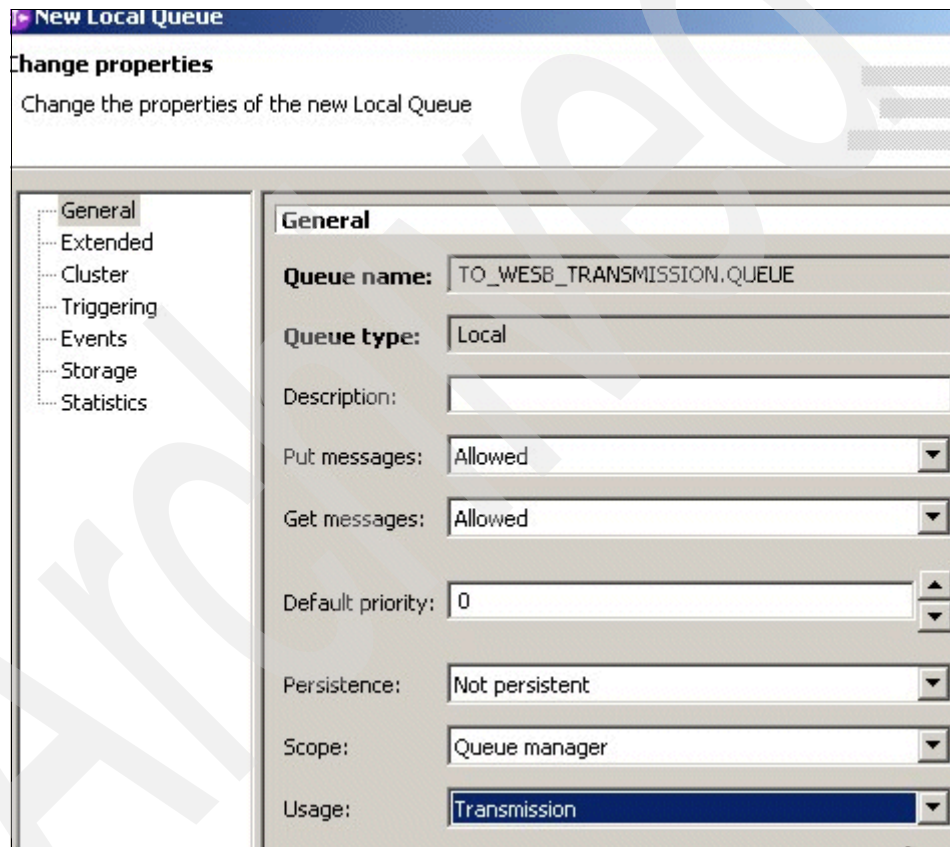


Figure 11-16 WebSphere MQ Queue Manager

Create a transmission queue

A transmission queue is a local queue on which prepared messages destined for a remote queue manager are temporarily stored. To create the transmission queue:

1. In the WebSphere MQ Explorer, navigate to **Queue Managers ITSOMART_QM → Queues**.
2. Right-click **Queues** and select **New → Local Queue**.
 - a. Type `TO_WESB_TRANSMISSION.QUEUE` in the Name field and click **Next**.
 - b. Change the Usage field to **Transmission**.



New Local Queue

Change properties
Change the properties of the new Local Queue

General

Queue name: `TO_WESB_TRANSMISSION.QUEUE`

Queue type: `Local`

Description:

Put messages: `Allowed`

Get messages: `Allowed`

Default priority: `0`

Persistence: `Not persistent`

Scope: `Queue manager`

Usage: `Transmission`

Figure 11-17 WebSphere MQ transmission queue

- c. Click **Finish**.

Create a sender channel

A sender channel is a channel that initiates transfers, removes messages from a transmission queue, and moves them over a communication link to a receiver or requester channel. To create the sender channel:

1. In the WebSphere MQ Explorer, navigate to **Queue Managers** → **ITSOMART_QM** → **Advanced** → **Channels**.
2. Right-click **Channels** and select **New** → **Sender Channel**.
 - a. Type `WMQ_TO_WESB.SENDER` in the Name field and click **Next**.
 - b. In the Connection Name field, type the IP address or host name of the system hosting the queue manger you want to send messages to, concatenated with its listener port number in parentheses.

The connection name for this example is `localhost(5558)`. This will connect the sender channel to the service integration bus.

The default listener port for the bus is 5558 for the first application server on a node. To check the port for your application server:

- i. Open the WebSphere administrative console.
- ii. Select **Servers** → **Application servers**.
- iii. Click the server name to open the details page.
- iv. Expand the **Ports** category under the Communications section. The port number used by the server is the `SIB_MQ_ENDPOINT_ADDRESS`.

- c. Enter `TO_WESB_TRANSMISSION.QUEUE` as the transmission queue (Figure 11-18). This is the queue defined earlier in “Create a transmission queue” on page 510.

New Sender Channel

Change properties
Change the properties of the new Sender Channel

General

Channel name: WMQ_TO_WESB.SENDER

Type: Sender

Description:

Transmission protocol: TCP

Connection name: localhost(5558)

Transmission queue: TO_WESB_TRANSMISSION.QUEUE

Local communication address:

Figure 11-18 WebSphere MQ sender channel

- d. Click **Finish**.

Create a receiver channel

The receiver channel is a channel that responds to a sender channel, taking messages from a communication link. To create the receiver channel:

1. In the WebSphere MQ Explorer, navigate to **Queue Managers** → **ITSOMART_QM** → **Advanced** → **Channels**.
2. Right-click **Channels** and select **New** → **Receiver Channel**.
 - a. Type `WESB_TO_WMQ.RECEIVER` in the Name field. Unlike the sender channel, the receiver channel does not need the connection name defined.

- b. Take the default for the transmission protocol of TCP. The properties used for our sample are shown in Figure 11-19.

New Receiver Channel

Change properties
Change the properties of the new Receiver Channel

General

Channel name: WESB_TO_WM.Q.RECEIVER

Type: Receiver

Description:

Transmission protocol: TCP

Figure 11-19 WebSphere MQ receiver channel

- c. Click **Finish**.

You should now see the sender and receiver channels in the WebSphere MQ Explorer (Figure 11-20).

WebSphere MQ Explorer - Content

Channels

Filter: Standard for Channels

Channel name	Channel type	Overall channel status	Conn name	Xmit protocol
WESB_TO_WM.Q.RECEIVER	Receiver	Inactive		TCP
WM.Q_TO_WESB.SENDER	Sender	Inactive	localhost(5558)	TCP

Figure 11-20 WebSphere MQ channels

Create local queues

The local queues are specific to the application. To create the local queues for ITSOMart:

1. Navigate to **Queue Managers** → **ITSOMART_QM** → **Queues**.
2. Right-click **Queues** and select **New** → **Local Queue**.

Type LOG.SUCCESS.QUEUE in the Name field and click **Finish**.

3. Create two more local queues with the following names:
 - LOG.DENIED.QUEUE
 - LOG.FAILURE.QUEUE

You should now see these three local queues and the transmission queue in the WebSphere MQ Explorer (Figure 11-21).

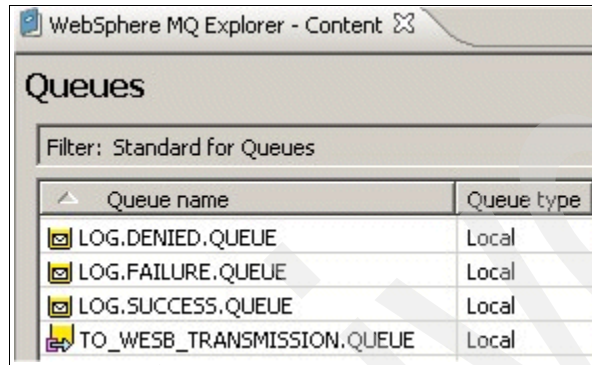


Figure 11-21 WebSphere MQ queues

11.9.2 Configure the bus

The following steps are needed to integrate the service integration bus with WebSphere MQ:

1. Define WebSphere MQ as a foreign bus.
2. Define a WebSphere MQ link.
3. Create alias queues.
4. Start the bus and WebSphere MQ connections.

This configuration is done using the WebSphere administrative console.

Define WebSphere MQ as a foreign bus

The next step is to define WebSphere MQ to the service integration bus. WebSphere MQ is represented as a foreign bus.

1. In the WebSphere administrative console, select **Service Integration** → **Buses**.
2. Click the bus name, **ITSOMartBus**, to open the detail page.
3. Click **Foreign Buses**.

4. Click **New**.
 - a. Type ITSOMART_QM in the Name field and click **Next**.
 - b. Select **Direct, WebSphere MQ link** and click **Next**. Click **Next** again.
 - c. Click **Finish** to add the new foreign bus definition, shown in Figure 11-22.

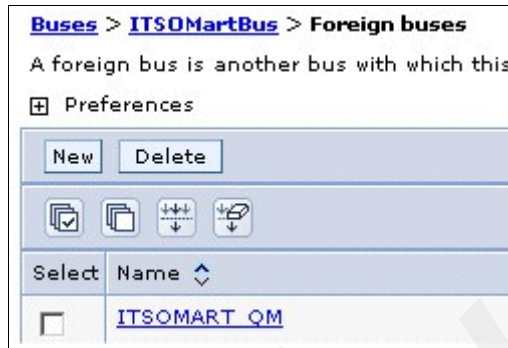


Figure 11-22 Foreign bus

5. Save the changes.

11.9.3 Define a WebSphere MQ link

A WebSphere MQ link enables the exchange of messages with a WebSphere MQ network. Defining an MQ link defines the sender and receiver channels used to transmit messages to and from WebSphere MQ.

WebSphere MQ links are defined at the messaging engine:

1. Select **Service Integration** → **Buses**. Click the bus name to open it.

2. Click **Messaging Engines**. The messaging engine created for the application server (when you added the server to the bus) should be in Started state. If not, click **Start**.

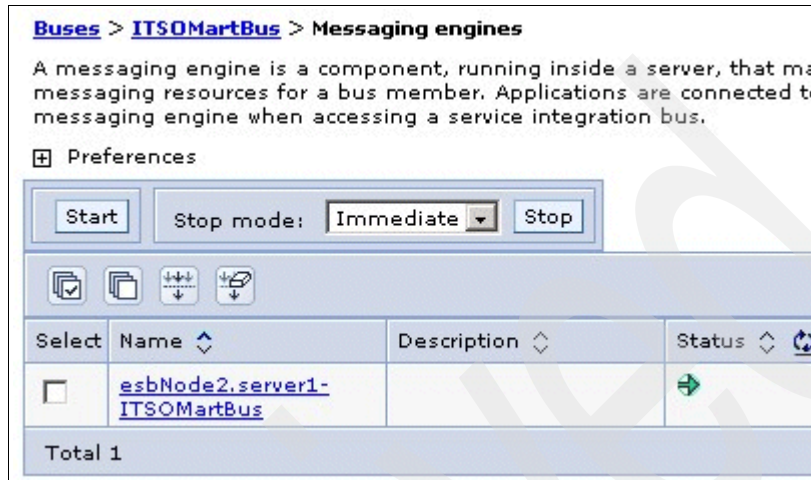
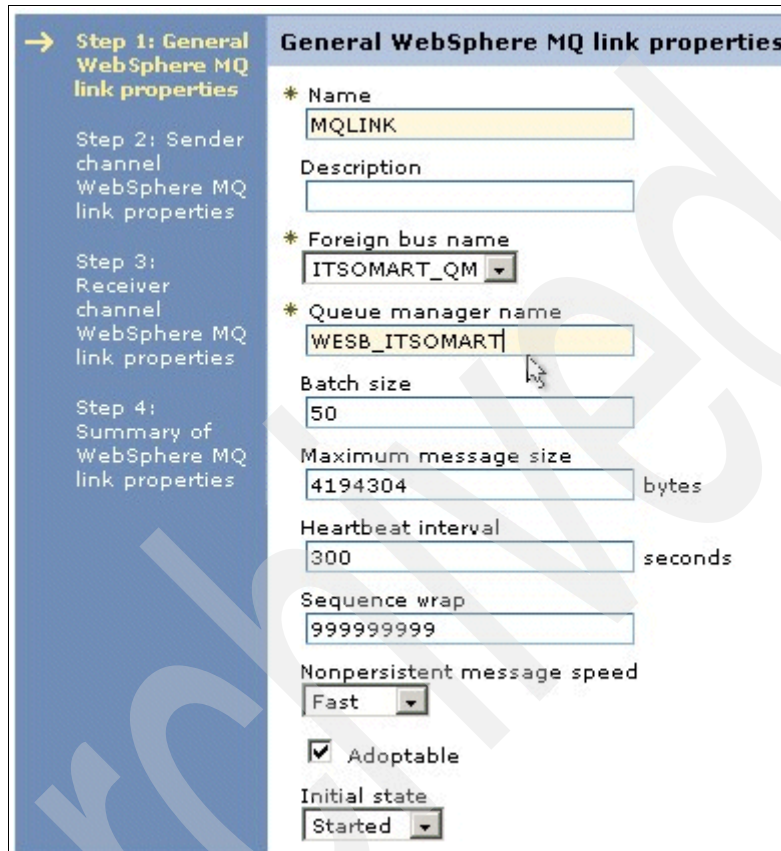


Figure 11-23 Messaging engine for server1

3. Click the messaging engine for your server to open the details page.
4. Click **WebSphere MQ Links**.
5. Click the **New** button.
6. In step 1 (Figure 11-24 on page 517):
 - a. Type MQLINK in the Name field.
 - b. Select **ITSOMART_QM** in the Foreign Bus field.

- c. Type WESB_ITSOMART in the Queue Manager Name field. This is the queue manager name that this service integration bus MQ Link will be known as to an external WebSphere MQ network.



→ **Step 1: General WebSphere MQ link properties**

Step 2: Sender channel WebSphere MQ link properties

Step 3: Receiver channel WebSphere MQ link properties

Step 4: Summary of WebSphere MQ link properties

General WebSphere MQ link properties

* Name: MQLINK

Description:

* Foreign bus name: ITSOMART_QM

* Queue manager name: WESB_ITSOMART

Batch size: 50

Maximum message size: 4194304 bytes

Heartbeat interval: 300 seconds

Sequence wrap: 999999999

Nonpersistent message speed: Fast

☒ Adoptable

Initial state: Started

Figure 11-24 WebSphere MQ link

Click **Next**.

7. In step 2 (Figure 11-25 on page 518):
 - a. Type WESB_TO_WMQL.RECEIVER in the Sender MQ Channel Name field. This defines the sender channel. The name used here must match the name you use for the partner receiver channel in WebSphere MQ.
 - b. Type the IP address of the WebSphere MQ host in the Host Name field.
 - c. Type the listener port number for the WebSphere MQ queue manager in the Port field. In this example, the WebSphere MQ queue manager is listening on the default port 1414.

d. Select **OutboundBasicMQLink** in Transport Chain field.

Step 1:
General
WebSphere
MQ link
properties

→ Step 2: Sender
channel
WebSphere
MQ link
properties

Step 3:
Receiver
channel
WebSphere
MQ link
properties

Step 4:
Summary of
WebSphere
MQ link
properties

Sender channel WebSphere MQ link properties

Sender MQ channel name
WESB_TO_WM.Q.RECEIVER

Host name
localhost

Port
1414

* Transport chain
OutboundBasicMQLink

Disconnect interval
900 seconds

Short retry count
10

Short retry interval
60 seconds

Long retry count
999999999

Long retry interval
1200 seconds

Initial state
Started

Figure 11-25 WebSphere MQ link - Sender channel

e. Click **Next**.

8. In step 3 (Figure 11-26):
 - a. Type `WMQ_TO_WESB.SENDER` in the Receiver MQ Channel Name field. This defines the receiver channel. The name used here must match the name you use for the partner sender channel in WebSphere MQ.

The screenshot shows a configuration window titled "Receiver channel WebSphere MQ link properties". On the left is a navigation pane with four steps: "Step 1: General WebSphere MQ link properties", "Step 2: Sender channel WebSphere MQ link properties", "Step 3: Receiver channel WebSphere MQ link properties" (which is highlighted with a yellow arrow), and "Step 4: Summary of WebSphere MQ link properties". The main area contains the following fields and dropdowns: "Receiver MQ channel name" with the text "WMQ_TO_WESB.SENDER" entered; "Inbound nonpersistent message reliability" set to "Reliable"; "Inbound persistent message reliability" set to "Assured"; and "Initial state" set to "Started". A mouse cursor is visible at the bottom right of the window.

Figure 11-26 WebSphere MQ link - Receiver channel

- b. Click **Next**.
9. In step 4, click **Finish**.
10. Save the changes.
11. Restart the server for the new WebSphere MQ link configuration to take effect.

11.9.4 Create alias queues

The next step prepares the bus for the application-specific queue requirements. In this example we create a local queue to hold the messages and an alias queue to represent the WebSphere MQ queue.

1. Select **Service Integration** → **Buses**.
2. Click the bus name.
3. Click **Destinations**.
4. Click **New**.
5. Select **Alias** and click **Next**.

6. In step 1 (Figure 11-27):
- a. Type `ITSOMart.LogSuccessQ` in the Identifier field.
 - b. Select **ITSOMartBus** in the Bus field.
 - c. In the Target Identifier field, select **other, please specify**, then type:
`LOG.SUCCESS.QUEUE@ITSOMART_QM`
 - d. Select the foreign bus, **ITSOMART_QM**, in the Target Bus field.

→ **Step 1: Set alias destination attributes**

Step 2: Confirm alias destination creation

Set alias destination attributes

Configure the attributes of your new alias destination

* Identifier
ITSOMart.LogSuccessQ

description

Bus
ITSOMartBus

* Target identifier
other, please specify CESS.QUEUE@ITSOMART_QM

Target bus
ITSOMART_QM

Figure 11-27 Alias destination

- e. Click **Next**.
7. In step 2, click **Finish**.
8. Create two more alias destinations with the values in Table 11-4.

Table 11-4 Values

Identifier	Bus	Target Identifier	Target bus
ITSOMart.LogDeniedQ	ITSOMartBus	LOG.DENIED.QUEUE@ITSOMART_QM	ITSOMART_QM
ITSOMart.LogFailureQ	ITSOMartBus	LOG.FAILURE.QUEUE@ITSOMART_QM	ITSOMART_QM

You should now see all three alias destinations under Buses → ITSOMartBus → Destinations (Figure 11-28).



Figure 11-28 Alias destinations

9. **Save** all changes.

JMS resource definitions: You must create the corresponding JMS queue connection factory and JMS queues. These are the same whether the queues reside on the bus or on WebSphere MQ and have an alias on the bus pointing to them.

Create the following if you have not already done so:

- ▶ The ITSOMart.LogQCF queue connection factory created in 11.8.2, “Create a queue connection factory” on page 503.
- ▶ The ITSOMart.LogSuccessq, ITSOMart.LogDeniedQ, and ITSOMart.LogFailureQ created in 11.8.3, “Create a JMS queue” on page 504.

11.9.5 Start the bus and WebSphere MQ connections

After creating the sender channel in WebSphere MQ and starting it, it should have gone into standby status waiting for the receiver channel on the bus to become active. Now that the corresponding channel definitions are defined in the bus, you will be able to start the sender channels on both sides.

In the WebSphere administrative console:

1. Select **Services Integration** → **Buses**. Then click the bus name to open its configuration.
2. Click **Messaging Engines**. Then click the messaging engine to open it.
3. Click **WebSphere MQ Links**. Then click the WebSphere MQ link name, **MLINK**, to open it.
4. Click **Sender Channel**. Note that the status of the channel is standby.
5. Check the box to the left of WESB_TO_WMQ.RECEIVER and click **Start**. The channel will start and you can see the status change to Started (Figure 11-29).

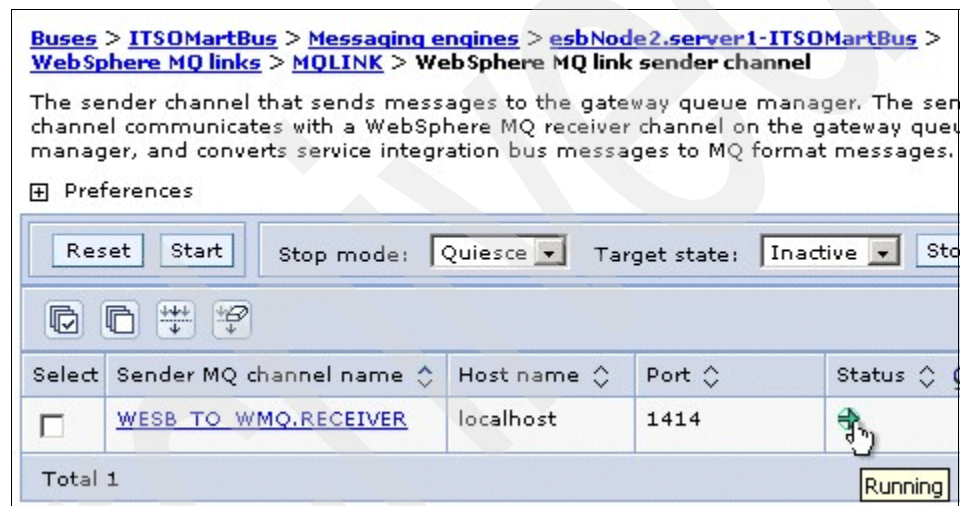


Figure 11-29 Start the sender channel in the bus

In WebSphere MQ Explorer right-click the sender channel **WMQ_TO_WESB.SENDER** and select **Start**.

When the channel starts successfully, the status will change to Running and the icon will turn green (Figure 11-30).

Channel name	Channel type	Overall channel status	Conn name
WESB_TO_WM.Q.RECEIVER	Receiver	Inactive	
WM.Q_TO_WESB.SENDER	Sender	Running	localhost(5558)

Figure 11-30 Channel status in WebSphere MQ Explorer

11.10 Deploying applications

Depending on the environment, a mediation module can be deployed in one of the following ways:

- ▶ If the server has been defined to WebSphere Integration Developer as a test environment server, you can deploy mediation modules directly from the workbench using the Add and remove projects option from the server context menu. See 6.5, “Testing mediations” on page 256, for more information.
- ▶ WebSphere Integration Developer can be used to generate a deployable EAR file. The file is then installed using the WebSphere administrative tools. See 6.6, “Packaging the mediation for deployment” on page 260, for information about generating a deployable EAR file from WebSphere Integration Developer.
- ▶ The deployable EAR file can be generated using a command-line utility, serviceDeploy, and then installed using the WebSphere administrative tools.

11.10.1 Use the serviceDeploy command

WebSphere ESB provides a command-line utility called serviceDeploy that can be used to build deployable mediation modules from zip or jar files containing service components. A mediation module can be exported from WebSphere Integration Developer to be used later by the serviceDeploy command for generation of deployable EAR file.

If a mediation module is exported from WebSphere Integration Developer as a zip file or a jar file, it will not contain any deployable code, only files that describe the module and its components.

Running `serviceDeploy` generates an installable EAR file containing all of the deployable code required for the module to run as a service application on WebSphere ESB. An example of using `serviceDeploy` is shown in Example 11-1.

Example 11-1 Usage of serviceDeploy

```
serviceDeploy.bat c:\temp\MyModule.zip -outputApplication MyModule.ear
```

The utility `serviceDeploy` is commonly used by development teams using a version control system. After developers check in their mediation module projects into a source code repository, the modules can be extracted and built into installable EAR files using `serviceDeploy`. This can be done for deploying the application in a system test environment or a production environment. Using this method ensures that the runtime code is not checked in, but generated when required.

11.10.2 Deploy an EAR file

To deploy the mediation module, open the WebSphere administrative console and do the following:

1. Select **Applications** → **Install New Applications**.
 - a. Browse to the EAR file you exported and click **Next**.
 - b. In the next window, labeled Preparing for the application installation, click **Next**.

The next series of steps will take you through the application installation. This process is the same as for any WebSphere application. The number of steps and complexity of choices depend on the application and environment. In the sample we use, all of the default values were correct.

Follow through with the installation process and save your changes.

2. To start the application select **Applications** → **Enterprise Applications**. Place a check in the box to the left of the mediation application and click **Start**.

Automating deployment: You can use ANT to automate the deployment of mediation modules. The `ServiceDeploy` task will create EAR files for application JAR files and the `InstallApplication` task will install the EAR files.

For more information see the *Deploying applications using ANT tasks* topic in the WebSphere Enterprise Service Bus 6.0.1 Information Center at:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wsps.dev.doc/doc/tdep_usingant.html

11.10.3 Installing the ITSOMart applications

The sample application contains the following EAR files:

- ▶ ITSOMartApp.ear
- ▶ ITSO_CreditRatingMedApp.ear
- ▶ ITSO_CreditRatingMedV2App.ear
- ▶ ITSO_CreditRatingServiceApp.ear
- ▶ ITSO_CreditScoreMedApp.ear
- ▶ ITSO_CreditScoreServiceApp.ear
- ▶ ITSO_CRMMedApp.ear
- ▶ ITSO_RegLogMedApp.ear
- ▶ ITSO_RegProcServiceApp.ear
- ▶ ITSO_RegShipMedApp.ear
- ▶ ITSO_ShipShipSvcApp.ear
- ▶ MessageLogApp.ear

Refer to Appendix A, “Sample application install summary” on page 589, for an overview of each application.

These instructions assume a standalone server. You can use the same instructions to install to a cluster, but will select the cluster instead of a server when you map modules to servers.

Follow these steps to install each EAR file into WebSphere ESB:

1. Log in to the administrative console.
2. Select **Applications** → **Install New Application**.

3. Browse to the EAR file location, select it, and click **Next**.

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:\ITSO\ITSOMart-02\IT Browse...

☐ Remote file system

Specify path

Browse...

Context root

Used only for standalone Web modules
(.war files)

Next Cancel

Figure 11-31 Select the EAR file

4. Check the box **Generate Default Bindings** and click **Next**.

5. The number of steps varies with the type of EAR file selected. Normally for a mediation module you will find eight steps, listed to the left in Figure 11-32.

installation options

Step 2 Map modules to servers

Step 3 Provide listener bindings for message-driven beans

Step 4 Provide JNDI Names for Beans

Step 5 Map JCA resource references to resources

Step 6 Map resource references to resources

Step 7 Map virtual hosts for Web modules

Step 8 Ensure all unprotected 2.x methods have the correct level of protection

Step 9 Summary

Specify the various options that are available to prepare and install your application.

☐ Pre-compile JSP

Directory to install application

☒ Distribute application

☐ Use Binary Configuration

☐ Deploy enterprise beans

Application name

☒ Create MBeans for resources

☐ Enable class reloading

Reload interval in seconds

☐ Deploy Web services

Validate Input off/warn/fail

☒ Process embedded configuration

Next **Cancel**

Figure 11-32 Install steps for a mediation module

- Click **Next** to continue the installation. You can directly click **Summary** if you do not have any specific settings to perform. The summary page shows the install options that will be used (Figure 11-33).

Summary	
Summary of installation options	
Options	Values
Use Binary Configuration	No
Create MBeans for resources	Yes
Cell/Node/Server	Click here
Reload interval in seconds	
Enable class reloading	No
Process embedded configuration	Yes
Application name	ITSO_CRMMedApp
Validate Input off/warn/fail	warn
Directory to install application	
Distribute application	Yes
Deploy Web services	No
Pre-compile JSP	No
Deploy enterprise beans	No

Figure 11-33 Install summary

- Click **Finish** to install the application. A report shows the status of the application install. After successful completion, a message is generated, as shown in Example 11-2.

Example 11-2 Successful application install message

ADMA5013I: Application ITSO_CRMMedApp installed successfully.
Application ITSO_CRMMedApp installed successfully.

- Save the changes to the master configuration.

Installing an application from the administrative console will not start it. You can start each application after install or wait until all are installed. To start the applications, check the box to the left of each application and click **Start**.

<div> <input type="button" value="Start"/> <input type="button" value="Stop"/> <input type="button" value="Install"/> <input type="button" value="Uninstall"/> <input type="button" value="Update"/> <input type="button" value="Rollout Update"/> <input type="button" value="Remove"/> </div>		
<div> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> </div>		
Select	Name	Status
<input type="checkbox"/>	DefaultApplication	
<input type="checkbox"/>	ESBSamplesGallery	
<input type="checkbox"/>	IBMUTC	
<input checked="" type="checkbox"/>	ITSO_MartApp	
<input checked="" type="checkbox"/>	ITSO_CRMMedApp	
<input checked="" type="checkbox"/>	ITSO_CreditRatingMedApp	
<input checked="" type="checkbox"/>	ITSO_CreditRatingMedV2App	
<input checked="" type="checkbox"/>	ITSO_CreditRatingServiceApp	
<input checked="" type="checkbox"/>	ITSO_CreditScoreMedApp	
<input checked="" type="checkbox"/>	ITSO_CreditScoreServiceApp	
<input checked="" type="checkbox"/>	ITSO_ReqLogMedApp	
<input checked="" type="checkbox"/>	ITSO_ReqProcServiceApp	
<input checked="" type="checkbox"/>	ITSO_ReqShipMedApp	
<input checked="" type="checkbox"/>	ITSO_RegistryApp	
<input checked="" type="checkbox"/>	ITSO_ShipReqSvcApp	
<input checked="" type="checkbox"/>	MessageLogApp	

Figure 11-34 Starting applications

Make sure that all the applications started successfully.

By default, the applications will start automatically on server startup.

11.11 Testing ITSOMart

The ITSOMart application Web interface can be started with the following URL:

<http://localhost:9080/ITSOMartWeb/faces/CustomerRegistration.jsp>

To register a new customer, fill in the values and click **Submit Registration**.

Please enter your information below. Your email address will become your unique identifier.	
Contact Name (required)	
First:	Last:
<input type="text" value="ITSOMart"/>	<input type="text" value="ITSOMart"/>
Company / Organization	
<input type="text" value="IBM"/>	
Email / User ID (required)	
Email:	Password:
<input type="text" value="itso@ibm.com"/>	<input type="text" value="1234"/>
Billing Address (required)	
Name:	
<input type="text" value="ITSOMart"/>	
Street:	
<input type="text" value="123"/>	
City:	State/Province:
<input type="text" value="Durham"/>	<input type="text" value="NC"/>
Zip / Postal Code:	Country:
<input type="text" value="27712"/>	<input type="text" value="US"/>

Figure 11-35 ITSOMart Customer Registration page

You will be taken to the Registration Confirmation page. This does not mean that the customer has been successfully registered. Use the MessageLogApp supplied with the sample to see whether the registration was successful and to see the messages logged from each mediation.

The server log contains messages produced by the application and any error messages that may have occurred. The log is at:

`<profile_home>/logs/<server_name>/SystemOut.log`

If you are running in a WebSphere Integration Developer test environment, you can use the Console view.

11.12 Network Deployment and clustering topologies

WebSphere Application Server Network Deployment provides failover, workload management, and scalability features through its ability to cluster application servers. Each application server in the cluster hosts the same applications, and workload is distributed among the servers in the cluster. Although the applications are the same on each server, the server features and configuration can be unique.

Workload management for an application server cluster involves the following:

- ▶ HTTP requests can be distributed across multiple Web containers in a cluster of application servers using the Web server plug-in.
- ▶ EJB requests can be distributed across multiple EJB containers in a cluster of application servers.

Failover occurs when a cluster member becomes unavailable. With multiple servers in the cluster to handle workload, the work continues on the remaining active servers.

Scalability can also be achieved through clustering. Defining multiple cluster members on the same machine provides vertical scaling. This allows you to make efficient use of the server and machine resources and provides software failover. Defining multiple cluster members across machines provides horizontal scaling. This provides software and hardware failover capabilities and increases the amount of software and hardware resources available to handle the workload.

These features do not change for WebSphere ESB. You can use clustering to provide failover, scalability, and workload management for mediation modules running on a cluster.

For an excellent reference on workload management, scalability, and failover in WebSphere Application Server environment, please see the following:

- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- ▶ *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688

An additional consideration for WebSphere ESB is managing workload associated with messages on queue destinations in the bus. As an ESB, you

may expect large numbers of messages to be handled by the mediation modules. Clustering the application servers that run the mediation modules will provide failover by default, but managing workload generated by messages on queue destinations will require you to do some extra configuration.

The following are topologies to consider:

- ▶ High availability with a single cluster

This configuration consists of a cluster of servers that host mediation modules. A single messaging engine attached to one application server in the cluster manages the queue destinations. If the application server fails, the messaging engine is activated on another application server.

There is no workload sharing in this configuration, as there is only one messaging engine to handle the traffic through the destination.

- ▶ Workload management with a single cluster

This topology provides both high-availability and workload management features. The messaging engines and mediation modules are hosted in the same cluster of application servers. Each server in the cluster has a messaging engine active on the bus, giving it local access to the bus and providing workload management. This topology cannot support deferred asynchronous responses.

- ▶ High availability with two clusters

In this topology, two clusters are used — one for hosting the messaging engine and a second for hosting the mediation modules. The cluster hosting the messaging engines does not have to be SCA-enabled. It is configured as a remote destination for the SCA-enabled cluster that hosts the mediation modules using the advanced configuration settings for the cluster.

This topology can support deferred asynchronous responses, as there is no partitioning of the destination. Note, however, that we only have a single messaging engine to support all of the application servers. This means that asynchronous messages are not distributed across the cluster. In addition, when a mediation module accesses a queue, unless it is executing in the server that has the messaging engine, the access to the queue is remote. Adding additional messaging engines to the cluster would lead to partitioned queues and with remote consumers, there is no guarantee that the consumers will be spread evenly over the partitions (or even that they will not all connect to one messaging engine).

The following section illustrates how clusters are created and configured for SCA and for load balancing.

11.12.1 Workload management with a single cluster

By default, only one server in a cluster has an active messaging engine on a bus. If the server fails, the messaging engine on another server in the cluster is activated. This provides failover, but no workload management. The server with the active messaging engine has local access to the bus, but the rest of the servers in the cluster access the bus remotely by connecting to the active messaging engine. Servers accessing the bus through a remote messaging engine will not consume asynchronous messages.

In this topology, each server in the cluster will be configured to have an active messaging engine, thus providing workload management as well as failover.

When a queue destination is assigned to the cluster, the queue is partitioned, with each messaging engine in the cluster owning a partition of the queue. A message sent to the queue will be assigned to one partition. The messaging engine that owns the partition is responsible for managing the message. This means that requests sent to a destination can be served on any of the messaging engines running on any of the servers in the cluster.

Considerations for this topology

Several things should be considered when deciding to use this topology:

- Issues with partitioned queues

Queues are used in the WebSphere ESB to implement asynchronous calls to SCA components, including mediation components. In this topology, the queues are partitioned with each messaging engine owning a partition. You should consider the following:

- Connections to the partitioned queues may not be balanced.

One example would be when the number of users of the queues is less than the number of server members of the cluster. In that case some server members may not be used. Another example is when a queue user has a role of either putting messages in or taking messages from the queue.

- Affinity.

When applications are stateless any cluster member can process the message. This means that there is no affinity between the message and the server member that needs to process the message. However, when the applications are stateful the message needs to go to the one cluster member that has the correct state to be able to process the message correctly. (See the consideration for deferred asynchronous responses below.)

- Message ordering.

When a user of a queue reconnects to the queue it may be routed to a different server member (partition) from where the user was connected previously. This movement between partitions creates a condition where some messages may be processed prior to subsequent messages.

Partitions also introduce the possibility that messages are processed out of order on a single server in the cluster. For example, if messages are sent in the order A-B-C and A and B are routed to server1 and C to server2, if A fills up the partition on server1, then B may be rerouted to server2, arriving after message C.

- ▶ Deferred responses

This topology cannot be used with mediation modules that may have deferred responses (for example, a service using SOAP/JMS that contains a request/response operation) cannot use this topology. There is no guarantee that responses will be sent back to the same partition of the destination as the requester is listening on. This is not the case in the ITSOMart application.

- ▶ Database requirements for Message Logger primitives

The Message Logger primitive requires a database. In the stand-alone server topology it is created as a Cloudscape database during the installation process, but in a clustered environment you will need to create it. SQL statements have been provided in `<WAS_HOME>/util/EsbLoggerMediation/<db_type>`.

For DB2:

- Locate the DDL file and add a NOT NULL attribute to the MEDIATIONNAME column in the CREATE TABLE statement.
- Enter the following commands:

```
db2cmd
db2 create db medlog
db2 -tvf table.ddl
```


Topology example

Figure 11-36 shows the topology for this example. The SCA.APPLICATION and SCA.SYSTEM buses are created with the deployment manager profile. In the ITSOMart application we use an application-specific bus, ITSOMartBus, which we added to the configuration.

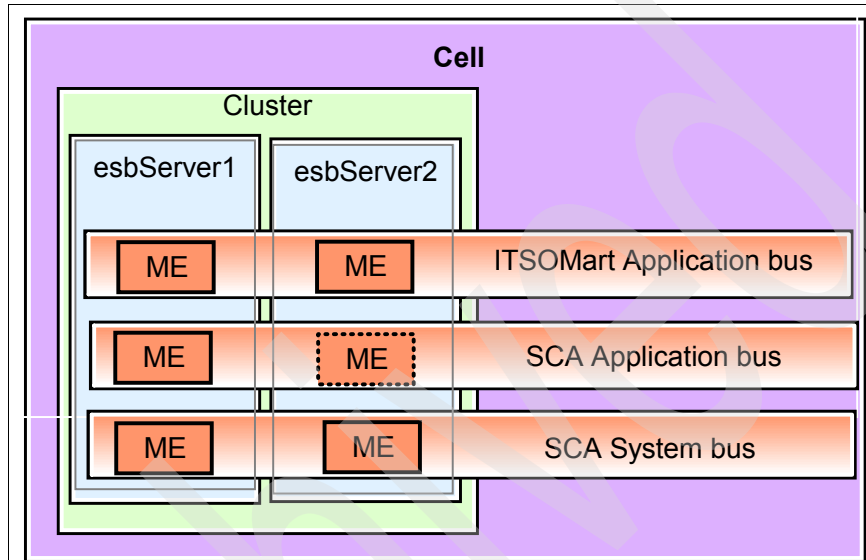


Figure 11-36 Single cluster topology

Install the product

During installation of WebSphere ESB you have the option to create an application server profile or to delay profile creation. If this is a new installation and you plan to use clustering, do the following:

1. Choose the **Custom install** option.
2. Do not install the samples or the javadocs.
3. Elect not to start the Profile Creation Wizard or the First Steps wizard.

This will delay the profile creation process.

If you plan to create clustered servers across multiple machines, you will have to install WebSphere ESB on those machines as well.

Create the messaging engine data stores

Each messaging engine has its own data store where it stores operating information and persists those objects that messaging engines need for recovery in the event of a failure. This data store consists of a set of tables in a database.

In a single-server environment, you have the option of using Cloudscape for the database and have it set up automatically when you add a member to a bus. In a clustered environment, you no longer have the option to use the Cloudscape JDBC provider and database. You will need to provide a JDBC provider (for example, DB2) and create the database manually.

Before getting started with creating profiles, we create the databases we need for the messaging engines. A common way of deploying these databases is to have one per bus used in the deployment configuration, and have the tables associated with every messaging engine created there. Each messaging engine uses a different schema to create entries in the database.

We created the following DB2 databases:

- ▶ SCADB - Persistent repository for the SCA application and system buses
- ▶ SCASINGL - Persistent repository for ITSOMartBus

Create the deployment manager

When building a distributed environment, you first need a deployment manager:

1. Start the profile creation wizard and create a deployment manager profile.

The options you select as you go through the wizard depend on your environment. The following are settings that we used. Our choices were primarily based on the fact that we were in a lab environment:

- We used the default port numbers.
 - We elected not to run the deployment manager as a Windows service.
 - We did not configure the bus for secure communication.
 - We used very short cell, node, and profile directory names (to avoid any potential problems with using long names in our deployment environment on Windows).
2. On the Summary panel, select **Next** and wait for the profile creation to complete.
 3. When the profile creation process is complete, check the **Launch First Steps** check box and click **Finish**.
 4. In the First Steps window, click the option to start the deployment manager and, once it has started, log in to the administrative console.

At this point the bus configuration of our cell looks like Figure 11-37. Note that the two SCA buses were created during the profile creation process.

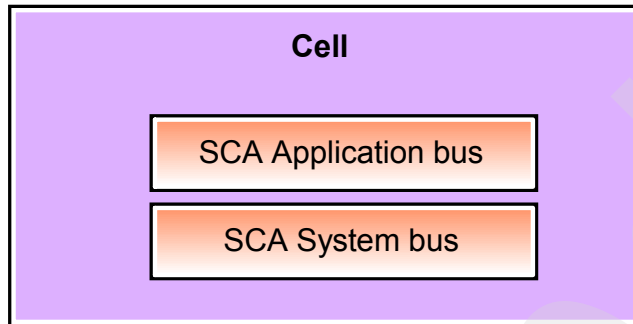


Figure 11-37 Cell configuration - deployment manager creation

Create the nodes

You must create a Custom profile that defines the node that will reside on each machine you will use in the cluster. Nodes can be defined with custom profiles and then added (federated) to the cell managed by the deployment manager.

A node contains a node agent that works with the deployment manager to synchronize configuration files and to manage the servers on the node. You would normally have one node per machine, but it is possible to create multiple nodes on one machine.

1. Start the Profile Creation Wizard and create a custom profile.
2. Leave “federate this node later using addNode” unchecked. If you did not use the default ports for the deployment manager, make sure the correct SOAP port is entered.

The wizard will federate this node to the deployment manager for you. You need to make sure the deployment manager is up and running before you continue.

3. On the Summary panel, select **Next** and wait for the profile creation to complete.
4. When the profile creation process completes, check the **Launch First Steps** check box and finish.

We created two custom profiles, and because this was a lab environment, these nodes resided on the same machine as the deployment manager:

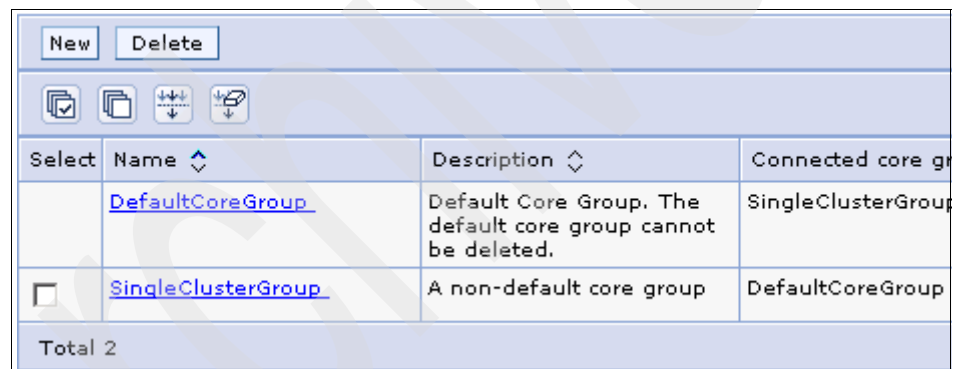
- ▶ Custom01 profile for Node01
- ▶ Custom02 profile for Node02

Create a core group

A core group is a high-availability domain that consists of a set of processes in the same cell that can directly establish high-availability relationships. Highly available components can only fail over to another process in the same core group and replication can occur only between members of the same core group. Each core group contains a set of high availability policies that are used to manage the highly available components within that core group.

The cell will have a default core group defined and you can use this core group if you wish. We opted to create a new core group. Later we will add a cluster to this core group and define a high-availability policy within this core group for the cluster.

1. Log in to the deployment manager administrative console and select **Servers** → **Core groups** → **Core group settings**.
2. Click **New**.
3. Enter a name for the core group (we used SingleClusterGroup) and click **OK** at the bottom.



<input type="button" value="New"/> <input type="button" value="Delete"/>			
<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>			
Select	Name	Description	Connected core group
<input checked="" type="checkbox"/>	DefaultCoreGroup	Default Core Group. The default core group cannot be deleted.	SingleClusterGroup
<input type="checkbox"/>	SingleClusterGroup	A non-default core group	DefaultCoreGroup
Total 2			

Figure 11-38 Adding a core group

4. Save and synchronize your changes with the nodes in the cell. To do this, click **Save**. Check the box associated with “Synchronize changes with nodes” and click **Save** again.

Create the cluster

We are now ready to create our cluster together with the first server member using the defaultESBServer template.

1. In the administrative console select **Servers** → **Clusters**.
2. Click **New**.

3. Enter the cluster name. We used SingleCluster. We will create new servers for this cluster, so select **Do not include an existing server in this cluster** and click **Next**.
4. This step lets you create one or more servers for the cluster:
 - a. Provide a name for the first server in the cluster. We used esbServer1.
 - b. Pick the node in which you want the server to reside.
 - c. Select the core group **SingleClusterGroup**.
 - d. Select the template **defaultESBServer**.

The contents of the panel for this step should look like Figure 11-39.

Step 1: Enter basic cluster information.

→ Step 2: Create cluster members

Step 3: Summary

Create cluster members

Enter information about this new cluster member, and click Apply to add this cluster member to the member list. Use the Edit function to edit the properties of a cluster member that are already included in this list. Use the Delete function to remove a cluster member from this list.

* Member name
esbServer1

Select node
Node01(6.0.2.5)

* Weight
2

Core Group
SingleClusterGroup

☒ Generate Unique Http Ports

Select template:
☒ Default application server template
Choose a server template from this list:
defaultESBServer

Figure 11-39 Creating a cluster - adding a cluster member

Click the **Apply** button. You could add additional servers by repeating this process, but for now, we will only create the first server. Click **Next**.

5. Verify your settings on the Summary panel and click **Finish**.
6. Save and synchronize your changes.

At this point the cell configuration looks like Figure 11-40.

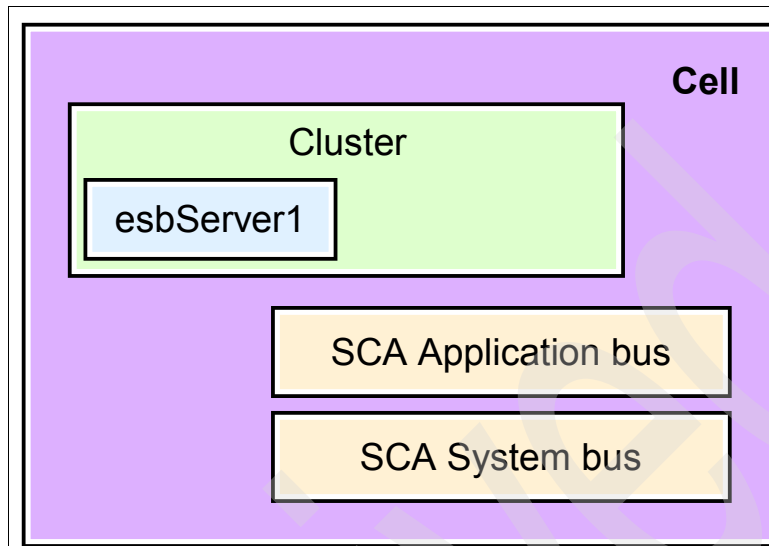


Figure 11-40 Cell configuration after creating the cluster

Create the data source for the messaging engines

You must create a JDBC provider and a data source for the database associated with the messaging engines on ITSOMartBus. The JDBC Provider and configuration for the SCA application and system buses, including the authentication alias to the database, are created when you complete “Configure the cluster to support SCA” on page 546.

We first create the J2C authentication data entry containing the user ID and password required to authenticate with DB2.

1. In the administrative console select **Security** → **Global Security**.
2. Under authentication, expand **JAAS Configuration**.
3. Select **J2C Authentication data** and create a new alias.
4. Provide a name for the alias (we used db2_alias) and provide a valid user ID and password for DB2.
5. Save and synchronize your changes.

We now define a JDBC provider for the DB2 database that is going to hold the messages for the ITSOMartBus.

1. In the administrative console select **Resources** → **JDBC Providers**.
2. Set the scope level to the new cluster.

3. Click **New**.
4. Select:
 - **DB2** for the database type
 - **DB2 Universal JDBC Provider** for the provider type
 - **Connection pool data source** for the implementation typeClick **Next**.
5. Click **Apply**.
6. Under Additional Properties select **Data sources**.
7. On the Data Sources panel, click **New**.
8. Provide the following input:
 - Name: DB2 DataSource for SingleCluster
 - JNDI Name: jdbc/single
 - Uncheck the “Use this Data source in Container Managed Persistence” check-box.
 - Component-Managed authentication alias: `<dmgr_node_name>/db2_alias`
Note that it is preferable to define the authentication alias on the messaging engine versus here. We defined it here so we could test the data source, but will remove it before going into production.
 - Database Name: SCASINGL
 - Driver type: 4
 - Server Name: `<DB2_hostname>`
 - Port Number: `<DB2_port_number>`Click **OK**.
9. Several of the default data source definition fields use standard WebSphere variables to describe the location of files. You will need to update your WebSphere variables to contain the correct paths. For a cluster, define these variables at the cluster scope if DB2 is installed in the same path on all nodes. Otherwise use the Node scope.
 - DB2_UNIVERSAL_JDBC_DRIVER_PATH = C:\IBM\SQLLIB\java
 - DB2_UNIVERSAL_JDBC_DRIVER_NATIVEPATH = C:\IBM\SQLLIB\java
10. Save and synchronize your changes.
11. Test the connection to the database by navigating back to the data source. Check the box to the left and click **Test Connection**.

Tip: Data sources that use WebSphere variables may function differently during the test connection function than they would during runtime. The difference is in how the WebSphere variables are resolved in runtime versus during the test. If you created the variables and data sources at the cluster scope, the test connection should work. However, if you defined the data source at the cluster level then the test connection will most likely fail, but the connection should function during runtime.

