WebSphere software

IBM

Patterns: Building Serial and Parallel Processes for IBM WebSphere Process Server V6

- Develop business process solutions using IBM WebSphere Integration Developer V6
- Follow design, development, and runtime guidelines
- Learn by going through practical scenarios

Martin Keen
Andy Garratt
Ed Grossman
Dennis Stilhoff
Torsten Wilms
Dan Zrobok

Redbooks

ibm.com/redbooks

Archived

Archived

International Technical Support Organization

Patterns: Building Serial and Parallel Processes for IBM WebSphere Process Server V6

April 2006

Archived

SG24-7205-00

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

Archived

First Edition (April 2006)

This edition applies to IBM WebSphere Process Server V6.0.1 and IBM WebSphere Integration Developer V6.0.1.

© 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 xi
Trademarks xii

Preface xiii
The team that wrote this redbook xiii
Become a published author xvi
Comments welcome xvi

Part 1. Patterns for e-business and business processes 1

Chapter 1. Welcome to this redbook 3
1.1 An introduction to this document 4
1.2 Patterns for e-business SOA series of redbooks 4
1.3 How to read this redbook 5

Chapter 2. Introduction to the Patterns for e-business 9
2.1 The Patterns for e-business layered asset model 10
2.2 How to use the Patterns for e-business 12
2.2.1 Selecting a Business, Integration, or Composite pattern, or a Custom design 12
2.2.2 Selecting Application patterns 16
2.2.3 Reviewing Runtime patterns 18
2.2.4 Reviewing product mappings 21
2.2.5 Reviewing guidelines and related links 22
2.3 Patterns for e-business naming conventions 22
2.4 Summary 23

Chapter 3. Introduction to business processes 25
3.1 Overview of business process management 26
3.2 Overview of SOA 28
3.2.1 Definition of a service 29
3.2.2 Web services and SOA 31
3.2.3 The advantages of SOA 32
3.2.4 SOA summary 32
3.3 WS-BPEL V2.0 standard 33
3.3.1 Partners 34
3.3.2 Endpoint references 34
3.3.3 Activities 34
3.3.4 Data handling 36

Archived

© Copyright IBM Corp. 2006. All rights reserved.

3.3.5 Correlation . . . 37
3.3.6 Scope . . . 37
3.4 Service Component Architecture . . . 39
3.4.1 Anatomy of SCA . . . 39

Chapter 4. Application Integration and Extended Enterprise patterns . . . 47
4.1 Application Integration pattern . . . 48
4.1.1 Process-focused Application Integration patterns . . . 49
4.1.2 Serial Process application pattern . . . 53
4.1.3 Parallel Process pattern . . . 56
4.1.4 Serial Workflow variation . . . 58
4.1.5 Parallel Workflow variation . . . 60
4.2 Extended Enterprise pattern . . . 62
4.2.1 Extended Enterprise application patterns . . . 62
4.2.2 Exposed Serial Process application pattern . . . 67
4.2.3 Exposed Serial Workflow pattern . . . 69

Chapter 5. Runtime patterns . . . 73
5.1 Application Integration runtime patterns . . . 74
5.1.1 Node types . . . 75
5.1.2 Serial Process pattern . . . 80
5.1.3 Serial Workflow variation . . . 82
5.1.4 Parallel Process pattern . . . 84
5.1.5 Parallel Workflow variation . . . 86
5.2 Extended Enterprise runtime patterns . . . 87
5.2.1 Exposed Serial Process pattern . . . 89
5.2.2 Exposed Serial Workflow pattern . . . 90

Chapter 6. Product descriptions . . . 93
6.1 Runtime products . . . 94
6.1.1 IBM WebSphere Process Server V6.0 . . . 94
6.1.2 IBM DB2 Universal Database Enterprise Server Edition V8.2 . . . 96
6.1.3 IBM Cloudscape . . . 96
6.1.4 IBM WebSphere Adapter for JDBC V6.0 . . . 97
6.1.5 IBM WebSphere MQ V5.3 . . . 98
6.2 Development products . . . 99
6.2.1 IBM WebSphere Integration Developer V6.0 . . . 99

Chapter 7. Product mappings . . . 101
7.1 Application Integration product mappings . . . 102
7.1.1 Serial Process product mapping . . . 102
7.1.2 Parallel Process product mapping . . . 104
7.1.3 Parallel Workflow product mapping . . . 105
7.2 Extended Enterprise product mappings . . . 106

Archived

7.2.1 Exposed Serial Process product mapping 106

Part 2. Business scenario and guidelines 109

Chapter 8. Business scenarios used in this book 111
8.1 Customer overview 112
8.1.1 Business profile 112
8.1.2 Existing environment 113
8.1.3 Non-functional requirements 114
8.2 Scenario overview 115
8.2.1 Scenario 1: Internal ordering on demand 117
8.2.2 Scenario 2: Internal ordering on demand in parallel 120
8.2.3 Scenario 3: Ordering on demand with multiple wholesalers 124
8.2.4 Scenario 4: Ordering on demand with approval workflow 127