For more information see “Test connection service” in the WebSphere ESB Information Center:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/topic/com.ibm.websphere.nd.doc/info/ae/ae/cdat_testcon.html

Create the bus for the mediations

We have chosen to create a new bus for the queue destinations used by ITSOMart. To create the bus:

1. From the WebSphere administrative console, select **Service integration** → **Buses**.
2. Click **New**.
3. Enter ITSOMartBus for the name and click **Apply**.
4. Click the **Bus members** link under Topology.
5. Click **Add**.

- Click **Cluster** and select **SingleCluster**. Enter jdbc/single as the data source JNDI name (Figure 11-41). Click **Next**.

→ **Step 1: Select server or cluster**

Step 2: Confirm the addition of a new bus member

Select server or cluster

Choose the server or cluster to add to the bus

☐ **Server**

Server
Node01:esbServer1

Data store

☒ Default

Data source JNDI name

☒ **Cluster**

Cluster
SingleCluster

Data store

* Data source JNDI name
jdbc/single

Figure 11-41 Adding the cluster as a ITSOMartBus member

- Click **Finish**. Click **Next**.
- Verify your settings on the Summary panel and click **Finish**.
- Save and synchronize the changes.

Since we are using one database to contain the entries from multiple messaging engines, each messaging engine must use a unique schema for the database. We also set the authentication alias for the data store.

To change the schema name used by SingleCluster, do the following:

- In the administrative console select **Service Integration** → **Buses** → **ITSOMartBus**.
- Under topology select **Bus Members**.
- Select the cluster member we just added, in our case SingleCluster.

4. When the messaging engines panel comes up select the messaging engine associated with the cluster member above, in our case, SingleCluster.000-ITSOMartBus.
5. Under Additional Properties, click **Data Store**.
6. Choose a schema name and change it here (remember the length of this type of names is restricted to eight characters).
7. Select the authentication alias for the data store. The panel should look like Figure 11-42.

The screenshot shows a web-based configuration interface for a messaging engine. At the top, a breadcrumb trail reads: **Buses > ITSOMartBus > Messaging engines > SingleCluster.000-ITSOMartBus > Data store**. Below this, a description states: "The persistent store for messages and other state managed by the messaging engine." A tab labeled "Configuration" is active. The main content area is divided into two sections: "General Properties" and "Related Items". Under "General Properties", there are several fields: "UUID" with the value "6C38CEFE6FCCE825", "* Data source JNDI name" with the value "jdbc/single", "Schema name" with the value "singled", and "Authentication alias" with a dropdown menu showing "DMnode/db2_alias". There is also a checked checkbox labeled "Create tables". At the bottom of the "General Properties" section are four buttons: "Apply", "OK", "Reset", and "Cancel". The "Related Items" section on the right contains a single link: "J2EE Connector Architecture (J2C) authentication data entries".

Figure 11-42 Changing the schema used by a messaging engine

Note that the Create Tables check box is checked, meaning the tables associated with the messaging engine will be created the first time the cluster is started. The schema name proves to be a good filter when you want to only view the tables associated with a particular messaging engine.

Note also that the messaging engine is associated with the cluster and not with any of the servers in the cluster. Later we assign a messaging engine to each server member of the cluster.

8. Click **OK**.
9. Save and synchronize your changes.
10. Restart the deployment manager.

Starting nodes and clusters: This may be a good point to start the cluster and verify that the messaging engine starts successfully. Before starting the cluster, ensure that the node agent for each cluster has started.

Start the node:

1. Open a command window and navigate to the bin directory for the custom profile, for example:

```
cd \pf\Custom01\bin
```

2. Check the status of the node agent:

```
serverStatus -all
```

3. If the status of nodeagent is stopped, start the node agent with the following command:

```
startNode
```

Once each node agent is up and running, you can start the cluster from the administrative console:

1. Select **Servers** → **Clusters**.
2. Check the box to the left of the cluster and click **Start**.

This starts each server in the cluster. This may take a few minutes. You can refresh the status column. Once you see the solid green arrow, the servers in the cluster have started.

To check the messaging engine for the cluster:

1. Select **Service integration** → **Buses**.
2. Click **ITSOMartBus** to open the details page.
3. Select **Messaging engines** under Topology.

You should see a solid green arrow by the messaging engine name indicating that it has started.

Our cell configuration at this point looks like Figure 11-43.

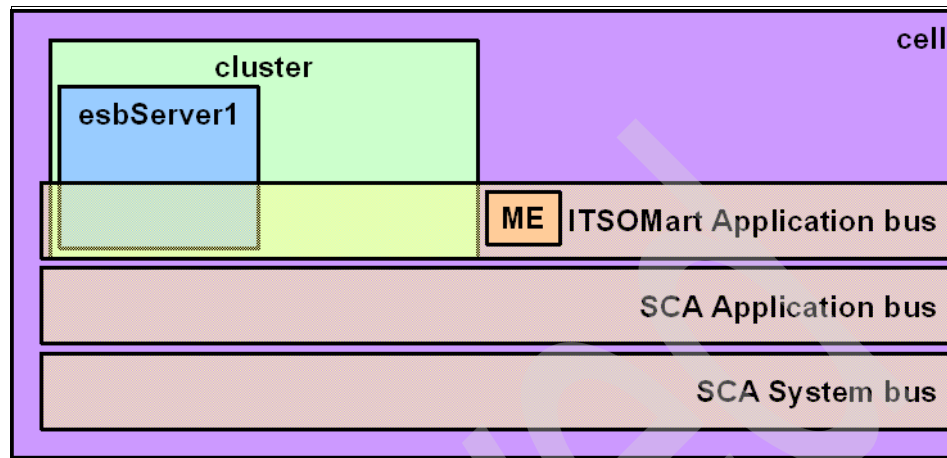


Figure 11-43 Cell configuration after adding a cluster as a member to ITSOMartBus

Configure the cluster to support SCA

Now we must configure the cluster to support SCA. This automatically adds the cluster to the SCA application and system buses.

1. Select **Servers** → **Clusters** → **SingleCluster**.
2. Under Additional Properties select **Advanced Configuration**.
 - a. Select the **Default Destination Location** check button to specify that the messaging engines are to be created locally. This has the effect of adding the cluster as a member of the two SCA application and system buses. A single messaging engine will be created on each bus for the cluster bus member.
 - b. Select the proper JDBC provider for your system.
 - c. Enter values for the following:
 - Application bus schema name: ASCH
 - System bus schema name: SSCH
 - DB2 user name and password

- d. Make sure the check box for Create tables is checked (Figure 11-44).

Service Component Architecture

☐ Do not configure to host SCA applications.

☐ Remote Destination Location

☒ Default Destination Location

JDBC Provider
DB2 UDB 8.1 & 8.2 (DB2 Universal JDBC Driver Provider (XA))

Application bus schema name: ASCH

System bus schema name: SSCH

User Name: db2admin

Password: *****

Application Bus Database Name:
databaseName=SCADB
driverType=2
serverName=localhost
portNumber=50000
description=SIB DataSource for SCA
traceLevel=
traceFile=
fullyMaterializeLobData=true
resultSetHoldability=2
currentPackageSet=

System Bus Database Name:
databaseName=SCADB
driverType=2
serverName=localhost
portNumber=50000
description=SIB DataSource for SCA
traceLevel=
traceFile=
fullyMaterializeLobData=true
resultSetHoldability=2
currentPackageSet=

Create tables ☒

Figure 11-44 Advanced Configuration panel

3. Click **OK**.
4. Save and synchronize your changes.

Our cell configuration at this point looks like Figure 11-45.

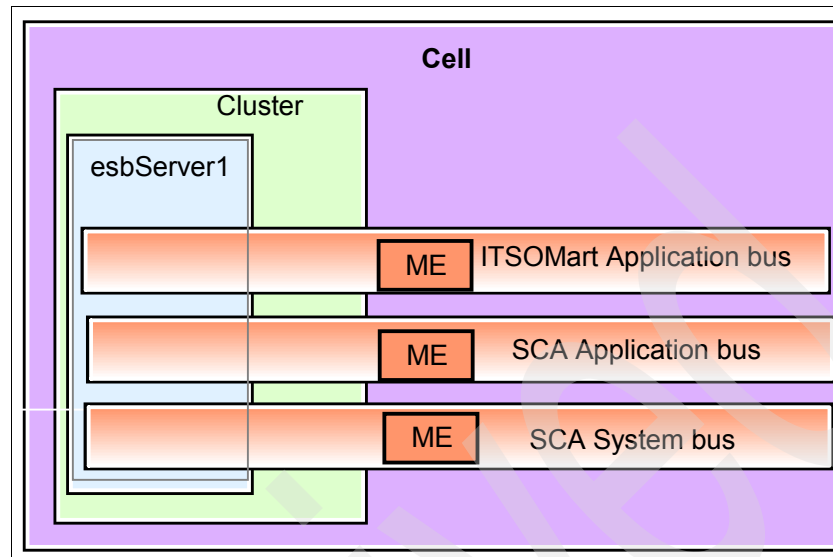


Figure 11-45 Cell configuration after enabling SCA for the cluster

Define the high-availability policy

By default, the messaging engine starts in the first available server in the cluster. To enable workload management for messaging engines in the cluster, we want each server in the cluster to have its own messaging engine. This is achieved by defining a high-availability (HA) policy that matches the messaging engine name and defining a preferred server (the server where the messaging engine is going to run). We should also enable fail-back so that if the messaging engine starts on another server, it will fail back to the preferred server when it becomes available.

1. In the administrative console select **Servers** → **Core groups** → **Core group settings**.
2. Go into the core group we defined for this topology, SingleClusterGroup.
3. Under Additional Properties select **Policies**.
4. Click **New**.
 - a. From the Configuration panel select **One of N policy** and click **Next**.
 - b. We want the name of the policy to reflect its association with a specific messaging engine, so we use the name of the messaging engine as the name of the policy. In our case it is SingleCluster.000-ITSOMartBus.
 - c. Check the **Fail-back** check box.
 - d. Click **Apply**.

- The links that were previously disabled under Additional Properties are available now, and there is a message in red reminding us that we have to complete the matching criteria for this policy.


 The policy must have at least one match criteria defined.

Figure 11-46 Warning before the matching criteria for the policy is defined

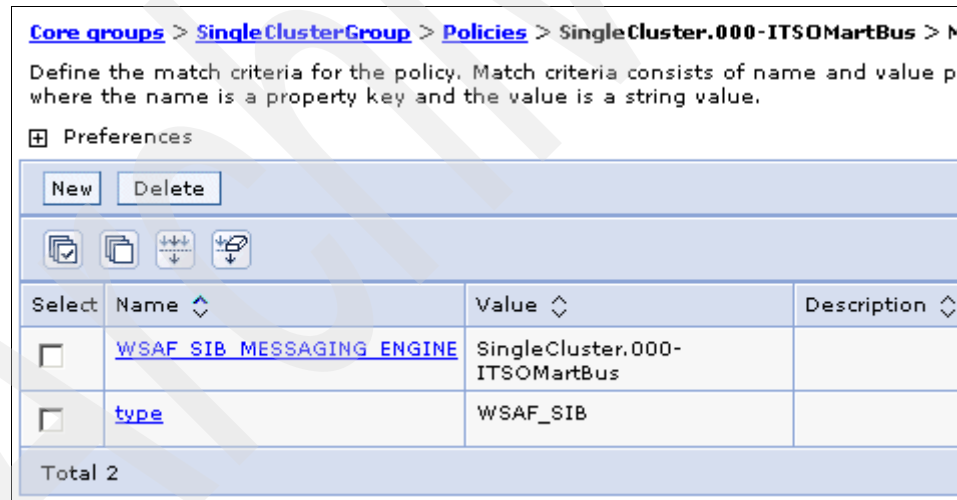
- Under Additional Properties select **Match Criteria**.
- When the Match Criteria panel comes up, click **New** and enter the following name/value pair:

- Name: type
- Value: WSAF_SIB

Enter a second name/value pair. Note that the value is the messaging engine name.

- Name: WSAF_SIB_MESSAGING_ENGINE
- Value: SingleCluster.000-ITSOMartBus

The list of match criteria for the policy should look like Figure 11-47.



Select	Name	Value	Description
<input type="checkbox"/>	WSAF_SIB_MESSAGING_ENGINE	SingleCluster.000-ITSOMartBus	
<input type="checkbox"/>	type	WSAF_SIB	

Total 2

Figure 11-47 Matching criteria for the policy

- Navigate back to the new policies details page and select **Preferred Servers** under Additional Properties.
- Add the cluster member esbServer1 as the preferred server.
- Click **OK**.

11. Repeat these steps to set the preferred server for the SCA System bus to esbServer1. In our example, the messaging engine name is SingleCluster.000-SCA.SYSTEM.ESB1.Bus.
12. Save and synchronize your changes.

Our cell configuration at this point looks like Figure 11-48.

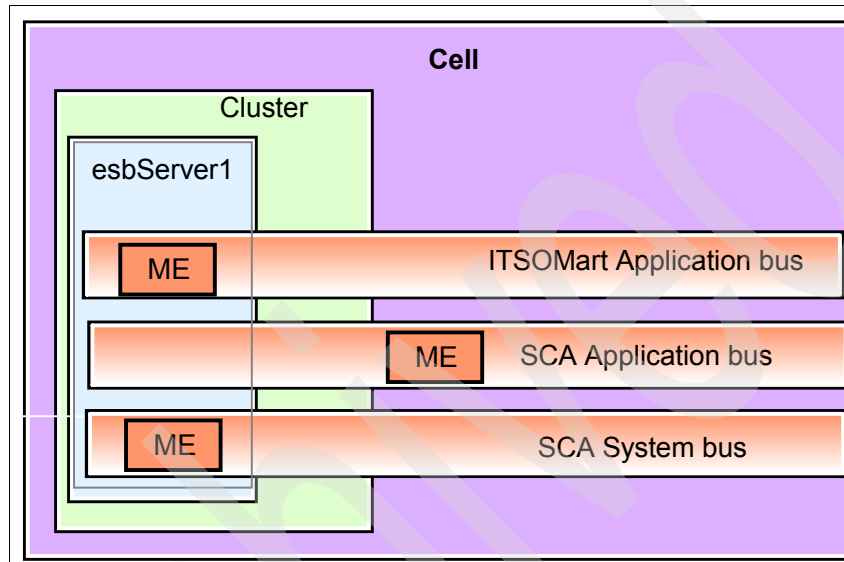


Figure 11-48 Cell configuration after the definition of the HA policy

Add additional members to the cluster

Now it is time to grow our cluster by adding a new member.

1. In the administrative console select **Servers** → **Cluster** → **SingleCluster**.
2. Under Additional Properties select **Cluster members**.
3. Click **New**.
 - a. Provide a member name. We used esbServer2.
 - b. Pick a node from the ones you just federated into the cell.
4. Click **Apply**.
5. Click **Next**.
6. Verify your settings on the Summary panel and click **Finish**.
7. Save and synchronize your changes.

Our cell configuration at this point looks like Figure 11-49.

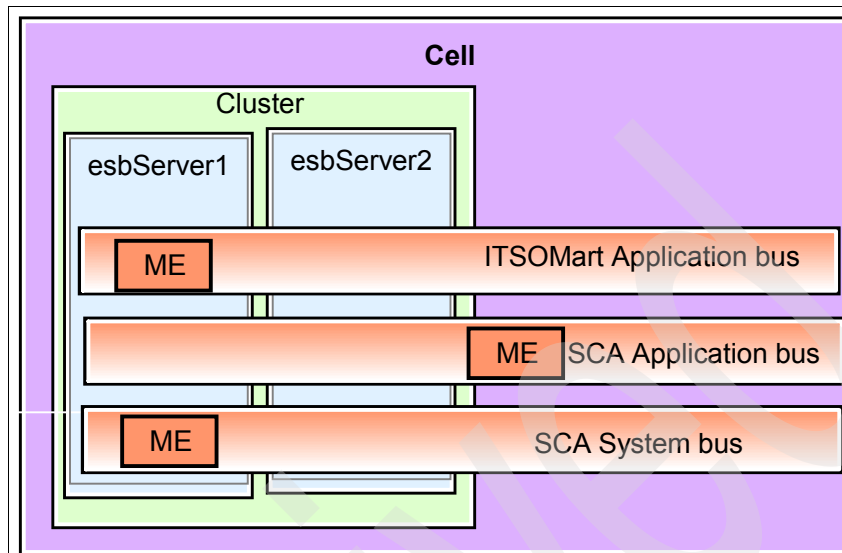


Figure 11-49 Cell configuration after adding the second server to the cluster

Add messaging engines

We now add a new messaging engine for ITSOMartBus and the SCA System bus and assign it to run on the new member of the cluster.

First add the new messaging engine for ITSOMartBus:

1. In the administrative console select **Service Integration** → **Buses** → **ITSOMartBus**.
2. Under Topology select **Bus members**.
3. Select the bus member cluster, **SingleCluster**.
4. When the Messaging Engine Panel comes up, click **Add messaging engine**.
5. Fill in the following values for the new messaging engine data source:
 - Data source JNDI name: jdbc/single
 - Schema Name: CLONE2
 - Authentication Alias: <node name>/db2_aliasClick **OK**.
6. Create a new high-availability policy with the name of the second messaging engine and match criteria for the messaging engine name, SingleCluster.001-ITSOMartBus. Select **esbServer2** as the preferred server. (see “Define the high-availability policy” on page 548).

7. Save and synchronize your changes.
8. Repeat these steps to create a new messaging engine for the SCA system bus and set the preferred server to be esbServer2. In our example, the new messaging engine name is SingleCluster.001-SCA.SYSTEM.ESB1.Bus.

This may be a good point to start the cluster and verify that all the messaging engines come up as follows:

- ▶ esbServer1 should come up with the SingleCluster.000-ITSOMartBus and the SingleCluster.000-SCA.SYSTEM.ESB1.Bus messaging engines.
- ▶ esbServer2 should come up the SingleCluster.001-ITSOMartBus messaging and the SingleCluster.001-SCA.SYSTEM.ESB1.Bus engines.
- ▶ The messaging engine for the SCA application bus could come up on either of the cluster members.

Our cell configuration at this point looks like Figure 11-50.

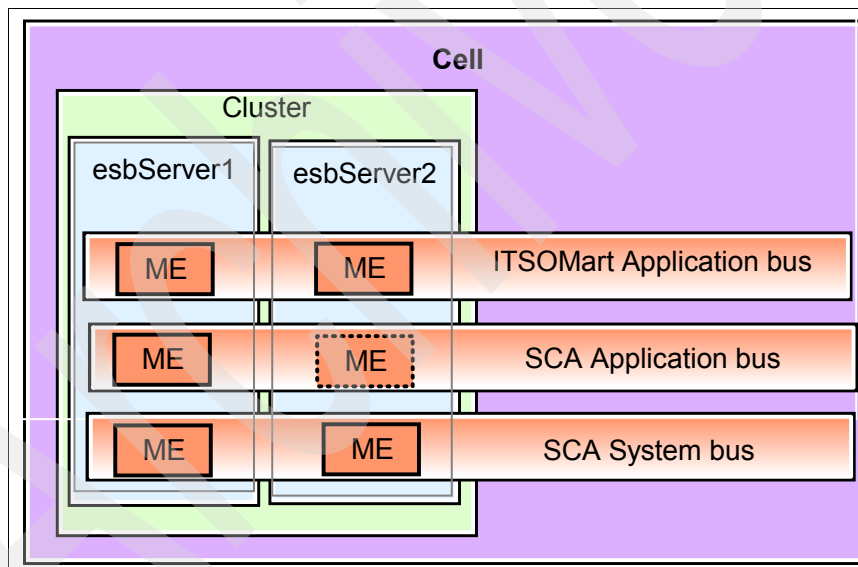


Figure 11-50 The single cluster topology

Deploy the mediations to the cluster

Deploying mediations to a cluster environment is basically the same as to a single server environment. The difference is in the scope you use to define resources, and the applications will be installed to a cluster versus a to a single server. Consider the following:

- ▶ Create virtual hosts for your servers and clusters or make sure that the alias list for the default_host virtual server is updated to include the Web container default host port.
- ▶ Create JDBC resources at the deployment manager node and cluster scope.
- ▶ Install the mediation applications to the cluster. You can select the cluster during the install during the *map modules to servers* step.
- ▶ Create the bus destinations required.
- ▶ If you add Web services support for the bus, you will not be able to use the default Cloudscape database for the SDO repository.

11.13 For more information

See the following resources:

- ▶ WebSphere Enterprise Service Bus 6.0.1 Information Center
<http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/topic/com.ibm.websphere.wesb.doc/info/welcome.html>
- ▶ *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212
- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- ▶ *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688



Service monitoring and management with IBM Tivoli Composite Application Manager SOA

This chapter is an overview of the IBM Tivoli Composite Application Manager for SOA. System administrators use IBM Tivoli Composite Application Manager for SOA along with the IBM Tivoli Enterprise Monitoring framework to monitor and manage Web services in a composite application environment.

This chapter includes a brief overview of the architecture of the composite application management environment. The ITSOMart application will be used to illustrate application management using IBM Tivoli Composite Application Manager for SOA.

This chapter includes the following topics

- Tivoli Composite Application Manager (ITCAM)
- IBM Tivoli Enterprise Monitoring framework
- IBM Tivoli Composite Application Manager for SOA
- Tracking performance with ITCAM for SOA
- Monitoring ITSOMart

12.1 Tivoli Composite Application Manager (ITCAM)

IBM Tivoli Composite Application Manager is the name for a suite of Tivoli monitoring products that are positioned for performance monitoring and management of composite applications. Typically composite applications consist of many components including Web servers, application servers, database servers, and back-end systems such as CICS and IMS running on mainframe systems. Traditional application monitoring tools provide the ability to observe each individual system layer but do not provide a way to perform end-to-end monitoring of the entire application. Figure 12-1 shows the complete Tivoli solution portfolio for storage, security, and systems management.

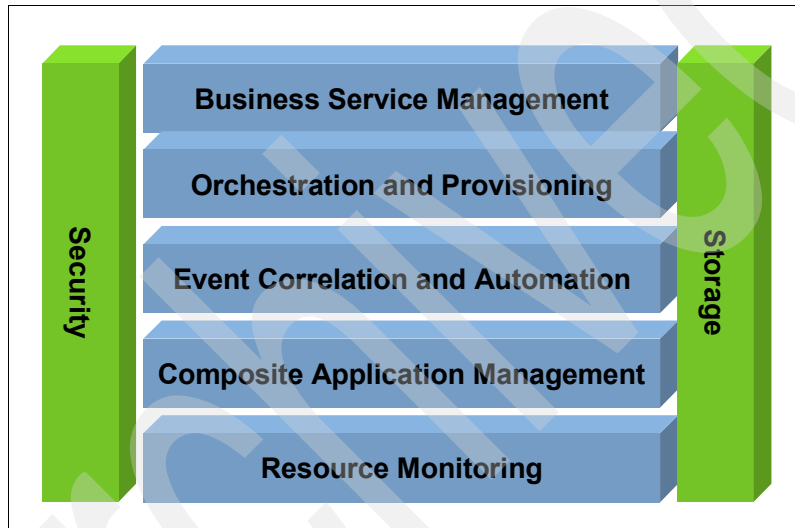


Figure 12-1 Tivoli solution stack

The complete solution stack offers the following features:

- ▶ **Resource Monitoring**
Measuring and managing IT resource performance including servers, databases, and middleware.
- ▶ **Composite Application Management**
Monitoring and managing an application and its components and providing a view on application availability.
- ▶ **Event Correlation and Automation**
Correlates and automates events or faults that are generated by resource monitoring and application monitoring to provide information about root-cause analysis of failure in the environment.

- ▶ **Orchestration and Provisioning**
Provides the ability to deploy components as required to fulfill processing needs, if the need arises as indicated by the correlation engine.
- ▶ **Business Services Management**
Provides a high-level view of business status as reflected by its underlying monitoring components. The view can either be in real time or based on a service level agreement (SLA).

The IBM Tivoli Composite Application Manager suite consists of the following products:

- ▶ **IBM Tivoli Composite Application Manager Response Time Tracking V6.0**
Proactively recognizes, isolates, and resolves transaction performance problems
- ▶ **IBM Tivoli Composite Application Manager for SOA 6.0**
Monitors and manages the Web services layer of an IT architecture
- ▶ **IBM Tivoli Composite Application Manager for WebSphere V6.0**
Can isolate the root cause of bottlenecks in a WebSphere application runtime environment
- ▶ **IBM Tivoli Composite Application Manager for CICS Transaction V6.0**
Provides data from CICS systems to be used in ITCAM for response time tracking
- ▶ **IBM Tivoli Composite Application Manager for IMS Transactions V6.0**
Provides data for IMS systems to be used in ITCAM for response time tracking
- ▶ **OMEGAMON XE for WebSphere Business Integration V1.1**
Can monitor WebSphere MQ family runtimes and provide automatic corrective actions to improve performance and availability

Table 12-1 Mapping between new IBM terminology and previous Candle terminology

IBM Tivoli terminology	Candle/OMEGAMON terminology
IBM Tivoli Monitoring Services	OMEGAMON Platform
IBM Tivoli Monitoring	OMEGAMON for distributed products
Tivoli Enterprise Monitoring Server	Candle® Management Server (CMS)
Tivoli Enterprise Monitoring Agent	OMEGAMON Monitoring Agent®
Tivoli Enterprise Portal Server	CandleNET Portal Server

IBM Tivoli terminology	Candle/OMEGAMON terminology
Tivoli Enterprise Portal	CandleNET Portal
Situation Event Console	Enterprise Event Console/Event Console
Tivoli Enterprise Portal	Candle Management Workstation
Database	Candle Data Warehouse

Table 12-1 on page 557 lists the mapping between the new IBM terminologies and the previous Candle terminology. It is worth noting that in the documentation slightly differing Candle terminologies may mean the same in the IBM terminologies.

12.1.1 Composite applications

Modern SOA applications are increasingly depend on multi-tier, composite applications. To fully realize the benefits of SOA, it is important that composite applications provide effective performance metrics in real time to manage availability and to meet business goals in terms of service level agreements (SLAs).

In general, composite applications are more difficult to build, test, and manage for high performance. Although techniques and best practices exist for designing high-performance systems, monitoring tools that can provide details of performance metrics for individual components as well as the end-to-end view of the composite application are needed. See Figure 12-2.

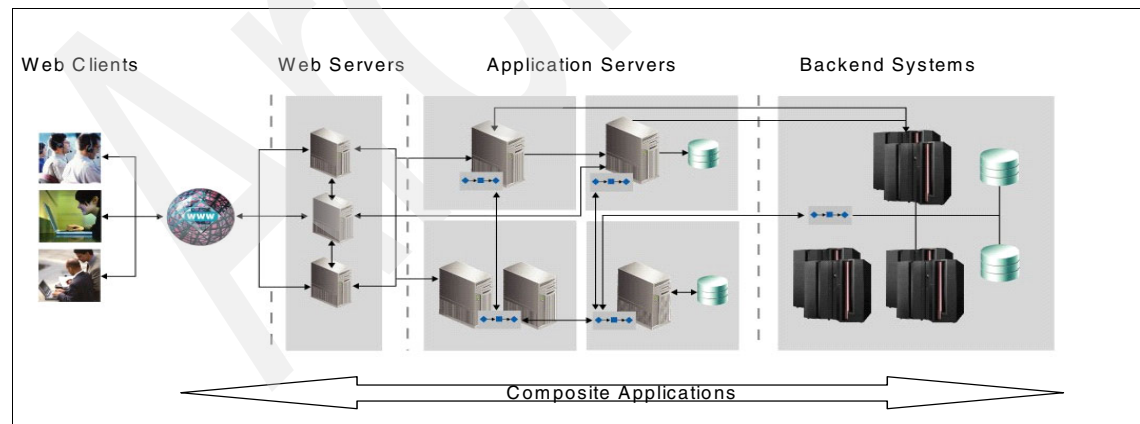


Figure 12-2 Composite Application consisting of several different types of components

To effectively manage composite applications you must consider three aspects of applications, as summarized in Figure 12-3. There is the management aspect for an end-to-end transaction, application perspective, and the individual resources perspective.

From a transaction point of view, it is important that the service level response time is met and that you are able to isolate problems that cause a degradation in the overall response time. The transaction perspective provides an end-user view of performance. ITCAM for response time tracking is positioned for this.

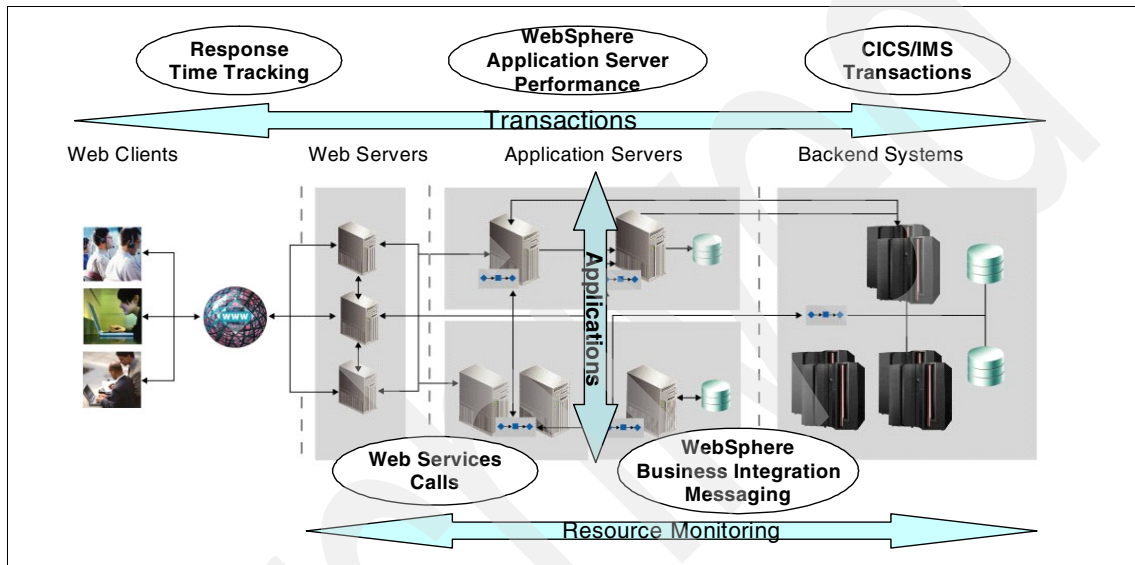


Figure 12-3 Three dimensions of effective application management

To support the end-to-end view, it is important to be able to isolate resource utilization issues at individual component levels such as application server metrics or Web service call utilization. This is where the other products in the family, such as ITCAM for WebSphere or ITCAM for CICS/IMS, come in. They can provide in-depth analysis of resource utilization and bottlenecks that are pertinent to that component.

12.2 IBM Tivoli Enterprise Monitoring framework

The ITCAM products are designed to work together within one framework, called the IBM Tivoli Enterprise Monitoring framework. The framework can be seen in Figure 12-4.

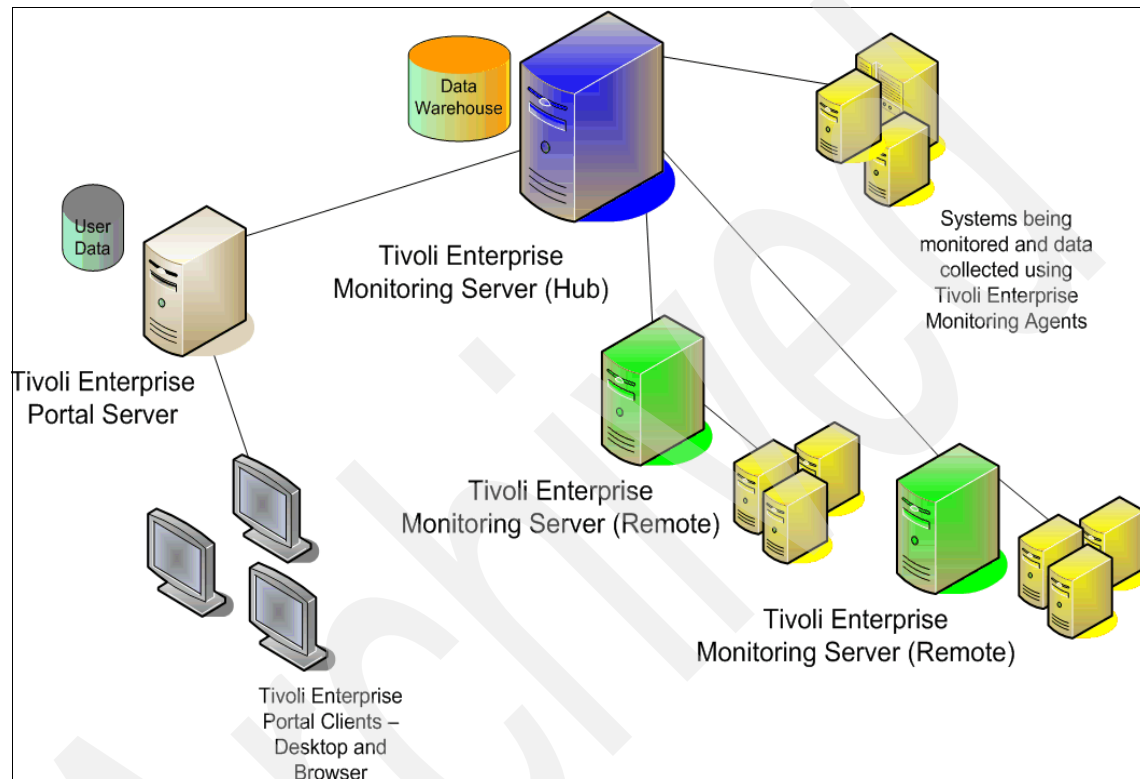


Figure 12-4 Components of the IBM Tivoli Enterprise Monitoring framework

Tivoli Enterprise Monitoring Server (TEMS)

The Tivoli Enterprise Monitoring Server (or monitoring server) is the initial component to install to begin building the IBM Tivoli Monitoring Services foundation. It is the component on which all other architectural components depend. The TEMS acts as a collection and control point for alerts received from agents, and collects their performance and availability data. The TEMS is responsible for tracking the heartbeat request interval for all Tivoli Enterprise Management Agents connected to it.

The TEMS stores, initiates, and tracks all situations and policies, and is the central repository for storing all active conditions and short-term data on every

Tivoli Enterprise Management Agent. Additionally, it is responsible for initiating and tracking all generated actions that invoke a script or program on the agent.

The TEMS stores its data in a collection of log files, which can be persisted in a relational database.

The primary TEMS is configured as a hub. All IBM Tivoli Monitoring installations require at least one TEMS configured as a hub. Additional remote TEMS can be installed later to introduce a scalable hierarchy into the architecture.

This hub/remote interconnection provides a hierarchical design that enables the remote TEMS to control and collect its individual agent status and propagate the agent status up to the hub TEMS. This mechanism enables the hub TEMS to maintain infrastructure-wide visibility of the entire environment. This visibility is passed to the Tivoli Enterprise Portal Server for formatting, ultimately displaying in the Tivoli Enterprise Portal client.

Tivoli Enterprise Portal Server (TEPS)

The Tivoli Enterprise Portal Server, also known as portal server, is a repository for all graphical presentation of monitoring data. The portal server database also consists of all user IDs and user access controls for the monitoring workspaces. The TEPS provides the core presentation layer, which allows for retrieval, manipulation, analysis, and formatting of data. It manages this access through user workspace consoles. The TEPS keeps a persistent connection to the hub TEMS, and can be considered a logical gateway between the hub TEMS and the Tivoli Enterprise Portal client. Any disconnection between the two components immediately disables access to the monitoring data used by the Tivoli Enterprise Portal client.

A database must be installed on the same host prior to the TEPS installation. This prerequisite is necessary because the TEPS installation will create the required TEPS database, along with the supporting tables. Also, an ODBC (data source name) is configured to connect directly to the Tivoli Data Warehouse database. This ODBC connection is used whenever a pull of historical data from the Tivoli Data Warehouse is requested.

Tivoli Enterprise Portal clients

The TEP client (referred to as the portal client) is a Java-based user interface that connects to the Tivoli Enterprise Portal Server to view all monitoring data collections. It is the user interaction component of the presentation layer. The TEP brings all of these views together in a single window so you can see when any component is not working as expected. There are two TEP clients, namely, the Java desktop client and an HTTP browser client.

To invoke the browser-based TEP client, use the URL:

`http://<hostname>:1920/cnp/kdh/lib/cnp.html`

Where `<hostname>` is the host name of the Tivoli Enterprise Portal Server.

The following products have integrated interfaces into the TEP:

- ▶ OMEGAMON z/OS®
- ▶ OMEGAMON Distributed
- ▶ IBM Tivoli Monitoring 5.1.2
- ▶ IBM Tivoli Monitoring 6.1
- ▶ NetView® for z/OS (release 5.2)
- ▶ IBM Tivoli Enterprise Console®
- ▶ IBM Tivoli Composite Application Manager for Response Time Tracking
- ▶ IBM Tivoli Composite Application Manager for WebSphere
- ▶ IBM Tivoli Composite Application Manager for SOA

This means that a consolidated view of a composite application has been made available through a single client.

Tivoli Enterprise Monitoring Agent (TEMA)

The Tivoli agents are installed on the systems or subsystems requiring data collection and monitoring. The agents are responsible for data gathering and distribution of attributes to the monitoring servers, including initiating the heartbeat status.

Monitoring applications are identified by product code, a three-letter identifier that starts with the letter K. Some identifiers are:

- ▶ KUM: Universal Agent
- ▶ KNT: Windows OS monitoring
- ▶ KD4: ITCAM for SOA
- ▶ KYN: ITCAM for WebSphere
- ▶ KT2: ITCAM for RTT

These agents test attribute values against a threshold and report these results to the monitoring servers. The TEP displays an alert icon when a threshold is exceeded or a value is matched. The tests are called *situations*.

When a portal workspace is opened or refreshed, the TEPS sends a sampling request to the hub TEMS. The request is passed to the monitoring agent if there is a direct connection or through the remote TEMS to which the monitoring agent connects. The monitoring agent takes a data sampling and returns the results through the monitoring server and portal server to the portal workspace.

The sampling interval for a situation (a test taken at your monitored systems) can be as often as once per second or as seldom as once every three months. When the interval expires, the monitoring server requests data samples from the agent and compares the returned values with the condition described in the situation. If the values meet the condition, the icons change on the navigation tree.

Optionally, the agents can be configured to transfer data collections directly to the Warehouse Proxy agent instead of using the remote TEMS. If firewall restrictions are disabled or minimal, you should configure all the agents to transfer directly to the Warehouse Proxy agent. Otherwise, firewall security is a key factor in the location of the Warehouse Proxy agent relative to the firewall zone and agents. Warehousing data through the remote TEMS is limited and should be used only as a last resort.

Tivoli Enterprise Management Agents are grouped into two categories:

- ▶ Operating system (OS) agents
Operating system agents retrieve and collect all monitoring attribute groups related to specific operating system management conditions and associated data.
- ▶ Application agents
Application agents are specialized agents coded to retrieve and collect unique monitoring attribute groups related to one specific application. The monitoring groups are designed around an individual software application, and they provide in-depth visibility into the status and conditions of that particular application.

Common management agents packaged with IBM Tivoli Monitoring include:

- ▶ Window OS Agent
- ▶ Linux OS Agent
- ▶ UNIX OS Agent
- ▶ UNIX Log Agent
- ▶ i5 OS Agent
- ▶ Universal Agent

The Universal Agent is a special agent that leverages an API to monitor and collect data for any type of software. The Universal Agent can monitor and retrieve data from any application that produces data values. Essentially, IBM Tivoli Monitoring can now monitor any unique application regardless of whether the base product supports it.

Common optional management agents that are packaged separately include:

- ▶ Monitoring Agent for IBM Tivoli Monitoring 5.x Endpoint
- ▶ DB2 Agent
- ▶ Oracle Agent
- ▶ MS SQL Agent
- ▶ MS Exchange Agent
- ▶ Active Directory® Agent

Tivoli Data Warehouse Proxy agent

The Warehouse Proxy agent is a unique agent that performs only one task: collecting and consolidating all historical data collections from the individual agents to store in the Tivoli Data Warehouse. If using the Tivoli Data Warehouse, one Warehouse Proxy agent is required for each IBM Tivoli Monitoring installation. It uses ODBC to write the historical data to a supported relational database.

Warehouse Summarization and Pruning Agent

The Summarization and Pruning agent is a unique agent that performs the aggregation and pruning functions for the historical raw data on the Tivoli Data Warehouse. It has advanced configuration options that enable exceptional customization of the historical data storage.

One Summarization and Pruning agent is recommended to manage the historical data in the Tivoli Data Warehouse. Due to the tremendous amounts of data processing necessary, we recommend that the Summarization and Pruning agent always be installed on the same physical system as the Tivoli Data Warehouse repository.

Tivoli Data Warehouse (TDW)

The Tivoli Data Warehouse is the database storage that contains all of the historical data collection. A Warehouse Proxy must be installed to leverage the TDW function within the environment. In large-scale deployments, a Tivoli Data Warehouse can be shared among monitoring installations.

12.3 IBM Tivoli Composite Application Manager for SOA

IBM Tivoli Composite Application Manager (ITCAM) for SOA v6.0 provides the management function for managing services in a SOA environment. It monitors and performs simple control of message traffic between Web services.

ITCAM works with several application server environments including IBM WebSphere Application Server, Microsoft .Net, and BEA WebLogic Server.

ITCAM supports production IT environments, and status and situation generation. This release provides Web services as *first class manageable resources* in the Tivoli Enterprise Portal environment.

ITCAM is installed and operates within the management infrastructure of the Tivoli Monitoring Services platform. The Tivoli Monitoring Services works in conjunction with Tivoli Enterprise Portal to provide graphical views of metrics of Web services calls.

The following list describes the primary components of ITCAM for SOA:

- ▶ A *monitoring agent* that interacts with the managed application servers and infrastructure middleware to collect data. The monitoring agent is installed locally on every application server environment where Web services are to be managed. The data collected by the monitoring agent is stored in a log file. The monitoring agent also stores data collector configuration data in a configuration file that is accessed by the data collectors on each managed system.
- ▶ A set of management data using logical table constructs.
- ▶ A set of queries and commands.

Figure 9-4 shows the user interface for managing the environment. This user interface is presented via the Tivoli Enterprise Portal.

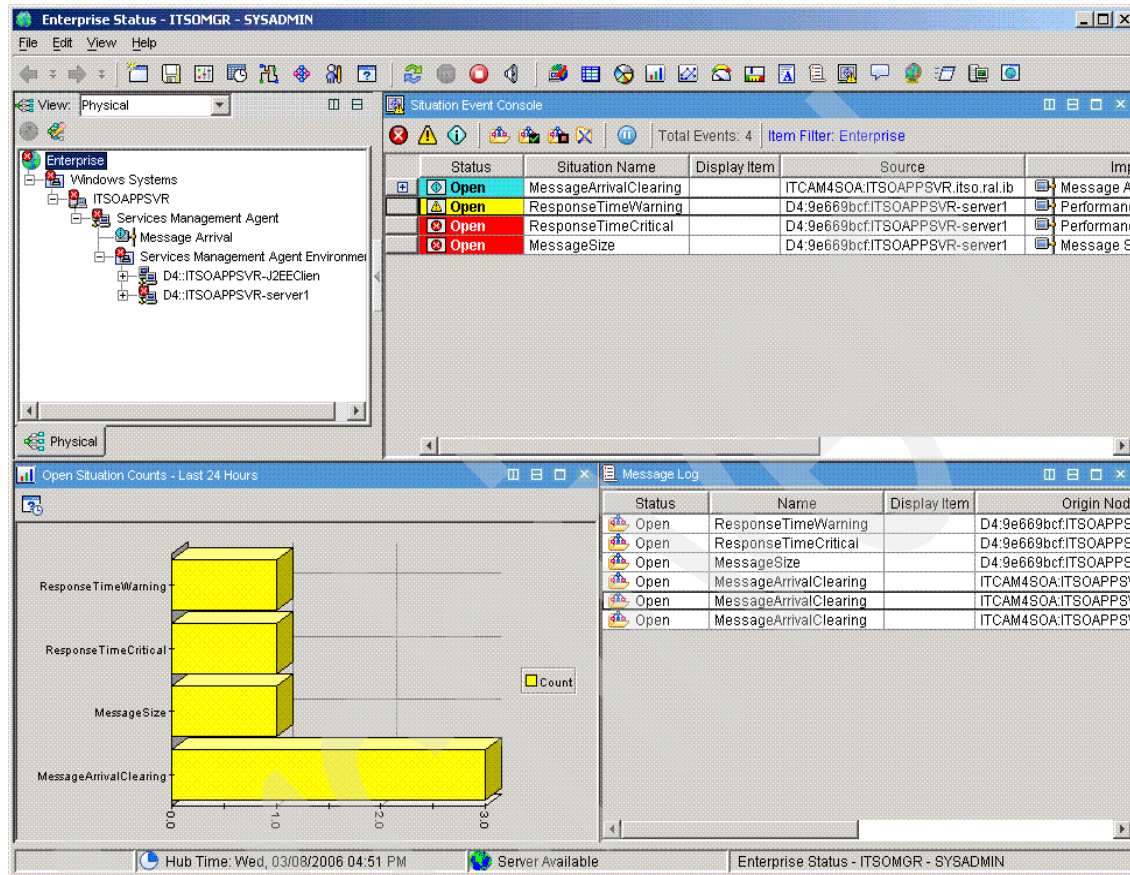


Figure 12-5 Sample ITCAM for SOA workspace displayed via Tivoli Enterprise Portal

12.4 Tracking performance with ITCAM for SOA

First we cover some of the concepts used in the Tivoli monitoring tools.

12.4.1 Workspaces

Workspaces are at the heart of Tivoli Enterprise Portal (TEP) and provide access to the collected data for your monitored Web services. Each workspace serves a unique purpose and displays a specific set of data to monitor or to provide an overview of all resource data (see Figure 12-6 on page 568). Various nodes

relating to the agent and the remote application server are highlighted in the picture. Notice the toolbar at the top, from which various editors can be invoked during customization of the workspace. Also, there are icons for tables and graphs, which can be used to display collected data.

Workspaces are split into multiple views. A view can be a table, graph, notepad, Web browser, 3270 terminal emulator, and so on. It is used to display data in a meaningful way. Each workspace contains the Tivoli Enterprise Portal Navigator view, fixed in the upper left corner of the workspace. The Navigator view provides tabbed support for one physical topology, or navigation tree, and zero or more business views.

Use the Navigator view to navigate through the physical topology of your enterprise environment. The Services Management Agent Environment node represents the standard Tivoli Enterprise Portal monitoring agent function provided by ITCAM for SOA. It is displayed in the Navigator tree under the managed system node. See Figure 12-6.

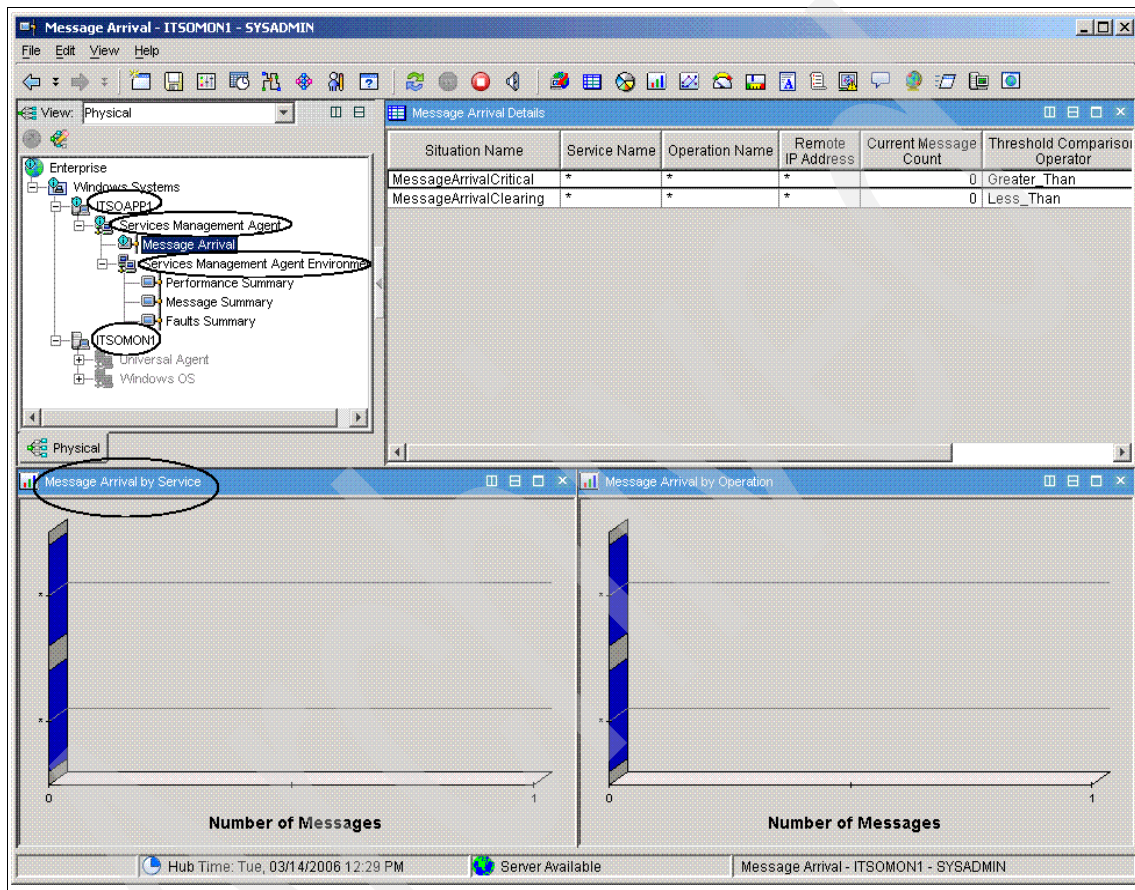


Figure 12-6 Example of Workspace in Tivoli Enterprise Portal

ITCAM for SOA comes with a set of predefined workspaces that you can select from the Navigator, each with its own set of views that display Web services data and metrics in various levels of detail:

- ▶ Message Arrival workspace
- ▶ Performance Summary workspace
- ▶ Message Summary workspace
- ▶ Faults Summary workspace
- ▶ Services Management Agent workspace
- ▶ Services Management Agent Environment workspace

The views included in the workspaces at the Services Management Agent Environment level or below, except the Fault Details view, are based on the Services Inventory table. This table contains summarization calculations for the monitored Web services traffic. The calculations are done on a five-minute interval. For each interval, the traffic is analyzed to calculate the fields available within the table. After the five-minute interval is complete, the record is marked as complete. All queries of the Services Inventory table that are provided with the product include a check for only complete records.

The Services Management Agent workspace

In the Navigator view, when you click the monitoring agent, the Service Management Agent workspace is displayed.

The Service Management Agent workspace displays the current configuration details for the monitoring agent data collectors that are configured in different application server instances. At the highest level this workspace contains the following views:

- ▶ Data Collector Global Configuration
- ▶ Data Collector Monitor Control Configuration
- ▶ Data Collector Filter Control Configuration

The Data Collector Global Configuration view includes the name and environment for the application server where the interception point is running, and flag settings to enable or disable data collection, tracing, and logging. In this view you can turn off monitoring by changing the data collector setting to off.

The Data Collector Monitor Control Configuration view includes information about which services and operations are being monitored for a specific application server in the specified environment. You can configure which services to monitor and for which services message content is to be logged. By default all services are monitored. If you change this setting, only those services and operations that you specify are monitored. The Message Logging Level column indicates the level of logging that is configured for each unique combination of service name and operation name.

Valid message logging levels are defined in Table 12-2.

Table 12-2 Message logging table

Messaging logging level	Comment
None	No information about the message is logged.
Body	Only the body of the message is logged.
Header	Only the header information is logged.
Full	Both body and header messages are logged.

Message Arrival workspace

The Message Arrival workspace provides a summary of the number of messages that arrive at the data collector for each combination of service name, operation name, and remote IP address that has been configured as a situation.

This workspace contains the following views:

- ▶ Message Arrival Details view
- ▶ Message Arrival by Service view
- ▶ Message Arrival by Operation view

Service Management Agent Environment workspace

The Services Management Agent Environment workspace represents the agent monitoring applications for all of the application servers on that system. This workspace provides a set of views that summarize the performance, message activity, and fault occurrences associated with the Web services traffic through this monitoring agent. This workspace contains the following views:

- ▶ Average Response Time by Operation view
- ▶ Number of Messages by Operation view
- ▶ Average Message Size by Operation view

Performance Summary workspace

The Performance Summary workspace provides the inventory of currently active and monitored services, as well as the response time of the services. This workspace contains the following views:

- ▶ Average Response Time by Operation view
- ▶ Services Inventory view

Messages Summary workspace

The Messages Summary workspace provides details about the number and size of messages received for services and service/operation combinations. This workspace contains the following views:

- ▶ Number of Messages by Service - Operation - Type view
- ▶ Average size of Messages by Service - Operation - Type view

Faults Summary workspace

The Faults Summary workspace provides a general faults summary. This workspace contains the following views:

- ▶ Faults Summary by Operation view
- ▶ Fault Details view

Customizing workspaces

Tivoli Enterprise Portal provides a set of predefined workspaces that include queries, thresholds, and views. You can customize these predefined workspaces to suit your particular needs or create additional workspaces. When you create custom views, you select a presentation type such as table, bar chart, pie chart, and so on and associate a query to that view. The query retrieves data collected by the monitoring agents and the view displays that data using parameters you specify.

12.4.2 Attributes

Attributes are measurements that are collected by the IBM Tivoli Monitoring V6.x family of products. ITCAM for SOA stores specific measurements or attributes that are pertinent to its functions. Attributes are grouped into tables. The tables that are available for long-term historical data collection are indicated in the description of the table and show historical reference information that identifies each attribute within each table.

Table 12-3 ITCAM for SOA attribute tables

Attributes table name	Description
KD4DCMT	Data collector monitor control configuration attributes. Associated Take Action commands: AddMntCntrl , De1MntCntrl , UpdMntCntrl
KD4DCT	Data collector global configuration attributes. Associated Take Action commands: EnableDC , DisableDC , updateLogging , updateTracing

Attributes table name	Description
KD4FCT	Data collector filter control configuration attributes. Associated Take Action commands: AddFltCntrl , DelFltCntrl
KD4FLCT	Fault log attributes
KD4INV	Service inventory attributes
KD4MATT	Message arrival threshold attributes
KD4SMT	Service message metric attributes

The columns of these tables correspond to the parameters that are required to be completed during a Take Action command. Most of the parameters are self-documenting. Some may require reference to the documentation for ITCAM for SOA.

Table 12-4 shows some of the parameters you may need.

Table 12-4 Key parameters used in attribute tables and Take Actions commands

Term reported for historic data	Attribute name	Description
APPSRVENV	Application Server Environment	Type of environment in which the data collector is running. Choose one of the following: <ul style="list-style-type: none">▶ WebSphere_Application_Server▶ .NET▶ WebLogic_Server
APPSRVNM	Application Server Name	Name of the application server from which the data is collected. If the Application Server Environment is set to WebSphere_Application_Server then the server name is something like server1 as the default or ITSOCARrentalSvr or ITSOWebSvcCARrentalSvr, if you have defined your own servers.
HOSTNAME	Host name	This is the fully qualified name of the host where the data collector is running. For example, itsoapp1.itso.ral.ibm.com If not sure, use an asterisk (*) (wild card).

For a full list of details on the attributes and associated tables, refer to the documentation for ITCAM for SOA, *IBM Tivoli Composite Application Manager for SOA Installation and User's Guide*, GC32-9492.

12.4.3 Situations

A situation is a type of an event that is generated in Tivoli Monitoring Services. ITCAM provides predefined situations. These situations are defined to help you

monitor critical activity and serve as a template for creating customized situations of your own.

A situation is a condition where a set of attributes are tested against a threshold. The situation evaluates the conditions at predefined intervals and invokes the appropriate automated responses and notification methods.

Predefined situations are activated when they are distributed to the node that you wish to monitor. Once they have been configured, the situation alerts provide ITCAM trigger event notification. You can modify an existing situation to create your own. The default sampling interval for a situation is 15 minutes. If you modify an existing situation to create your own, do not forget to change the sampling time.

The predefined situations provided with ITCAM for SOA are:

- ▶ **Fault:** Monitors the messages in the Web services flow to determine whether a Web service fault has occurred.
- ▶ **Message Arrival Critical:** Raises an alert when an excessive amount of traffic occurs (when the number of messages received from one or more remote clients exceeds a user-defined threshold).
- ▶ **Message Arrival Clearing:** Clears a previously triggered Message Arrival Critical situation. The situation can also be used to alert when message arrival falls below a specified threshold; perhaps this may be used to alert a lack of activity.
- ▶ **Message Size:** Monitors the length in bytes of each message during the Web services flow. If the length of the message is greater than the threshold value, then the situation is triggered.
- ▶ **Response Time Critical:** Monitors in milliseconds the elapsed round-trip response time for the completion of a Web services request.
- ▶ **Response Time Warning:** Monitors in milliseconds the response time for the completion of a Web services request.

The default situations are based on service calls stored in the Service_Metric table. Analyzing summarized information in the Service_Inventory table can reduce some of the monitoring overhead.

12.4.4 Policies

Advanced automation uses policies to perform actions, schedule work, or automate manual tasks. A policy consists of a series of activities. Activities are automated steps that are connected together to create a workflow. Tivoli Enterprise Portal gives you a Workflows window for designing and managing policies.

12.4.5 Take Action commands

Take Action commands are directives that can be run from the Tivoli Enterprise Portal desktop client or included in a situation policy. When included in a situation, the command runs when the situation becomes TRUE. When you enable a Take Action command in a situation, you automate the response to system conditions.

The ITCAM for SOA agent includes the following predefined Take Action commands that change the configuration file for the agent and control the operation of the data collector (Table 12-5).

Table 12-5 Predefined Take Action commands for ITCAM

Take Action command	Description
AddFiltrCntrl	Creates new filter control settings to reject messages
AddMntrCntrl	Creates new monitor control settings
DelFiltrCntrl	Deletes existing filter control settings
DelMntrCntrl	Deletes existing monitor control settings
DisableDC	Disables data collection and the ability to reject messages
EnableDC	Enables data collection and the ability to reject messages
updateLogging	Defines the level of logging information
UpdMntrCntrl	Updates existing message logging levels for monitor control
updateTracing	Enables or disables tracing

Take Action commands with SI- or SM- prefixes

When you are viewing data in the Services Management Agent Environment workspace or in any of its lower-level workspaces, several of these Take Action commands appear in the Action selection list, with the command name prefixed with either SI- (for example, SI-AddMntrCntrl) or SM- (for example, SM-DelFiltrCntrl). These additional versions of these commands perform the same function as the regular commands. The difference between these commands is in how the input fields for each command are pre-filled for you when you select a command from the list.

Figure 12-7 shows the relationship between the Take Action commands and the effect on attribute tables maintained by the agent.

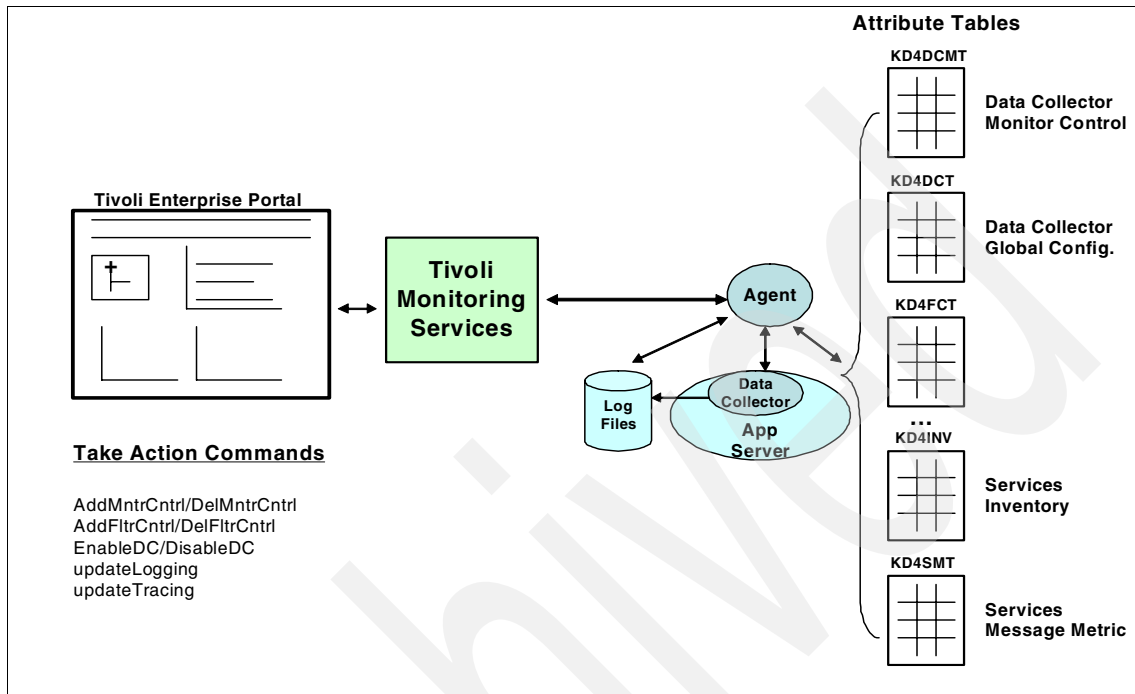


Figure 12-7 Relationship between the agent, data collection tables, and Take Action commands

The agent maintains a set of logical tables whose attributes can be manipulated by the Take Action commands such as AddMntrCntrl. It would be difficult to remember every single attribute that can be manipulated. Fortunately, the Take Action commands issued in the TEP for predefined workspaces come pre-filled with parameters such as the name of the application server. You only have to change the name of a specific operation or the name of the Web service to effect the behavior during data collection.

12.4.6 Log files

Log files are created as a standard action when starting agents and Tivoli Enterprise Monitoring Server. Proper management of log files is important, as their size and the number can grow depending on the managed systems and the level of activities in the systems.

Some examples of log files are:

- ▶ TEMS error log file
- ▶ Logs for Tivoli Enterprise Portal or Tivoli Enterprise Portal Server log
- ▶ Agent log files for ITCAM for SOA, including metric logs, content logs, action logs, operation logs and trace logs

12.5 Monitoring ITSOMart

This section illustrates how you can use the Tivoli Enterprise Portal to monitor Web services traffic. It uses the ITSOMart solution as an example. DB2 is providing the warehouse function.

1. Ensure that the following services are running on the monitoring server node (use the Manage Tivoli Enterprise Monitoring Services utility to start and stop services):
 - Tivoli Enterprise Portal Server
 - Warehouse Proxy
 - Tivoli Enterprise Monitoring Server
2. Ensure that the ITCAM for SOA Monitoring Agent is running on the enterprise service bus node (use the Manage Candle Services utility to start and stop the agent).
3. Launch the Tivoli Enterprise Portal Desktop client as shown in Figure 12-8 by double-clicking it.

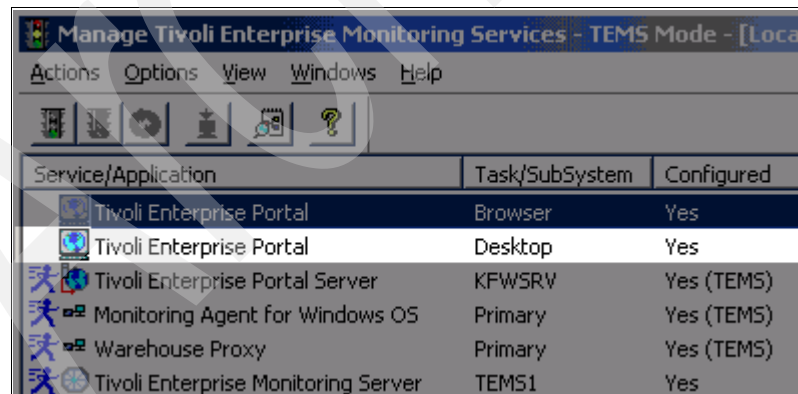


Figure 12-8 Launching the portal desktop client

4. In the portal Logon screen, enter sysadmin as the logon ID and leave the password field blank. See Figure 12-9.

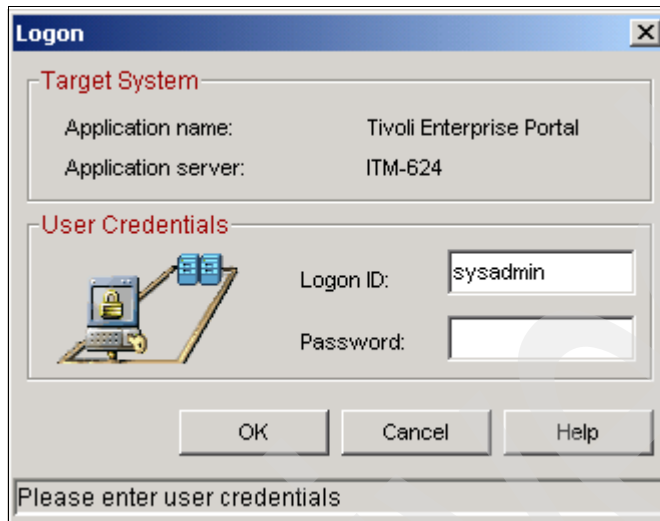


Figure 12-9 Tivoli Enterprise Portal login screen

5. Expand the navigation tree on the top right corner of the screen. Do not worry if you do not see an entry corresponding to the application server, you may need to generate traffic for the enterprise portal to recognize the monitoring agent.

Note: If the monitoring agent is running and you see an entry in the navigator but the Service Management Agent Environment is grayed out, the monitoring agent may not be able to connect to the monitoring server. To look at agent logs, right-click the **ITCAM for SOA** entry in the Manage Candle Services window on the application server node and select **Advanced** → **View Trace Log** from the menu. The trace log is also located in `<tivoli_agent_home>\cma\logs\kd4ras1.log`.

12.5.1 Configure data collection

Next we adjust the settings for data collection. We set the collection interval to 5 minutes. This causes data to be collected every 5 minutes. We also set the warehouse interval to 1 hour. This means that historical data will be sent to the warehouse every hour. If you do not have a warehouse, set this to off. Last, we start the data collection.

To set these intervals:

1. Click the **Configure History Button** in the toolbar.

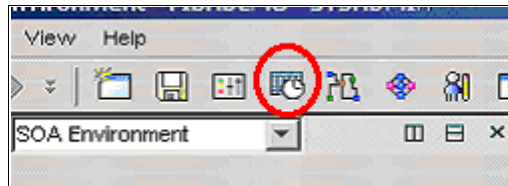


Figure 12-10 Configure History

2. In the dialog, select **ITCAM for SOA** from the drop-down menu at the top.
3. Select **Services_Inventory** from the list of Attribute Groups and change the collection interval to 5 minutes and set the Warehouse Interval to 1 hour.
4. Click **Configure Groups** at the bottom of the dialog.
5. Notice that the collection interval and warehouse interval settings you specified are now in the attributes grid, as shown in Figure 12-11.

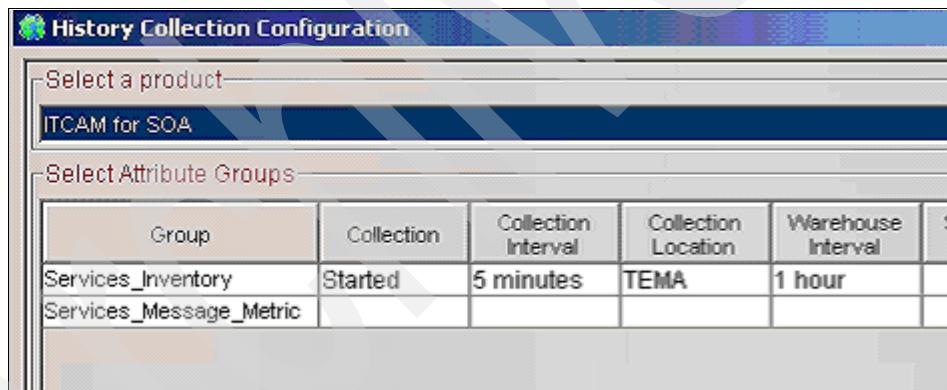


Figure 12-11 Configure ITCAM SOA data collector settings

6. Select the **Services_Inventory** again and click **Start Collection** at the bottom of the dialog. Make sure that the Collection column shows Started next to the Services_Inventory entry.
7. Repeat this process for the Services_Message_Metric.

12.5.2 Generate Web services traffic

Now that the data collection has started, we can start generating traffic to our Web services.

1. Start the server hosting the Web service.
2. Generate Web services traffic by navigating to the customer registration page and submitting a customer registration.

The URL for the ITSOMart customer registration page is
<http://localhost:9080/ITSOMartWeb/faces/CustomRegistration.jsp>.

3. Once you have submitted a customer registration, return to the enterprise portal client. A highlighted green arrow in the top right corner of the screen (Figure 12-12) indicates that the navigator has an update. Click the green arrow to update the navigator.

Note: You may need to wait several minutes for the data collector to post data to the monitoring server.



Figure 12-12 Navigator update available

4. Go to the portal client and expand the Services Management Agent node to see information about the Web services data collection settings.

- Click **Message Arrivals** to see details about incoming Web service requests (Figure 12-13).

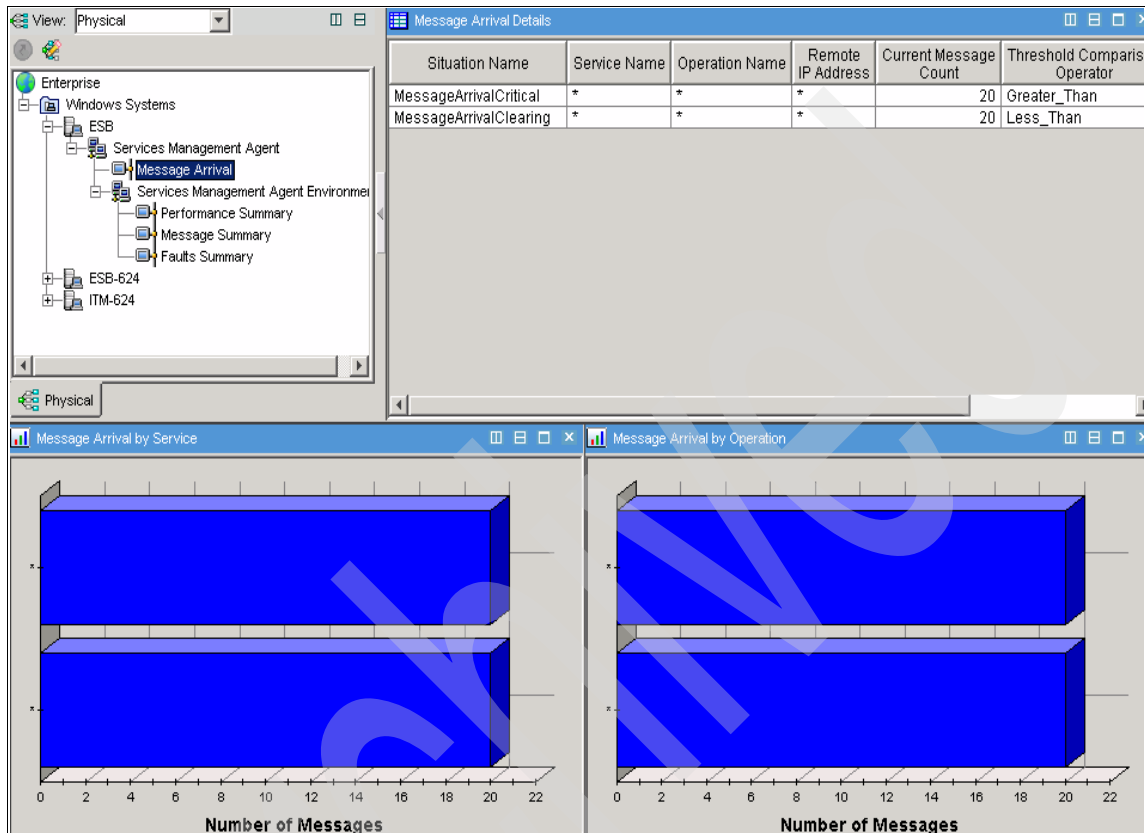


Figure 12-13 Message Arrival Summary view

- Expand the Services Management Agent Environment view to see additional details about the Web services requests, and then browse through each of the other views (Message Summary, Performance Summary, and so on).

12.5.3 Enable Web service data logging

Enabling the data logger for a particular service operation allows a more detailed inspection of service requests, including message headers and payloads. Logged content may also be used within the Web Services Navigator for additional analysis.

- Navigate to the Performance Summary view and select a service from the Services Inventory list at the bottom of the screen.

2. Right-click the item and select **Take Action** → **Select** from the menu.
3. From the Take Action dialog, select **SI-AddMntrlCntl** from the Name drop-down list.
4. You will now see a dialog allowing you to configure the monitor parameters. Notice that many of the parameters are already specified. You need to supply a value for `DataCollectorMessageLoggingLevel`. Valid values include Full, Body, Header, and None. Enter `Full` into this field, as in Figure 12-14.

Name	Value
Services_Inventory.Application_ServerName	server1
Services_Inventory.Application_ServerEnv	WebSphere_Application_Server
Services_Inventory.Local_Hostname	esb.ned.ibm.com
Services_Inventory.Service_Name	esbNode_server1_SOAPHTTPChannel1_InboundPort
Services_Inventory.Operation_Name	getCreditRating
DataCollectorMessageLoggingLevel	Full

Figure 12-14 Edit Monitor Argument Values

5. Click **OK**.
6. Select a destination system from the list (there should only be one system).
7. Click **OK** to enable the trace logger.
Once the command has executed, you will see a status dialog. A return code of 0 indicates success. Other codes indicate errors, and appropriate messages accompany the error codes.
8. Select the **Services Management Agent** in the navigator.
9. The data collector monitor you just enabled should now appear in the Data Collector Monitor Control Configuration at the bottom left handle corner of the screen.

Service Name	Operation Name	Message Logging Level
esbNode_server1_SOAPHTTPChannel1_InboundPort	getCreditRating	Full
*	*	None

Figure 12-15 Monitor control configuration

10. Go back to the ITSOMart application and create another customer registration request.
11. You should now see a file whose name ends with content.log in the `<tivoli_agent_home>/cma/kd4/logs` directory. This file contains the data that was collected. You may open this file using a text editor to view the content. This file may also be imported using Web Services Navigator for further analysis.
12. The content logging also generates an additional log at `<tivoli_agent_home>/cma/logs/kd4inv`. This log contains the agent data log, including Web services statistics and service details.

Note: Be aware that this file grows quickly. Do not leave logging enabled for extended periods of time, as this will degrade performance on the application server node.

Disable Web service data logging

Content logging should only be used briefly to collect data for error diagnosis or traffic pattern analysis. Once you have collected sufficient content, you will need to disable the content logging.

1. Select the **Service Management Agent** in the navigator.
2. Right-click the monitoring entry in the bottom left-hand corner of the screen.
3. Select **Take Action** → **Select** from the menu.
4. Select **DellMntrCtrl** from the Action Name drop-down list.
5. Select the system from the Destination Systems list and click **OK**.
6. Once the command has been executed, you will see a status window. Click **OK**.
7. Refresh the workspace using the refresh icon in the toolbar at the top of the screen. The Data Collector Monitor Control Configuration should no longer contain the monitor control you added earlier.

12.5.4 Content filtering

In the same way that content logging may be enabled, you can also control content filtering. Content allows you to selectively accept and reject Web service requests. This allows you to block denial of service (DOS) attacks against your Web services layer.

1. Navigate to the Performance Summary view and select a service from the Services Inventory list at the bottom of the screen.
2. Right-click the item and select **Take Action** → **Select** from the menu.

3. From the Take Action dialog, select **SI-AddFiltrCntrl** from the Name drop-down list.
4. You will now see a dialog allowing you to configure the filter parameters. Note that many of the parameters are already specified. You need to supply a value for **RemoteIPAddress**. This specifies the remote IP addresses to reject. Enter an asterisk (*) to indicate that all clients should be rejected.
5. Click **OK** and run the command against the agent by selecting the application server from the destination systems.
6. Once you have received confirmation that the command was executed, navigate to the Services Management Agent workspace in the navigator and verify that a new filter has been created in the Data Collector Filter Control Configuration view in the lower right-hand side.
7. If you generate traffic once against using the Web application, you should now see faults in the Faults Summary workspace. The summary should display a SOAP fault indicating that a service request was rejected.

Disable filtering

To disable filtering:

1. Right-click the filter entry in the Service Manage Agent area.
2. Select **Take Action** → **Select** from the menu.
3. Select **DelFiltrCntrl** and select the destination systems and click **OK**.
4. Refresh the workspace and verify that the filter has now been removed.

12.5.5 Using Web Services Navigator to analyze data

There are two ways of importing data into Web Services Navigator for analysis: importing from generated log files and importing from the data warehouse.

Web Services Navigator basics

In order to get started with Web Services Navigator, you must first create a project:

1. Start the Web Services Navigator.
2. Select **File** → **New** → **Project** from the menu.
3. Select **Simple** → **Project** and click **Next**.
4. Give the project a name such as LogProject.
5. Click **Finish**.

The project is created within the default workspace directory. The default workspace is `<NAVIGATOR_HOME>/workspace`, where `<NAVIGATOR_HOME>` is the directory in which the Web Services Navigator is

installed. On Windows, the default directory is C:\Program Files\IBM\ITCAM for SOA 6.0.0\IBM Web Services Navigator.

Once you have created a project, you can move on to populating the project with log files captured from the monitoring agent or import data from the data warehouse.

Import from log files

In order to import data from log files, you must enable content capture in the monitor control configuration. Once you have captured log files for your run, you need to assemble them into a single log file for use by Web Services Navigator using a tool called the Log Assembler. Finally, you can copy these files into your project directory and open them using the Web Services Navigator.

The steps to do this are:

1. Gather the log files for analysis. The log files are located on the application server node in the `<tivoli_agent_home>/cma/kd4/logs` directory and have file names ending with `.log`. In addition, there are archived Web services metric log files that are stored in the `KD4.DCA.CACHE/archive` directory within the logs directory. These files have filenames ending with `metric.log.<timestamp>`. Copy all of the log files to the server where Web Services Navigator is installed.
2. You must now run the Log Assembler tool. The Log Assembler tool is a Java program that combines all of the data in the log files into a single log file for use within Web Services Navigator. The Log Assembler tool is located in `<NAVIGATOR_HOME>/IWSNavigator/samples/runLogAssembler.bat` on Windows. The usage for this tool is:

```
runLogAssembler.bat <logdir> <logfile>
```

Where `<logdir>` is the directory where you have copied all of the log files from the application server node and `<logfile>` is the name of the combined log file for use within Web Services Navigator.

3. Once you have generated a combined log file, copy this file to the project directory for the project you created earlier.
4. Right-click your project in the Navigator view of the Web Services Navigator and select **Refresh** from the menu. You should now see the combined log file you created.
5. Double-click this file to begin analysis.

Import from a data warehouse

In order to import data from a data warehouse, you must set up a database connected to the Tivoli Data Warehouse. This can be done during the import.

1. Create a project using the instructions provided.
2. Right-click the Navigator view and select **Import** from the menu.
3. Select **ITCAM for SOA** as the import source and click **Next**.
4. Select your project (or a folder within your project) as the parent folder and enter a file name. This is the name of the file that will be created for you as part of the data warehouse import. Click **Next**.

Note: The file name of the file you create must end with .log.

5. Select a date range for the data import. Only metrics gathered during that date range will be imported. Click **Next**.
6. You are now asked to create or use a connection to the data warehouse database. If you have an existing connection already set up, you may select **Use existing connection** at the bottom of the screen and select that connection. Otherwise, provide the connection information for the data warehouse data. In this case, you must enter values for the fields shown in Table 12-6.

Table 12-6 Data warehouse connection information

Field	Value
Database	WAREHOUS
User ID	itmuser
Password	The password you selected when creating the data warehouse data source
Database Vendor Type	DB2 Universal Database™ 8.2
JDBC Driver	DB2 Universal Driver
Host	localhost

7. Click **Finish** to import the data.

Once the data warehouse import is complete, you will be able to open the log file for analysis.

Note: If you select a date range that contains no metric information, you will receive a warning to that effect. In most cases, you must generate some Web service traffic and wait at least one hour (based on data collector history configuration) before data is moved to the data warehouse. If you need to analyze traffic for a specific use case, it may be easier to enable content logging and use the log assembler to capture the Web services metric data.

Analysis

Once you have imported data into the Web Services Navigator, you can begin to use the analysis tools to better understand the Web services in your system. There are three primary views of the data: Service Topology, Transaction Flows, and Flow Patterns.

The Service Topology view provides a call graph that displays the sequence of service calls throughout the application. In the ITSOMart application, all calls originate from the enterprise service bus (where the Web application is running) and flow to the various services through the Register Customer process (which orchestrates other services).

The Transaction Flow view provides a view of the service's transactions as they flow through the system and displays usage statistics and timing information. This allows you to observe individual transactions as the application information moves from one service to another. Using this view, you can observe the amount of time that is spent on each service relative to the length of the overall transaction. This can help identify problematic performance areas.

The Flow Patterns view helps you identify service call patterns and understand the relationships between services (for example, if the Credit Rating mediation always calls the Credit Rating Service). Thus, that particular service call flow represents a call pattern. More complex patterns exist in the application, for example, if a user enters the Register Customer process but receives a bronze credit rating, that user will always be directed to the credit denied flow. By understanding the usage patterns of the services in your application, you can better organize service calls to provide the most optimal environment to suit your application usage patterns.

12.6 Summary

Modern composite applications require end-to-end monitoring tools in order to adequately meet and maintain system performance. The IBM Tivoli Composite Application Manager helps system administrators meet the challenges of monitoring large multi-system application environments. In the case of

WebSphere ESB, it is especially critical to be able to monitor the Web services traffic flowing through the bus. IBM Tivoli Composite Application Manager for SOA provides the Web services monitoring and management capabilities needed to properly maintain the enterprise service bus.

In addition to online management and monitoring, IBM Tivoli Composite Application Manager also includes Web Services Navigator, an offline Web services analysis tool, to help understand the interactions between various Web services in a composite application.

12.7 For more information

See the following IBM Redbooks for more information:

- ▶ *Getting Started with IBM Tivoli Monitoring 6.1 on Distributed Environments*, SG24-7143
- ▶ *IBM Tivoli Composite Application Manager V6.0 Family: Installation, Configuration, and Basic Usage*, SG24-7151

For information about the IBM Tivoli Composite Application Manager products, see:

- ▶ IBM Tivoli Composite Application Manager Basic for WebSphere home page
<http://www-306.ibm.com/software/tivoli/products/composite-application-mgr-basic-websphere/>
- ▶ IBM Tivoli Composite Application Manager for Response Time Tracking home page
<http://www-306.ibm.com/software/tivoli/products/composite-application-mgr-rtt/>
- ▶ IBM Tivoli Composite Application Manager for SOA home page
<http://www-306.ibm.com/software/tivoli/products/composite-application-mgr-soa/>
- ▶ IBM Tivoli Composite Application Manager for WebSphere home page
<http://www-306.ibm.com/software/tivoli/products/composite-application-mgr-websphere/>

Sample application install summary

The ITSOMart sample is shipped as a set of project interchange files suitable for import into WebSphere Integration Developer V6.0.1. You can download the zip file containing these files from:

<ftp://www.redbooks.ibm.com/redbooks/SG247228/>

This appendix tells you how to import the project interchange files and set up the runtime environment for the ITSOMart solution contained in ITSOMartPIFs.zip.

Overview

The ITSOMart solution includes a front-end Web application, an application that manages the customer registration process, and several mediation modules and services. Figure A-1 shows the high-level overview of the ITSOMart solution.

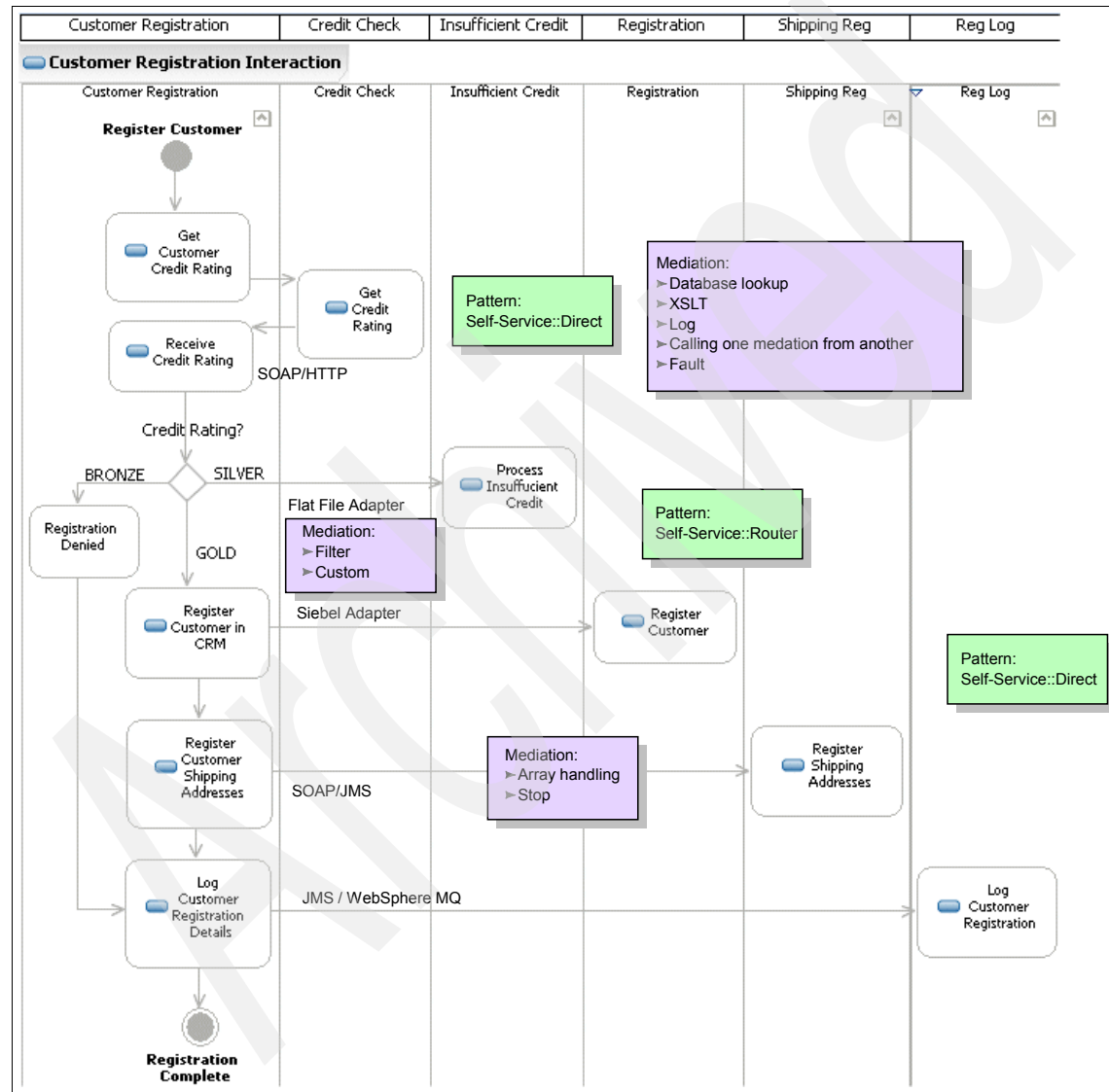


Figure A-1 ITSOMart solution overview

Prepare the development environment

To import the project interchange files and test them on a WebSphere ESB server, you should upgrade your environment to the latest levels. The ITSOMart solution has been tested at the following level: WebSphere Integration Developer 6.0.1, plus the following interim fixes:

- ▶ 6.0.1 interim fix 001
- ▶ 6.0.1 interim fix 002
- ▶ 6.0.1 interim fix 003

Configure the workbench

Certain settings are recommended in the WebSphere Integration Developer workbench when working with mediation modules. You can see these recommendations in 6.3, “Development environment settings” on page 224.

Import the projects into the workbench

In order to prepare the application for runtime, you must have access to the application projects inside WebSphere Integration Developer.

1. Extract ITSOMartPIFs.zip, ITSOMart.sql, and ITSOMART_Cloudscape.zip from SG247228.zip into a temporary directory.
2. Open WebSphere Integration Developer and create a new workspace.
3. Update the workspace preference settings as outlined in 6.3, “Development environment settings” on page 224.
4. Import the project interchange files included by doing the following:
 - a. Select **File** → **Import**.
 - b. Select **Project Interchange** and click **Next**.

- c. Browse to ITSOMartPIFs.zip and select all the project interchange files, or if you only want to see the files associated with one scenario, select the files you need from Table A-1.
 - d. Click **Finish**.
5. Perform a clean build of all the projects. **Project** → **Clean All**.

Table A-1 Project interchange files

	Project interchange files
Mediation utilities	
Common files needed for most or all scenarios	MessageLogApp
	MessageLogWeb
	ITSOMartLib
	ITSOMartUtils
Front-end applications	
ITSOMart Web interface application	ITSOMartApp
	ITSOMartWeb
	ITSOMart_Proxies
Registration processor application	ITSO_RegProcServiceApp
	ITSO_RegProcServiceEJB
	ITSO_RegProcService_Proxies
Mediations	
Credit Rating mediation (Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263) ^{1,3}	CRMed ^{1,3}
Credit Score mediation (Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263) ^{1,2,3}	CSMed ^{1,2,3}
	CRMedV2 ^{1,3}
CRM mediation (Chapter 8, “Building the CRM mediation” on page 347) ¹	ITSO_CRMMed ¹
¹ Requires ITSOMartLib ² Requires ITSOMartUtils ³ The module name has been shortened from the sample instructions to bypass the Windows limit on path names longer than 256 characters.	

	Project interchange files
Register Shipping mediation (Chapter 10, “Building Log Registration mediation” on page 449)	RSMed ^{1,3}
	SRSvc ^{1,3}
Log Registration mediation (Chapter 10, “Building Log Registration mediation” on page 449)	ITSO_RegLogMed ¹
	websphere_default_messaging_provider
Imported Services	
Credit Rating mediation (Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263) ^{1,3} (Referred to as ITSO_CreditRatingService)	CRsvcApp ³
	CRsvc ³
Credit Score mediation (Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263) ^{1,3} (Referred to as ITSO_CreditScoreService)	CSSvcApp ³
	CSSvc ³
¹ Requires ITSOMartLib ² Requires ITSOMartUtils ³ The module name has been shortened from the sample instructions to bypass the Windows limit on path names longer than 256 characters.	

Prepare the runtime environment

The sample application must be deployed to a WebSphere ESB server or to a WebSphere ESB test environment on WebSphere Integration Developer. In both instances, the runtime environment is configured using the WebSphere administrative console. The difference is in the method of deployment and in the location of the runtime files for WebSphere ESB.

We recommend that you create a new WebSphere ESB server for your test environment in order to isolate the ITSOMart runtime from any other applications you may have in test. You can do this using the instructions in “Creating a new server in the test environment” on page 640.

For the sample to run, we assume the following:

- ▶ You have a running standalone application server named `server1` using port 9080 for access to the Web container.
- ▶ You are using a Windows system. If you are not using Windows, most of these instructions will still work as documented, but you may need to refer to the WebSphere ESB product documentation for platform-specific information such as command locations or extensions.
- ▶ The installation directory for WebSphere Integration Developer will be denoted with `<wid_install>`.
- ▶ WebSphere ESB file locations will be denoted with `<wesk_install>`. For example, if you have WebSphere ESB installed, `<wesk_install>` will look similar to `C:\WebSphere\AppServer`. In the WebSphere Integration Developer test environment, this will be `<wid_install>\runtimes\bi_v6`.

Installing applications: As you go through the steps for setting up and deploying the ITSOMart sample, you will be asked to install applications to the server.

To install an application to the WebSphere Integration Developer unit test environment, perform the following steps.

1. Select the server in the Servers view, right-click, and select **Add and remove projects**.
2. Select the application in the list of available projects and click **Add >**.
3. Click **Finish**.

To install an application to WebSphere ESB:

1. Export the mediation module as an *integration module* from WebSphere Integration Developer to the `<wesk_install>installableApps` directory. The integration module will be packaged as an EAR file.
2. In the administrative console, select **Applications** → **Enterprise Applications**.
3. Click **Install** and follow the wizard.

Create a service integration bus

The application sample uses a service integration bus as the default JMS provider. You need to create the bus and add the application server where the application will run as a member of the bus.

1. Start the server. In the test environment, you can start the server in the Servers view. Right-click **WebSphere ESB Server v6.0** and select **Start**.

Switch to the Console view to view messages. When you see Server server1 open for e-business, the server is started.

2. Log in to the administrative console. You can do this from the Servers view. Right-click **WebSphere ESB Server v6.0** and select **Run administrative console**. When the console login screen appears, click **Log in** (no name is required).
3. Create a bus called ITSOMartBus and add server1 as a member of the bus using the instructions found in 11.5, “Creating a service integration bus” on page 490.

If you prefer, you can skip this step and use the SCA.APPLICATION.<cell>Bus bus instead. Just be sure to substitute this bus wherever you see references to ITSOMartBus later.
4. Save the configuration.

MessageLogApp application

The MessageLogApp application is not a mediation application, but rather a simple program to allow you to easily check the contents of the service integration bus queues and the messages logged by Message Logger primitives. We used it to simplify checking the results of our testing.

This application contains a JavaServer Faces (JSF) Web application that provides the following:

- ▶ **ESB Mediation Message Log viewer**

This is a simple interface for querying the ESB Mediation Message Log database for messages that have been logged by Message Logger mediation primitives in mediation flows.

This viewer can be accessed with the following URL:

<http://localhost:9080/MessageLogWeb/faces/MessageLogView.jsp>

- ▶ **Queue browser**

This provides a basic interface for browsing messages on JMS queues.

The queue browser can be accessed with the following URL:

<http://localhost:9080/MessageLogWeb/faces/QueueBrowser.jsp>

Install MessageLogApp

No runtime resources are required for this application. Simply install the MessageLogApp application to the server.

Registration processor service

The mediations described in this book can each be developed, deployed, and tested individually. However, to build a meaningful end-to-end solution, we need an application that drives the registration process. This application is called ITSOMartApp.

Runtime

To prepare the runtime you must configure the following on the application server:

- ▶ A queue named `ITSOMart.RegistrationProcessorServiceQ` defined to the `ITSOMartBus` service integration bus. See 11.8.1, “Create a queue destination on the bus” on page 502.
- ▶ A JMS queue connection factory:
 - Name: `ITSOMart.RegistrationProcessorServiceQCF`
 - JNDI Name: `jms/RegistrationProcessorServiceQCF`
 - Bus name: `ITSOMartBus`

See 11.8.2, “Create a queue connection factory” on page 503.

- ▶ Another JMS queue connection factory:
 - Name: `ITSOMart.RegistrationProcessorServiceReplyQCF`
 - JNDI Name: `jms/RegistrationProcessorServiceReplyQCF`
 - Bus name: `ITSOMartBus`

See 11.8.2, “Create a queue connection factory” on page 503.

- ▶ A JMS queue:
 - Name: `ITSOMart.RegistrationProcessorServiceQ`
 - JNDI Name: `jms/RegistrationProcessorServiceQ`
 - Bus name: `ITSOMartBus`.
 - Queue name: `ITSOMart.RegistrationProcessorServiceQ`

See “Create a JMS queue” on page 504.

- ▶ A JMS activation specification:
 - Name: `ITSOMart.RegistrationProcessorServiceActivationSpec`
 - JNDI name: `jms/RegistrationProcessorServiceActivationSpec`
 - Destination type: `Queue`.
 - Destination JNDI name: `jms/RegistrationProcessorServiceQ`
 - Bus name: `ITSOMartBus`

See 11.8.4, “Creating a JMS activation specification” on page 505.

Save the changes to the configuration.

Install the application

Install the ITSO_RegProcServiceApp application. All of the mediation modules provided for the sample must be installed and configured for this Web service application to work.

ITSOMart application

Install the application ITSOMartApp. This application contains the Web front-end for driving the ITSOMart sample scenario. The Registration Processor Service application and all of the mediation module applications must be installed for this application to work.

The application can be accessed from a Web browser using the following URL:

<http://localhost:9080/ITSOMartWeb/faces/CustomerRegistration.jsp>

Credit check mediations

Chapter 7, “Building the Credit Rating and Credit Score mediations” on page 263, contains two scenarios:

- ▶ Get Credit Rating scenario
- ▶ Get Credit Score scenario

The first is a simple call to a Web service. The second expands on the first to perform more complex functions, including a database lookup.

The Get Credit Rating scenario does not require any special runtime setup. The Get Credit Score scenario, however, requires a database and the supporting JDBC provider and data source resources defined to the application server.

The ITSOMart sample, as shipped, is set up to use the Get Credit Score scenario.

Create the database and configure the JDBC data source

The Credit Score mediation uses a database named ITSOMART. This database holds the ITSOMart credit rating values (bronze, silver, gold) that match numeric credit scores returned by the Credit Score Service. Instructions are provided here for configuring the ITSOMART database in DB2 and Cloudscape. You will only need to use one.

Create the DB2 database

Use these steps to create the database:

1. Copy the ITSOMART.sql file to a temporary location, for example, C:\temp.
2. Open a DB2 command window and enter the information in Example A-1.

Example: A-1 Creating the database

```
cd C:\temp

db2cmd

db2 create database itsomart
db2 connect to itsomart
db2 -tvf ITSOMART.sql
db2 connect reset
```

Create the Cloudscape database

Using a Cloudscape database eliminates the need for access to a DB2 server. The sample includes a ready-to-use Cloudscape database. Cloudscape databases can be viewed and manipulated using the cview.bat tool in the `<wesh_install>\cloudscape\bin\embedded` directory.

The Cloudscape database is contained in ITSOMART_Cloudscape.zip. Unzip this file into the <wesp_install>\cloudscape\databases directory. Figure A-2 shows the directory structure when unzipped into the directory structure for the WebSphere Integration Developer test environment.

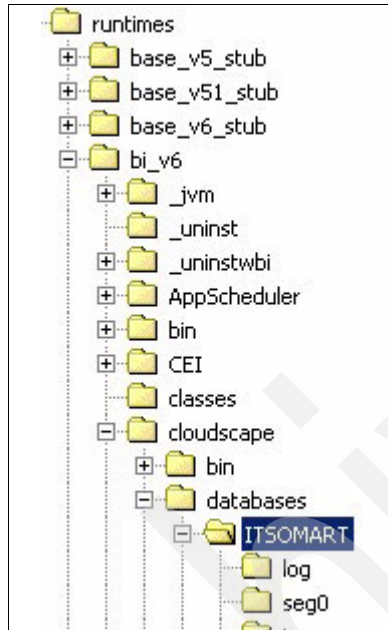


Figure A-2 Cloudscape database

Create the J2C authentication entry (for DB2)

If you choose to use DB2, you will need a J2C authentication entry containing the user ID and password required to authenticate with DB2. This user ID must have write access to the database.

Use the instructions in 11.6.1, “Create a J2C authentication data entry for the database” on page 491, to create an entry.

Create the JDBC provider

If you chose to use DB2, use the instructions in 11.6.2, “Create a JDBC provider” on page 492, to create a JDBC provider for DB2. Use the defaults and the following properties:

- ▶ Database type: DB2
- ▶ Provider type: DB2 Universal JDBC Driver Provider
- ▶ Implementation type: Connection pool data source

Be sure to define the DB2_JDBC _DRIVER_PATH WebSphere variable to the proper location for your system.

Create the data source

Use the instructions in 11.6.3, “Create a data source” on page 494, to create a data source for the database. You will need to do this regardless of the database type.

Note that each data source requires a unique JNDI name. You will only be able to create one or the other of these (DB2 or Cloudscape) unless you use unique JNDI names. If you choose to do this, be sure to change the JNDI name in the Database Lookup primitive.

Cloudscape

Using the predefined JDBC Provider *Cloudscape JDBC Provider* at the server scope, define the Cloudscape ITSOMart database. Use the defaults and the following properties:

- ▶ Name: ITSOMartDataSource
- ▶ JNDI name: jdbc/ITSOMartDataSource
- ▶ Database name: databases\ITSOMART

DB2

For DB2, use the defaults and the following properties:

- ▶ Name: ITSOMart DataSource
- ▶ JNDI name: jdbc/ITSOMartDataSource
- ▶ Check **Use this Data Source in container managed persistence**.
- ▶ Component-managed authentication alias: <node_name>/DB2 ITSOMART USER
- ▶ Database name: ITSOMART
- ▶ Driver type: 4
- ▶ Server name: <your_server>

Install the applications

The Get Credit Score scenario requires the following to be installed on the server:

- ▶ CSMedApp
This application contains the Credit Score mediation module.
- ▶ CRMedV2App
This application contains version 2 of the Credit Rating mediation module.

- ▶ CSSvcApp

This application contains the Credit Score Service, a soap/http Web service. The Credit Score mediation mediates calls to this Web service.

CRM mediation

Install the mediation module application ITSO_CRMedApp. This application contains the CRM Mediation module.

Note that the sample is not shipped with the WebSphere Adapters and the bindings for the exports have been removed. The processor code that calls these services has been commented out.

Register Shipping mediation

Install the following applications:

- ▶ RSMedApp

This application contains the Register Shipping mediation module.

- ▶ SRSvcApp

This application contains the Register Shipping Service.

Registration Log Mediation

The Registration Log Mediation Module requires three queue destinations to which it will send registration success, denied, and failure messages. You must configure the supporting queues and JMS resources on the server.

Create the bus destinations

Create three queue destinations on the ITSOMartBus with the following identifiers:

- ▶ ITSOMart.LogSuccessQ
- ▶ ITSOMart.LogDeniedQ
- ▶ ITSOMart.LogFailureQ

Save the configuration.

For information about how to create a bus destination, see 11.8.1, “Create a queue destination on the bus” on page 502.

Create the JMS queue connection factory

Create a JMS queue connection factory under the Default Messaging JMS provider with the following values:

- ▶ Name: ITSOMart.LogQCF
- ▶ JNDI Name: jms/ITSOMart/LogQCF
- ▶ Bus name: ITSOMartBus

For information about how to create a JMS queue connection factory, see 11.8.2, “Create a queue connection factory” on page 503.

Create the JMS queues

Create three JMS queues under the Default Messaging JMS provider with the following values (Table A-2).

Table A-2 Registration Log Mediation JMS queues

JMS queue name	JNDI name	Bus name	Queue name
ITSOMart.LogSuccessQ	jms/ITSOMart/LogSuccessQ	ITSOMartBus	ITSOMart.LogSuccessQ
ITSOMart.LogDeniedQ	jms/ITSOMart/LogDeniedQ	ITSOMartBus	ITSOMart.LogDeniedQ
ITSOMart.LogFailureQ	jms/ITSOMart/LogFailureQ	ITSOMartBus	ITSOMart.LogFailureQ

Save the configuration changes.

For information about how to create a JMS queue, see 11.8.3, “Create a JMS queue” on page 504.

Install the application

Install ITSO_RegLogMedApp.

Common errors:

The following is a common error and how to correct it:

► Error

```
[5/5/05 8:45:59:188 EDT] 00000037 StaleConnecti A CONM7007I: Mapping the  
following SQLException, with ErrorCode -30082 and SQLState 08001, to a  
StaleConnectionException: java.sql.SQLException: SQL30082N Attempt to  
establish connection failed with security reason "24" ("USERNAME AND/OR  
PASSWORD INVALID"). SQLSTATE=08001  
DSRA0010E: SQL State = 08001, Error Code = -30,082
```

► Tip

Check the user name and password in the authentication alias defined for the data source

Tips and techniques

This appendix contains helpful tips for working with mediations, Web services, and the runtime environment. These are issues that we came across during the process of creating the ITSOMart sample and think that they will be of use to others. This appendix contains the following:

- ▶ Creating a top-down SOAP/JMS Web service
- ▶ Server errors in the test environment
- ▶ Errors using XML Mapper without Internet connectivity
- ▶ Creating a new server in the test environment
- ▶ Installing WebSphere MQ Explorer as a plug-in

Creating a top-down SOAP/JMS Web service

This section provides instructions for taking a WSDL created in the Business Integration perspective using the interface editor and generating a top-down SOAP/JMS Web service from it.

The service you will create here is the Register Customer process used in the sample application provided with this book. This Web service has a one-way operation that is invoked asynchronously using SOAP/JMS and is passed a Customer business object. This operation provides the implementation for processing customer registration requests.

Important: Make sure you have configured your workspace with the settings outlined in 6.3.3, “Configure Web services workspace preferences” on page 226.

Create the business object

This service uses one business object called RegistrationProcessorRequest. This object, in turn, uses the Customer business object built in “Create the Customer business object” on page 273 and stored in the ITSOMartLib library.

This process assumes that you have a library called ITSOMartLib, containing the Customer business object, in your workspace.

To build the new business object:

1. In your WebSphere Integration Developer workspace, open the Business Integration perspective.
2. Create a business object named RegistrationProcessorRequest in the ITSOMartLib library.
3. Add one attribute called customer with a type of Customer to the RegistrationProcessorRequest business object.
4. In the properties for this business object, select the **Description** tab and give this business object a unique namespace. We used the name `http://ITSOMartLib/RegistrationProcessorRequest`.

This step is important for uniquely identifying your business objects. Remember that the underlying representations of these business objects are in the XML schema definition, and this is the namespace that will be used in that schema when these business object types are imported into WSDL interface definitions.

The completed business object should look like Figure B-1.

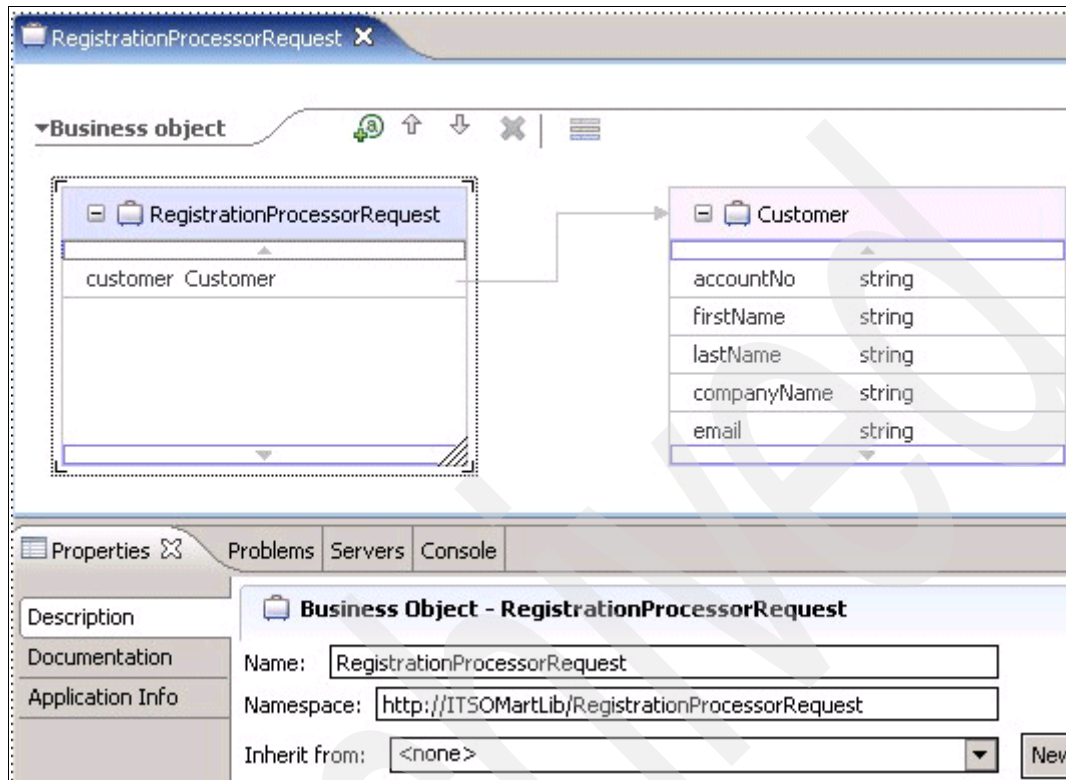


Figure B-1 RegistrationProcessorRequest business object

Build the interface

Next we need an interface for the service:

1. Create an interface named `RegistrationProcessor` in the `ITSOMartLib` library.
2. Define a one-way operation named `registerCustomer`.

3. Add an input parameter named `registrationRequest` to the `registerCustomer` operation and set its type to `RegistrationProcessorRequest`. The `RegistrationProcessor` interface should look like Figure B-2.

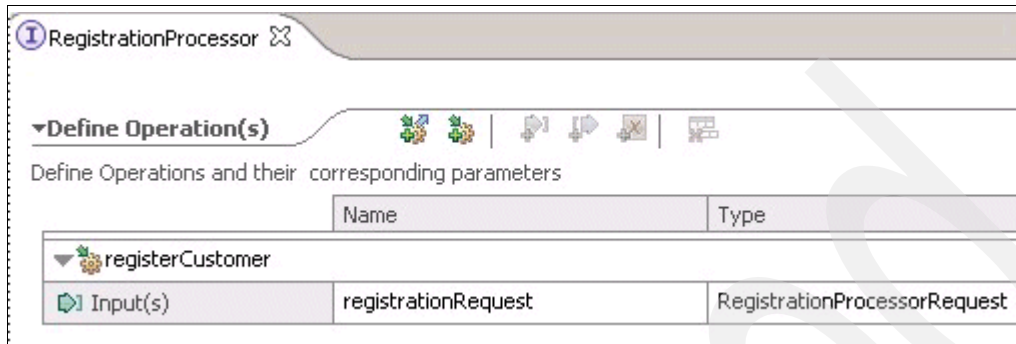


Figure B-2 *RegistrationProcessor interface*

Create an EJB project

When you create a SOAP/JMS Web service, a message-driven bean is created that will act as the listener for SOAP/JMS messages. For this reason, when creating a top-down SOAP/JMS Web service using WebSphere Integration Developer, you have to create an EJB project into which your service will be generated.

1. Open the J2EE perspective by selecting **Window** → **Open Perspective** → **Other** → **J2EE**.
2. In the Project Explorer, right-click **EJB Projects** and select **New** → **EJB Project**

If you do not have the Enterprise Java capability enabled for your workspace, you will be prompted with the message shown in Figure B-3. Select **Always enable capabilities and do not ask me again** and click **OK**.

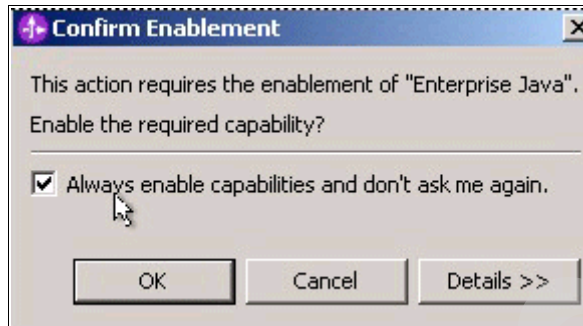


Figure B-3 Confirm Enablement

3. In the New EJB Project wizard do the following:
 - a. Enter a name for the EJB project. We used the name ITSO_RegProcService for our sample application.
 - b. Click **Show Advanced**.
 - c. Select a Target Server of **WebSphere ESB Server v6.0**.
 - d. Enter a name for the EAR project this EJB project will be a part of. We used the name ITSO_RegProcServiceApp.
 - e. De-select the check box **Create an EJB Client JAR Project to hold the client interfaces and classes**.

- f. Select the check box **Create a default stateless session bean** (Figure B-4).

You will not be using this default stateless session bean for anything, but you will get errors when using the Web Service Wizard to generate your Web service if you do not have at least one EJB defined in your EJB project. One of the first things the Web Service Wizard will do is deploy this EJB project to the test environment server, and if you do not have at least one EJB in the project, an error will occur deploying the application and the Web Service Wizard will fail.

Name: ITSO_RegProcService

Project location: C:\projects\esb-redbook\workspace-staging\ITSO_RegProcServ Browse...

Hide Advanced <<

EJB version: 2.1

Target server: WebSphere ESB Server v6.0 New...

☒ Add module to an EAR project.

EAR project: ITSO_RegProcServiceApp New...

☐ Create an EJB Client JAR Project to hold the client interfaces and classes.

☐ Add support for annotated Java classes

☒ Create a default stateless session bean

Figure B-4 New EJB Project

- g. Click **Finish** to create the EJB Project.
4. Close and do not save the default.dnx file that is opened in the editor after the project is created. This is the UML class diagram for the EJB project and we will not be using it here.

Modify the WSDL file

The WSDL file that was created for the interface you defined in the Business Integration perspective needs to be modified before you can successfully generate a top-down Web service from it.

In the following steps you will copy the WSDL and XSD files from the ITSOMartLib library project into your EJB project. You will then modify the WSDL to define a binding, port, and service definition that will be used for generating the top-down Web service implementation.

1. Create a new folder under your EJB project to copy the WSDL and XSD files into.
 - a. In the Project Explorer, right-click **EJB Projects** → **ITSO_RegProcService** and select **New** → **Source Folder**.
 - b. Enter a folder name of **wsdl**.

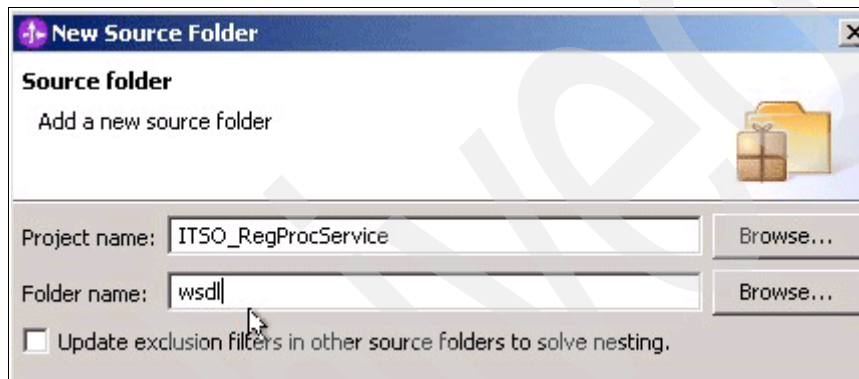


Figure B-5 New source folder

- c. Click **Finish** to create the new folder.
2. Copy the WSDL and XSD files from ITSOMartLib library into the new **wsdl** folder you created in the EJB project.
 - a. In the Project Explorer, navigate to **Other Projects** → **ITSOMartLib**.

b. Copy the following files into EJB Projects/ITSO_RegProcServices/ wsdl:

- Address.xsd
- BillingAddress.xsd
- Customer.xsd
- RegistrationProcessor.wsdl
- RegistrationProcessorRequest.xsd
- ShippingAddress.xsd

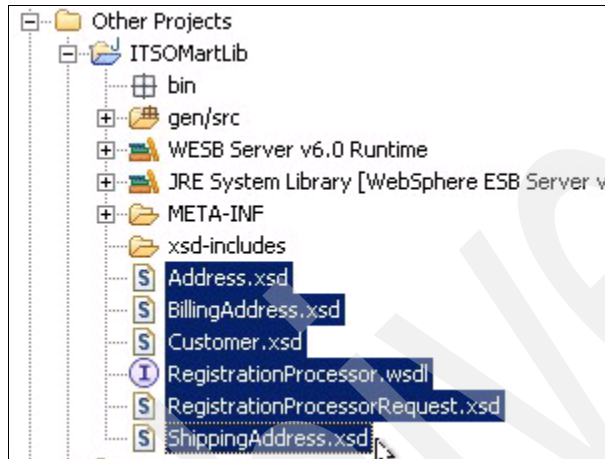


Figure B-6 ITSOMartLib WSDL and XSD files

3. Under the wsdl folder in your EJB project, right-click **RegistrationProcessor.wsdl** and select **Open with → WSDL Editor**.

4. Under Bindings, right-click and select **Add Binding**.

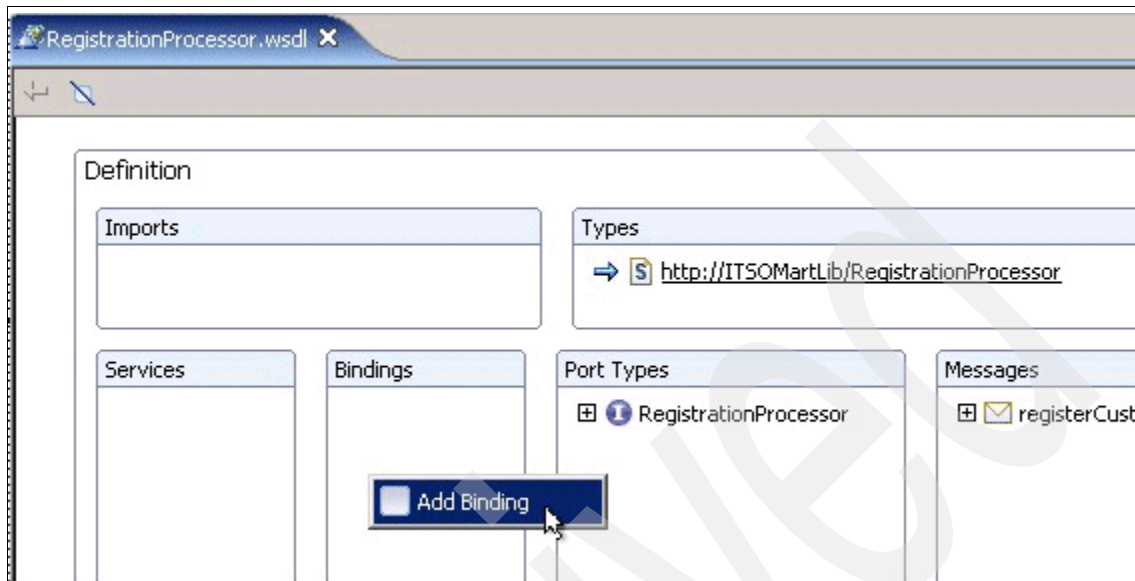


Figure B-7 Add binding

5. In the Binding Wizard:
 - a. Specify a binding name, such as RegistrationProcessorBinding.
 - b. Select the existing Port Type: **tns:RegistrationProcessor**.
 - c. For the Protocol, select **SOAP**.
 - d. Under SOAP Bindings Options, select **document literal**.

e. Click **Finish** to create the binding.

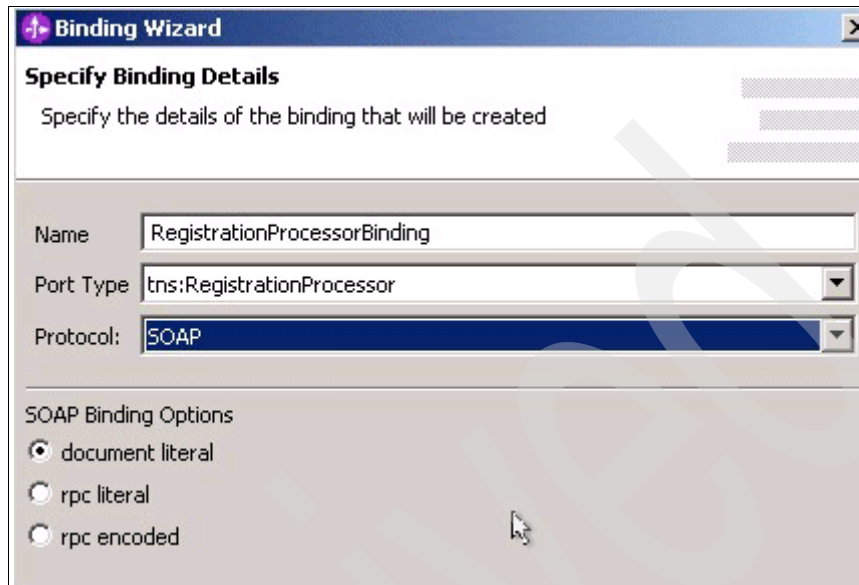


Figure B-8 Binding Wizard

6. Save the RegistrationProcessor.wsdl with the new binding.
7. In the Problems view, you will see two validation errors for the new binding definition you just added.

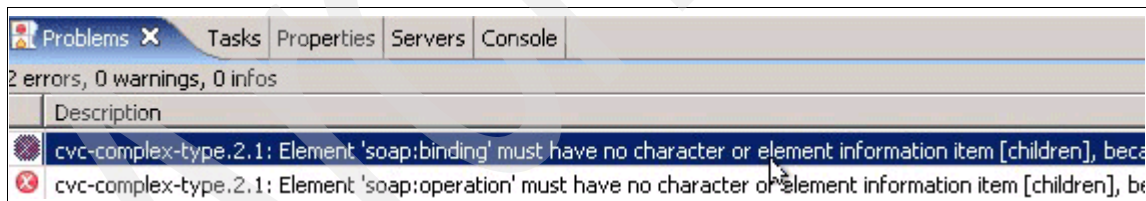


Figure B-9 Errors in binding

To correct these errors:

- a. Click the **Source** tab in the WSDL Editor to show the WSDL source and scroll down to the errors.

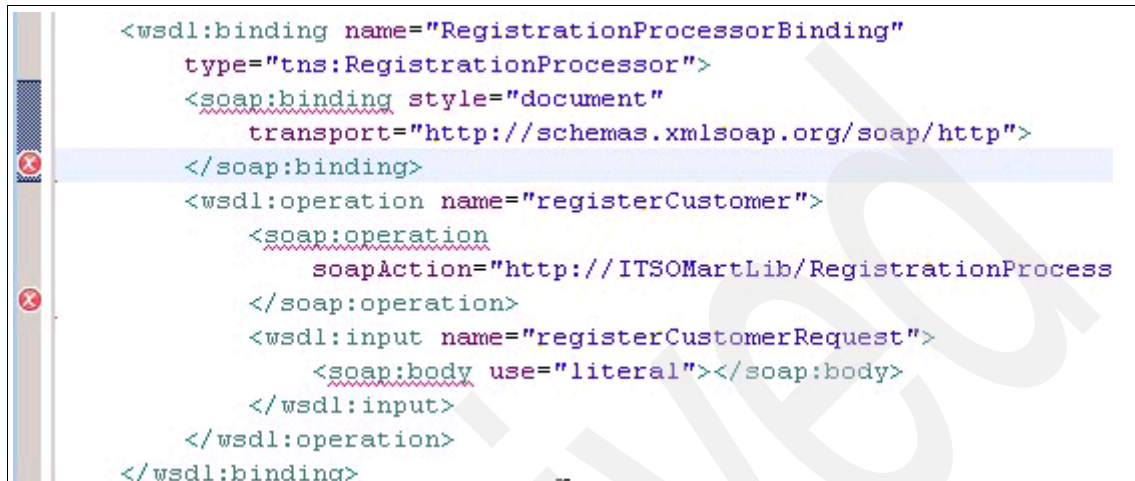


Figure B-10 Errors in WSDL source

- b. The errors are indicating that the <soap:binding>, <soap:operation>, and <soap:body> elements cannot have empty values. Fix these errors by simply removing the end tags for these elements and closing the tag within the beginning element tag. For example, instead of:

```
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http">
</soap:binding>
```

use:

```
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
```

The corrected source should look like Example B-1.

Example: B-1 Corrected WSDL source

```
<wsdl:binding name="RegistrationProcessorBinding"
  type="tns:RegistrationProcessor">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="registerCustomer">
    <soap:operation
      soapAction="http://ITSOMartLib/RegistrationProcessor/registerCustomer" />
    <wsdl:input name="registerCustomerRequest">
```

```

        <soap:body use="literal" />
    </wsdl:input>
</wsdl:operation>
</wsdl:binding>

```

8. Switch back to the Graph tab in the WSDL Editor, and under Services, right-click and select **Add Service**.

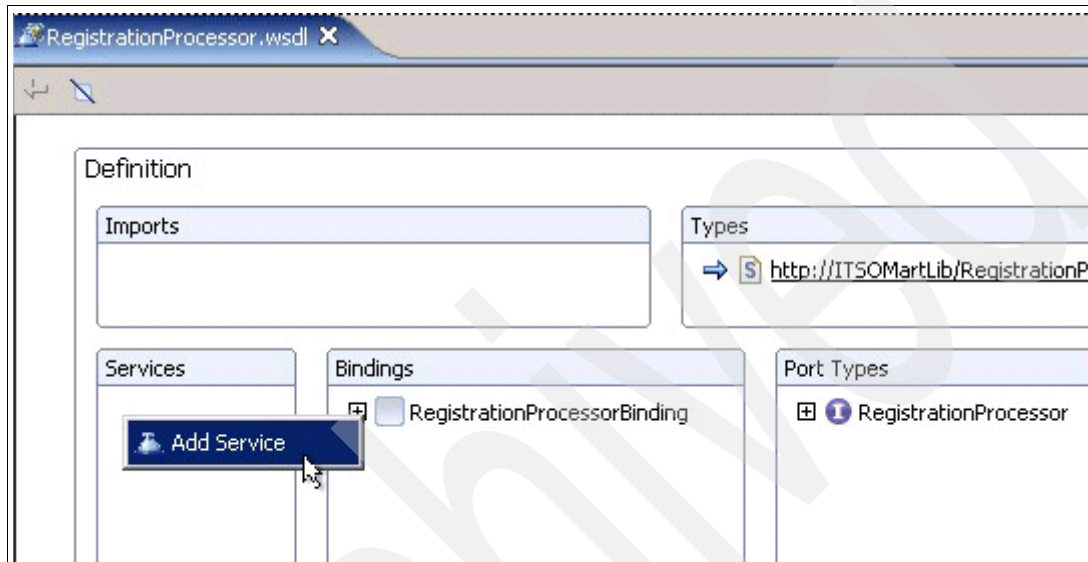


Figure B-11 Add Service

9. In the New Service window, specify a service name such as RegistrationProcessorService and click **OK**.

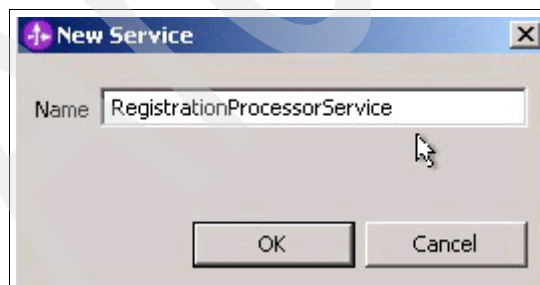


Figure B-12 New Service

10. Right-click **RegistrationProcessorService** and select **Add Port**.

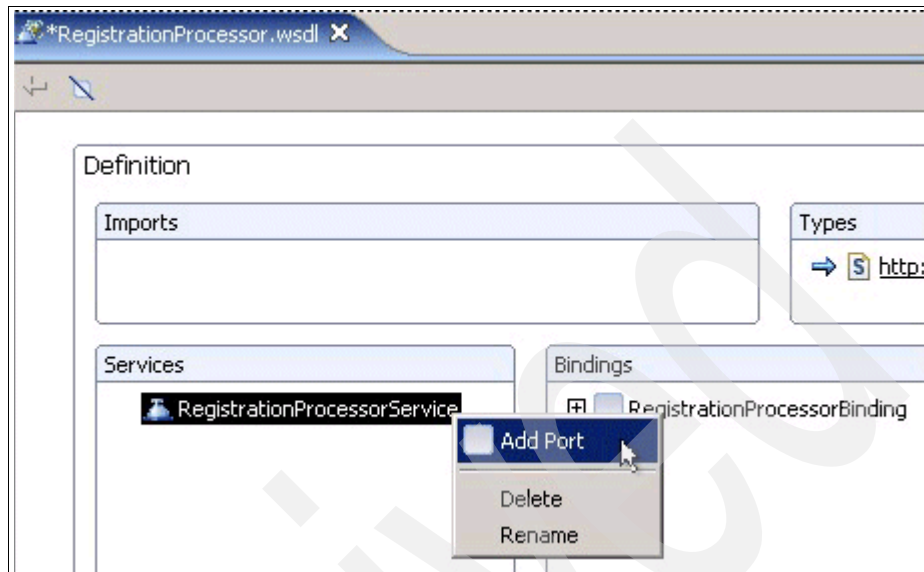


Figure B-13 Add Port

11. In the Port Wizard:

- a. Specify a port name, such as **RegistrationProcessorJMSPort**.
- b. Select the binding you created in the previous step, **tns:RegistrationProcessorBinding**.
- c. For the Protocol, select **SOAP**.
- d. Take the default for the Location field. You will update this value in the next step to use a SOAP/JMS URI.

e. Click **Finish** to create the port.

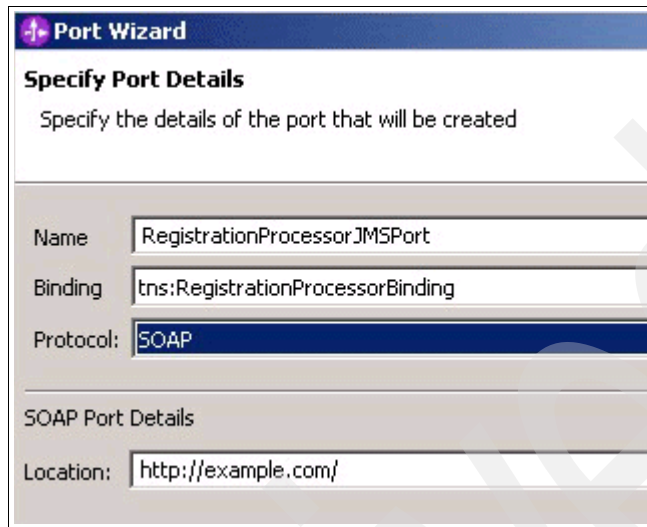


Figure B-14 Port Wizard

12. Update the soap:address location with a SOAP/JMS URI.

a. Under RegistrationProcessorService, select **RegistrationProcessorJMSPort** → **soap:address**.

b. In the Properties view, enter the following URI for the location property:

```
jms:/queue?destination=jms/RegistrationProcessorServiceQ&connectionFactory=jms/RegistrationProcessorServiceQCF&targetService=RegistrationProcessorServiceJmsPort
```

- *jms* indicates that this is a SOAP/JMS address.
- *queue* indicates that SOAP messages will be placed on a JMS queue (rather than a JMS topic).
- The destination property is the JNDI name of the JMS queue on which this Web service will receive SOAP messages.
- The connectionFactory property is the JNDI name of the JMS queue connection factory from which connections will be established to the JMS destination.
- The targetService property is the name of the Web service port that will be invoked when a SOAP message arrives on the JMS destination.

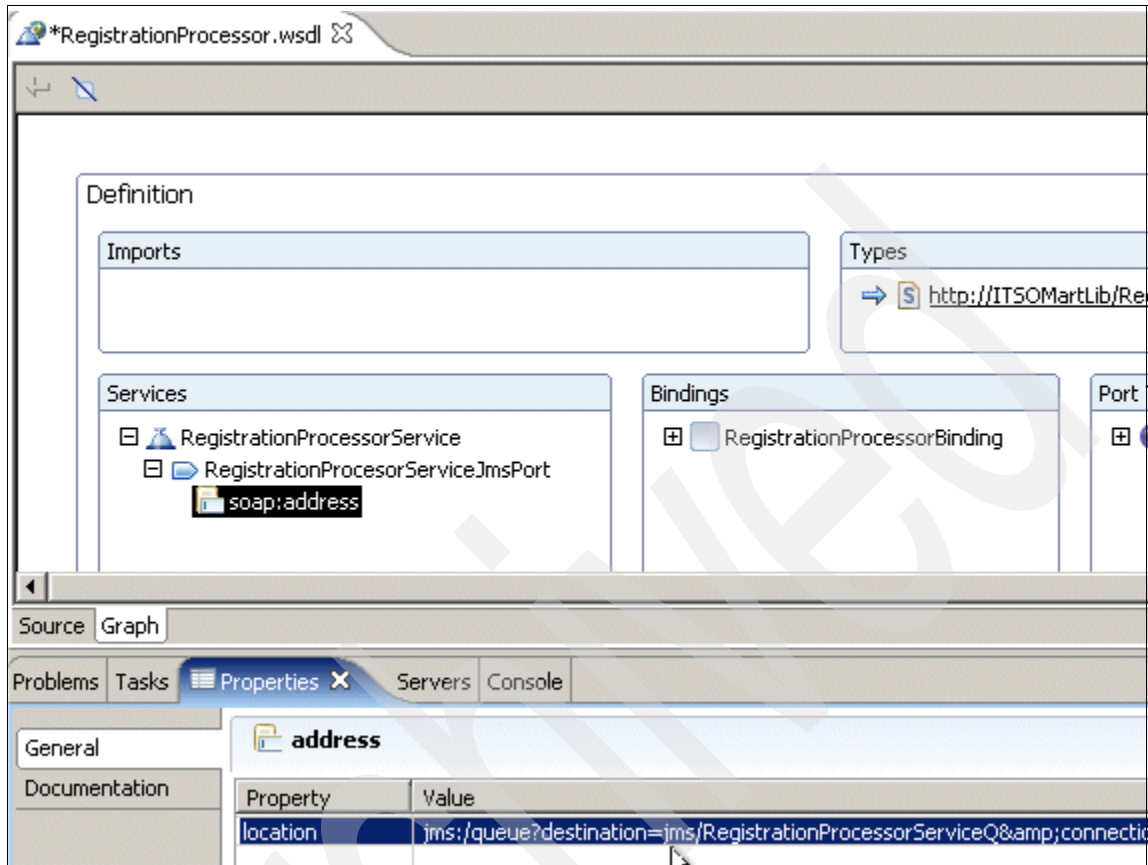


Figure B-15 SOAP address location

13. Save and close the RegistrationProcessor.wsdl file.

Create JMS resources

Create the JMS resources you are going to use for the SOAP/JMS Web service.

The following is a summary of the JMS resources needed. Detailed instructions for configuring these JMS resources for the sample application are provided in 11.8, “Configuration for JMS bindings” on page 501.

Create the following JMS objects in the test environment server:

- A JMS queue connection factory with the JNDI name:

jms/RegistrationProcessorServiceQCF

This is the JNDI name you specified for the `connectionFactory` property in the SOAP address location (step 12 on page 618).

- ▶ Another JMS queue connection factory with the JNDI name:

```
jms/RegistrationProcessorServiceReplyQCF
```

This will be the queue connection factory used by the Web service message-driven bean for sending reply messages if the Web service contains request-reply operations.

- ▶ A JMS queue with the JNDI name:

```
jms/RegistrationProcessorServiceQ
```

This is the JNDI name you specified for the destination property in the SOAP address location (step 12 on page 618).

- ▶ A JMS activation specification. This is required for the message-driven bean that gets created for you by the Web Service Wizard, which will listen on a JMS destination for SOAP messages. This activation specification needs to be configured with the following values:

- JNDI name:

```
jms/RegistrationProcessorServiceActivationSpec
```

This JNDI name can be anything you like, but it must match the value you enter into the Web Service Wizard for your SOAP/JMS Web service.

- Destination JNDI name:

```
jms/RegistrationProcessorServiceQ
```

This is the JNDI name of the JMS destination the message-driven bean will listen on for SOAP messages.

Note: If you do not define these resources before you generate the Web service using the Web Service Wizard, then WebSphere Integration Developer will hang and you will have to kill its Java process. This will occur when the Web Service Wizard attempts to deploy the application to the test environment and the deployment fails because the JMS activation specification specified for the message-driven bean is not configured.

Create the Web service

All of the previous steps have been in preparation for running the Web Service Wizard to create a top-down SOAP/JMS Web service. In this section you will generate a skeleton EJB session bean implementation for the Web service defined in the WSDL you have been working with.

1. Start the WebSphere ESB Server in the test environment.

Tip: Starting the test environment server before running the Web Service Wizard will make generating your Web service go much quicker. One of the first things the Web Service Wizard does is deploy the application to the server. If the server is stopped, then it will be started automatically, but the Web Service Wizard shows no indication that it is waiting for the server to start. So it will appear that the wizard is hung until the server is started.

2. In the J2EE perspective, navigate to the WSDL file in EJB Projects/
ITSO_RegProcService/wsdl/RegistrationProcessor.wsdl.
3. Right-click **RegistrationProcessor.wsdl** and select **Web Services** → **Generate Java bean skeleton**.
4. In the Web Service Wizard:
 - a. For the Web service type, select **Skeleton EJB Web Service**.
 - b. De-select **Start Web service in Web project**.

This option will not be needed since there is no Web project for this application and you are creating a SOAP/JMS Web service that runs in a EJB project instead of a Web project.
 - c. Select **Overwrite files without warning**.

This option allows you to complete the Web Service Wizard without getting prompted to overwrite files that get updated through the wizard. For example, the EJB Deployment Descriptor gets updated with the new Web service skeleton session bean.
 - d. Click **Next**.

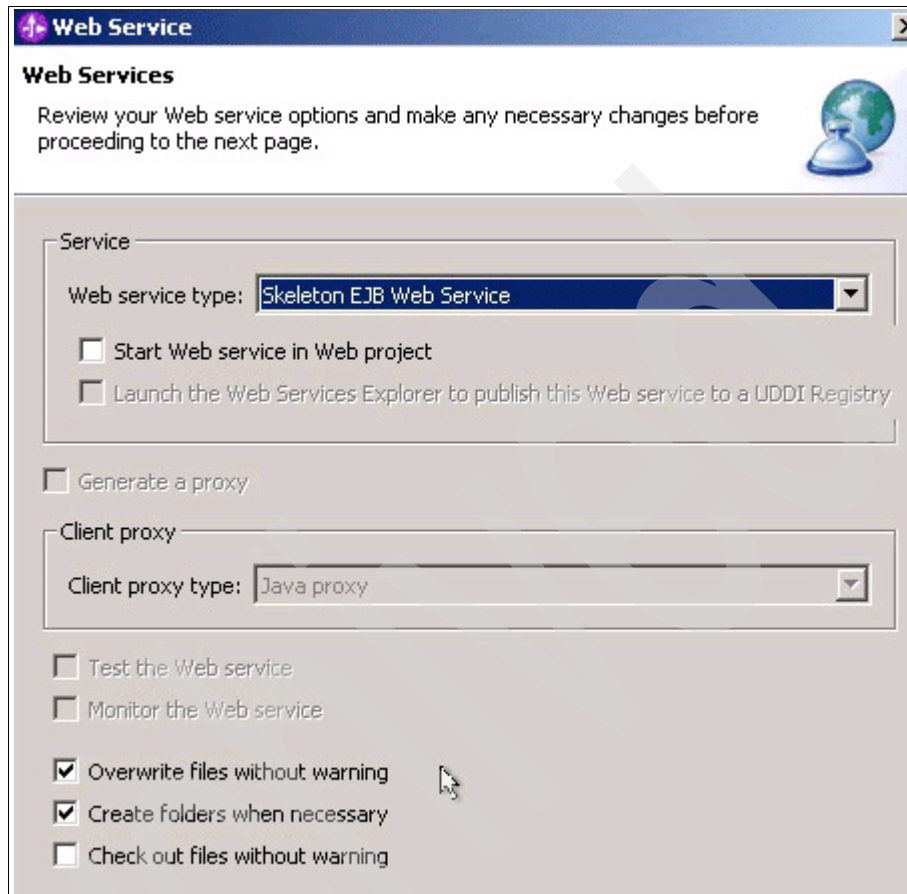


Figure B-16 Web Service Wizard: Web Services

5. On the Object Selection Page, ensure that the WSDL file location is correct and click **Next**.

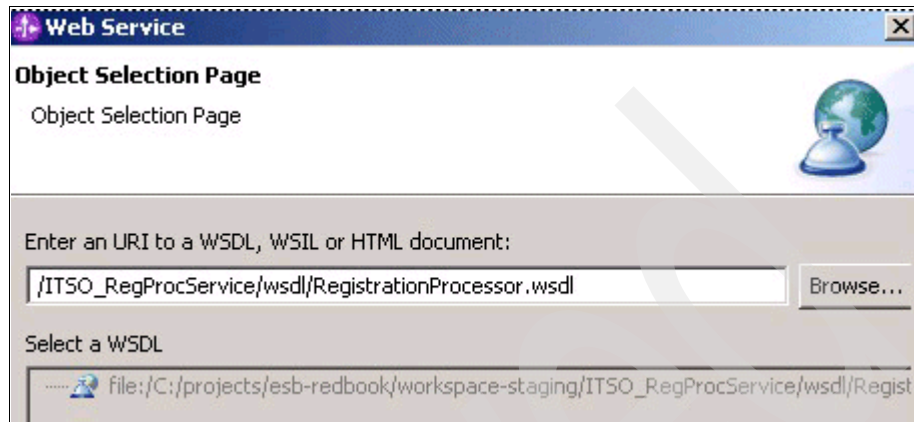


Figure B-17 Web Service Wizard: Object Selection Page

6. On the Service Deployment Configuration page, the Server-Side Deployment Selection section should have the settings that you specified in your workspace preferences, and the project names should already be set to the project from which you launched the Web Service Wizard:

- Web service runtime: IBM WebSphere
- Server: WebSphere ESB Server v6.0
- J2EE version: 1.4
- Service project: ITS0_RegProcService

Remember, for a SOAP/JMS Web service, this must be an EJB project.

- EAR project: ITS0_RegProcServiceApp

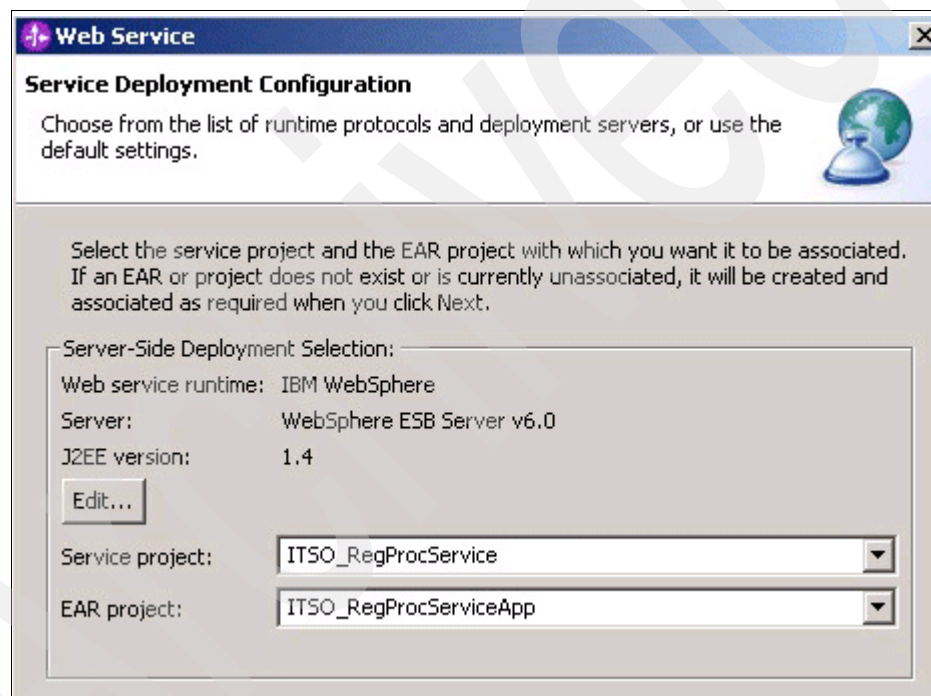



Figure B-18 Web Service Wizard: Service Deployment Configuration

7. Click **Next** on the Service Deployment Configuration page, and the wizard then publishes the EAR project ITS0_RegProcServiceApp to the server you specified. As mentioned earlier, if this server is not started, then the wizard will not continue to the next page until the server is started.

8. On the Web Service Skeleton EJB Configuration page:
 - a. For Select Router Project, enter the name of your EJB project, `ITS0_RegProcService`.

Your EJB project will not be in the drop-down selection, so you have to type this name into the field.
 - b. Under Select transports, SOAP over JMS should already be selected because you specified this as the preferred transport in your workspace preferences and the SOAP address location in your WSDL file contains a SOAP/JMS URI.
 - c. Notice that most of the JMS URI Properties are pre-filled with the values you specified in the URI you provided for the SOAP address location in the WSDL:
 - Queue kind: queue
 - WSDL service name: `RegistrationProcesorServiceJmsPort`
 - Connection factory: `jms/RegistrationProcessorServiceQCF`
 - Destination: `jms/RegistrationProcessorServiceQ`
 - d. For the MDB deployment mechanism, select **JMS ActivationSpec**.
 - e. For ActivationSpec JNDI name, enter:
`jms/RegistrationProcessorServiceActivationSpec`
 - f. Click **Next**.


Web Service

Web Service Skeleton EJB Configuration
 Define a skeleton EJB as a Web service.

WSDL folder /ITSO_RegProcService/ejbModule/wsdl
 WSDL file name RegistrationProcessor.wsdl

Select Router Project ITS0_RegProcService

Security Configuration No Security

☐ Define custom mapping for namespace to package.

Select transports:
☐ SOAP over HTTP ☒ SOAP over JMS

JMS URI Properties
 Queue kind queue
 WSDL service name RegistrationProcessorJMSPort
 Connection factory jms/ITSOMart/RegistrationProcessorServiceQCF
 Destination jms/ITSOMart/RegistrationProcessorServiceQ
 MDB deployment mechanism JMS ActivationSpec
 ActivationSpec JNDI name jms/ITSOMart/RegistrationProcessorServiceActivationSpec
 Listener port name

Figure B-19 Web Service Wizard: Web Service Skeleton EJB Configuration

- On the next page, accept the defaults and click **Finish** to close the Web Service Wizard.

The skeleton session bean implementation class file is automatically opened for you in the editor for you to implement the operations that were defined in your WSDL. You will implement this class in the next section.

10. Look at what the Web Service Wizard generated for you:

a. In the Project Explorer, expand

EJB Projects/ITSO_RegProcService/Deployment Descriptor.

- Under Session Beans, you will see the stateless session bean that was generated with the interface defined in the WSDL, `RegistrationProcessorBindingImpl`. This bean name comes from the service binding name in the WSDL.
- Under Message-Driven Beans, you will see a bean named `WebServicesJMSRouter`. This bean routes SOAP messages that arrive on the JMS queue to the session bean that implements the Web service.

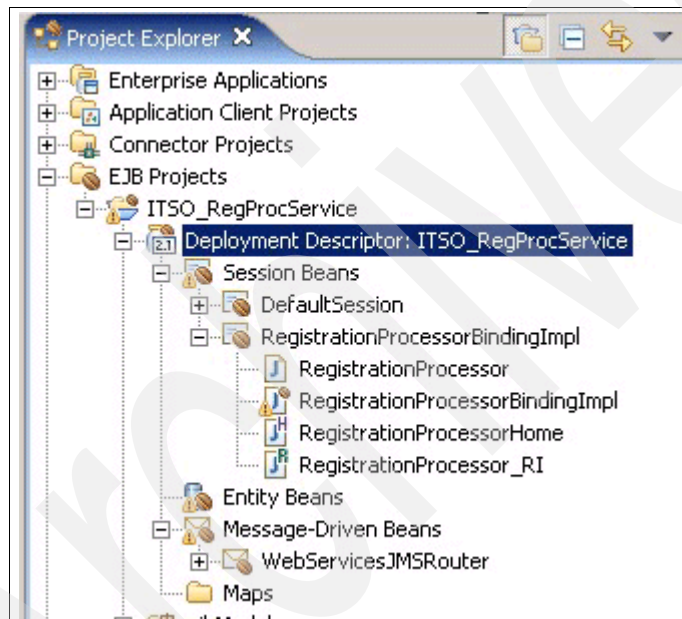


Figure B-20 J2EE artifacts generated by the Web Service Wizard

- The deployed WSDL and XSD files for the service were copied into the ejbModule/META-INF/wsdl directory.

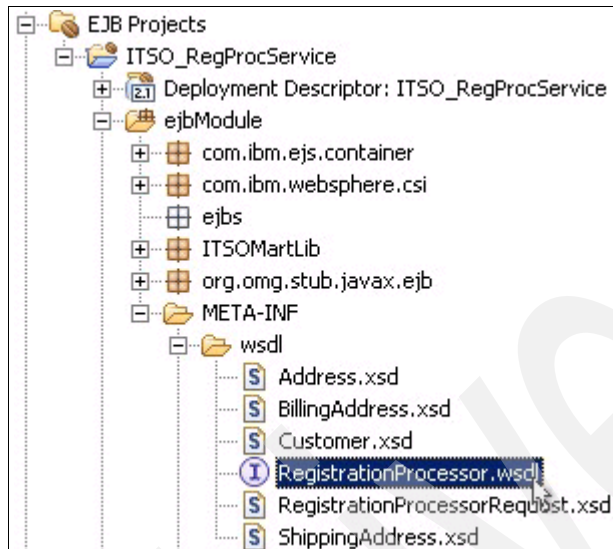


Figure B-21 Deployed WSDL for the service

11. You can delete the DefaultSession session bean now if you like. It will not be used for anything in this sample.
12. Double-click **Deployment Descriptor** to open it in the EJB Deployment Descriptor editor.
 - a. In the Bean tab, select the **RegistrationProcessorBindingImpl** bean. Under WebSphere Bindings enter the JNDI name for this bean:
ejb/ITSMart/RegistrationProcessor

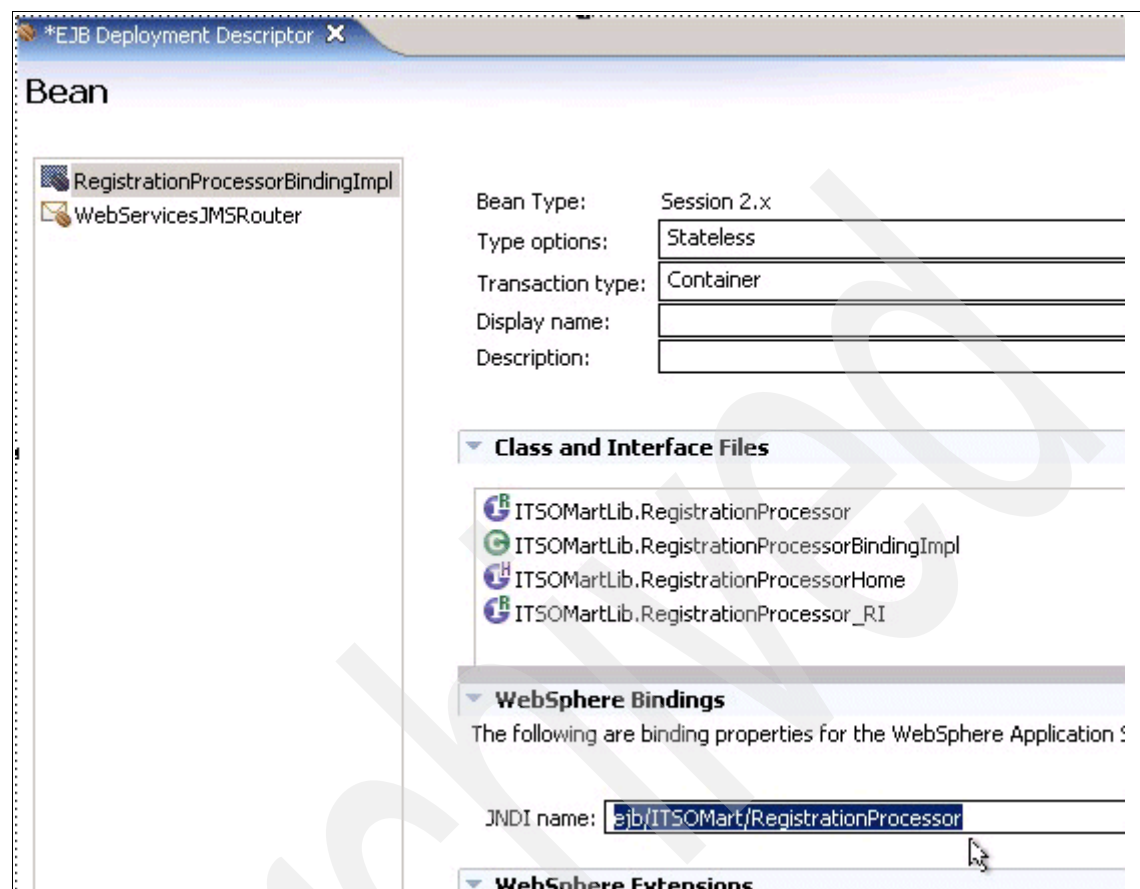


Figure B-22 Session bean JNDI name

- b. Select the **WebServicesJMSRouter** bean and notice that the JMS activation specification that you specified in the Web Service Wizard is specified for this bean under the WebSphere Bindings.

The screenshot shows the 'EJB Deployment Descriptor' window. On the left, a tree view shows 'RegistrationProcessorBindingImpl' and 'WebServicesJMSRouter'. The main area displays the configuration for 'WebServicesJMSRouter'.

Bean Type: Message-Driven
Transaction type: Container
Display name:
Description:

Message-Driven Destination
Destination type:

Activation Configuration
The following activation configurations are defined for the message driven bean:

destinationType	Name
<input type="radio"/> Listener Port	ListenerPort name:
<input checked="" type="radio"/> JCA Adapter	ActivationSpec JNDI name: jms/RegistrationProcessorServiceActivationSpec ActivationSpec Authorization Alias:

At the bottom left are 'Add...' and 'Remove' buttons.

Figure B-23 Message-driven bean ActivationSpec

- c. Switch to the deployment descriptor References tab. Select the queue connection factory, **ResourceEnvRef**, defined under WebServicesJMSRouter.

Under WebSphere Bindings, change the JNDI name to `jms/RegistrationProcessorServiceReplyQCF` (Figure B-24).

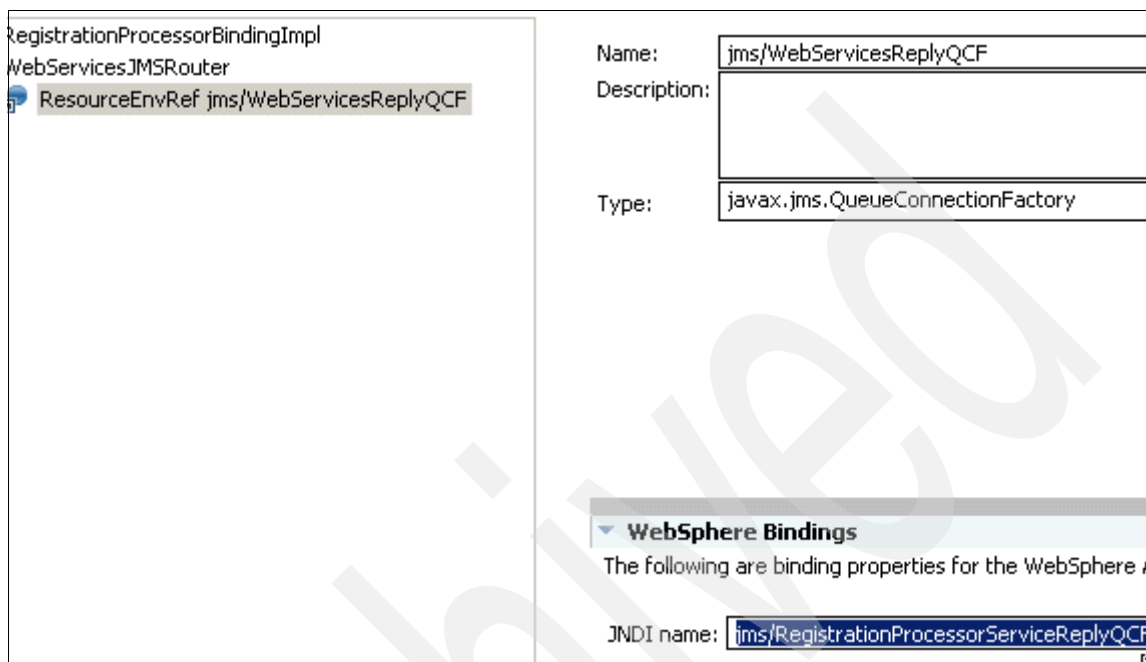


Figure B-24 Message-driven bean resource reference

- d. Save and close the EJB Deployment Descriptor.

Implement the Web service

The last step is to add code to implement the Web service:

1. In the Project Explorer, expand:

EJB Projects/ITS0_RegProcService/Deployment Descriptor/Session Beans/RegistrationProcessorBindingImpl

Double-click **RegistrationProcessorBindingImpl** to open the session bean implementation class in the Java editor.

2. Scroll down to the registerCustomer method.

Notice that this method has the same signature as the interface defined in the WSDL.

Delete the throws clause (throws java.rmi.RemoteException) that the editor and the Problems view mark with a warning because this throws clause does not comply with the EJB 2.0 specification.

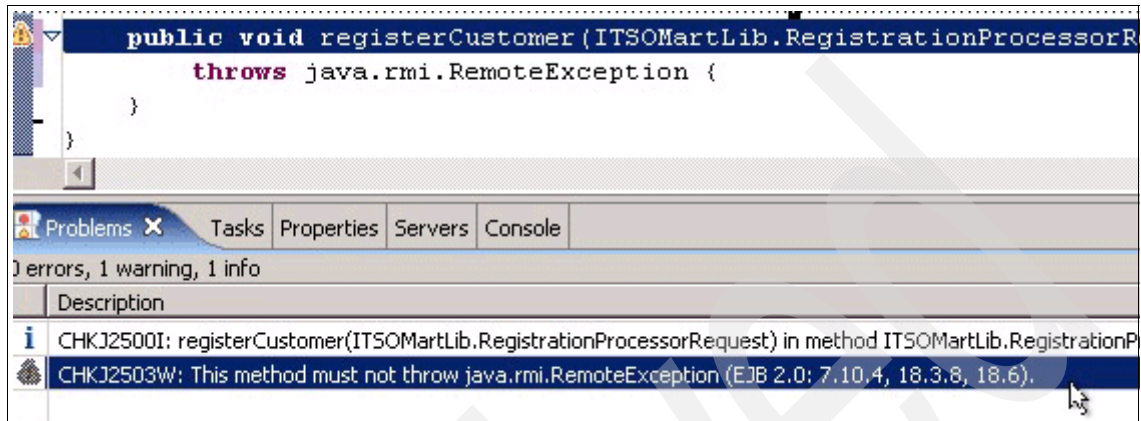


Figure B-25 Warning for throws clause in session bean method

3. Implement the registerCustomer method. For example, enter the following lines of code to print customer information out to the log (Example B-2).

Example: B-2 Print customer information

```
System.out.println("ITSOMart RegistrationProcessorService >> registerCustomer");
if (registrationRequest != null) {
    Customer customer = registrationRequest.getCustomer();
    System.out.println("\t customer - " + customer.getEmail());
}
```

4. Save and close the file.

Create the Web service client

The next step is to write a client application that invokes your SOAP/JMS Web service. The Web Services Explorer provided with WebSphere Integration Developer can only be used to test SOAP/HTTP Web services. It cannot make SOAP/JMS invocations. So in order to test your SOAP/JMS Web service you have to generate a Web service client proxy for it, then use that proxy in an application.

The Web application for ITSOMart has a JSF page that invokes a proxy for the Register Customer process (ITSO_RegProcServiceApp). The ITSOMart application consists of the following:

- ▶ ITSOMartApp
- ▶ ITSOMartWeb
- ▶ ITSOMart_Proxies

This is the Java utility project for the application that contains only the generated Web service client proxy. You will re-generate the client proxy into this project in the following steps.

For this example, we assume that you have these files loaded into your workspace.

1. Copy the deployed WSDL and XSD files for the service into the client application.

In this case, the files are located under the ITSO_RegProcService/ejbModule/META-INF/wsdl folder and must be copied into the ITSOMart_Proxies/wsdl folder (Figure B-26).

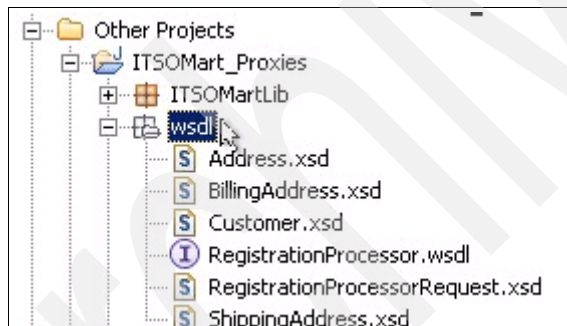
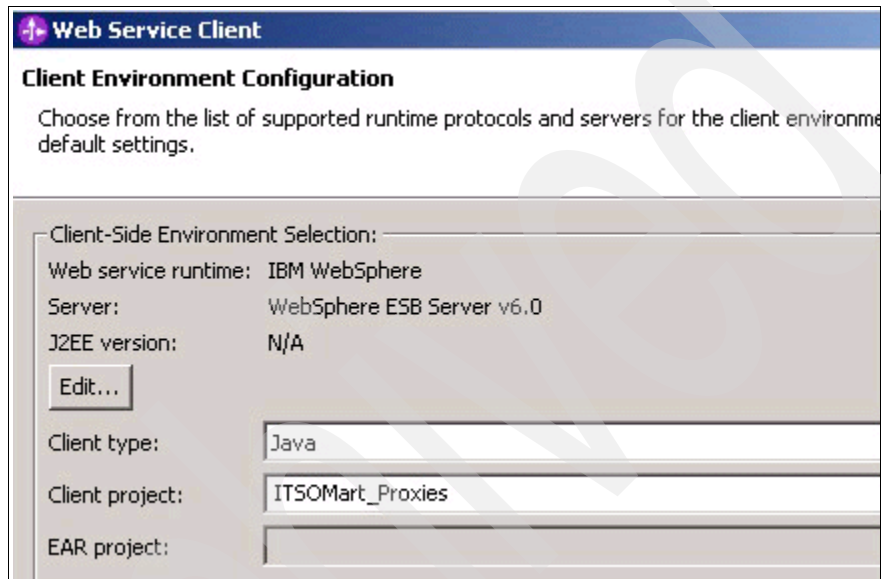


Figure B-26 ITSOMart_Proxies wsdl

2. Start the test environment server.
3. Right-click the WSDL file **ITSOMart_Proxies/wsdl/RegistrationProcessor.wsdl** and select **Web Services** → **Generate Client**.
 - a. In the Web Service Client wizard, for the client proxy type select **Java proxy** then click **Next**.
 - b. On the Web Service Selection page, the WSDL that you selected should already be filled in. Click **Next**.

- c. On the Client Environment Configuration page, select the following values that determine how the client proxy will be generated:
- Web service runtime: **IBM WebSphere**
 - Server: **WebSphere ESB Server v6.0**
 - J2EE version: **N/A**
 - Client type: **Java**
 - Client project: **ITSOMart_Proxies**



The screenshot shows a dialog box titled "Web Service Client" with a sub-header "Client Environment Configuration". Below the sub-header is a text prompt: "Choose from the list of supported runtime protocols and servers for the client environment default settings." The main area of the dialog is titled "Client-Side Environment Selection:" and contains several configuration fields. The "Web service runtime:" is set to "IBM WebSphere", the "Server:" is "WebSphere ESB Server v6.0", and the "J2EE version:" is "N/A". There is an "Edit..." button next to these three fields. Below these, the "Client type:" is set to "Java", the "Client project:" is "ITSOMart_Proxies", and the "EAR project:" field is empty.

Client-Side Environment Selection:	
Web service runtime:	IBM WebSphere
Server:	WebSphere ESB Server v6.0
J2EE version:	N/A
<input type="button" value="Edit..."/>	
Client type:	Java
Client project:	ITSOMart_Proxies
EAR project:	

Figure B-27 Web Service Client: Client Environment Configuration

- d. Click **Finish** to generate the Web service client proxy. Acknowledge any warning messages you receive during the Web service client generation.

4. Open the Web perspective, then open the sample JSF page by double-clicking the following file:

Dynamic Web Projects/ITSMartWeb/Web Content /CustomerRegistration.jsp

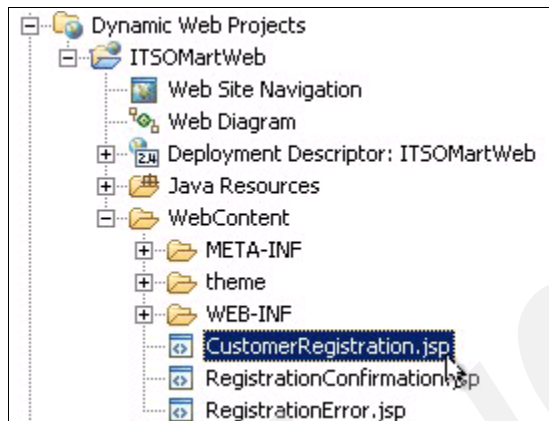


Figure B-28 Sample JSF page

This JSF page opens in the editor (Figure B-29).

CustomerRegistration.jsp

CustomerRegistration.jsp - ITSOMart...pplication - Customer Registration

tpl:put Standard

ITSOMart **Self-Service Application**

Customer Registration

Please enter your information below. Your email address will become your unique identification in our system.

Contact Name (required)

First: Last:

{firstName} {lastName}

{Error Message for firstName}

Company / Organization:

{companyName}

Figure B-29 Sample JSF page

5. Scroll down to the bottom of the page and click the **Submit Registration** button (Figure 12-16). Then in the Properties view, click the **QuickEdit** tab.

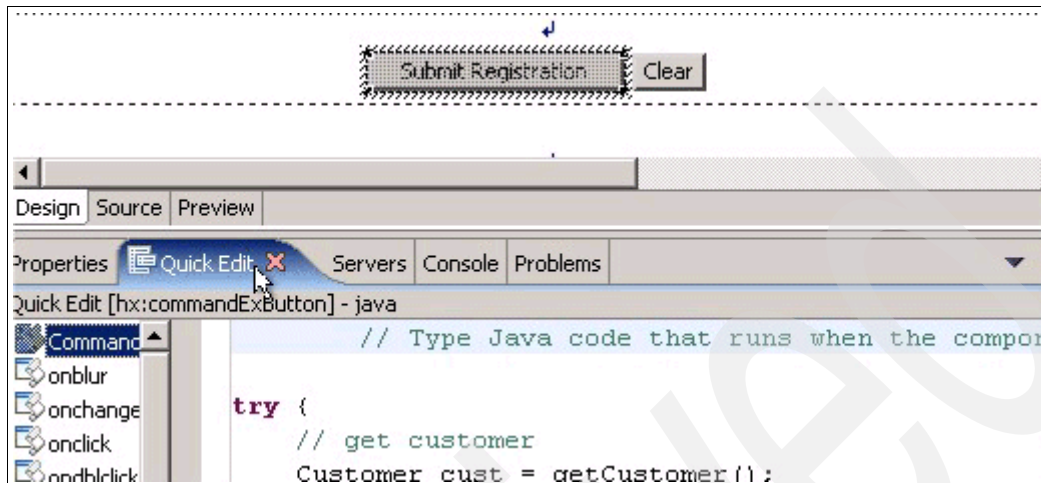


Figure 12-16 Submit Registration QuickEdit

6. In the QuickEdit view, you can see the piece of code that populates a RegistrationProcessorRequest with the form data from the page and invokes the registerCustomer method of RegistrationProcessorProxy.

Example: B-3 Sample code for invoking the RegistrationProcessorProxy

```
try {
    // get customer
    Customer cust = getCustomer();
    if (cust != null) {
        // billingAddress
        BillingAddress bAddress = getBillingAddress();
        if (bAddress != null) cust.setBillingAddress(bAddress);

        Vector v = new Vector();
        // shippingAddress1
        ShippingAddress sAddress1 = getShippingAddress1();
        if (sAddress1 != null) {
            v.add(sAddress1);

            // shippingAddress2
            ShippingAddress sAddress2 = getShippingAddress2();
            if ((sAddress2 != null) && (sAddress2.getName() != null) &&
                (!sAddress2.getName().equals(""))) v.add(sAddress2);
        }
        ShippingAddress[] sAddresses = new ShippingAddress[v.size()];
        v.copyInto(sAddresses);
```

```

        cust.setShippingAddress(sAddresses);
    }

    // invoke RegistrationProcessorService (soap/jms)
    System.out.println("ITSOMart CustomerRegistration >> submitting registration for " +
getCustomer().getEmail());
    RegistrationProcessorProxy proxy = new RegistrationProcessorProxy();
    RegistrationProcessorRequest request = new RegistrationProcessorRequest();
    request.setCustomer(cust);
    proxy.registerCustomer(request);

    return "success";
} catch (Exception e) {
    System.out.println("ITSOMart CustomerRegistration [Submit Registration] >> exception: " +
e);
    return "error";
}

```

Test the Web service

Now that you have got a client application that invokes the Register Customer process using a generated Web service client proxy, you can test the Web service in the test environment.

1. Right-click **Dynamic Web Projects/ITSOMartWeb/ Web Content/ CustomerRegistration.jsp** and select **Run** → **Run on Server**.

The ITSOMartApp will be published to the test environment and the JSF page is displayed in the browser.

2. Click the **Submit Registration** button to submit the customer details and invoke the Registration Processor Service. You should see the following messages printed in the console for the server log:

```

ITSOMart CustomerRegistration >> submitting registration for carla@ibm.com
ITSOMart RegistrationProcessorService >> registerCustomer
customer - carla@ibm.com

```


Server errors in the test environment

In WebSphere Integration Developer 6.0.1, you will see the following error occur when starting a WebSphere ESB Server in the test environment (Example 12-1).

Example 12-1 Error

```
[3/17/06 12:55:19:359 MST] 0000000a WsServerImpl A WSVR0002I: Server server1 open for
e-business, problems occurred during startup
[3/17/06 12:54:06:609 MST] 0000000a SystemErr R com.ibm.ws.exception.ConfigurationWarning:
C:\pf\esb2\config\cells\esbNode02Cell\cell-wbi.xml (The system cannot find the file specified)
at
com.ibm.wbiserver.relationshipservice.model.RelationshipServiceConfig.initialize(RelationshipSe
rviceConfig.java:189)
    at com.ibm.ws.runtime.component.ContainerImpl.initializeComponent(ContainerImpl.java:1160)
    at com.ibm.ws.runtime.component.ContainerImpl.initializeComponents(ContainerImpl.java:979)
    at com.ibm.ws.runtime.component.ServerImpl.initialize(ServerImpl.java:277)
    at com.ibm.ws.runtime.WsServerImpl.bootServerContainer(WsServerImpl.java:173)
    at com.ibm.ws.runtime.WsServerImpl.start(WsServerImpl.java:133)
    at com.ibm.ws.runtime.WsServerImpl.main(WsServerImpl.java:387)
    at com.ibm.ws.runtime.WsServer.main(WsServer.java:53)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:85)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:58)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:60)
    at java.lang.reflect.Method.invoke(Method.java:391)
    at com.ibm.ws.bootstrap.WSLauncher.run(WSLauncher.java:218)
    at java.lang.Thread.run(Thread.java:568)
```

You can ignore this error. This is caused, as it says in the exception, by the file cell-wbi.xml not existing under the WebSphere ESB Server profile. This is a configuration file that is only needed for WebSphere Process Server functionality.

The problem has been reported and has a fix available at:

<http://www-1.ibm.com/support/docview.wss?uid=swg24011345>

Errors using XML Mapper without Internet connectivity

In early releases, there was a defect in WebSphere Integration Developer when working with XSL Transformation primitives in the Mediation Flow editor on a workstation that does not have Internet connectivity. The XML map file that gets generated by the XML Mapper for the XSLT refers to the following schema:

<http://www.w3.org/2001/xml.xsd>

This schema was not included in WebSphere Integration Developer. If you tried to open the XML Mapper to edit the XSLT, WebSphere Integration Developer hung waiting for a connection to:

<http://www.w3.org>

A PMR was opened on this issue.

Important: If you experience this problem and must work without Internet connectivity, update your WebSphere Integration Developer to 6.0.1.1.

Creating a new server in the test environment

This section contains instructions for creating and using a WebSphere ESB Server profile in the WebSphere Integration Developer test environment.

In these instructions, we assume the following: File locations denoted with `<wid_install>` represent the directory in which you have WebSphere Integration Developer installed.

During the installation of WebSphere Integration Developer, you are presented with the option of creating a WebSphere ESB Server profile. If you selected this option, then you will already have a profile created under the directory `<wid_install>\pfesb`. You will also have a WebSphere ESB Server already defined for you in the test environment under the Servers view.

If you installed WebSphere Integration Developer on Windows into the default installation directory located under the `c:\Program Files` directory, then it is very likely that you will encounter problems running mediation module applications in the test environment due to the Windows limit on path names longer than 256 characters. This occurs because deployment artifacts for your applications that you run in the test environment get placed under long directory structures under the profile directory.

Instructions are provided here to show you how to configure a new profile into a short directory structure so that you can get around this restriction.

Create the WebSphere ESB Server profile

First you must have a new application server profile:

1. Launch the WebSphere ESB profile creation wizard located under `<wid_install>\runtimes\bi_v6\bin\ProfileCreator_wbi`. The executable `esbpcatWindows.exe` (on Windows) or `esbpcatLinux.bin` (on Linux) is the WebSphere ESB profile creation wizard.

2. The first page of the wizard indicates that this is the WebSphere Process Server 6.0 Profile Wizard (Figure B-30), but do not be fooled — as long as you launched the correct executable in step 1 that begins with *esb*, then you are actually running what should have been labeled the WebSphere ESB Server Profile Wizard.



Figure B-30 Profile Wizard

3. In the Existing profile detection page, select **Create a new WebSphere Process Server profile** (Figure B-31). Again, this is simply mislabeled, and you will actually be creating a WebSphere ESB Server profile.

Existing profile detection

An existing profile has been detected. You can choose to create a new profile or augment an existing profile. If you augment an existing WebSphere Application Server profile, it will then become a WebSphere Process Server profile.

☒ Create a new WebSphere Process Server profile

☐ Augment an existing WebSphere Application Server profile

Figure B-31 Profile Wizard: Existing profile detection

4. In the Profile type selection page, select **Stand-alone profile** (Figure B-32).

Profile type selection

A profile defines a runtime environment. Choose the type of profile to create or augment best for your needs. Although you can create or augment just one profile at a time, you can rerun the Profile Wizard multiple times to create or augment additional profiles.

Deployment manager profile

☐ The first step in setting up a Network Deployment environment is to create a deployment manager. A deployment manager administers process servers and application servers that are federated into (made a part of) its cell. The next step is to create additional profiles and federate them.

Custom profile

☐ A custom profile contains an empty node, which does not contain an administrative console or any servers. The typical use for a custom profile is to federate its node to a deployment manager. After federating the node, use the deployment manager to create a server or a cluster of servers within the node.

Stand-alone profile

☒ A stand-alone profile runs your enterprise applications. It can be managed from its own administrative console and function independent of other WebSphere Process Server profiles and deployment managers.

Figure B-32 Profile Wizard: Profile type selection

5. In the Profile name page, enter the name you want to call your profile. This profile name must be unique within the WebSphere ESB installation. The default WebSphere ESB Server profile created when you installed WebSphere Integration Developer is named *esb*.

Enter a value (for example, of *esb2*) (Figure B-33).

Leave “Make this profile the default” unchecked.

Profile name

Provide a unique name for the profile.

Profile name:

esb2

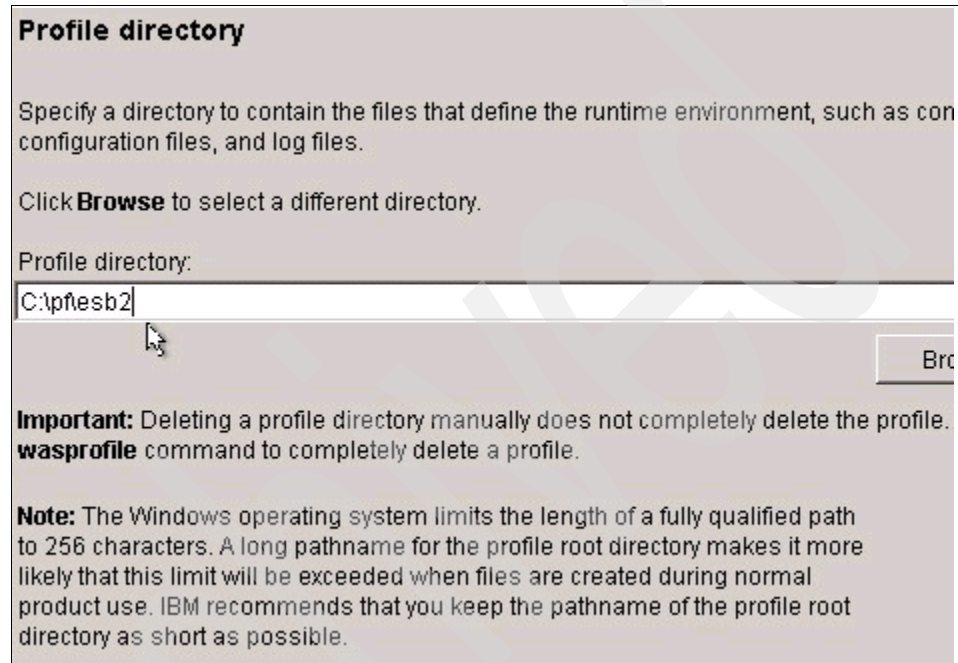
☐ Make this profile the default.

Each installation of WebSphere Process Server always has one default profile. Components that run without referring to a specific profile will use the default profile. Selecting this option makes this profile the new default.

Figure B-33 Profile Wizard: Profile name

6. On the Profile directory page, enter a very short path on your file system for where your new profile will be located (for example, enter c:\pf\esb2).

This step is the main reason we suggest that you create another profile. The shorter the profile directory path, the less likely you will encounter problems with the Windows limit on path names longer than 256 characters.



Profile directory

Specify a directory to contain the files that define the runtime environment, such as configuration files, and log files.

Click **Browse** to select a different directory.

Profile directory:

C:\pf\esb2

Important: Deleting a profile directory manually does not completely delete the profile. The **wasprofile** command to completely delete a profile.

Note: The Windows operating system limits the length of a fully qualified path to 256 characters. A long pathname for the profile root directory makes it more likely that this limit will be exceeded when files are created during normal product use. IBM recommends that you keep the pathname of the profile root directory as short as possible.

Figure B-34 Profile Wizard: Profile directory

7. On the Node and host names page, the node name will be pre-filled with a unique node name and the host name with the host name of your machine. For consistency in naming, change the node name to something that more closely matches the name of your profile. A profile represents the configuration of a WebSphere node.

Enter a node name (for example, of esbNode2) (Figure B-35).

Also, notice the note in this wizard page that discusses the Windows path length restriction.

Node and host names

Specify a node name and a host name for this profile. Refer to the installation and migration information for detailed field descriptions and migration considerations.

Node name:

Host name:

Node name: The node name is used for administration. If the node is federated, the name must be unique within the cell.

Host name: The host name is the domain name system (DNS) name (short or long) or the IP address of this computer.

Note: The Windows operating system limits the length of a fully qualified path to 256 characters. Long node and host names make it more likely that this limit will be exceeded when files are created during normal product use. IBM recommends that you specify a maximum of eight characters for each name.

Figure B-35 Profile Wizard: Node and host names

8. On the Port value assignment page, all of the port values will be pre-filled with unique values within the WebSphere ESB installation so that you do not encounter port conflicts if you happen to run servers in multiple profiles at the same time. Because we are creating a profile specifically for use in the WebSphere Integration Developer test environment, we know that we will not be running multiple WebSphere ESB Servers at the same time.

Change all of the port values to the default values indicated next to each port name in parenthesis (Figure B-36).

Port value assignment	
The values in the following fields define the ports for this profile and do not conflict with other profiles in this installation. Another installation of WebSphere Application Server or other programs might use the same ports. To avoid runtime port conflicts, verify that each port value is unique.	
Administrative console port (Default 9060):	9060
Administrative console secure port (Default 9043):	9043
HTTP transport port (Default 9080):	9080
HTTPS transport port (Default 9443):	9443
Bootstrap port (Default 2809):	2809
SOAP connector port (Default 8880):	8880
SAS SSL ServerAuth port (Default 9401):	9401
CSIV2 ServerAuth listener port (Default 9403):	9403
CSIV2 MultiAuth listener port (Default 9402):	9402
ORB listener port (Default 9100):	9100
High availability manager communication port (Default 9353):	9353
Service Integration Port (Default 7276):	7276
Service Integration Secure Port (Default 7286):	7286
Service Integration MQ Interoperability Port (Default 5558):	5558
Service Integration MQ Interoperability Secure Port (Default 5578):	5578

Figure B-36 Profile Wizard: Port value assignment

9. On the Windows service definition page (if you are creating a profile on Windows), de-select **Run the WebSphere Process Server process as a Windows service** (Figure B-37 on page 647). In a test environment, it is better to stop and start the server manually.

Windows service definition

Choose whether to use a Windows service to run the WebSphere Process Server. Windows Services can start and stop the WebSphere Process Server, and configure startup and actions.

☐ Run the WebSphere Process Server process as a Windows service

☒ Log on as a local system account

☐ Log on as a specified user account

User name:

Password:

Startup type:

The user account that runs the Windows Service must have the following user rights:

- Act as part of the operating system
- Log on as a service

Figure B-37 Profile Wizard: Windows service definition

10. On the Service Component Architecture page, leave “Configure the Service Integration Bus in a secured mode” unchecked. You can always configure security on the service integration bus after the profile is created using the administrative console.

Service Component Architecture configuration

WebSphere Process Server 6.0 provides the capability for components to communicate asynchronously. Please provide a user name and password to be used to connect to Service Integration Bus in a secured mode.

☐ Configure the Service Integration Bus in a secured mode

Figure B-38 Profile Wizard: Service Component Architecture configuration

11. On the Common Event Infrastructure page:
 - User ID and password fields: Even though in the previous step you selected to not configure security on the service integration bus (which provides WebSphere Messaging), you must provide a value for these

fields. Otherwise, the wizard will not let you continue to the next page. So enter anything into these fields.

- WebSphere server name: server1
- Choose a database product: Cloudscape V5.1

Common Event Infrastructure Configuration

Configure the Common Event Infrastructure EJBs, Default Messaging, and database.

User ID to authenticate with WebSphere Messaging queue manager:

wsdemo

Password (the password for WebSphere Messaging authentication):

Password confirmation:

WebSphere server name:

server1

Choose a database product:

Cloudscape V5.1

Figure B-39 Profile Wizard: Common Event Infrastructure Configuration

12. On the Business Process Choreographer Configuration page, leave “Configure a sample Business Process Choreographer” unchecked. Technically, this page should not even be in this WebSphere ESB Profile Wizard since WebSphere ESB does not support the use business processes that execute in the Business Process Choreographer (Figure B-40).

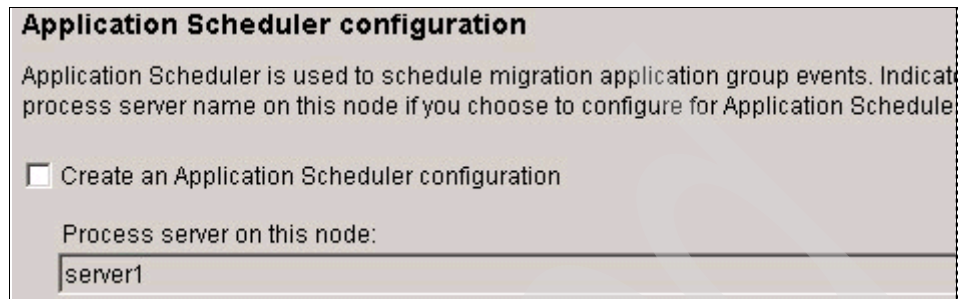
Business Process Choreographer Configuration

Please provide a user name and password for the sample Business Process Choreo configuration to connect to the Service Integration Bus in a secured mode as well as th security role for the business process system administrator. A Cloudscape database used for this configuration.

☐ Configure a sample Business Process Choreographer

Figure B-40 Profile Wizard: Business Process Choreographer Configuration

13. On the Application Scheduler configuration page, leave “Create an Application Scheduler configuration” unchecked, as this is another feature of WebSphere Process Server that is not used by WebSphere ESB.



Application Scheduler configuration

Application Scheduler is used to schedule migration application group events. Indicate the process server name on this node if you choose to configure for Application Scheduler.

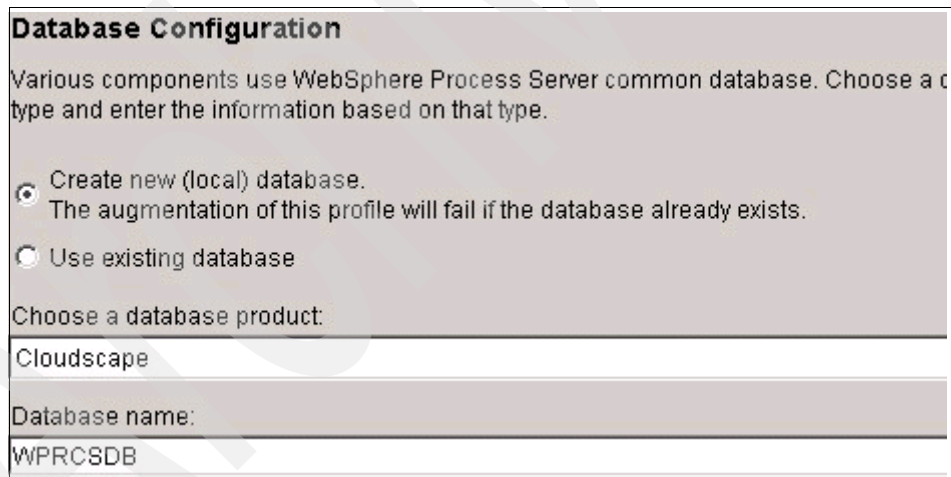
☐ Create an Application Scheduler configuration

Process server on this node:

server1

Figure B-41 Profile Wizard: Application Scheduler configuration

14. On the Database Configuration page, take all the defaults (Figure B-42). For a WebSphere ESB profile, this database will not get created because it is not needed.



Database Configuration

Various components use WebSphere Process Server common database. Choose a database type and enter the information based on that type.

☒ Create new (local) database.
The augmentation of this profile will fail if the database already exists.

☐ Use existing database

Choose a database product:

Cloudscape

Database name:

WPRCSDB

Figure B-42 Profile Wizard: Database Configuration

15. The Profile summary page shows the details of the name and location of the profile that will be created (Figure B-43). Again, this profile will actually be a stand-alone WebSphere ESB profile, not a WebSphere Process Server profile as the wizard indicates.

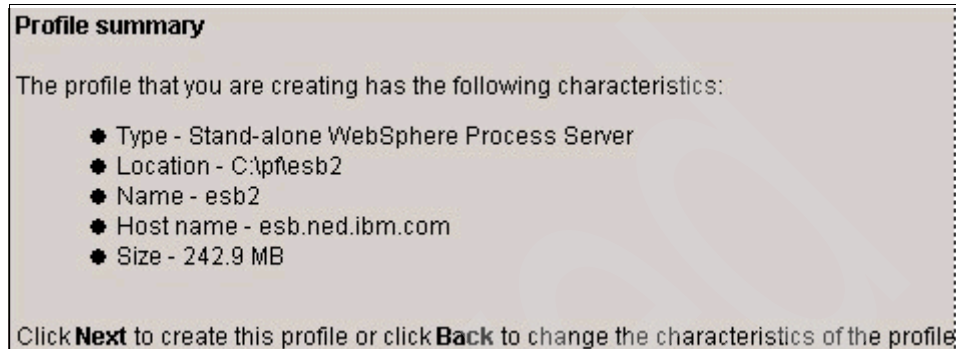


Figure B-43 Profile Wizard: Profile summary

16. It will take several minutes for the profile to be created. If it seems that the wizard is hung, be patient — some pieces of the profile creation take a while, especially on slower machines.

Eventually, you will see the Profile creation is complete page. De-select "Launch the First Steps console."

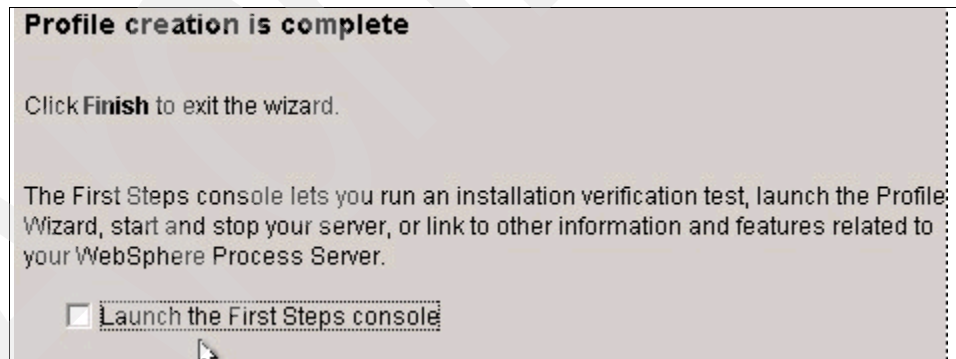


Figure B-44 Profile Wizard: Profile creation is complete

17. Now you can see the new esb2 profile created under the directory you specified, and you can navigate down through the configuration directories to see that the esbNode2 and the server1 were created under this profile.

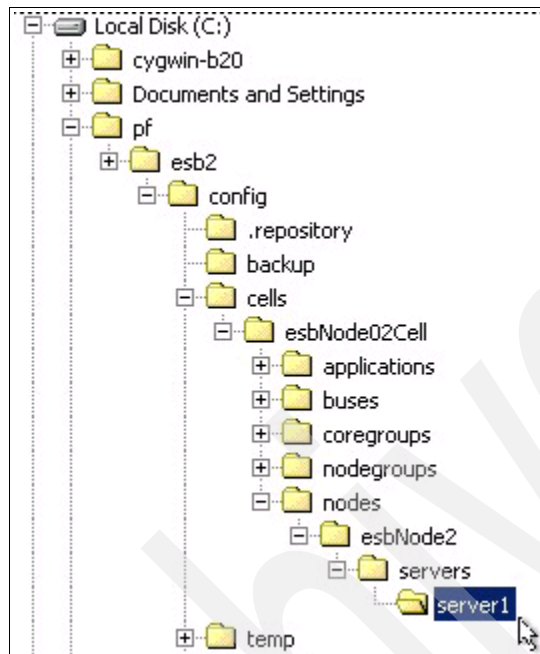


Figure B-45 New esb2 profile directory

Changing WebSphere Integration Developer to use the new profile

Follow these instructions for changing an existing WebSphere ESB Server defined in your WebSphere Integration Developer test environment to use your new profile.

1. In your workspace, make sure the existing WebSphere ESB Server is stopped (Figure B-46).

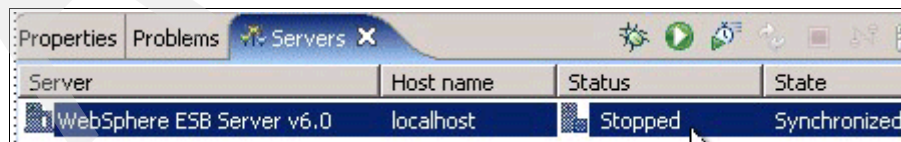
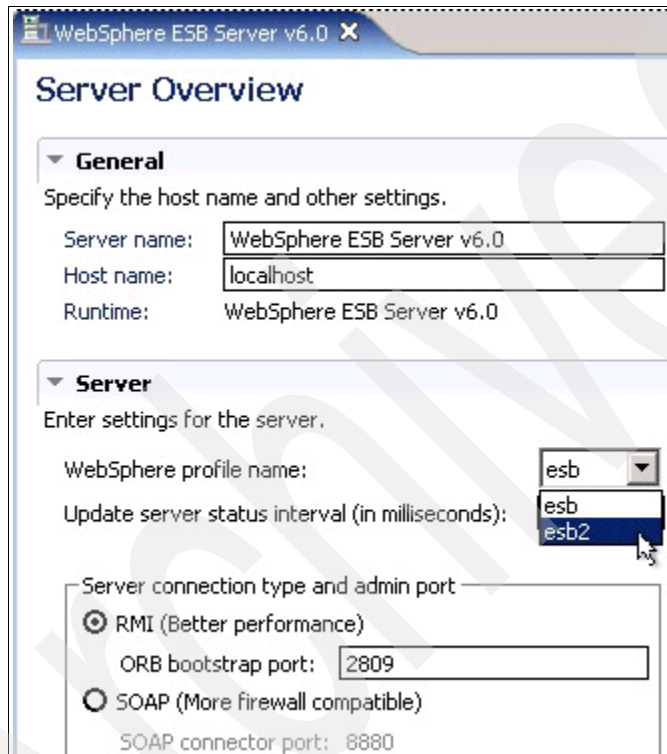


Figure B-46 WebSphere ESB Server stopped

2. Double-click your server to open the Server Overview editor.

3. In the Server Overview:
 - a. For the WebSphere profile name, select your new profile name in the drop-down list (Figure 12-17).
 - b. Under Server connection type and admin port, make sure the ORB bootstrap port is set to the same value that you specified when you created the profile in the profile creation wizard. If you took the default ports, then this value should be 2809.



WebSphere ESB Server v6.0

Server Overview

General

Specify the host name and other settings.

Server name: WebSphere ESB Server v6.0

Host name: localhost

Runtime: WebSphere ESB Server v6.0

Server

Enter settings for the server.

WebSphere profile name: esb

Update server status interval (in milliseconds): esb

Server connection type and admin port

☒ RMI (Better performance)

ORB bootstrap port: 2809

☐ SOAP (More firewall compatible)

SOAP connector port: 8880

Figure 12-17 Existing WebSphere ESB Server: Server Overview

4. Save and close the Server Overview, and the next time you start the server, it will be using your new profile.

Create a new server in the test environment to use the new profile

Follow these instructions if you do not have a WebSphere ESB Server defined in your WebSphere Integration Developer test environment, or you would like to define a another server to use your new profile.

1. In the Servers view, right-click and select **New** → **Server**.

2. In the New Server dialog, select **WebSphere ESB Server**, then **Next** (Figure B-47).

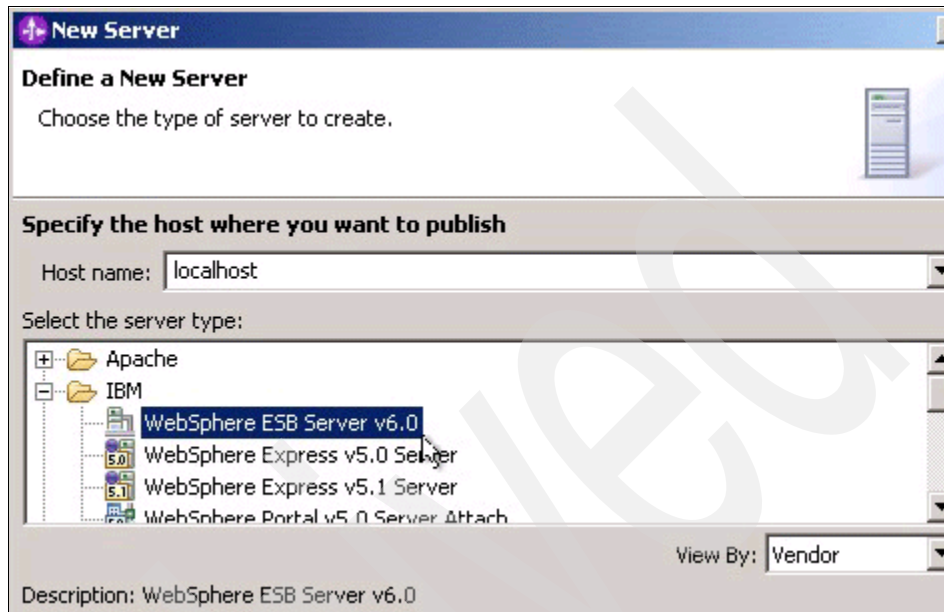


Figure B-47 New Server: Define a New Server

3. On the WebSphere Server Settings page:
 - a. For the WebSphere profile name, select your new profile name in the drop-down list.
 - b. Under Server connection type and admin port, make sure the ORB bootstrap port is set to the same value that you specified when you created the profile in the profile creation wizard. If you took the default ports, then this value should be 2809.

c. Click **Finish** to create the new server.

New Server

WebSphere Server Settings

Input settings for the new WebSphere server.

WebSphere profile name: esb

Server connection type: esb2

☒ RMI (Better performance)

ORB bootstrap port: 2809

☐ SOAP (More firewall compatible)

SOAP connector port: 8880

☒ Run server with resources within the workspace

☐ Security is enabled on this server

Current active authentication settings:

User ID:

Password:

Server name: server1

Figure B-48 New Server: WebSphere Server Settings

Installing WebSphere MQ Explorer as a plug-in

The ITSOMart solution will connect to a WebSphere MQ network. To define the queues, ITSOMart will need to use the WebSphere MQ Explorer.

The prerequisite for support of the WMQ Explorer plug-ins is stated as WebSphere Eclipse Platform Version 3.0.1. The WebSphere MQ V6 installation provides an option for installing the MQ Explorer into an existing Eclipse workbench, but it does not recognize Rational Software Architect or WebSphere Integration Developer as a valid Eclipse workbench.

However, we found that it is possible to merge them after the install by manually copying the plug-ins from WebSphere MQ v6 installation (for example, C:\Program Files\IBM\WebSphere MQ\eclipse\plugins) to <wid_install>/eclipse/plugins.

Important! This is not a supported configuration. If any problems are encountered with the WebSphere MQ Explorer plug-ins in such an environment, you may be required to recreate those problems in a supported environment before raising an issue with IBM service.

If multiple Eclipse-based IBM products are installed on the same machine, you must determine the "true" eclipse/plugins directory. This is because IBM Eclipse-based products like WebSphere Integration Developer and Rational Application Developer do *shell sharing*. This means that the Eclipse/plugins directory under WebSphere Integration Developer's install location may not be the "true" Eclipse/plugins directory where new plug-ins needed to go. For example, if you installed Rational Software Architect and then WebSphere Integration Developer, the true plug-ins directory will be <RSA_install>/eclipse/plugins.

Restart WebSphere Integration Developer for the new plug-ins to be available. If you do not see the WebSphere MQ Explorer perspective in the list of available perspectives, try launching WebSphere Integration Developer with the **-clean** command. For example, type `wid.exe -clean` from the command line.

Installation details

This appendix shows the installation process used to install the products needed for the ITSOMart solution.

- ▶ Installing WebSphere Integration Developer
- ▶ Installing WebSphere Adapters
- ▶ Installing Tivoli Composite Application Manager

For installation information about WebSphere ESB, we recommend that you refer to *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212.

Installing WebSphere Integration Developer

This section takes you through all the steps of installing a default configuration of WebSphere Integration Developer V6.0.1 on a Windows system. Before beginning the installation, ensure that you are logged in as a user with administrative privileges, then perform the following.

Silent installation: Instead of using the installation wizard, it is possible to install WebSphere Integration Developer non-interactively, using a response file. Silent installation could be very useful when setting up multiple development environments.

1. Start the launchpad.exe from disk 1 of the installation CDs. This opens the launchpad shown in Figure C-1. We recommend that you review the readme file and release notes, as they contain late breaking information.

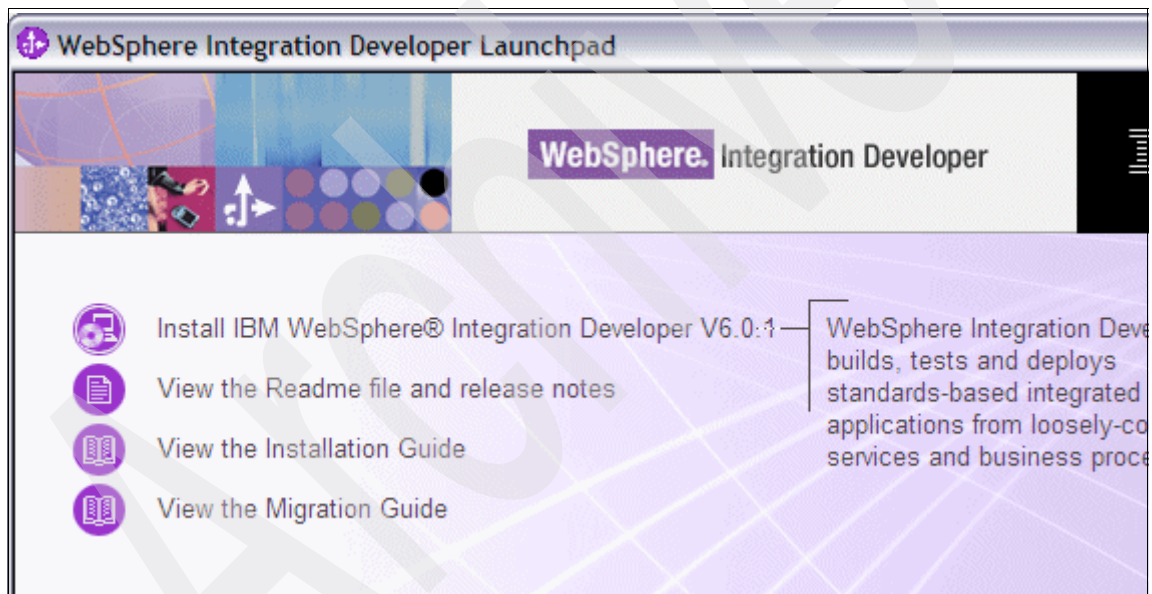


Figure C-1 Install wizard - Launchpad

2. Click **Install IBM WebSphere Integration Developer V6.0.1** to start the installation.
3. The IBM WebSphere Integration Developer V6.0.1 Installer will launch and display a welcome page. Click **Next**.
4. The Software License Agreement will be displayed. Accept the agreement, then click **Next**.

5. Specify an installation directory, as shown in Figure C-2, then click **Next**.

Note: We recommend that you change the installation directory to a short directory name to avoid problems with the Windows limit on path names longer than 256 characters.

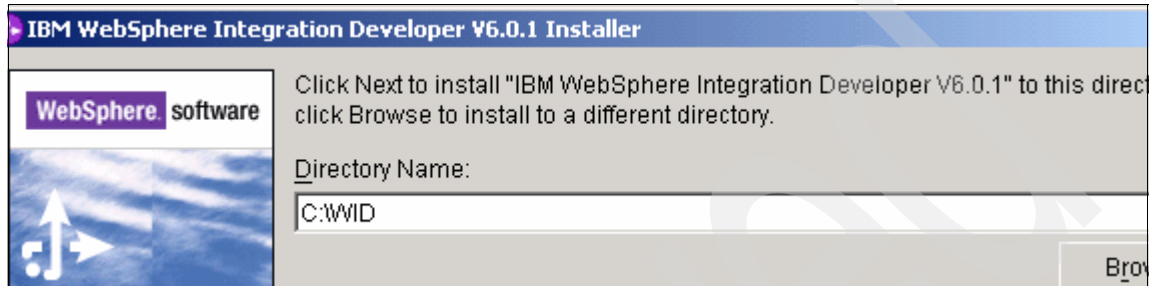


Figure C-2 Install wizard - Install directory

6. The installation of WebSphere Integration Developer is divided into two parts. The first part is to install the Integrated Development Environment. This part is required. The second part is the Integrated Test Environment, which allows you to deploy, run, and test artifacts you have built in the Integrated Development Environment. This part is optional but strongly recommended if you want to unit test your artifacts. Click **Integrated Test Environment** and click **Next** (Figure C-3).

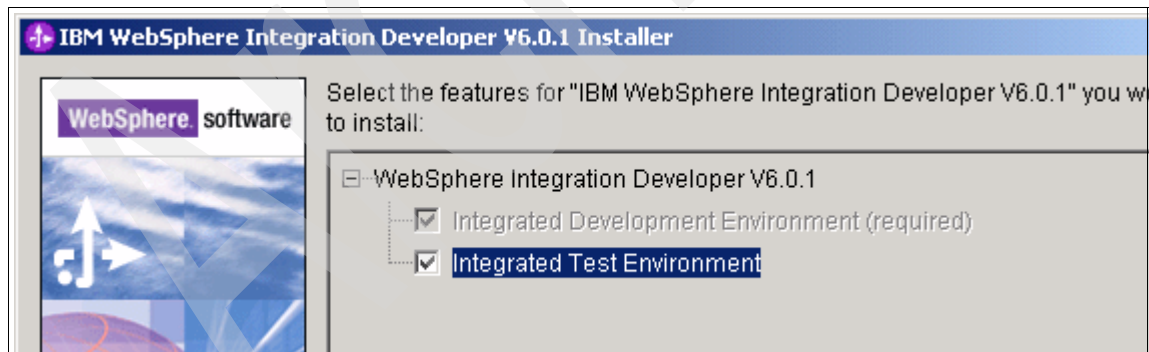


Figure C-3 Install wizard - Integrated Test Environment

7. The Integrated Test Environment of WebSphere Integration Developer offers two server types: WebSphere Process Server and WebSphere Enterprise Service Bus. We recommend installing both. Select **WebSphere Enterprise**

Service Bus in addition to WebSphere Process Server and click **Next** (Figure C-4).



Figure C-4 Install wizard - Select Integration Test Environment

8. The next screen shows a summary of the installation and shows the disk space required depending on the options selected (Figure C-5). Click **Next** to start the installation.

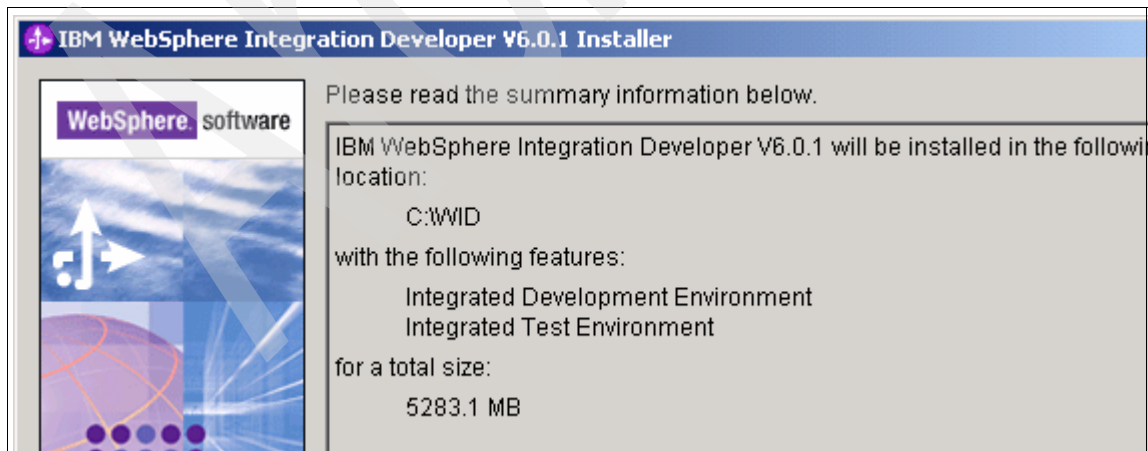


Figure C-5 Install wizard - Summary and disk space required

9. At the end of the installation, a summary is shown along with the status for each of the components, as shown in Figure C-6. Click **Next**.

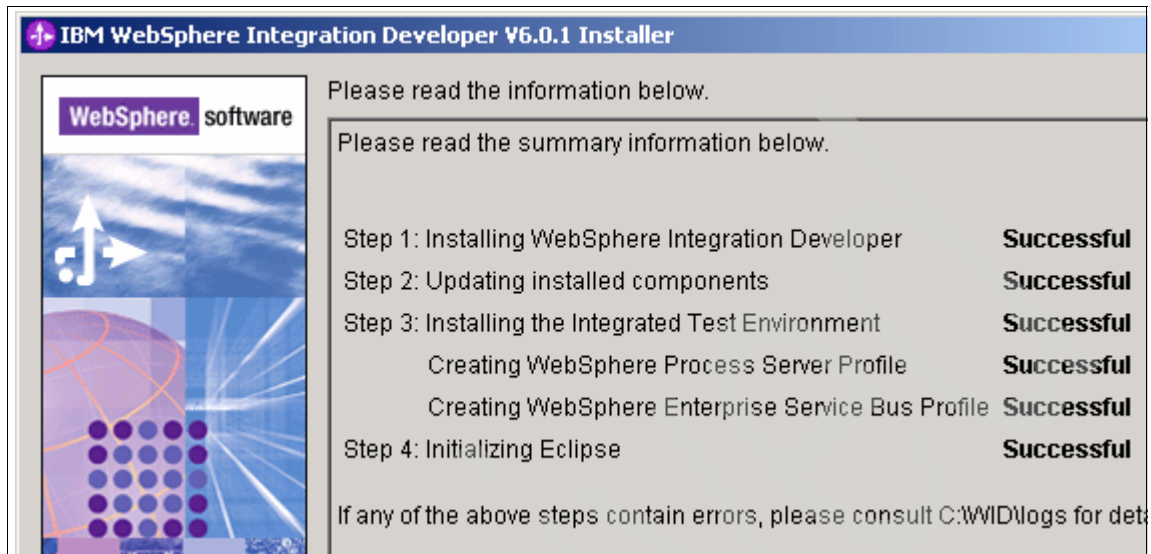


Figure C-6 Install wizard - Complete install summary

10. You will be asked if you wish to view the readme file containing late-breaking information. Click **Next**, then **Next** again.
11. Finally you will be asked if you wish to launch the Rational Product Updater. You can use this tool to apply interim fixes and product updates. “Using Rational Product Updater” on page 662 provides more information about this tool. We recommend that you select **Launch Rational Product Updater** (Figure C-7). Click **Finish**.

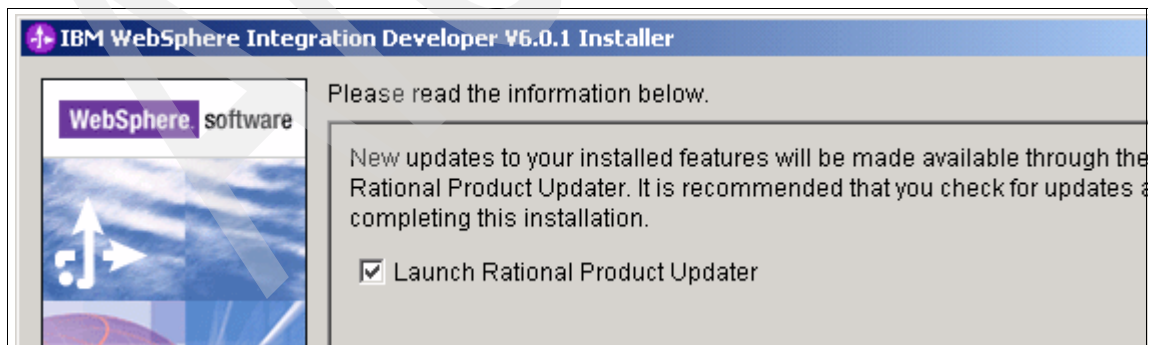


Figure C-7 Install wizard - Launch product updater

12. The directory structure from a WebSphere Integration Developer installation is as shown in Figure C-8.

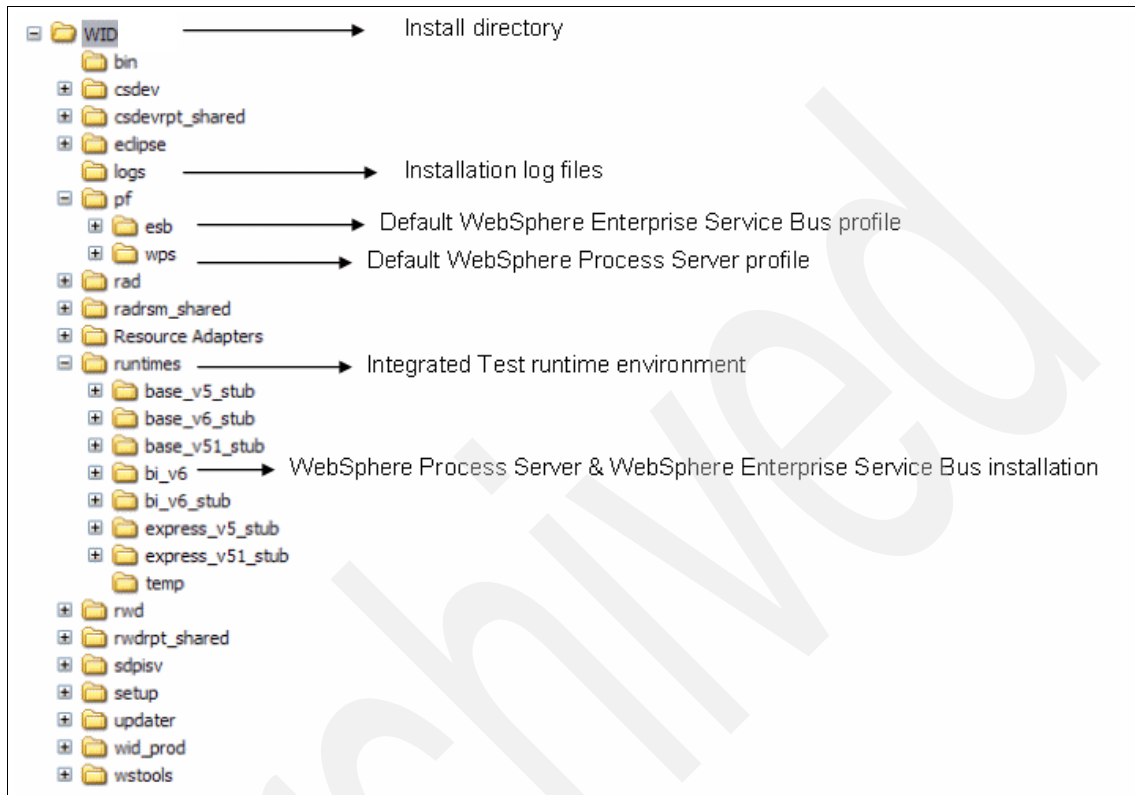


Figure C-8 Directory structure of WebSphere Integration Developer installation

Note: You can create stand-alone server profiles using the profile creation wizard located in:

- On Windows:
`<WID_INSTALL>\runtimes\bi_v6\bin\ProfileCreator_wbi\esbpcatWindows.exe`
- Or on Linux:
`<WID_INSTALL>\runtimes\bi_v6\bin\ProfileCreator_wbi\esbpcatLinux.bin`

Using Rational Product Updater

Rational Product Updater is the tool provided to install maintenance updates for WebSphere Integration Developer as well as other products based on Rational

Software Development Platform. This tool accesses the update server on the internet and locates and installs product updates as well as optional new features. After completing a successful install, we strongly recommend checking for product updates so that you can install fixes and avoid encountering known problems.

Note: It is possible to change the update site preference within Rational Product Updater so updates can be installed from a local or a network drive rather than from the Internet update server. You must download the updates to a local driver and change the update site preference.

The recommended updates for WebSphere Integration Developer can be downloaded from:

<http://www-1.ibm.com/support/docview.wss?rs=2308&uid=swg27006685>

Further information about changing the update site preference can be found at:

<http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.orca.updater.doc/topics/tupdatesites.html>

After completing WebSphere Integration Developer install, we installed updates using Rational Product Updater. We have downloaded the updates and used the update policy shipped with the fix. Below is each step for installing an update using Rational Product Updater:

1. If the Rational Product Updater is not running, launch it from **Start → Programs → IBM WebSphere → Integration Developer V6.0.1 → Rational Product Updater**. This opens the Rational Software Development Platform Product Updates screen, as shown in Figure C-9.

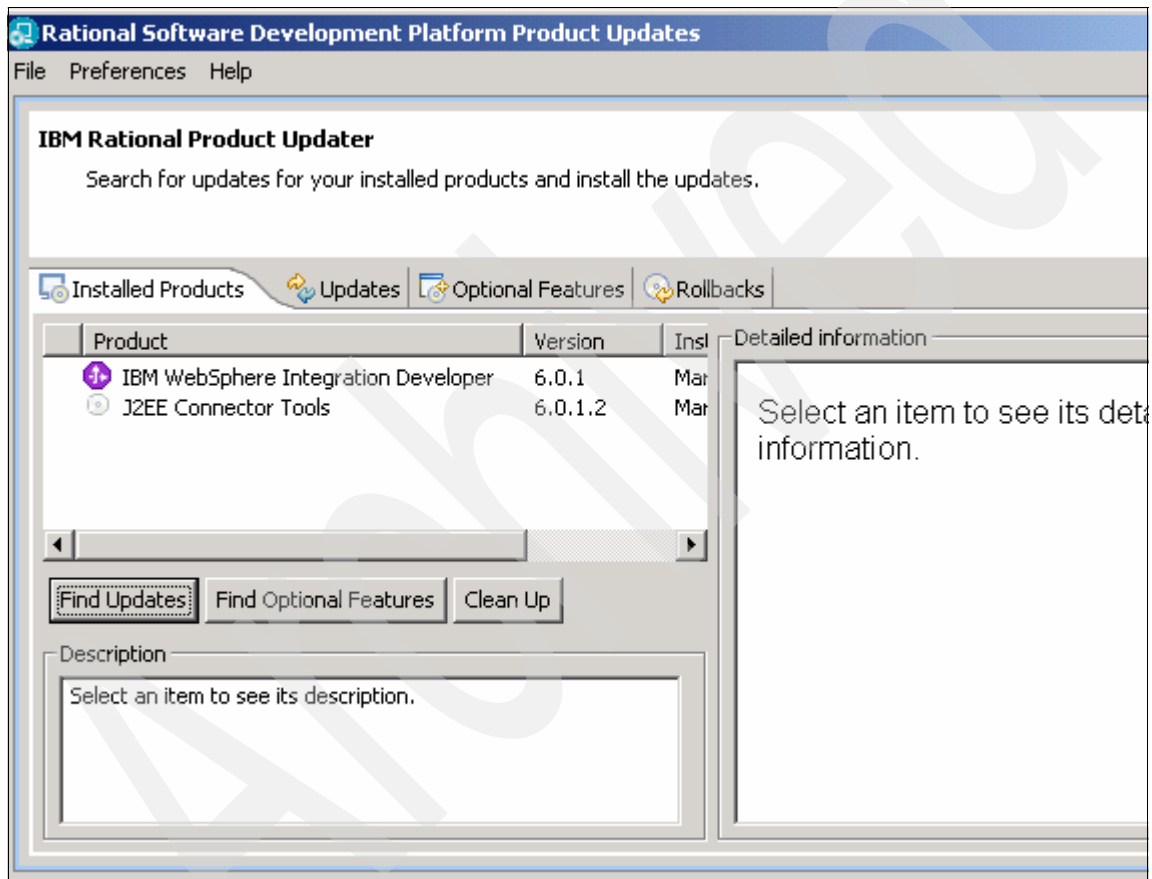


Figure C-9 Rational Product Updater

2. If you wish to provide a local update policy file instead of downloading one using the default Internet update server, perform the following:
 - a. Click **Preferences → Update Sites**.

- b. Use **Browse** to locate your policy name, as shown in Figure C-10.
- c. Click **OK**.

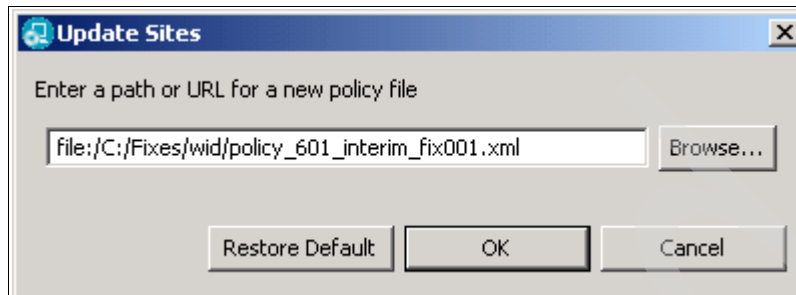


Figure C-10 Provide your update policy

3. Under the Installed Products tab, highlight **IBM WebSphere Integration Developer** and click **Find Updates**.

4. Once the updates have been located (either locally or using the Internet), the Product Updater will switch to the Updates view (Figure C-11).

Highlighting each update shown in the update window will provide a brief description of that update in the Description box, and detailed information about that update in the Detailed information box. You can select or deselect an update to install using the check box. After completing selections, click the **Install Updates** button to start the install.

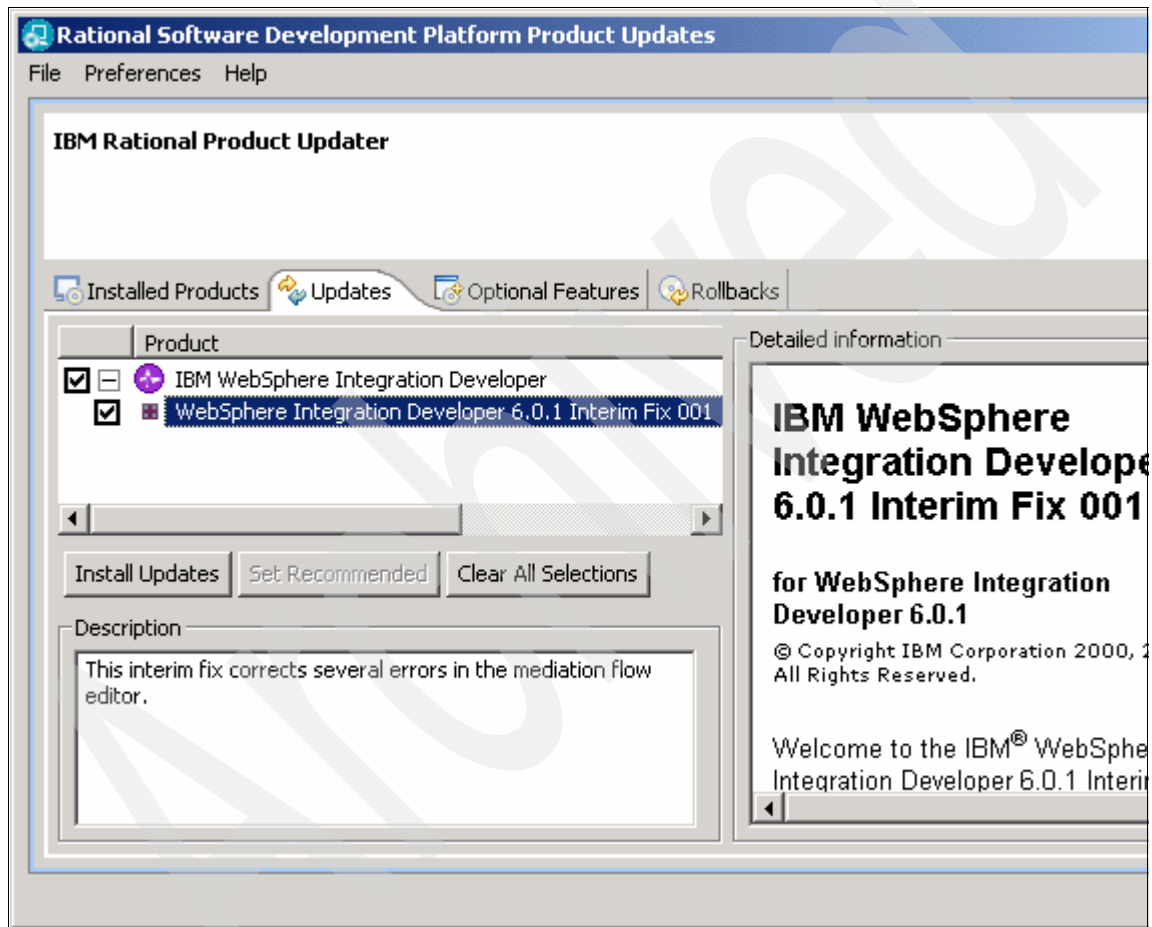


Figure C-11 Selecting the updates to install

5. After installation is complete, the installed updates will show on the Installed Products pane, as shown in Figure C-12 on page 667.

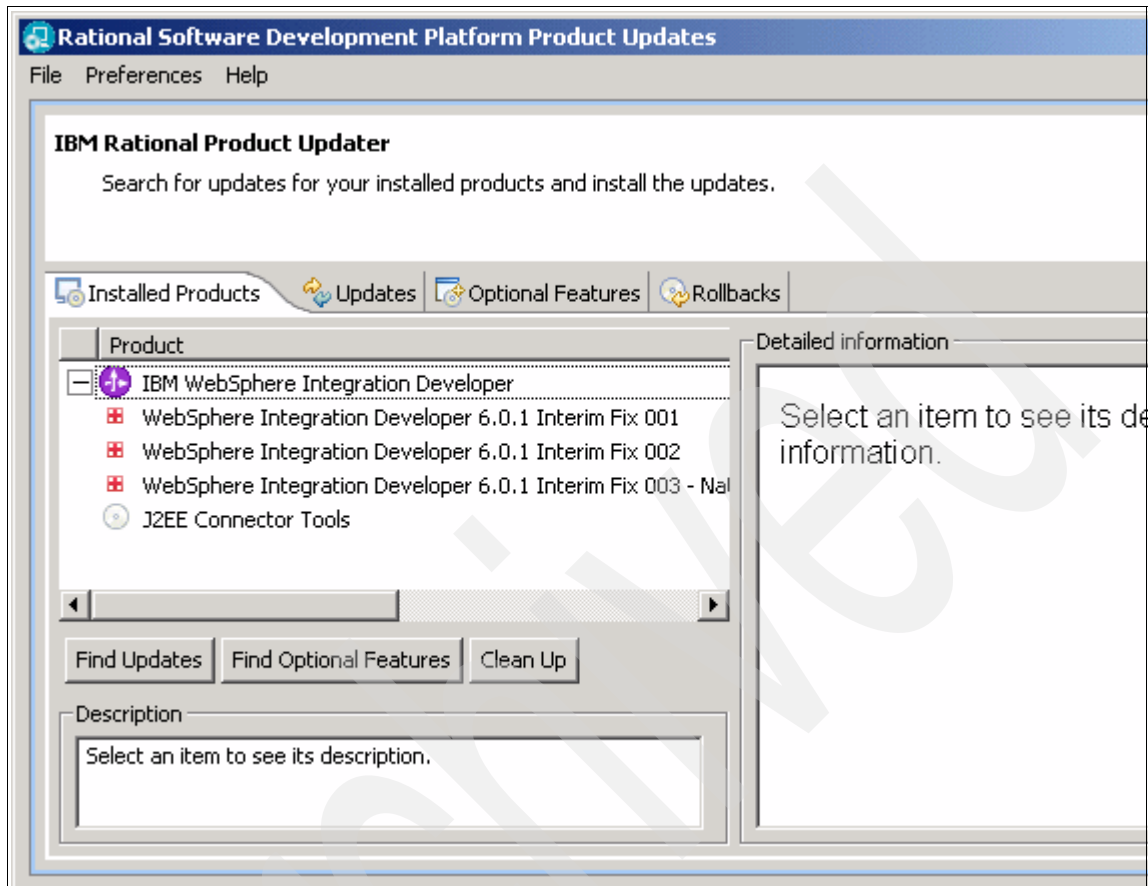


Figure C-12 Install updates

Installing WebSphere Adapters

This topic describes the steps to install and configure the IBM WebSphere Adapter for Siebel Business Applications V6.0.0 and IBM WebSphere Adapter for Flat Files.

The Adapter installer is available for the Windows and Linux platforms (the same as the WebSphere Integration Developer platforms). To use the adapter on other platforms, install on Windows or Linux, build your application using WebSphere Integration Developer, and then export it to an EAR file. Now you can install this EAR to any of the platforms supported by WebSphere Process Server.

IBM WebSphere Adapter for Siebel Business Applications

To install the adapter, do the following on the WebSphere Integration Developer system:

1. Click the **launchpad_win.exe/launchpad.sh**.
2. Select the language for the launchpad. Click **OK**.
3. When the Launchpad window opens, click **Install Product** (Figure C-13).

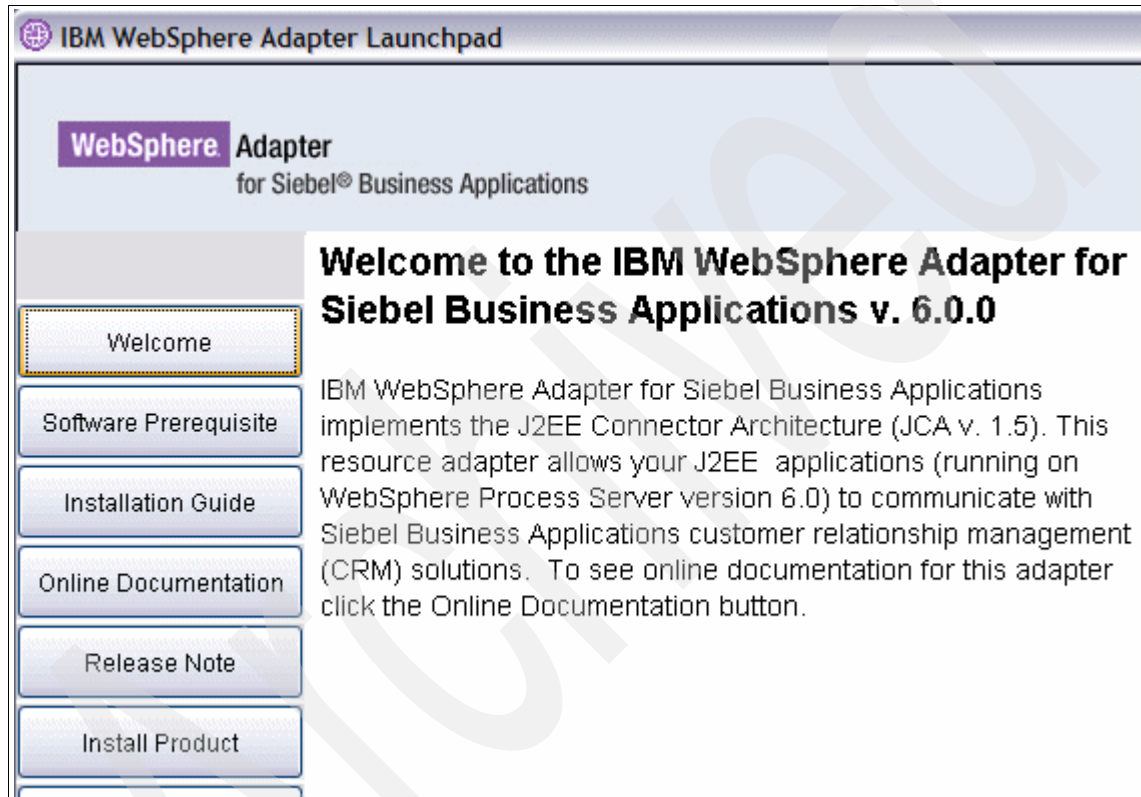


Figure C-13 WebSphere Adapter launchpad

4. Select the language for the adapter installation and click **OK**.
5. The next page is the adapter installer welcome page. Click **Next**.
6. Accept the license agreement and click **Next**.

7. Select the folder where you want to install the adapter. Click **Next**.

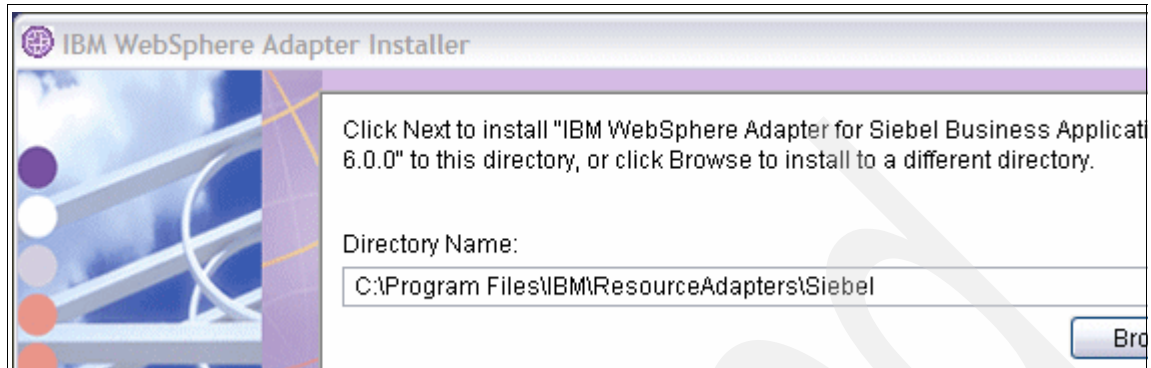


Figure C-14 Install location

8. Click **Next** on the summary page (Figure C-15).

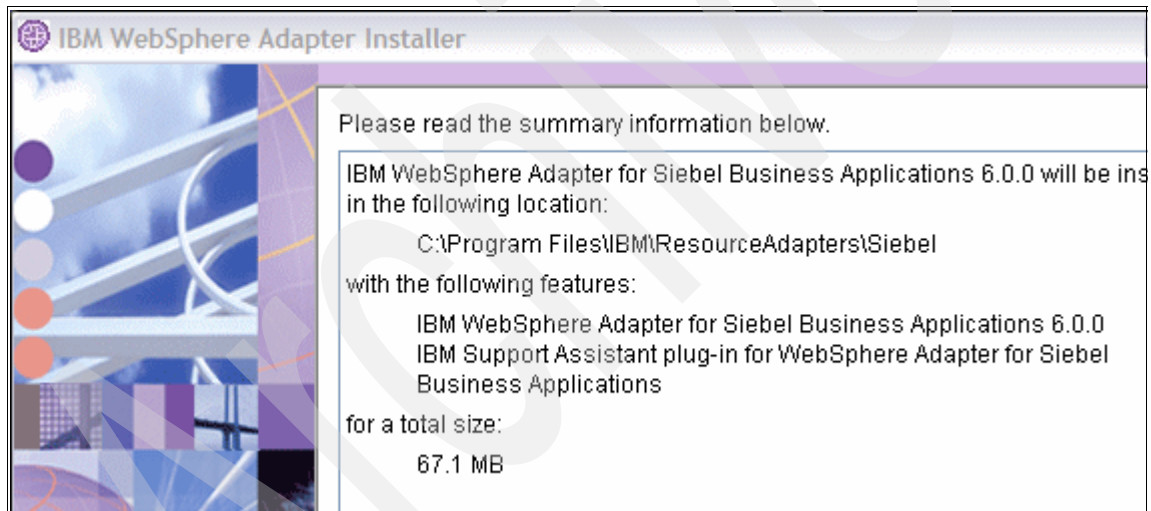


Figure C-15 Summary page

9. Click **Finish**. The installation will complete and you should see the message that indicates that the install was successful.

IBM WebSphere Adapter for Flat Files

To install the adapter, do the following on the WebSphere Integration Developer system:

1. Click the **launchpad_win.exe/launchpad.sh**.

2. Select the language for the launchpad. Click **OK**.
3. When the Launchpad window opens, click **Install Product**. (Figure C-13 on page 668).

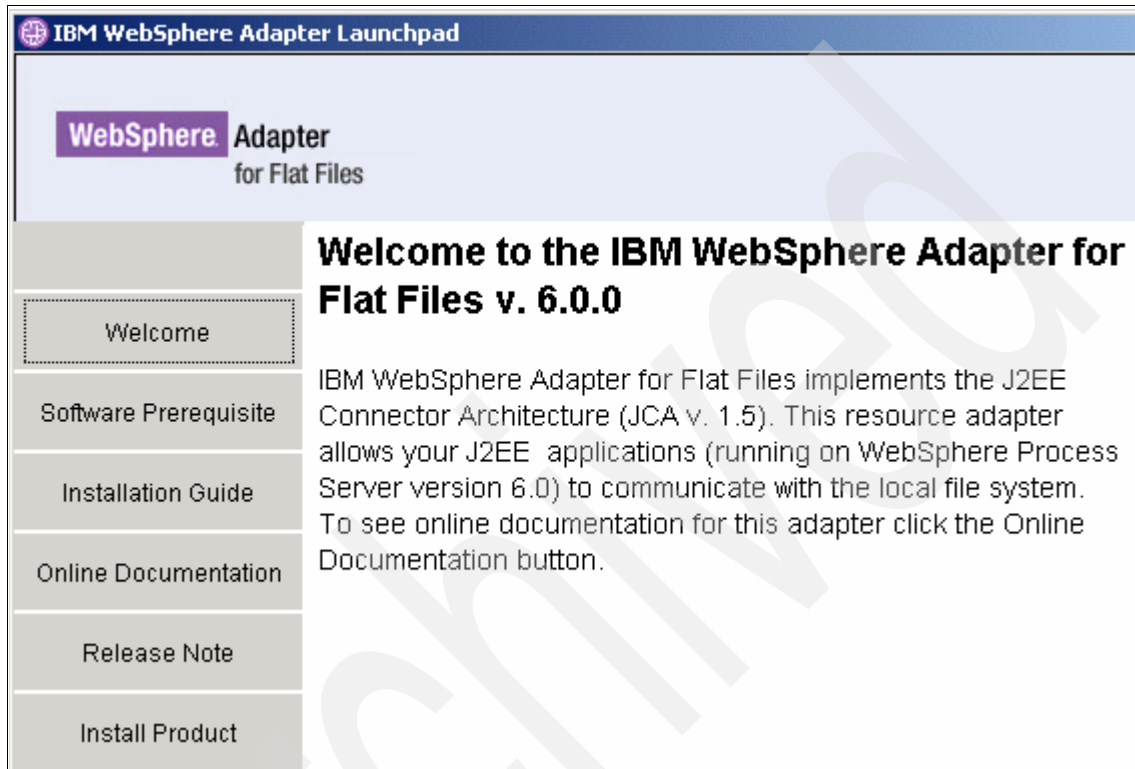


Figure C-16 WebSphere Adapter launchpad

4. Select the language for the adapter installation and click **OK**.
5. The next page is the adapter installer welcome page. Click **Next**.
6. Accept the license agreement and click **Next**.
7. Select the folder where you want to install the adapter.

The default (on Windows) is:

C:\Program Files\IBM\ResourceAdapters\FlatFiles

Click **Next**.

8. Click **Next** on the summary page (Figure C-15 on page 669).

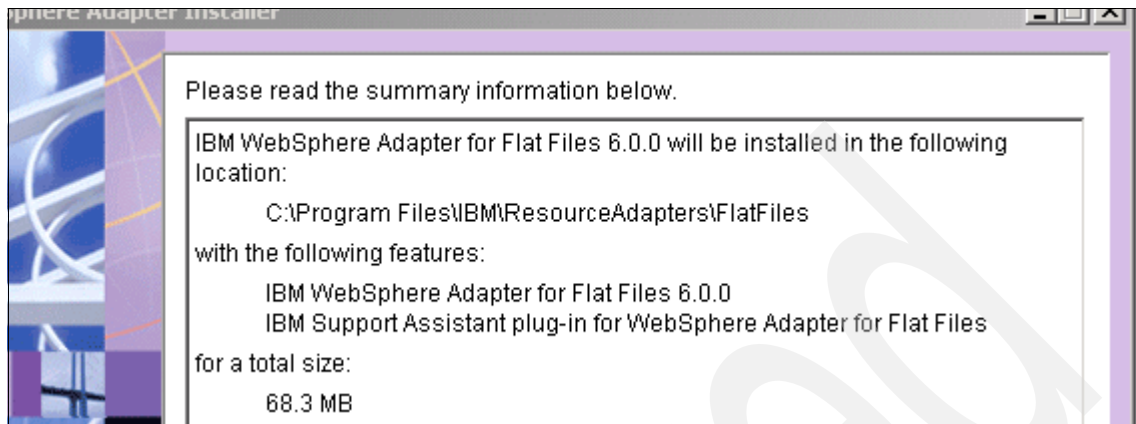


Figure C-17 Summary page

9. Click **Finish**. The installation will complete and you should see the message that indicates the install was successful.

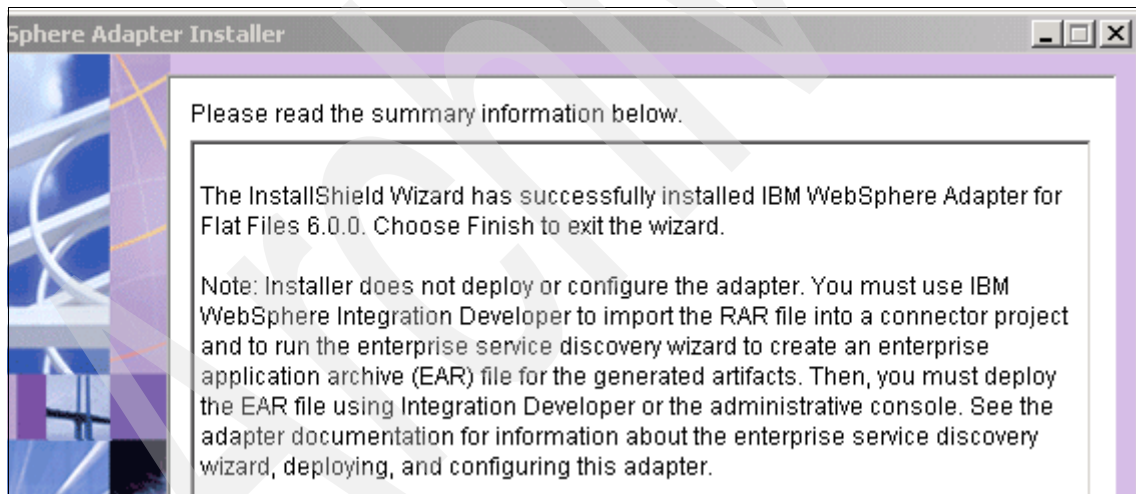


Figure 12-18 Installation complete

Installing Tivoli Composite Application Manager

The Tivoli environment for this solution required the following:

- IBM DB2 Universal Database installation

- ▶ IBM Tivoli Monitoring installation, including:
 - Tivoli Enterprise Monitoring Agent Framework
 - Tivoli Enterprise Monitoring Server
 - Tivoli Enterprise Portal Server
 - Tivoli Enterprise Portal Desktop Client
- ▶ IBM Tivoli Composite Application Manager for SOA installation, including:
 - ITCAM for SOA Application Support installation
 - ITCAM for SOA Monitoring Agent installation and configuration
- ▶ Web Services Navigator installation

Tivoli product documentation: You can find documentation about the installation and use of the Tivoli Monitoring products at:

<http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?toc=/com.ibm.itm.doc/toc.xml>

IBM DB2 Universal Database installation

This section provides details about installing IBM DB2. IBM Tivoli Enterprise Monitoring requires a database for runtime operations as well as historical data warehousing. For information about other supported database products see the Tivoli product documentation.

1. Log on to the system as a user with the correct installation privileges (root in UNIX and Linux operating systems, a user with Administrator privileges on Microsoft Windows operating systems).
2. Launch the IBM DB2 UDB installation executable as appropriate for the target operating system (setup.exe on Windows).
3. Click **Install Product** in the DB2 Setup Launchpad.

4. Accept the default product selection in the Select the product you would like to install screen and click **Next**, as shown in Figure C-18.

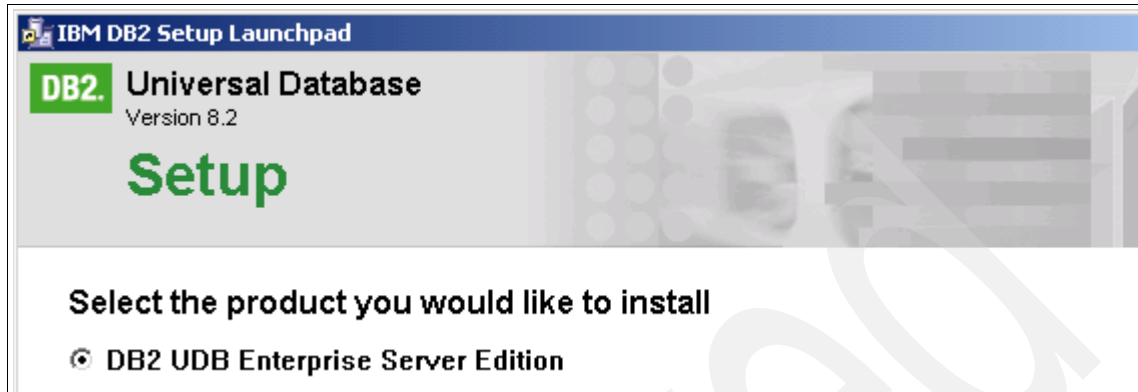


Figure C-18 DB2 setup product selection

5. The next screen is the welcome screen, which confirms the product that the installer is going to set up. Click **Next**.
6. Carefully read the license agreement and select **I accept the terms in the license agreement** and click **Next**.

7. In the Select the installation type screen select **Typical** as the installation type and click **Next**. Figure C-19 shows the installation type selection screen.

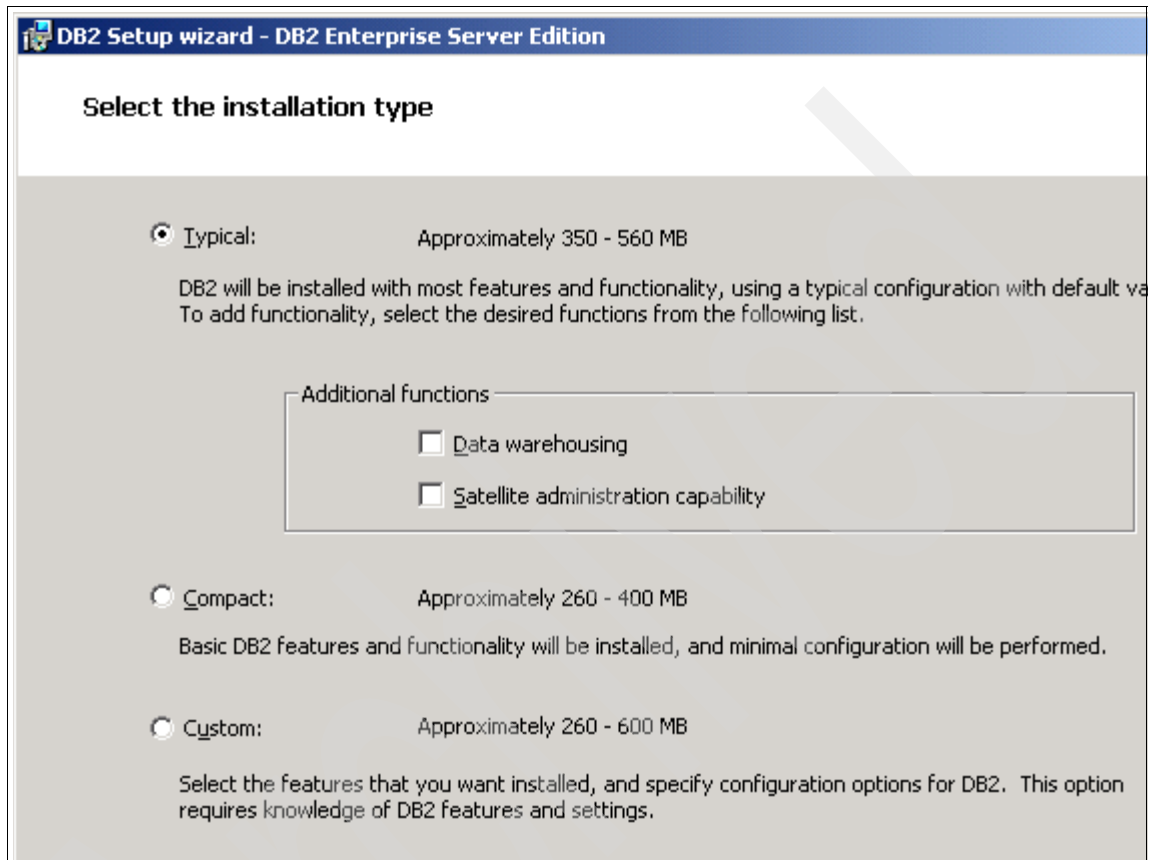


Figure C-19 DB2 Installation type

8. Leave the default values in the Select the installation action screen and click **Next**.

9. In the next screen, select the directory where DB2 should be installed. The default installation directory for Microsoft Windows is C:\PROGRAM FILES\IBM\SQLLIB. For optimal performance, DB2 should be installed to a disk drive other than the drive that contains the operating system. Click **Next**. Figure C-20 shows the installation directory.

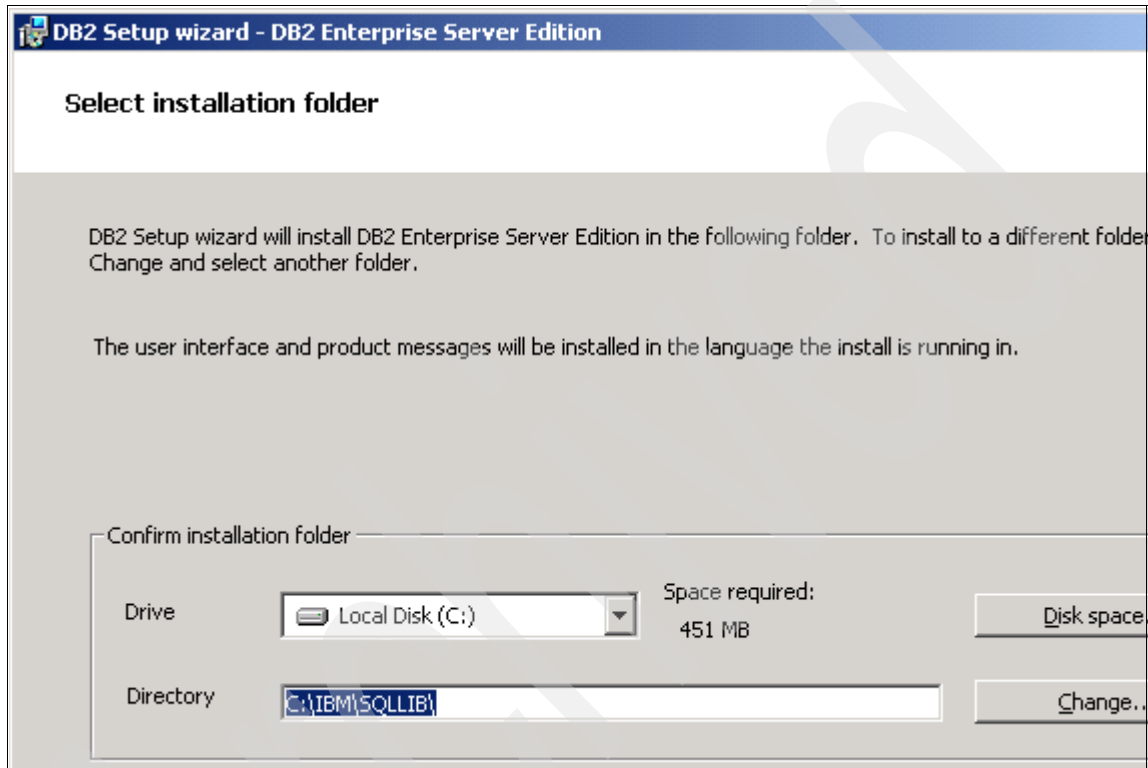


Figure C-20 DB2 Installation location

10. Enter the user name and password for the user that the DB2 Administration Server will use to log on to the operating system. The default user name is db2admin. Enter db2admin as the password and do so once more as the

confirmation password. Ensure that “use the same username and password for the remaining DB2 services” is checked, as shown in Figure C-21, and click **Next**.

DB2 Setup wizard - DB2 Enterprise Server Edition

Set user information for the DB2 Administration Server

Enter the user name and password that the DB2 Administration Server (DAS) will use to log on to your system. You can use a local user or a domain user.

User information

Domain: [dropdown menu]

User name: db2admin

Password: *****

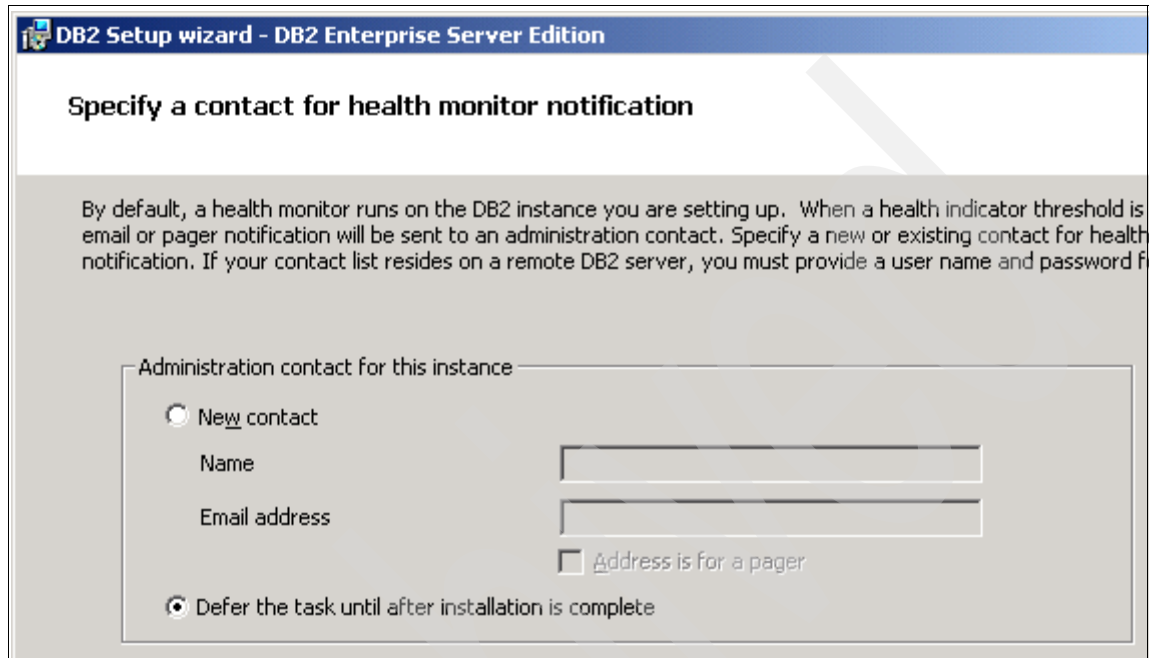
Confirm password: *****

☒ Use the same user name and password for the remaining DB2 services

Figure C-21 DB2 user name setup

11. Leave the default values in the Setup the administration contact list screen and click **Next**.
12. You may receive a warning that the SMTP notification server has not been configured. This is fine for now. Click **OK** to continue.
13. Leave the default values in the Configure DB2 instances screen and click **Next**.
14. Leave the default values in the Prepare DB2 tools catalog screen and click **Next**.
15. In the next screen, you will be asked to enter contact information for the health monitor. DB2 uses this information to send alerts when certain

database health thresholds have been exceeded. For the moment, select **Defer the task until after installation is complete** and click **Next**, as shown in Figure C-22.



DB2 Setup wizard - DB2 Enterprise Server Edition

Specify a contact for health monitor notification

By default, a health monitor runs on the DB2 instance you are setting up. When a health indicator threshold is exceeded, email or pager notification will be sent to an administration contact. Specify a new or existing contact for health monitor notification. If your contact list resides on a remote DB2 server, you must provide a user name and password for that server.

Administration contact for this instance

☐ New contact

Name

Email address

☐ Address is for a pager

☒ Defer the task until after installation is complete

Figure C-22 Specify administrative contact

16. The next screen shows a summary of the installation process. Click **Install** to proceed with installation.
17. Click **Finish** once the setup is complete.
18. The next screen is the DB2 First Steps Launchpad screen. You can perform initial post-installation tasks such as creating a sample database. Click **Exit First Steps** to close this window.
19. If you are installing on Microsoft Windows, you may need to restart the computer to complete post-installation cleanup.
20. IBM DB2 is now installed and ready for use by Tivoli Monitoring.

IBM Tivoli Monitoring installation

The following sections provide detailed information about installing a Hub TEMS and performing the initial configuration.

1. Log on to the system as a user with the correct installation privileges (root in UNIX and Linux operating systems, a user with administrator privileges on Microsoft Windows operating systems).
2. Launch the IBM Tivoli Monitoring installation executable as appropriate for the target operating system (setup.exe on Windows).

Note: The setup.exe is located within the Windows directory of the Tivoli Monitoring installation CD.

3. Click **Accept** to accept the license agreement.
4. Choose the directory in which you want to install the product. The default directory in Windows is C:\IBM\ITM. We strongly recommend that you install IBM Tivoli Monitoring 6.1 in a different drive from one that holds the operating system. Click **Next**.
5. The next window asks you to type a 32-bit encryption key. You can use the default key.

Notes: This encryption key is used to establish a secure connection (using SSL protocol) between the Hub TEMS and the other components of the IBM Tivoli Monitoring 6.1 environment as the Remote TEMS connected to the hub. Do not use any of the following characters in your key:

- ▶ =
- ▶ ,
- ▶ |

Ensure that you document the value you use for the key. Use this key during the installation of any components that communicate with this monitoring server.

6. Click **Next**, then **OK** to confirm the encryption key.

7. Select the components that you want to install. Figure C-23 shows the components for the sample installation. Click **Next**.



The screenshot shows a list of components for selection, each with a checked checkbox. The components are organized into four main sections, each starting with a checked checkbox followed by a list of sub-components, also with checked checkboxes.

- ☒ Tivoli Enterprise Monitoring Agents
 - ☒ Tivoli Enterprise Monitoring Agent Framework
 - ☒ Monitoring Agent for Windows OS
- ☒ Tivoli Enterprise Monitoring Server
 - ☒ Tivoli Enterprise Monitoring Server
 - ☒ Windows OS Support
- ☒ Tivoli Enterprise Portal Server
 - ☒ Tivoli Enterprise Monitoring Portal Server Framework
 - ☒ Windows OS Support
- ☒ Tivoli Enterprise Portal Desktop Client
 - ☒ Tivoli Enterprise Portal Desktop Client
 - ☒ Windows OS Support

Figure C-23 ITCAM component selection

8. Accept the default in the agent deployment selection screen and click **Next**.
9. Select a program folder and click **Next**. The default program folder name is IBM Tivoli Monitoring.

10. Review the installation summary details. This summary identifies what you are installing and where you have chosen to install it. Click **Next** to begin the installation of components, as shown in Figure C-24.



Figure C-24 Installation summary details

11. After the components are installed, a configuration window (Figure C-25) will display a list of components that can be configured. Select all the components and click **Next**.

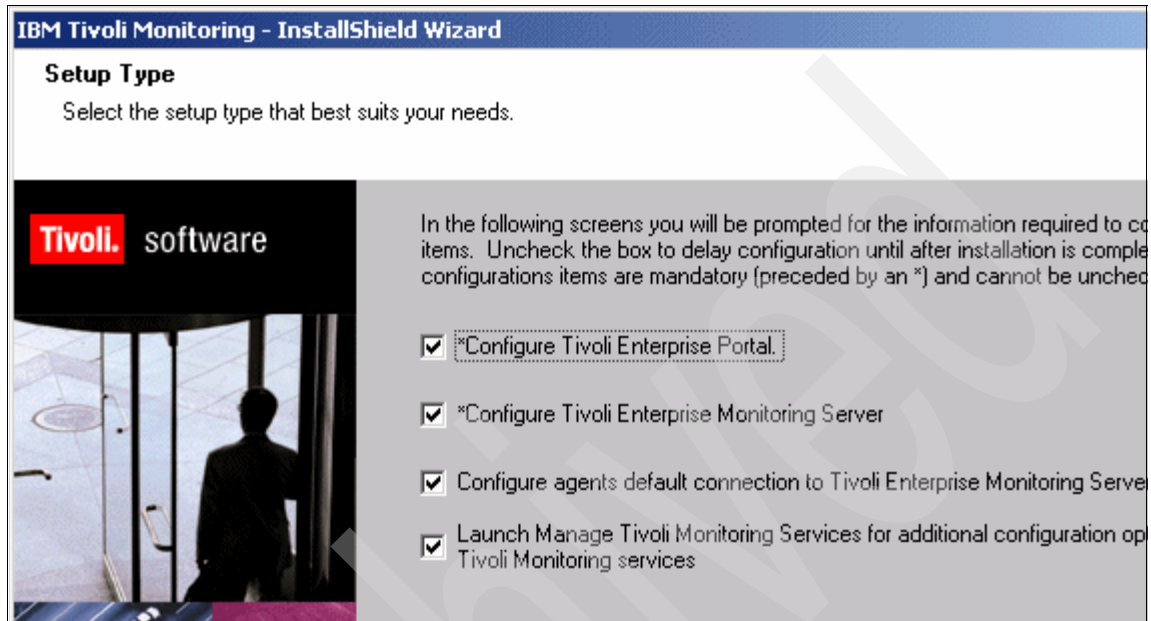


Figure C-25 Setup configuration

12. Enter the host name of the Tivoli Enterprise Monitoring Server. The default value (the host name of the current machine) should be correct. Click **Next**.
13. The next screen is the database configuration screen for the Tivoli Enterprise Portal. Enter the password for the DB2 administrative user (selected during DB2 installation) and enter a password for the Tivoli Enterprise Portal Server user (leave the default value for the user name), as shown in Figure C-26.

TEPS Data Source Config Parameters - DB2

Data source name:

OK

Cancel

Please enter your Database Administrator ID (up to 8 characters) and password below:

Admin User ID:

Admin Password:

Please accept the default Database User ID or enter your own TEPS Database User ID (up to 8 characters) and password:

Database User ID:

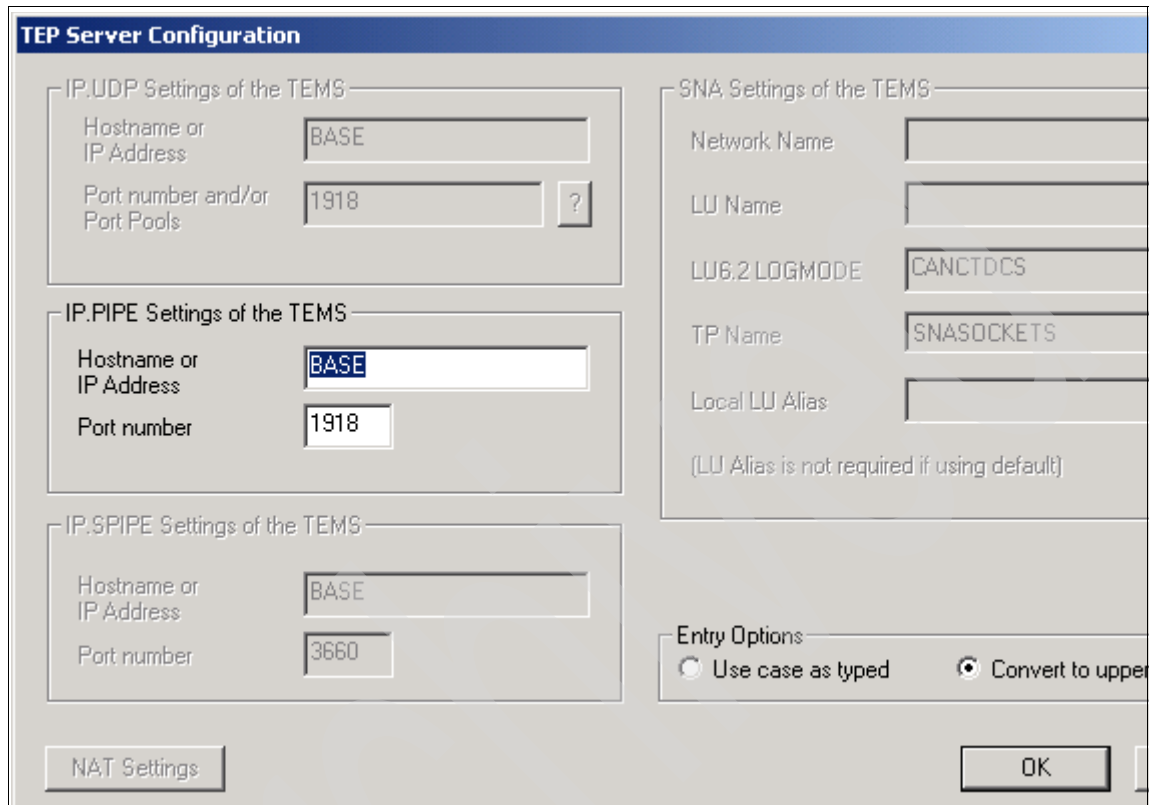
Database Password:

Reenter Password:

Figure C-26 Configure data warehousing

14. A notification window will appear. Click **OK**.
15. Leave the default users in the warehouse ID and password for Tivoli Enterprise Portal Server and click **Next**
16. Leave the default communication protocol in the TEP Server Configuration screen and click **OK**.

17. Leave the default settings in the protocol configuration screen (Figure C-27) and click **OK**.



The image shows a 'TEP Server Configuration' dialog box with a blue title bar. It contains several sections for configuring the TEP server. On the left, there are three sections for IP settings: 'IP.UDP Settings of the TEMS', 'IP.PIPE Settings of the TEMS', and 'IP.SPIPE Settings of the TEMS'. Each section has fields for 'Hostname or IP Address' and 'Port number and/or Port Pools'. The 'IP.UDP' section has 'BASE' and '1918' with a '?' button. The 'IP.PIPE' section has 'BASE' and '1918'. The 'IP.SPIPE' section has 'BASE' and '3660'. On the right, there is a 'SNA Settings of the TEMS' section with fields for 'Network Name', 'LU Name', 'LU6.2 LOGMODE' (set to 'CANCTDCS'), 'TP Name' (set to 'SNASOCKETS'), and 'Local LU Alias'. Below this is a note: '(LU Alias is not required if using default)'. At the bottom right, there is an 'Entry Options' section with two radio buttons: 'Use case as typed' and 'Convert to upper' (which is selected). At the bottom left is a 'NAT Settings' button, and at the bottom right is an 'OK' button.

Section	Field	Value
IP.UDP Settings of the TEMS	Hostname or IP Address	BASE
	Port number and/or Port Pools	1918 ?
IP.PIPE Settings of the TEMS	Hostname or IP Address	BASE
	Port number	1918
IP.SPIPE Settings of the TEMS	Hostname or IP Address	BASE
	Port number	3660
SNA Settings of the TEMS	Network Name	
	LU Name	
	LU6.2 LOGMODE	CANCTDCS
	TP Name	SNASOCKETS
	Local LU Alias	
Entry Options		<input type="radio"/> Use case as typed <input checked="" type="radio"/> Convert to upper

Figure C-27 Monitoring server configuration

18. If you are asked to reconfigure the Warehouse Proxy information for the Enterprise Portal Server, click **No**.

19. Figure C-28 shows the different options of monitoring the type of Tivoli Enterprise Monitoring Server that can be selected. Select **Hub**.

The screenshot shows the 'Tivoli Enterprise Monitoring Server Configuration' dialog box. It has a title bar with a close button. The main area is divided into several sections. The 'TEMS Type' section has two radio buttons: 'Hub' (selected) and 'Remote'. The 'TEMS Name' section has a text field containing 'HUB_BASE'. The 'Protocol for this TEMS' section has three checkboxes for 'Protocol 1', 'Protocol 2', and 'Protocol 3'. 'Protocol 1' is checked and its dropdown menu shows 'IP.PIPE'. The other two protocols are unchecked. To the right of this section is a 'Configure Hot Standby TEMS' section with its own three protocol checkboxes and dropdowns, all of which are unchecked. Above the 'Configure Hot Standby TEMS' section are five unchecked checkboxes: 'Configuration Auditing', 'Security: Validate User', 'Address Translation', 'TEC Event Integration Facility', and 'Disable Workflow Policy/Tivoli Emitter Agent Event Forwarding'. At the bottom right are 'OK' and 'Cancel' buttons.

Figure C-28 Tivoli Enterprise Monitoring Server configuration

20. Verify that the name of this monitoring server is correct in the TEMS field. If it is not, change it. The default name is `HUB_HOSTNAME`.
21. Identify the communications protocol for the monitoring server. Possible choices are `IP.UDP`, `IP.PIPE`, `IP.SPIPE`, and `SNA`. If multiple methods for communication are specified, the TEMS server will use alternate communication protocols when the primary protocol fails. For example, if the method you have identified as Protocol 1 fails, Protocol 2 is used.

Note: IP.PIPE protocol uses TCP. Thus a permanent connection is established between the TEMS and the remote servers. This could have an impact on the server performance because of the number of RPCs that it needs to handle. If using UDP will not cause security breaches in your environment, we recommend that you set up the first protocol as IP.UDP. Otherwise use IP.PIPE.

If a firewall is between your TEMS and your agents, you cannot use IP.UDP.

22. Accept the default values in the protocol configuration window and click **OK**.

23. Specify the location of the monitoring server. There are two options:

- On this computer
- On a different computer

Chose the first option and click **OK**.

24. Because the monitoring server is not currently running, it will start automatically before the process begins. Click **OK** when you see Figure C-29.



Figure C-29 Monitoring server start confirmation windows

25. Select the data that you want to add to the monitoring server. By default, all available application support is selected. Leave all components selected so that they can be seeded. Click **OK**.

Note: Seeding adds product-specific data from the monitored resources to the monitoring server. For Windows, you can seed the monitoring server both during install and through Manage Tivoli Monitoring Services.

During this process, fields are created in the TEMS database (a flat file/Btrieve database, not the relational database installed for the TEPS) for the agents you have chosen. This enables the TEMS to work with the data from these agents. The same goes for the TEPS, except here of course the necessary tables are created in the relational database of choice.

If the seed data is for an agent that reports to a remote monitoring server, complete this process for both the hub and the remote monitoring server. A hub monitoring server should be running before proceeding with a remote monitoring server seed.

26. Verify that each application support added for the components has a return code (rc) equal to 0, as shown in Figure C-30. Click **Next**.

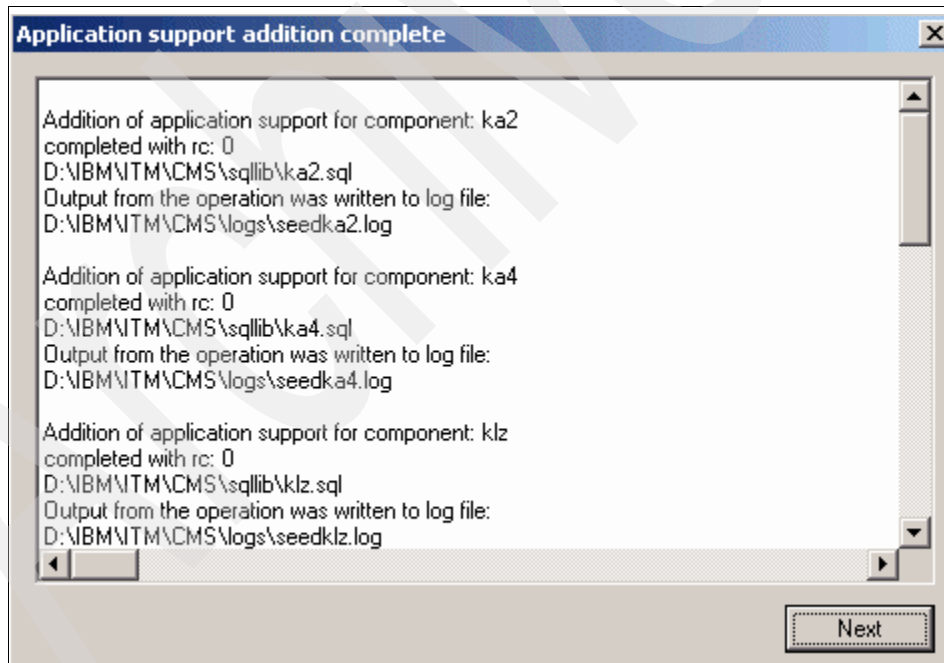
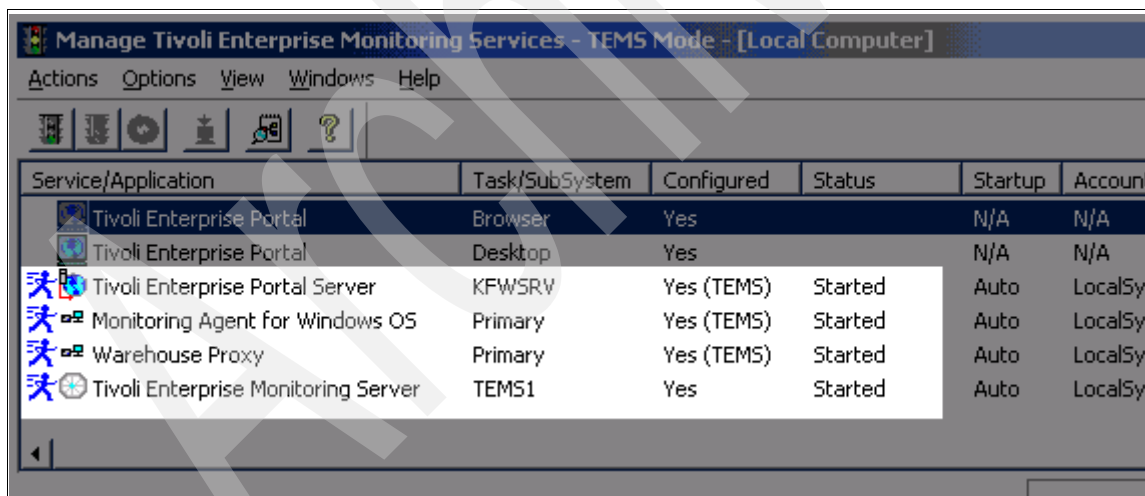


Figure C-30 Application addition support window

The next configuration step (Figure C-31) configures the default communication between any IBM Tivoli Monitoring component and the hub monitoring server.

27. Specify the default values for IBM Tivoli Monitoring components to use when they communicate with the monitoring server.
 - a. If agents must cross a firewall to access the monitoring server, select **Connection must pass through firewall**.
 - b. Identify the type of protocol that the agents use to communicate with the hub monitoring server. Your choices are IP.UDP, IP.PIPE, IP.SPIPE, and SNA. Leave the default choice and click **OK**.
28. Click **OK** in the protocol configuration screen.
29. Once the installation is complete, click **Finish**.
30. The Manage Tivoli Enterprise Monitoring Services window now appears. Verify that the following services are configured and started, as shown in Figure C-31.
 - Tivoli Enterprise Portal Server
 - Monitoring Agent for Windows OS
 - Warehouse Proxy
 - Tivoli Enterprise Monitoring Server



Service/Application	Task/SubSystem	Configured	Status	Startup	Account
Tivoli Enterprise Portal	Browser	Yes		N/A	N/A
Tivoli Enterprise Portal	Desktop	Yes		N/A	N/A
Tivoli Enterprise Portal Server	KFWSRV	Yes (TEMS)	Started	Auto	LocalSy
Monitoring Agent for Windows OS	Primary	Yes (TEMS)	Started	Auto	LocalSy
Warehouse Proxy	Primary	Yes (TEMS)	Started	Auto	LocalSy
Tivoli Enterprise Monitoring Server	TEMS1	Yes	Started	Auto	LocalSy

Figure C-31 Services that should be started

ITCAM for SOA Application Support installation

In order to view data collected by the ITCAM for SOA monitoring agents in the Tivoli Enterprise Portal, Tivoli Enterprise Monitoring Server and Tivoli Enterprise

Portal Server must have application support for ITCAM for SOA installed. Application support consists of the following:

- ▶ Data structure definition for Tivoli Enterprise Monitoring Server attributes and attribute groups (tables). ITCAM for SOA contains two tables: Services_Metrics and Services_Inventory.
- ▶ Situation definitions that allow proactive monitoring to be performed in the IBM Tivoli Monitoring environment.
- ▶ Presentation information to be installed in the Tivoli Enterprise Portal Server, including help resources and workspace definitions.
- ▶ Additional resources such as sample workflow and historical collection information.

The installation for ITCAM for SOA application support is on the CD, which provides multiple-platform support (Windows, AIX, and Solaris binaries). The wizard will copy the CD-ROM's files into the disk. If you install this from disk, copy the CD content into the *same* path; otherwise the installation wizard will fail.

1. Navigate to the Application Support Installer directory on the ITCAM for SOA product CD. The installer is provided to support IBM Tivoli Monitoring V6.1 installation. Select the appropriate operating system platform:

Windows	setupwin32.exe
AIX	setupaix.bin
Solaris	setupSolaris.bin

2. The first screen identifies the product to be installed. Click **Next**.
3. Enter the installation directory and the location of the installable media. The default directory installation directory is C:\IBM\ITM.
4. Because the Tivoli Enterprise Monitoring Server, Tivoli Enterprise Portal Server, and Tivoli Enterprise Portal are installed on one system, all of the components are selected to install to the local machine. See Figure C-32.

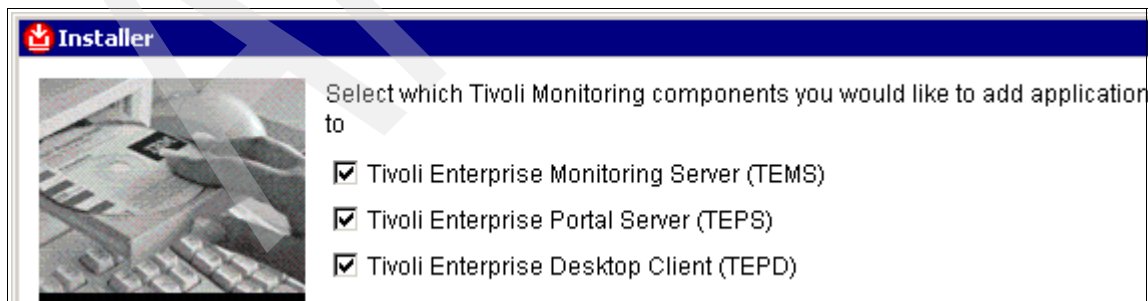


Figure C-32 Installation features

Note: If you select any of the application support components that are already installed, a warning window is displayed. You have the choice of overwriting the existing installation files or unchecking the desired component to avoid overwriting the files.

5. Select the application support for ITCAM for SOA to be installed in the IBM Tivoli Monitoring Services environment (Figure C-33).

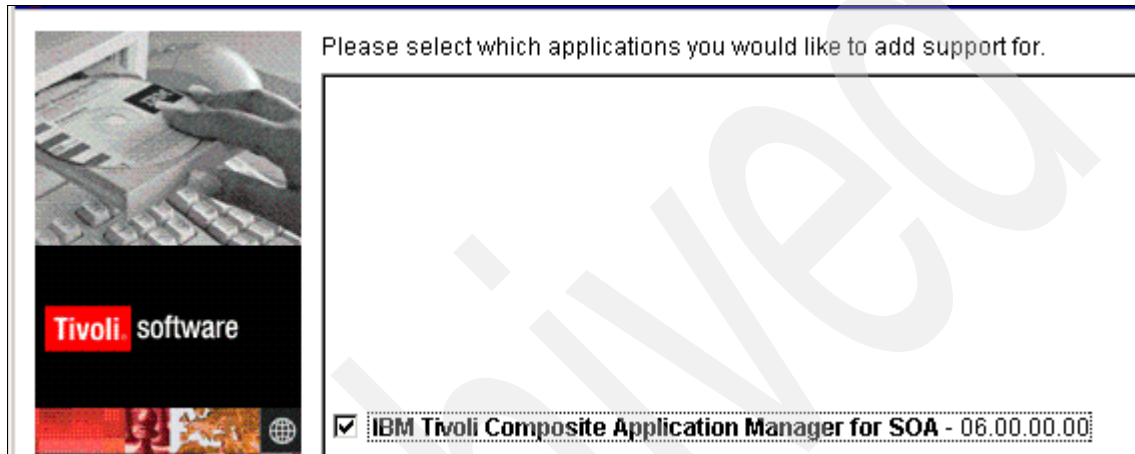


Figure C-33 Application support component

6. The installation summary provides a brief list of actions it will perform. Click **Next**. See Figure C-34.

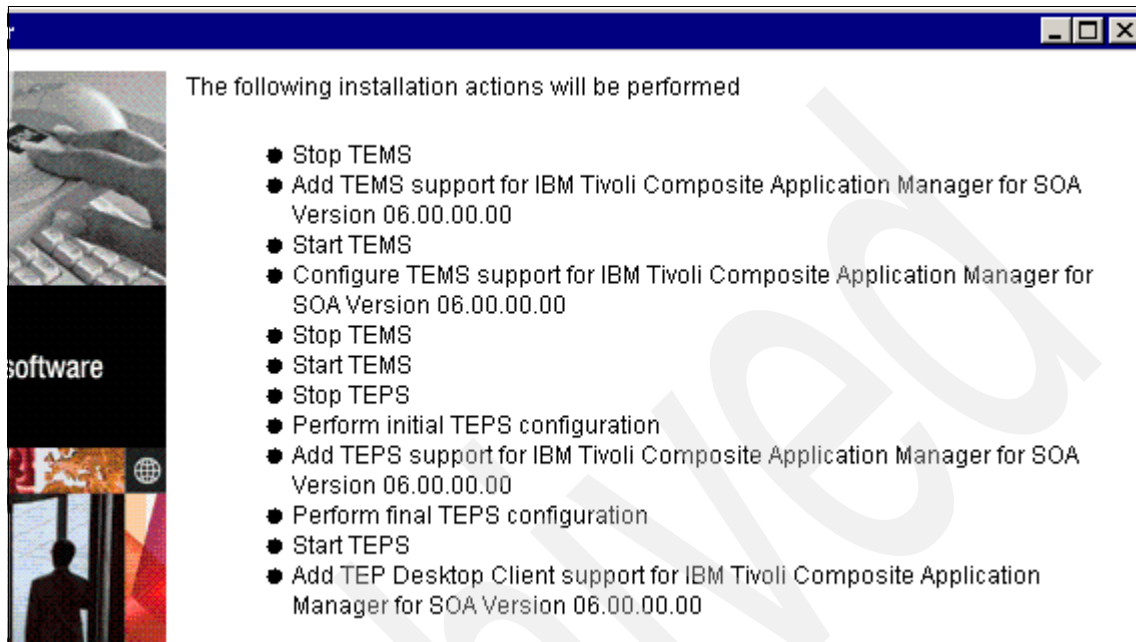


Figure C-34 Installation action list

7. The final installation panel displays the progress bar and the installation details output (Figure C-35). Once the installation is complete, click **Finish**.

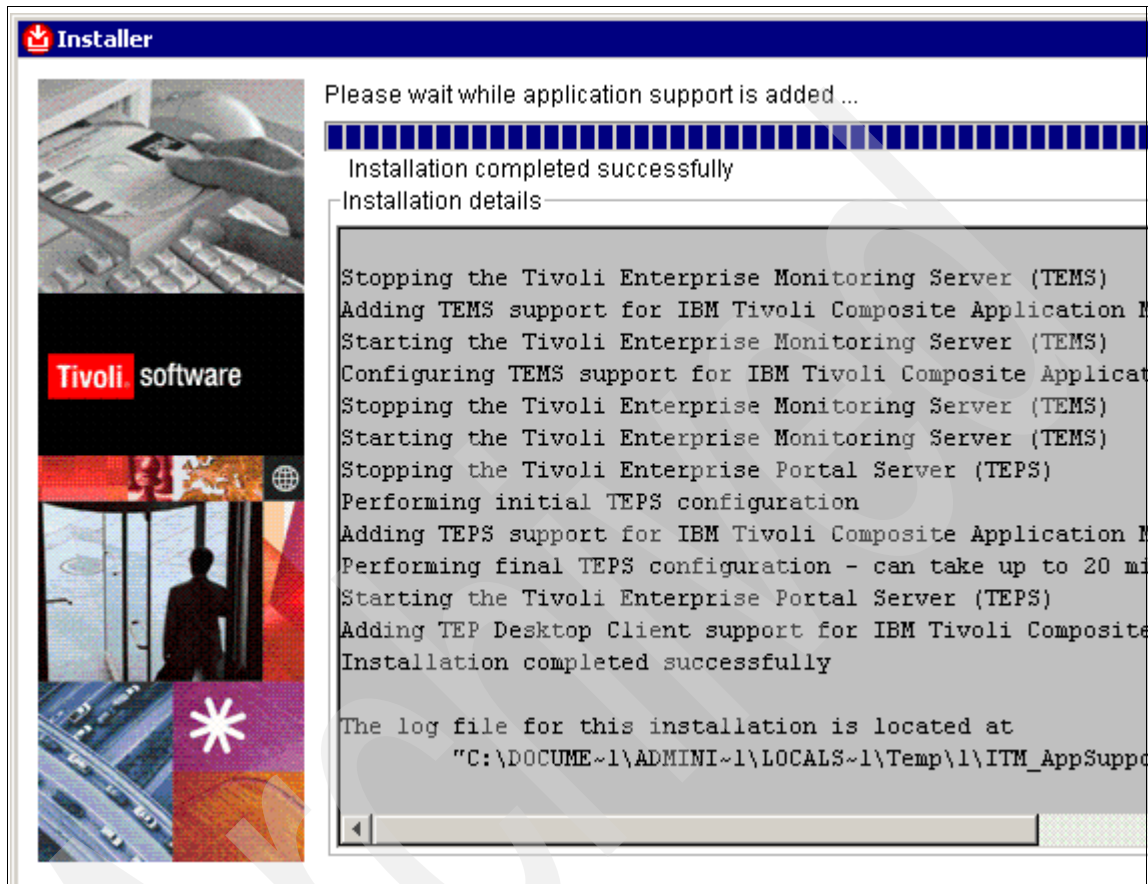


Figure C-35 Installation completion summary

ITCAM for SOA Monitoring Agent installation and configuration

This section gives an overview of management agent installation. ITCAM for SOA Monitoring Agent installation includes the data collector, which is installed into each application server environment where Web services traffic is to be monitored.

1. Launch the ITCAM for SOA installation:
 - For Windows installation, navigate to the /WINDOWS directory and select **setup.exe** from the ITCAM for SOA product CD.

- For UNIX installation, open a command session and navigate to the root directory of the ITCAM for SOA CD. Issue this command:

```
./install.sh
```

2. At the welcome screen, click **Next**.
3. Click **Next** at the prerequisites screen.
4. Carefully read the software license agreement and click **Accept**.
5. Select the directory where ITCAM SOA should be installed (Figure C-36) and click **Next**.



Figure C-36 Installation location

6. Select **Agent Support** for ITCAM for SOA Agent to be installed on the local system (Figure C-37).

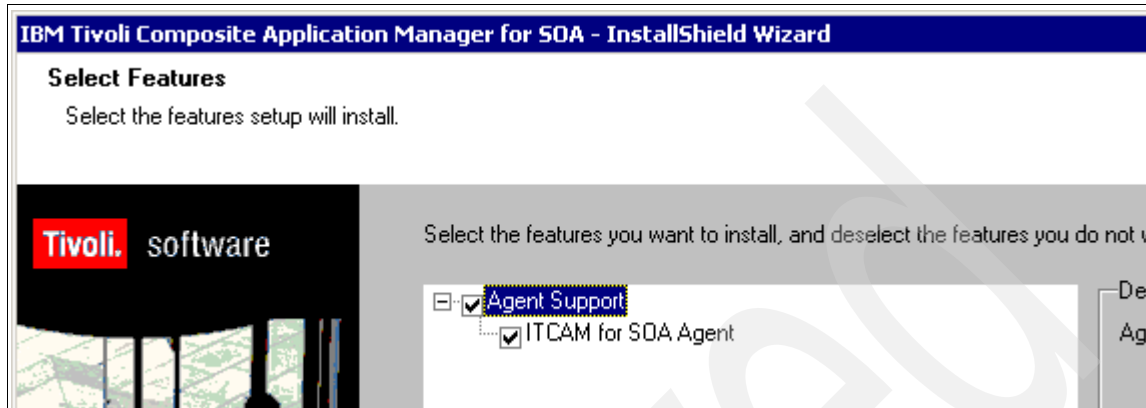


Figure C-37 Agent installation

7. Accept the default program folder on the next screen and click **Next**.
8. The installation summary displays a brief list of tasks for the current installation. Click **Next**.
9. Select both options in the Setup Type screen to configure the ITCAM for SOA monitoring agent and launch the Tivoli Monitoring Services for additional configurations (Figure C-38).

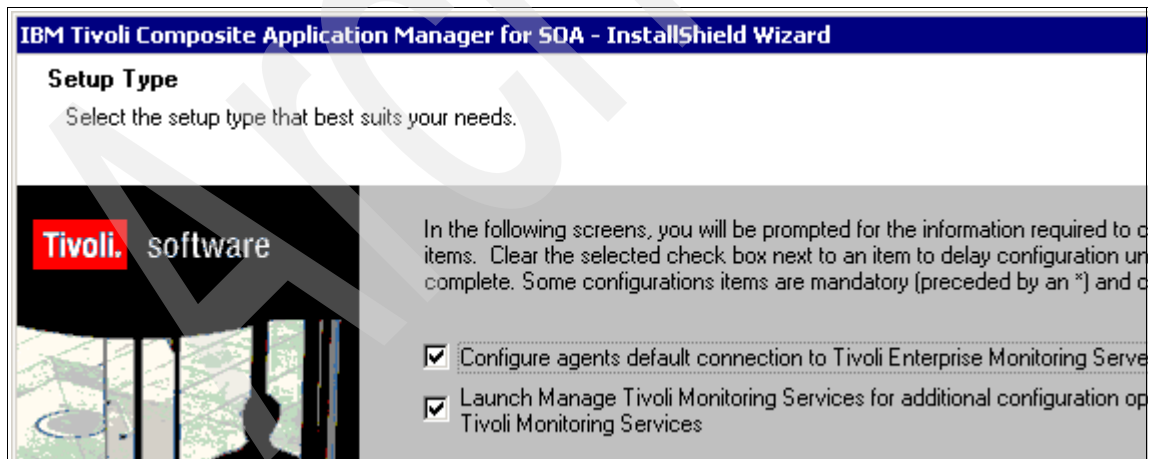
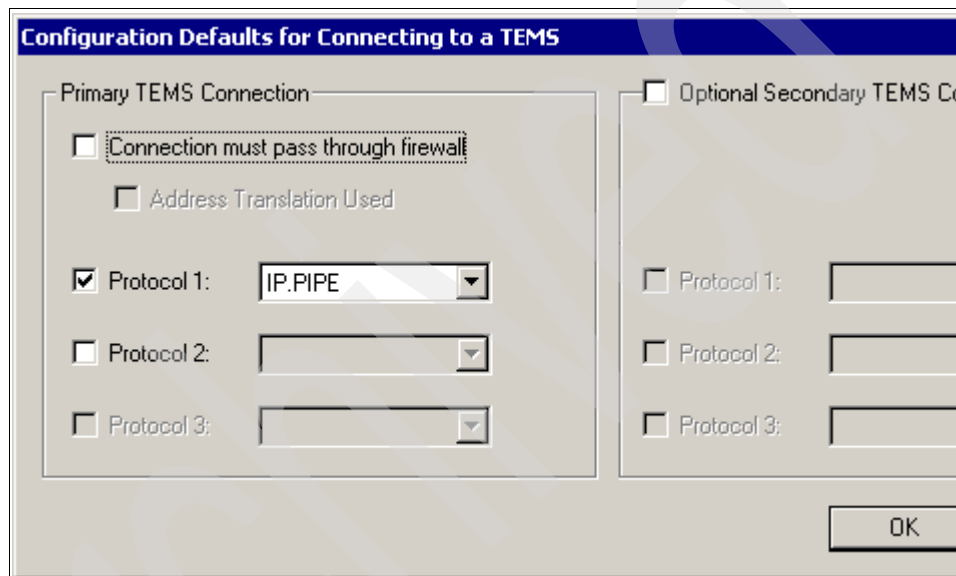


Figure C-38 Setup type

10. The installation configures the communication protocol for the ITCAM for SOA monitoring agent to connect to the Tivoli Enterprise Monitoring Server (Figure C-39).

Note: The default protocol type to connect to the Tivoli Enterprise Monitoring Server using TCP protocol is IP.PIPE. Depending on your Tivoli Enterprise Monitoring Server configuration, you may select another protocol as appropriate.



The image shows a Windows-style dialog box titled "Configuration Defaults for Connecting to a TEMS". It is divided into two main sections: "Primary TEMS Connection" and "Optional Secondary TEMS Connection".

Primary TEMS Connection:

- ☐ Connection must pass through firewall
- ☐ Address Translation Used
- ☒ Protocol 1: IP.PIPE (dropdown menu)
- ☐ Protocol 2: (empty dropdown menu)
- ☐ Protocol 3: (empty dropdown menu)

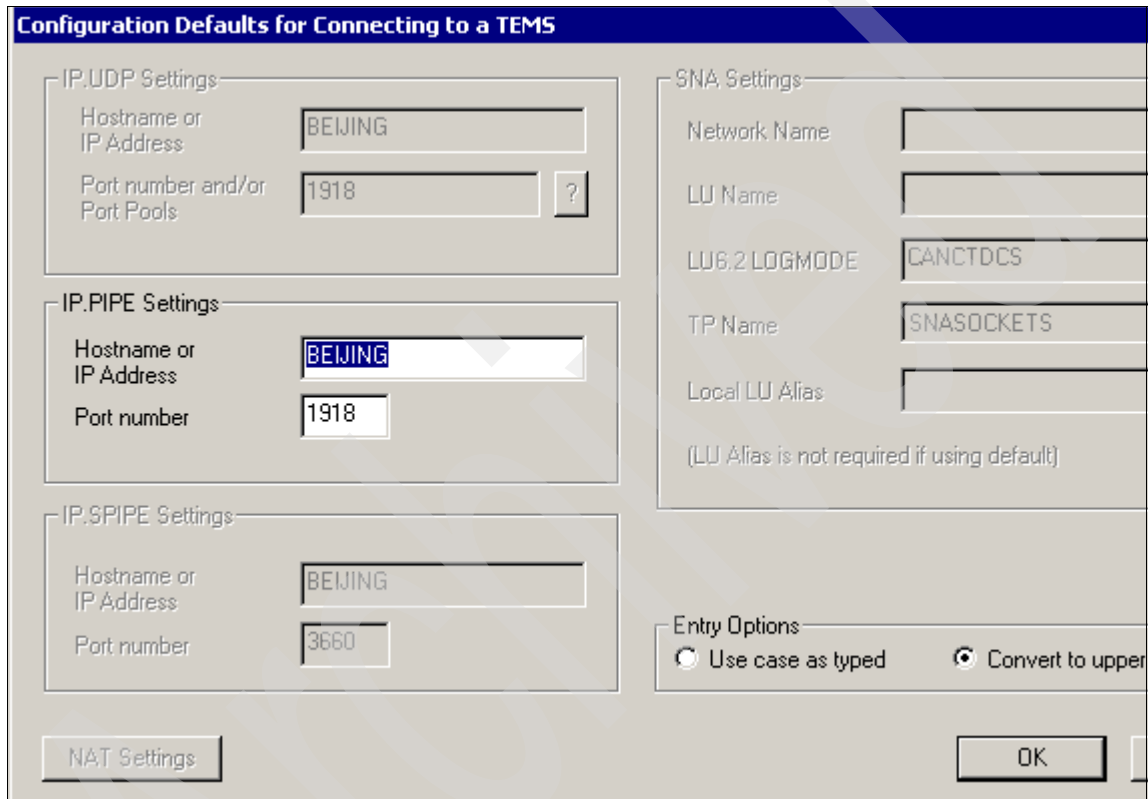
Optional Secondary TEMS Connection:

- ☐ Optional Secondary TEMS Connection
- ☐ Protocol 1: (empty dropdown menu)
- ☐ Protocol 2: (empty dropdown menu)
- ☐ Protocol 3: (empty dropdown menu)

An "OK" button is located at the bottom right of the dialog box.

Figure C-39 Communication configuration

11. In the Tivoli Enterprise Monitoring Server configuration window (Figure C-40), specify the Tivoli Enterprise Monitoring Server host name and IP address of the Enterprise Monitoring Server. Keep the default port number at 1918 for the ITCAM for SOA monitoring agent connection unless a different IP.PIPE port number was specified during the Tivoli Enterprise Monitoring Server installation.



The image shows a configuration window titled "Configuration Defaults for Connecting to a TEMS". It contains several sections for setting up connections to the Tivoli Enterprise Monitoring Server.

- IP.UDP Settings:** Hostname or IP Address: BEIJING; Port number and/or Port Pools: 1918.
- IP.PIPE Settings:** Hostname or IP Address: BEIJING; Port number: 1918.
- IP.SPIPE Settings:** Hostname or IP Address: BEIJING; Port number: 3660.
- NAT Settings:** A button labeled "NAT Settings".
- SNA Settings:** Network Name, LU Name, LU6.2 LOGMODE: CANCTDCS, TP Name: SNASOCKETS, Local LU Alias. A note below states: "(LU Alias is not required if using default)".
- Entry Options:** Two radio buttons: "Use case as typed" and "Convert to upper" (which is selected).
- Buttons:** "OK" and a partially visible "Cancel" button.

Figure C-40 Connection to the Tivoli Enterprise Monitoring Server

12. Click **Finish** to complete the installation.

13. When the Tivoli Enterprise Monitoring Server configuration for ITCAM for SOA is complete, the Manage Tivoli Enterprise Monitoring Services utility launches. The ITCAM for SOA will be listed as not configured and the status will be blank. See Figure C-41.

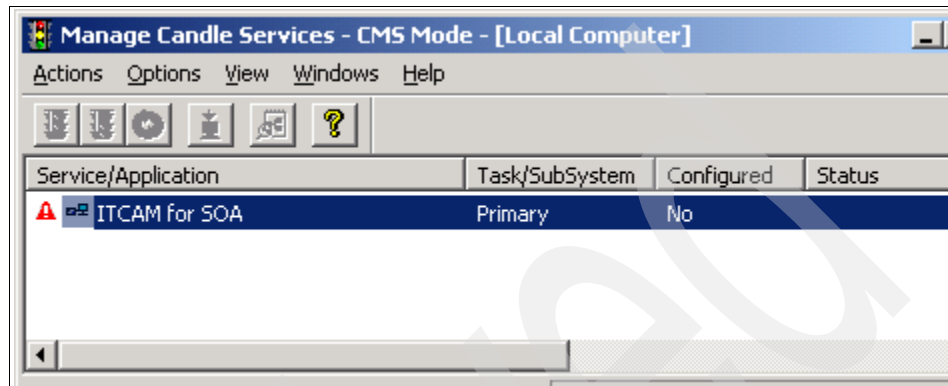


Figure C-41 Manage IBM Tivoli Monitoring Services

Note: You do not have to configure and start the ITCAM for SOA monitoring agent until the monitoring agent support is enabled for the target application server. If you start the monitoring agent prior to enabling application server support, the agent will start but will have to be stopped and restarted after enabling the application server support.

ITCAM for SOA Monitoring Agent configuration

This section describes how to enable the ITCAM for SOA monitoring agent data collector handler in the appropriate application server.

After the ITCAM for SOA monitoring agent is installed, the data collector directory structure is created in the Tivoli Enterprise Monitoring Agent base directory as follows:

- ▶ For Windows: <tivoli_agent_home>\CMA
- ▶ For UNIX: <tivoli_agent_home>/<OS_INTERP>/kd4

This directory contains the structure shown in Figure C-42.

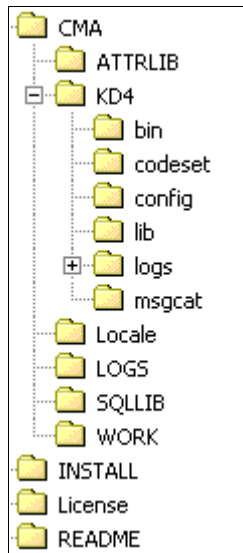


Figure C-42 Agent directory structure

These directories contain all files required to run the data collector on IBM WebSphere Application Server, Microsoft .NET, and BEA WebLogic.

A monitoring agent that has been installed on an application server host must be configured. On J2EE servers, the configuration step installs a JAX-RPC handler to intercept all Web services calls. To configure the monitoring agent, run the following command:

```
KD4configDC.bat -enable -env <x> <application server specific arguments>
```

Note: On non-Windows servers, you would use KD4ConfigDC.sh or as appropriate for your operating system.

The <x> argument after -env specifies the target application server. <x> must be one of the following:

- ▶ 1 = IBM WebSphere Application Server
- ▶ 2 = Microsoft .Net platform
- ▶ 3 = BEA WebLogic Server

The application server specific arguments for this command vary depending on the application server environment. Refer to the *IBM Tivoli Composite Application Manager for SOA Installation and User's Guide*, GC32-9492, for more detailed information about the KD4configDC options.

Once the monitoring agent configuration is complete, the application server may need to be restarted in order for the agent to begin reporting monitoring data to the monitoring server.

Configuring for WebSphere Application Server data collector

For IBM WebSphere Application Server, run this command at a command prompt:

```
KD4configDC -enable -env 1 <WAS_HOME>
```

Note: If you have set the WAS_HOME environment variable, the configuration program can be invoked without any argument. The WAS_HOME environment variable is typically set using the setupCmdLine program from WebSphere.

This command enables IBM WebSphere Application Server to use the KD4 data collector as a JAX-RPC handler. The kd4dcagent.jar file is installed into the <WAS_HOME>/lib/ext directory.

After the ITCAM for SOA monitoring agent data collector has been configured, stop and restart IBM WebSphere Application Server.

Web Services Navigator installation

The Web Services Navigator is a stand-alone application based on the Eclipse framework. You can install the application into an existing Eclipse framework application such as Rational Software Architect or WebSphere Integration Developer or you can install a copy of the Eclipse framework to run Web Services Navigator.

For the purposes of this sample, you will install Web Services Navigator on the monitoring server node.

1. Launch the installer by executing the installer executable. On Windows, this is setupwin32.exe.
2. Click **Next** on the Welcome screen.
3. Accept the terms of the license agreement and click **Next**.
4. Select a directory in which to install the Web Services Navigator and click **Next**.
5. Select **Install Eclipse as part of this installation** and click **Next**.
6. The next screen summarizes the installation about to be performed. Click **Next**.

Once the installation is complete, click **Finish** to exit the installer.

Verify the installation

To verify the installation, you will need to verify that both the monitoring agent and the monitoring server have been installed correctly.

Verifying the monitoring agent installation

After running the KD4configDC script, ensure the following:

- Check for a file named KD4BaseDirConfig.properties that is created automatically during installation in the system configuration directory. For Windows operating systems, this file should be located in %SYSTEMROOT%\system32\drivers\etc. Verify that the file exists and has the following entry, assuming the monitoring agent is installed in <tivoli_agent_home>.

```
INSTALLDIR=C:\\<tivoli_agent_home>\\CMA\\
```

The value in the INSTALLDIR variable represents the directory location where the \KD4 folder is located. The \KD4 folder contains logging and properties files used with ITCAM for SOA.

- For WebSphere Application Server (and servers that are built on WebSphere Application Server such as WebSphere ESB or WebSphere Process Server) the JAX-RPC handler should be installed into the application server as well. The file kd4agent.jar should be copied to <WAS_HOME>/lib/ext.

Verify the monitoring server installation

Open the Manage Tivoli Enterprise Monitoring Services utility and verify that the Tivoli Enterprise Monitoring Server and Tivoli Enterprise Portal Server are started. If you are planning to write Web services data to the data warehouse using the Warehouse Proxy, make sure that the Warehouse Proxy is started as well. Finally, started either the Enterprise Portal Desktop client or Web client.

Note: The default user name for the portal client is sysadmin. There is no password by default.

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/SG247228>

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, SG247228.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
WID_Resources.zip	This is the project interchange file that contains the WID_Resources project referred in “Errors using XML Mapper without Internet connectivity” on page 639.
ITSOMart_UML_Model.zip	This zip file contains UML model project interchange files that can be imported into a Rational Software Architect V6 workspace. These models are discussed in Chapter 5, “Model with Rational Software Architect” on page 125.
ITSOMartPIFs.zip	This zip file contains ITSOMart application project interchange files that can be imported into a WebSphere Integration Developer V6 workspace. Instructions for using these files can be found in Appendix A, “Sample application install summary” on page 589.
TopDownAppendix.zip	This zip file contains project interchange files with the projects built using the instructions in “Creating a top-down SOAP/JMS Web service” on page 606. These files can be imported to a WebSphere Integration Developer V6 workspace.
ITSOMart.sql	This file contains the SQL statements required to define the ITSOMART database used in the Get Credit Score scenario.
ITSOMART_Cloudscape.zip	This zip file contains the Cloudscape version of the ITSOMART database used in the Get Credit Score scenario.

System requirements for downloading the Web material

The following system configuration is recommended:

Operating system	Windows or Linux
Software	WebSphere Integration Developer V6.0.1, Rational Software Architect V6.0.1, WebSphere ESB V6.0.1

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.

Archived

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 about ordering these publications, see “How to get IBM Redbooks” on page 709. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- ▶ *WebSphere Application Server V6 System Management & Configuration Handbook*, SG24-6451
- ▶ *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6*, SG24-6494
- ▶ *Patterns: Implementing Self-Service in an SOA Environment*, SG24-6680
- ▶ *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688
- ▶ *Patterns: Model-Driven Development Using IBM Rational Software Architect*, SG24-7105
- ▶ *Patterns: Extended Enterprise SOA and Web Services*, SG24-7135
- ▶ *Getting Started with IBM Tivoli Monitoring 6.1 on Distributed Environments*, SG24-7143
- ▶ *IBM Tivoli Composite Application Manager V6.0 Family: Installation, Configuration, and Basic Usage*, SG24-7151
- ▶ *Enabling SOA Using WebSphere Messaging*, SG24-7163
- ▶ *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212
- ▶ *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240

Other publications

These publications are also relevant as further information sources:

- ▶ Gamma, Erich, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1997. ISBN 0-201-63361-2
- ▶ *IBM Tivoli Monitoring Installation and Setup Guide*, GC32-9407
- ▶ *IBM Tivoli Composite Application Manager for SOA Installation and User's Guide*, GC32-9492

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ IBM Patterns for e-business
<http://www-128.ibm.com/developerworks/patterns/>
- ▶ Rational Application Developer system requirements
<http://www-306.ibm.com/software/awdtools/developer/application/sysreq/index.html>
- ▶ Rational Software Architect home page
<http://www.ibm.com/software/awdtools/architect/swarchitect/index.html>
- ▶ Rational Software Architect system requirements
<http://www.ibm.com/software/awdtools/architect/swarchitect/sysreq/index.html>
- ▶ Tivoli Composite Application Manager Basic for WebSphere home page
<http://www-306.ibm.com/software/tivoli/products/composite-application-mgr-basic-websphere/>
- ▶ Tivoli Composite Application Manager for Response Time Tracking home page
<http://www-306.ibm.com/software/tivoli/products/composite-application-mgr-rtt/>
- ▶ Tivoli Composite Application Manager for SOA home page
<http://www-306.ibm.com/software/tivoli/products/composite-application-mgr-soa/>
- ▶ Tivoli Composite Application Manager for WebSphere home page
<http://www-306.ibm.com/software/tivoli/products/composite-application-mgr-websphere/>
- ▶ WebSphere Adapter home page
<http://www-306.ibm.com/software/integration/wbiadapters/>

- ▶ WebSphere Adapter product documentation
<http://www-306.ibm.com/software/integration/wbiadapters/library/infocenter/doc/index.html>
- ▶ WebSphere Enterprise Service Bus system requirements
<http://www-306.ibm.com/software/integration/wsesb/sysreqs/>
- ▶ WebSphere Enterprise Service Bus home page
<http://www-306.ibm.com/software/integration/wsesb/>
- ▶ WebSphere Enterprise Service Bus product documentation
<http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.websphere.wesb.doc/info/welcome.html>
- ▶ WebSphere Enterprise Service Bus 6.0.1 Information Center
<http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/topic/com.ibm.websphere.wesb.doc/info/welcome.html>
- ▶ WebSphere Integration Developer system requirements
<http://www-306.ibm.com/software/integration/wid/sysreqs/>
- ▶ *What Is an ESB, and Do You Really Need One?*
<http://www.computerworld.com/developmenttopics/development/webservices/story/0,10801,108478,00.html>
- ▶ *ESB in Practice*
http://www-128.ibm.com/developerworks/websphere/library/techarticles/0509_flurry1/0509_flurry1.html
- ▶ IBM WebSphere Developer Technical Journal: *Building an Enterprise Service Bus with WebSphere Application Server V6 -- Part 7*
http://www-128.ibm.com/developerworks/websphere/techjournal/0509_reinitz/0509_reinitz.html
- ▶ The Open Applications Group Integration Specification (OAGIS)
<http://www-128.ibm.com/developerworks/xml/library/x-oagis/>
- ▶ *Simplify integration architectures with an Enterprise Service Bus*
<http://www-128.ibm.com/developerworks/webservices/library/ws-esbia/index.html>
- ▶ *Web services programming tips and tricks: Learn simple, practical Web services design patterns, Part 4*
<http://www-128.ibm.com/developerworks/webservices/library/ws-tip-altdesign4>
- ▶ *SOA programming model for implementing Web services, Part 4: An introduction to the IBM Enterprise Service Bus*
<http://www-128.ibm.com/developerworks/library/ws-soa-progmodel4/>

- ▶ WebSphere Adapters
<http://www-306.ibm.com/software/integration/wbiadapters/>
- ▶ WebSphere DataPower SOA Appliances
<http://www-306.ibm.com/software/integration/datapower/index.html>
- ▶ *Toward a pattern language for Service-Oriented Architecture and Integration, Part 1: Build a service eco-system*
<http://www-128.ibm.com/developerworks/webservices/library/ws-soa-soi/>
- ▶ *SOA programming model for implementing Web services, Part 4: An introduction to the IBM Enterprise Service Bus*
<http://www-128.ibm.com/developerworks/webservices/library/ws-soa-progmodel4/index.html>
- ▶ *XQuery: An XML query language*
<http://www.research.ibm.com/journal/sj/414/chamberlin.pdf>
- ▶ IBM WebSphere Enterprise Service Bus
<http://www-306.ibm.com/software/info/middleware/applications/index.jsp>
- ▶ developerWorks Architecture Integration
<http://www-128.ibm.com/developerworks/architecture/application.html>
- ▶ Rational Software Architect
<http://www-306.ibm.com/software/awdtools/architect/swarchitect/>
- ▶ Rational Software Architect: SOA design resources
<http://www-306.ibm.com/software/info/developer/solutions/soadev/dtoolkit2.jsp>
- ▶ *UML Profile for Software Services, RSA Plug-In*
http://www-128.ibm.com/developerworks/rational/library/05/510_svc/
- ▶ *Architecting on demand solutions, Part 11: Build ESB connectivity with Rational Software Architecture (RSA) WebSphere Platform Messaging Patterns*
<http://www-128.ibm.com/developerworks/ibm/library/i-odoebp11/>
- ▶ OMG Model Driven Architecture
<http://www.omg.org/mda>

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

Archived

Index

Symbols

679
./install.sh 692
.NET 112
.Net 485, 564

Numerics

3270 terminal emulator 567
80/20 situation 27

A

Access Integration pattern 29
access services 77
action log 577
activation specification 195, 201
activation spec 501
Active Directory Agent 564
activity diagram 140, 142
actor 144
adapter 85
 runtime configuration 498
adapter services 247
adapter-specific dependent JAR files 354
AddFltrCntrl 575
AddMntCntrl 571
AddMntrCntrl 575
administrative console
 starting 487
affinity 533
agent 560
agent data log 583
agent log 578
agent log files for ITCAM for SOA 577
agent status 561
alert 560, 562
alias destination 521
alias queue 474, 519
analysis model 128–129, 137, 144, 151
ANT 524
App Server/Services node 42, 57, 59
application
 deploy 523

 gateways 41
 install 525
 patterns 44, 60
 server 40
 start 524, 529
application agent 563
Application Integration 38, 52
Application Integration pattern 29
Application pattern 28, 38–39
application server profile 486
architectural design 128, 149
array 417, 428, 435, 437, 445
assemble mediations 220
assembly
 WebSphere Integration Developer 104
assembly diagram 161, 224, 284, 315, 336, 401, 403
asset-based development 129
asynchronous 213
attribute 234, 270, 273
 enumerations 455
attribute tables 571
attributes 571
Audit System 164
Audit system 163
Audit System service 451
augment 486
augmentation 152
automatic build 224
autonomic rules 54, 60
availability data 560
Average Message Size by Operation view 570
Average Response Time by Operation view 570
Average size of Messages by Service - Operation - Type view 571

B

Binding
 SCA 336
 Web service 283, 315, 442
binding 248, 611, 613
 Adapter 247
 JMS 246–247, 452, 459, 501

- SCA 246, 315
 - Web service 245–246, 284, 372, 458
 - Web service binding 428, 432
- blank model 137
- blank model template 187
- BO location 362
- BO Services 219
- BO Type Metadata 219
- body 220, 436
- boundary component 149
- boundary service 152
- broker 38, 95
 - connectivity 37
- Broker application pattern 56, 58
- broker domain 95
- Btrieve 686
- build project 225
- built-in node 94
- bus component 190–192
- bus member 491, 502
- business application services 77
- business data mapping rules 54, 60
- business graph 219
- Business Integration perspective 222–223
- Business Integration view 224
- business object 165, 216, 218–219, 232–234, 269, 272, 276, 369, 459
 - create 269–270, 273, 276, 313, 422, 429
 - create 454
- business object framework 218–219
- Business pattern 27, 29
- Business Process Execution Language (BPEL) 104
- Business Service Directory
 - UDDI directory 41
- business services management 557
- business tier transformations 127

C

- C 485
- C++ 485
- caching 484
- Call Connection variation 54, 61
- callout fault node 298
- callout node 287, 329
- callout response node 255, 298
- capabilities 226
- default terminal 465
- cell 486
- CICS 556–557
- CICS Transaction Server 42
- class 171
- class diagram 128, 134, 140, 144, 174
- classpath 386
- clean 225
- client caching 484
- Cloudscape 338, 491, 494, 497–498, 648
- cluster 118, 532
 - add members 550
 - create 538
 - start 545
- clustering 480
- collaboration 188
- Collaboration business pattern 29
- collection interval 579
- com.ibm.j2ca.siebel.SiebelManagedConnection-Factory 389
- commands
 - ./install.sh 692
 - setup.exe 691
 - setupaix.bin 688
 - setupSolaris.bin 688
 - setupwin32.exe 688
- Common Business Event (CBE) 485
- Common Event Infrastructure 647
- Common Event Infrastructure (CEI) 485
- CommonEventInfrastructure_Bus 486
- commonj.sdo.ChangeSummary 218
- commonj.sdo.DataObject 217, 219
- component 144–145, 212
 - wiring 250
- component diagram 129, 140, 144, 148, 151, 158–159
- component test 259
- component-managed authentication alias 495
- composite application 556, 558
- composite application management 556
- Composite pattern 27
- concat function 296
- Configuration manager 95
- connection factory 199, 207
- connection rules 54
- connectionFactory 618
- connectivity 78
 - WebSphere ESB 85
- Connector node 42
- Console view 259, 487, 531

- content filtering 583
- content log 577
- content-based message routing 88, 91
- core group 538
- Credit Rating mediation 155, 265, 267, 275
- Credit Rating Service 265, 309
 - Interface 278
- Credit Score mediation 267, 309
- Credit Score Service 267, 313, 315
- CRM 351
- CRM mediation 161, 349, 363
- CRM Registration scenario 348
- CRM Registration Service 349
- CurrencyCode 381
- Custom binding 483
- custom install 535
- custom Java bean 320
- Custom mediation 347
- Custom primitive 253, 351, 375, 396–397, 501
 - implementation 400, 403
 - interface 399
- custom profile 486, 537
- Custom XPath 464
- custom XPath 324–325
- Customer Registration 151
- Customer Relationship Management (CRM) 348
- customer relationship management (CRM) 142
- CWYEB_SiebelAdapterEAR 386
- CWYFF_FlatFileEAR 412

D

- Data access service (DAS) 216
- data collection 569, 575
- data collector 570
- data collector filter control configuration attributes 572
- Data Collector Filter Control Configuration view 569
- data collector global configuration attributes 571
- Data Collector Global Configuration view 569
- data collector monitor control configuration attributes 571
- Data Collector Monitor Control Configuration view 569
- data graph 217, 219
- Data Integration 53
- data object 217
- data serialization 459
- data source 338, 540

- create 494
- data warehouse 586, 672, 699
- data warehousing 682
- database 161, 263
- Database Lookup primitive 253, 323–324, 326, 329, 338, 491, 495
 - fail terminal 331
 - key not found 328
- database name 339
- DataCollectMessageLoggingLevel 582
- DataPower 98–99
- DataPower Toolkit 14
- DataPower XS40 14
- DB2 113–114, 540, 682
 - Install 672
- DB2 Agent 564
- DB2_UNIVERSAL_JDBC_DRIVER_NATIVEPATH 494, 541
- DB2_UNIVERSAL_JDBC_DRIVER_PATH 494, 541
- decomposition 38
- Decomposition application pattern 44, 49–50
- default configuration 658
- default flow 392
- default messaging provider 467, 484
- default terminal 392
- default.dnx 610
- deferred asynchronous response 532
- deferred response 534
- DelFiltrCntrl 575, 584
- DelMntrCntrl 583
- DelMntCntrl 571
- DelMntrlCntrl 575
- demilitarized Zone 40
- denial of service (DOS) 583
- dependency
 - JAR files 354, 364
 - Java 311–312
 - library 231–232, 242–243, 268, 281, 310, 352, 454
- deploy
 - WebSphere Application Server 102
- deployment diagram 129
- deployment location 340
- deployment manager 486, 536–537
- deployment manager profile 486
- design 76
- design model 128, 137, 149
- destination 502

- destination JNDI name 620
- develop
 - Message Brokers Toolkit 95
- development 104
- development services 76
- Diagram Navigator view 133
- direct connection 38
- Direct Connection application pattern 53, 55–56
- direct connectivity 36
- Directly Integrated Single Channel 156
- Directly Integrated Single Channel application pattern 38, 44
- Directory services 40
- DisableDC 571, 575
- disaster recovery 118
- Distribution mode
 - All 435
- distribution mode 377, 464
- DNS See Domain Name Server
- document literal 613
- domain analysis 128, 144, 148
- Domain Name Server 40
- dynamic routing 26

E

- EJB cClient 91
- EJB client 485
- element
 - Add to an array 445
- EnableDC 571, 575
- encryption key 678
- Enterprise Application Integration (EAI) 52
- Enterprise Java Beans (EJBs) 102
- enterprise Java capability 609
- enterprise level IT transformation 72
- Enterprise Resource Planning (ERP) 53
- enterprise service bus 32, 35, 82, 102, 480
 - SOA Foundation Reference Architecture 102
- Enterprise Service Bus pattern 84, 123
- Enterprise Service Discovery wizard 247, 353, 355, 364, 386
- enterprise Web services 483
- Enterprise Web Services (JSR 109) 102
- enumerated values 465
- enumeration 455
- error log file 577
- ESB 57, 150–151, 154
- ESB Mediation Message Log viewer 595

- ESB Pattern 32
- ESB runtime pattern 41–42
- esbpcatLinux.bin 487, 640, 662
- esbpcatWindows.exe 487, 640, 662
- event correlation and automation 556
- execution group 95
- Existing applications 41
- export 215–216, 247, 426, 432, 445, 457
 - binding 245
 - JMS binding 246–247, 501
 - SCA binding 246, 315
 - Web service binding 246, 284, 372, 428, 442, 458
- Exposed Broker 38
- Exposed Direct Connection application pattern 38, 61
- Exposed Direct Connection runtime pattern
 - SOA profile
 - Exposed ESB Gateway 43
 - Service Consumers 43
 - Service Providers 43
- Exposed ESB Gateway node 43
- Exposed Router variation of the Broker 38
- Extended Enterprise business pattern 29, 38, 60
- Extended Structured Query Language (ESQL) 94

F

- Fail primitive 253, 331–332
- failover 118, 480, 531–533
- Fault 237–239
- fault 263, 278, 300, 556, 574
- Fault Details view 569, 571
- fault handling 449
- Fault Log attributes 572
- fault occurrences 570
- Faults Summary by Operation view 571
- Faults Summary workspace 569, 571, 584
- federate 486, 537
- filter control settings 575
- Filter pattern 464–465
- Filter primitive 375–376, 417
- Firewall
 - Protocol 41
- firewall 687
 - and IP.UDP 685
- first class manageable resource 565
- First Steps 536
- flat file 350, 372, 396, 501

Flow Patterns view 587
foreign bus 507
 create 514

G

Gateway 13
gateway 35
generate Implementation 424
generate implementation 427, 433
generic JMS provider 484
generic profile 39
generic SOAP elements 484
generic use case 26, 38
Get Credit Rating scenario 264, 266

H

HACMP 118
header 220
heartbeat request interval 560
heartbeat status 562
hierarchical business object 219
high availability 118, 532
high availability policy 548, 551
historical data collection 564
horizontal scaling 531
HTTP browser client 561
HTTPS 117
hub 561
hub monitoring server 686–687
Hub TEMS 678

I

i5 OS Agent 563
IBM Patterns for e-business (P4eb) 27
IBM Rational RequisitePro 128
IBM Tivoli Composite Application Manager (ITCAM) 556
IBM Tivoli Composite Application Manager for CICS Transaction 557
IBM Tivoli Composite Application Manager for IMS Transactions 557
IBM Tivoli Composite Application Manager for Response Time Tracking 562
IBM Tivoli Composite Application Manager for SOA 14, 16, 18, 562
IBM Tivoli Composite Application Manager for SOA 6.0 557

IBM Tivoli Composite Application Manager for WebSphere 557, 562
IBM Tivoli Composite Application Manager Response Time Tracking 557
IBM Tivoli Enterprise Console 562
IBM Tivoli Enterprise Monitoring framework 560
IBM Tivoli Monitoring 562
IBM WebSphere Adapter for Flat Files 350, 352, 364, 498
IBM WebSphere Adapter for Siebel Business Applications 350, 352, 355, 498
 import to the workspace 353
 Install 667
implementation 213
import 87, 214–215, 247, 283, 361–362, 364, 372, 457
 adapter binding 247
 binding 245
 JMS binding 247, 452, 459, 501
 no binding 458
 SCA binding 336
 Siebel 355
 Web service binding 245, 283, 315, 432
import binding 215
imported service 215
IMS 556
IMS Transaction Manager 42
Information Aggregation business pattern 29
infrastructure services 78
in-line reference 214
input 238–239
input fault node 287, 298
input node 252, 287
input response node 287, 298
input terminal 255
install fixes 663
install updates 666
InstallApplication 524
Installation wizard 658
Integrated Development Environment 659
Integrated Test Environment 659
integration assembly 76
Integration pattern 27
Integration Test Client 259, 303, 337, 467
Integration test client 220
Integration Tier Transformations 127
interaction services 77
interaction style 213
interface 215, 232, 237, 247, 276, 315, 369, 422,

- 432, 457, 501
 - add 284
 - create 236, 370, 456
 - create from WSDL 313
 - Custom primitive 399
 - define 277–278, 313
 - definition 213
 - EIS 352
- interfaces
 - create 236
- Interim fix 591
- IP.PIPE 684–685, 687, 695
- IP.SPIPE 684, 687
- IP.UDP 684–685, 687
- ITCAM for CICS/IMS 559
- ITCAM for Response Time Tracking 559
- ITCAM for RTT 562
- ITCAM for SOA 113, 562, 564
 - Agent installation 691
 - Application support 688
 - application support 114
 - data collector 114
 - Enable the monitoring Agent 696
 - Enable the monitoring agent 698
 - Installation 687
 - installation 113–114
 - Monitoring Agent 114
- ITCAM for SOA application support 113
- ITCAM for SOA Monitoring Agent 577
- ITCAM for SOA monitoring agents 113
- ITCAM for WebSphere 559, 562
- ITSOMart
 - URL 530
- ITSOMart database 337–338
- ITSOMart.LogDeniedQ 503
- ITSOMart.LogFailureQ 503
- ITSOMart.LogSuccessQ 503
- ITSOMart.RegistrationProcessorServiceQ 503
- ITSOMartBus.jacl 206
- ITSOMartLib 269–270, 277, 279–281, 313–314, 352, 372, 421, 453–454
- ITSOMartUtils 268, 311–312, 320

J

- J2C authentication data 363
- J2C authentication data entry 386, 491, 498, 500, 540
- J2CA0009E 389

- J2EE application 481
- J2EE client 485
- J2EE client support 91
- JAAS configuration 499
- JACL script 205, 208
- JAR files dependency 354, 364
- Java API for XML-based RPC (JAX-RPC) 102
- Java component 423–424
- Java dependencies 311–312
- Java dependency 312
- Java desktop client 561
- Java project 311
- Java proxy 633
- java.lang.ClassNotFoundException 389
- java.rmi.RemoteException 632
- JavaServer Faces (JSF) 595
- JavaServer Pages (JSPs) 45
- JAX-RPC 483–485, 698
- JAX-RPC handler 483
- JDBC driver 338
- JDBC provider 492, 540, 546
- JMS 164, 449
- JMS activation specification 505, 620, 630
- JMS binding 215–216, 247
- JMS client 91, 485
- JMS Connection 195
- JMS Connection component 196
- JMS Connection pattern 194–195, 198
- JMS destination 461
- JMS message header 220
- JMS messaging domain 459
- JMS provider 460, 484
- JMS queue 180–181, 207, 468, 501, 504, 521, 620
- JMS queue connection factory 467, 472
- JMS queues 484
- JMS resource 204, 206, 208, 619
- JNDI lookup name 460–461
- JNDI Name 620
- JNDI name 495
- JSF 633
- JSR 101 483
- JSR 109 483
- JSR 235 216
- JVM classpath 386

K

- KD4 562
- kd4agent.jar 699

KD4BaseDirConfig.properties 699
KD4configDC 698–699
key column 324
key Not Found 450
key not found 323, 328, 341
kiosk 47
KNT 562
KT2 562
KUM 562
KYN 562

L

Launchpad 658, 668–669
library
 create 232, 268
 include in a mediation module 242–243
library dependency 231–232, 242–243, 268, 281, 310, 352, 454
license agreement 678
Lightweight Directory Access Protocol (LDAP) 116
Lightweight Third Party Authentication (LTPA) 116
Linux OS Agent 563
listener port 507, 511, 517
listener port number 508
Local integration 15
local queue 474, 510, 513–514
Log Assembler 585
log file 585
log files 576–577
Log Registration mediation 163, 451
logging 152, 569
logging level 575
loose coupling 26, 35

M

Manage Tivoli Enterprise Monitoring Services 687, 699
Manage Tivoli Monitoring Service 686
managed node 486
Mapping editor 292
Mapping folder 232
match mapping 437
MDD 126
mediation 83, 152, 154
 packaging for deployment 260
 test 256
mediation flow 87, 213, 250–251, 253, 255, 426, 433

 create 285, 316, 373, 462
mediation flow component 159, 212–213, 215, 250–252, 283, 315–316, 336, 372, 426, 432–433, 457
 create 240–241, 281, 310, 352, 423, 430, 454
Mediation Flow Editor 251
mediation module 86–87, 215, 230, 480–481, 532
 call from another module 333
 copy 333
 create 240, 281, 310, 352, 423, 430, 454
 deploy 490, 523–524
 Install 525
 start 524
mediation primitive 150
Merge implementation 401
message
 viewing on the bus 473
message activity 570
Message Arrival by Operation view 570
Message Arrival by Service view 570
Message Arrival Clearing 574
Message Arrival Critical 574
Message Arrival Details view 570
Message Arrival Summary view 581
message arrival threshold attributes 572
Message Arrival workspace 569–570
Message Arrivals 581
message augmentation 90–91
Message Brokers Toolkit 20, 95
Message Connection variation 54, 61
Message Filter primitive 253, 255, 351, 375–376, 433–434, 447, 463–464
message flow 91, 94
message format transformation 89, 91
message header 220, 581
Message Logger primitive 253, 275, 288–290, 298, 300, 318, 322–323, 328–329, 331, 341, 351, 375–376, 390, 427, 463, 534
message logging 263, 347, 449, 570
message model 91
message order 534
message protocol transformation 88, 91
message routing 347
message service client 485
Message Service Clients 90
Message Service Clients for C/C++ and .NET 90
message Size 574
Message Summary workspace 569
message transformation 48, 152

- message type 255, 290, 301
- message-driven bean 505, 608, 620, 627
- MessageLogApp 303, 305, 595
- Messages Summary workspace 571
- messaging bus 482
- messaging engine 515, 522, 532–533, 540, 544, 548
 - add 551
 - schema 543
 - status 545
- messaging engine data store 535
- metric log 577
- mMatching criteria 549
- Model Driven Architecture (MDA) 126
- Model Explorer view 132, 136
- model-driven development 126
- modeling 128
 - business objects 164
 - messaging resources 180, 186
- Modeling perspective 129
- module assembly 214, 224, 244, 281, 315, 335, 372
- module test 259
- monitor control settings 575
- monitoring 119
- monitoring agent 113, 562, 565, 578
- Monitoring Agent for IBM Tivoli Monitoring 5.x End-point 564
- Monitoring Agent for Windows OS 687
- monitoring server 560, 562
 - communications protocol 684
- MQ link 484, 507, 515
- MS Exchange Agent 564
- MS SQL Agent 564

N

- namespace 606
- Navigator view 177, 205
- Netegrity SiteMinder 98
- NetView for z/OS 562
- Network infrastructure node 42
- no binding 247
- node
 - start 545
- node agent 537, 545
- Number of Messages by Operation view 570
- Number of Messages by Service - Operation - Type view 571

O

- Object Management Group (OMG) 126
- OMEGAMON Distributed 562
- OMEGAMON XE for WebSphere Business Integration 557
- OMEGAMON z/OS 562
- On Demand Business Transformation 72
- one of N policy 548
- one-way asynchronous message 145
- one-way operation 237–238, 422, 429, 449, 456, 463, 501
- Operating System (OS) Agent 563
- operation 237–238
- operation connections 252
- operation log 577
- Oracle Agent 564
- orchestration 557
- organize imports 425
- Outline view 133
- output 238–239
- output terminal 255

P

- partitioned queue 533
- partner reference 214–215
- Pattern Explorer view 187, 194
- Patterns 127
- Patterns for e-business 27
 - Web site 29
- payload 581
- pcatAIX.bin 487
- pcatHPUX.bin 487
- pcatLinux.bin 487
- pcatLinuxPPC.bin 487
- pcatSolaris.bin 487
- pcatWindows.exe 487
- PDA *See* personal digital assistant
- performance 118, 570
- performance data 560
- performance metric 558
- Performance Summary view 581, 583
- Performance Summary workspace 569–570
- personal digital assistant 39
- perspective 223
- pervasive computing 39
- Physical Resources view 333, 431
- PKI *See* Public Key Infrastructure
- Plain Old Java Objects (POJOs) 102

- point-to-point 459
- point-to-point connection 26
- policies 574
- port 611, 617, 646
- port type 613
- Portal client 561
- Portal Composite pattern 29
- Portal server 561
- Portal server database 561
- preferred server 548–549
- Presentation Tier Transformations 127
- primitive 253–255
- process component 149
- Process Integration 53
- process services 77
- Product mapping 28, 44, 47
- profile 257, 486
- profile creation wizard 486–487, 536
- profile directory 644
- ProfileCreator_wbi 486, 640
- project build 225
- Properties view 133, 224, 235
- protocol 1 684
- protocol 2 684
- protocol conversion 48, 57
- Protocol firewall 41
- protocol transformation 59
- provisioning 557
- Proxy 343, 445
- proxy 341, 415, 475
- Public Key Infrastructure 40

Q

- quality of service qualifiers 215
- queue 192, 195, 200, 471, 473, 502–503
 - alias 474
 - Local 474
- Queue Browser 595
- queue component 190, 192–194, 197–198
- queue connection factory 208, 460, 467, 472, 501, 503, 521, 619–620, 630
- queue destination 207, 467, 501, 531
- queue manager 507–508, 517

R

- RAR file 353, 364
- RAS perspective 128, 130
- RAS repository 128

- Rational Application Developer 105, 108, 481
- Rational design patterns 127
- Rational Product Updater 661, 664
- Rational Software Architect 103–105, 108, 125
 - installation 107
- Rational Software Development Platform 107
- Rational Unified Process (RUP) 126, 130
- Rational Unified Process Analysis profile 127
- receiver channel 507, 512–513, 515, 519, 521
- Redbooks Web site 709
 - Contact us xix
- reference 214
- regenerate implementation 425
- Register Customer process 264, 266, 348, 418, 451
- Register Shipping mediation 163
- Register Shipping scenario 417
- Registration Processor 151
- remote monitoring server seed 686
- remote TEMS 562
- RemotelPAddress 584
- request flow 252, 255, 286–287, 317–318, 374–375, 392, 396
- request/response operation 238, 252, 263, 277, 347, 374, 502, 534
- resource adapter 355
- resource monitoring 556
- response flow 252, 255, 286, 297, 317–318, 322, 374, 463
- Response Time Critical 574
- Response Time Warning 574
- Reusable Asset Specification (RAS) 130
- role-based permission qualifier 213
- root 219, 291, 327, 375, 398
- router 38, 58
- Router application pattern 44, 47, 162, 164
- router connectivity 37
- Router variation of the Broker 38, 58
- routing 152, 434
- routing messages 449
- RSA transformations 127
- Rules Directory node 43
- Runtime pattern 28, 39
- runtime services 77

S

- sampling interval 563
- SCA 159, 213

- Define cluster support 546
- SCA binding 215–216
- SCA message header 220
- SCA module 423, 481
 - View from the administrative console 489
- SCA.APPLICATION..Bus 467–468, 486, 490, 501
- SCA.Application..Bus 467
- SCA.APPLICATION.Bus 482, 595
- sca.module 334
- SCA.SYSTEM..Bus 482, 486
- scalability 118, 480, 531
- scdl
 - module 334
- schema 170, 543–544
- schema stereotype 166
- screening routers 41
- SDO 213, 218
- SDO data graph 219
- SDO DataObject 220
- security 99, 116
 - messaging 117
 - transport 117
 - Web services 483
- Security services 40
- seeding 686
- self
 - node() 377, 464
- Self Service 38
- Self Service business pattern 44
- Self-Service business 29
- Self-Service business pattern 29, 44
- sender channel 507, 511–513, 515, 517, 519, 521–522
 - start 521
- sequence diagram 128–129, 140, 144, 146–147
- servers
 - local test environments 256
 - remote test environments 257
- Servers view 224, 257, 640, 652
- serverStatus 545
- service 144, 616
- service component 151, 212–213
- Service Component Architecture (SCA) 212
- Service Component Architecture (SCA). See also SCA
- Service Component Definition Language (SCDL) 216
- Service Connectivity scenario 11, 21, 31, 38
- Service Data Objects (SDO) 212, 216
- Service Data Objects (SDO). See also SDO
- service definition 611
- service implementation 76
- service integration bus 180–181, 186–187, 190–192, 204, 206, 208, 452, 481, 484, 507
 - create 490
 - foreign bus 514
- Service Integration Bus pattern 188, 190, 192
- Service Integration collaboration 190, 193
- Service interface 213
- Service Inventory attributes 572
- Service Level Agreement (SLA) 557
- Service Level Agreements (SLAs) 558
- service management 78
- Service Management Agent Environment workspace 570
- Service message metric attributes 572
- Service message objects (SMO) 219
- Service message objects (SMO). See also SMO
- service modules 214
- service provider 481
- service reference 213
- service requester 481
- Service Topology view 587
- service wires 213
- serviceDeploy 523–524
- ServiceMessageObject 219
- services 149, 151
- Services Inventory table 569
- Services Inventory view 570
- Services Management Agent Environment 568–569
- Services Management Agent Environment workspace 569, 575
- Services Management Agent workspace 569, 584
- Services Oriented Architecture (SOA) 71
- Services_Inventory 579, 688
- Services_Metrics 688
- services-based implementation 71
- session bean 627
- setup.exe 691
- setupaix.bin 688
- setupSolaris.bin 688
- setupwin32.exe 688
- SI-AddFiltrCntrl 584
- SI-AddMntrlCntrl 582
- SIB_MQ_ENDPOINT_ADDRESS 511
- Siebel 162, 354–356, 359, 372, 379, 386, 389,

- 498–499
- Siebel.jar 386
- SiebelJL_enu.jar 354, 386
- simple business object 219
- single-machine installation 111
- situation 562–563, 573, 688
- skeleton EJB session bean implementation 620
- Skeleton EJB Web Service 621
- SMO 327, 376, 405
- SNA 684, 687
- SOA Foundation Reference Architecture 75, 78, 101
- SOA profile 39
- SOAP 613, 617
- soap
 - address 618
- SOAP transport 228
- SOAP/HTTP 57, 157, 159, 162, 215–216, 263, 265, 268, 284, 372, 482, 632
- SOAP/JMS 57, 163–164, 180, 186, 215–216, 427, 442, 449, 452, 458, 482, 501, 534, 608, 618, 620
- SOAP/JMS URI 617
- stand-alone profile 642
- stand-alone reference 214
- standalone server 111
- standby 521
- startNode 545
- state machine diagram 129
- stateless session bean 610, 627
- stateless session bean binding 215
- static routing 26
- stereotype 148, 166, 170
- Stop primitive 253, 417, 440, 465–466
- Structured Query Language (SQL) 94
- subflow node 94
- subsystem 149
- Summarization and Pruning agent 564
- synchronous 213
- system management 119

T

- Take Action command 575
- Take Action commands 576
- target server 609
- targetNamespace 178
- targetService 618
- TEMS error log file 577
- terminal

- add 376, 434, 463
- default 465
- default 440
- test 303, 385, 394, 412, 442, 444, 467–468
 - mediation module 303
- test client 259
 - starting 303
- Test connection 498
- test environment 116, 256
- threshold 562
- Tivoli Access Manager 14, 98, 116, 485
- Tivoli agent 562
- Tivoli Composite Application Manager 113, 115, 119
- Tivoli Composite Application Manager for SOA 120, 485
- Tivoli Data Warehouse 114, 561, 564
- Tivoli Data Warehouse (TDW) 564
- Tivoli Data Warehouse Proxy Agent 564
- Tivoli Enterprise Management Agent 563
- Tivoli Enterprise Management Agents 560
- Tivoli Enterprise Monitoring 98, 113, 672
- Tivoli Enterprise Monitoring Agent (TEMA) 562
- Tivoli Enterprise Monitoring Agents
 - Install 679
- Tivoli Enterprise Monitoring framework 113
- Tivoli Enterprise Monitoring Server 113–114, 119, 577, 687–688, 699
 - Configuration 682, 684
 - configuration 695
 - Hub 678, 684
 - Install 678
- Tivoli Enterprise Monitoring Server (TEMS) 560
- Tivoli Enterprise Portal 113–114, 485, 565–566, 687
 - Configuration 682
- Tivoli Enterprise Portal client 561
- Tivoli Enterprise Portal clients 561
- Tivoli Enterprise Portal Desktop client 577
- Tivoli Enterprise Portal monitoring agent 568
- Tivoli Enterprise Portal Navigator 567
- Tivoli Enterprise Portal Server 107, 113–114, 119, 561–562, 577, 687–688, 699
- Tivoli Enterprise Portal Server (TEPS) 561
- Tivoli Management Framework 119
- Tivoli Management framework 119
- Tivoli Monitoring Server 107
- top-down Web service 611
- trace log 577–578

- trace logger 582
- tracing 569, 575
- Transaction Flow view 587
- transformation 126–127, 129, 204–205
 - UML to JACL 180
 - UML to XSD 164
- transient context 318–319, 327
- transmission protocol 513
- transmission queue 507, 510, 514

U

- UDDI directory 41
- UML
 - transform to XSD 164
- UML diagram 137, 165
- UML model 135, 187
- UML Model Editor 168
- UML Model template 135
- UML profile 126
- UML project 134
- UML to JACL transformation 204
- UML to XSD transformation 175
- Unified Modeling Language (UML) 126
- Unified Modeling Language 2.0 editor 126
- Universal Agent 562–563
- Universal Discovery Description and Integration (UDDI) 482
- UNIX Log Agent 563
- UNIX OS Agent 563
- updateLogging 571, 575
- updateTracing 571, 575
- UpdMntCntrl 571
- UpdMntrCntrl 575
- upsert 360, 384
- USD 381
- use case diagram 140–141
- use case Model 140
- use case model 137, 144
- User node 39
- user-defined node 94

V

- vertical scaling 531
- view 223, 567
- voice response units (VRUs) 47

W

- Warehouse Interval 579
- Warehouse Proxy 577, 687, 699
- Warehouse proxy 683
- Warehouse Proxy agent 563–564
- Warehouse Summarization and Pruning Agent 564
- Web server redirector 40
- Web service
 - create 620
 - implement 631
- Web service binding 216
- Web Service Bindings folder 232
- Web service client
 - create 632
- Web service client proxy 343
- Web service data logging 581, 583
- Web Service Developer capability 226
- Web service message header 220
- Web service port 246, 314–315, 428, 432, 442
- Web service proxy 341, 415, 445, 475
- Web services Client 91
- Web services client 485
- Web Services Client for C++ 91
- Web Services Description Language (WSDL) 236, 482
- Web Services Explorer 632
- Web Services Gateway 483–484
- Web Services Navigator 581, 583–584
 - Install 698
- Web Services Security (WS-Security) 102
- Web Services Transactions (WS-TX) 102
- WebLogic Server 564
- WebSphere Adapter binding 215–216
- WebSphere Adapter For Flat Files 85
- WebSphere Adapter for JDBC 85
- WebSphere Adapter for PeopleSoft Enterprise 85
- WebSphere Adapter for SAP Applications 85
- WebSphere Adapter for Siebel Business Applications 85
- WebSphere Adapter message header 220
- WebSphere Adapters 49, 347
 - Install 667
- WebSphere Application Server 102
- WebSphere Application Server Network Deployment 107, 110, 480, 482
- WebSphere Enterprise Service Bus 16, 84, 110
- WebSphere Enterprise Service Bus. See also WebSphere ESB
- WebSphere ESB 49, 98, 102, 114

- administration 481
- client support 485
- Installation 109–110
- Introduction 480
- messaging support 484
- Security 116
- Web services support 482
- WebSphere ESB test server 257
- WebSphere Integration Developer 16, 18, 108, 211, 220, 303
 - Assembly 104
 - Closing files 236
 - Install 658
 - Installation 107
 - Installation planning 107
 - Saving files 236
 - starting 220
 - System requirements URL 108
- WebSphere Message Broker 52, 91
- WebSphere MQ 49, 449, 452, 474, 484, 503, 506–508, 521, 557
 - configuration 508
 - foreign bus 514
- WebSphere MQ Explorer 474, 654
- WebSphere MQ link 516, 519, 522
 - create 515
- WebSphere Platform Messaging Patterns 180–181, 183, 187
- WebSphere Process Server 102, 104, 212, 214, 220, 232, 256–257, 426
- WebSphere Studio Application Developer Integration Edition 107
- WebSphere variable 494, 541
- Window OS Agent 563
- Windows OS monitoring 562
- Windows service 646
- wire 284, 289–290, 298, 315, 318, 329, 377, 432, 441, 457
- wiring a mediation flow 255
- Workbench 222
- workload management 118, 531–533
- workspace 220, 224, 229, 566
- workspace clean 225
- workspace preferences 226
- WSAF_SIB 549
- WSDL 278, 284, 313, 431, 458, 606, 610
 - Import 431
- WSDL Editor 612
- WS-I compliance 229

- WS-I Simple SOAP Basic Profile 229
- WS-I SSBP compliance level 229
- WS-Interoperability Basic Profile (WS-I) 102
- WS-Security 483

X

- XML mapping 292, 300, 380, 393, 436
- XML schema 164–165, 178
- XML schema definition 606
- XML transformation 263, 347
- XPath 324–325, 377–378, 405–406, 435
- XPath Expression Builder 435, 465
- XSD 177–178, 611
- XSD Building Blocks package 177
- XSD Model 167
- XSD model 137, 166, 177
- XSD model template 164
- XSD schema stereotype 164
- XSD Transformation 169
- XSDProfile 164, 168
- XSL
 - Generate 300, 303
- XSL style sheet 303, 321, 328, 437
 - Generate 297
- XSL Transformation primitive 275, 289–292, 298, 300, 318, 323, 326, 328, 375, 379, 384, 392, 410, 417, 433, 436
- XSLT function 294, 320, 381, 412
- XSLT Function wizard 330
- XSLT primitive 253, 289–292, 298, 300, 318, 323, 326, 328, 351, 375, 379, 384, 392, 410, 417



Redbooks

Patterns: SOA Foundation Service Connectivity Scenario



Redbooks

Patterns: SOA Foundation Service Connectivity Scenario

**Learn key concepts
and architecture of
the IBM SOA
Foundation**

**Apply patterns to the
Service Connectivity
scenario**

**Service Connectivity
using WebSphere
ESB**

The IBM SOA Foundation is a reference architecture used to build new applications or extend the value of existing applications and business processes. The IBM SOA Foundation includes an integration architecture, best practices, patterns, and SOA scenarios to help simplify the packaging and use of IBM open standards-based software.

A set of SOA scenarios is being developed by IBM that describe key architectural scenarios for SOA solutions and bridge the gap between SOA and the IBM products that can be used to implement these architectures.

This IBM Redbook focuses on the Service Connectivity scenario, which describes architectural solutions using an ESB. The focus of this scenario is the integration of service consumers and service providers across multiple channels.

**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