Chapter 9. Technology descriptions 131
9.1 Service Component Architecture 132
9.2 Service Data Objects 132
9.3 J2EE Connector Architecture 133
9.4 Web services 134
9.4.1 XML Schema Definition 136
9.4.2 Web Services Description Language 137
9.4.3 SOAP 137
9.4.4 Universal Description, Discovery, IntegrationI 138
9.4.5 Web Services Business Process Execution Language 139
9.4.6 WS-Security 139
9.5 J2EE 140
9.5.1 Java Message Service 141
9.5.2 Web services for J2EE 141
9.5.3 Java API for XML-based RPC 141
9.6 Transport protocols 142
9.6.1 HTTP 142
9.6.2 HTTPS 142

Part 3. Scenario implementation 143

Chapter 10. Serial Process scenario 145
10.1 Business scenario 146
10.2 Design guidelines 147
10.2.1 Vocabulary 148
10.2.2 Analysis of the business requirements 148
10.2.3 Pattern selection 149
10.2.4 Analysis of the design options 153
10.3 Development options 158

10.4 Development guidelines 162
10.4.1 Development plan 164
10.4.2 Setting up the workspace 164
10.4.3 Creating the business logic for ITSO Electronics 171
10.4.4 Creating supporting services for the business process. 215
10.5 Runtime guidelines 229

Chapter 11. Exposed Serial Process scenario 235
11.1 Business scenario 236
11.2 Design guidelines 238
11.2.1 Analysis of the business requirements 238
11.2.2 Pattern selection 239
11.2.3 Analysis of the design options. 243
11.3 Development guidelines 249
11.3.1 Development plan 251
11.3.2 Setting up the workspace 252
11.3.3 Creating the business state machine 253
11.3.4 Creating the business rule BSMChooseWh 280
11.3.5 Creating the ITSOElectronics assembly diagram 284
11.3.6 Creating the Wholesaler service providers 285
11.3.7 Linking WholesalerD and WholesalerE to the state machine 298
11.3.8 Creating the Exposed Serial business process. 302
11.3.9 Linking the business process in the assembly diagram 308
11.3.10 Adding the ITSO_CheckDate business rule 310
11.3.11 Exposing the business process as a Web service 311
11.4 Runtime guidelines 312
11.4.1 Configuring an HTTP server for SSL pass-through 312
11.4.2 Using WS-Security 318
11.4.3 Testing the application 319

Chapter 12. Parallel Process scenario 331
12.1 Business scenario 332
12.2 Design guidelines 333
12.2.1 Analysis of the business requirements 334
12.2.2 Pattern selection 334
12.2.3 Analysis of the design options. 337
12.3 Development guidelines 348
12.3.1 Setting up the workspace 348
12.3.2 Creating the Parallel Process module. 348
12.3.3 Constructing a solution in the Assembly Editor. 350
12.3.4 Creating the component implementations. 356
12.3.5 Configuring the JDBC adapter with the business process 375
12.4 Runtime guidelines 388

Archived

12.4.1 Configuring DB2 as Business Process Choreography database . 388
12.4.2 Configuring the WebSphere Adapter for JDBC. 389
12.4.3 Testing the application 389

Chapter 13. Parallel Workflow scenario 393
13.1 Business scenario 394
13.2 Design guidelines 395
13.2.1 Analysis of the business requirements 395
13.2.2 Pattern selection 395
13.2.3 Analysis of the design options. 399
13.3 Development guidelines 404
13.3.1 Setting up the workspace 406
13.3.2 Creating the business objects 407
13.3.3 Creating the interfaces 409
13.3.4 Building the PlaceOrderProcess Parallel Workflow process. 410
13.3.5 Building the ITSOPModule assembly diagram 466
13.4 Runtime guidelines 484
13.4.1 Adding the required modules to the test server 484
13.4.2 Testing the Parallel Workflow process 485
13.4.3 Testing the Parallel Workflow compensation and fault handling. . 488
13.4.4 Testing the Parallel Workflow event handling 491
13.4.5 Testing Parallel Workflow ApproveScheduleDate Human Task . . 494

Part 4. Appendixes 499

Appendix A. Building Wholesaler A and B service implementations . . 501
Scenario implementation overview. 502
Creating the service implementation for Wholesaler A 504
Creating business objects for the WhDelReqWeb_I interface 506
Creating the WhDelReqWeb_I interface 512
Creating the business objects for the remaining interfaces 516
Creating the remaining interfaces 519
Creating the business logic 521
Creating the assembly diagram 546
Examining the generated WSDL files 551
Testing Wholesaler A 552
Creating the service implementation for Wholesaler B 556
Copying the business objects and interfaces from WholesalerA 558
Creating the business logic using the assembly diagram 558

Appendix B. Deploying and testing modules 571
Deployment 572
Starting run time and deploying modules to the test environment using the Integration Test Client 572

Archived

Adding and removing projects to the server 575
Installing an EAR file using the administrative console 577
Testing 578
The Integration Test Client 579
The Business Process Container Explorer 586
Managing deployed components 590
The Business Rules Manager 590
The administrative console for selectors 594
The Common Base Event Event Browser 596
The Failed Event Manager 599
The Relationship Manager 602
Configuring the server 603
Security 603
WebSphere Adapter for JDBC 618
Debugging 623
Starting the server in debug mode 623
Adding breakpoints to components 624

Appendix C. Relationships and cross-referencing 627
Relationships: What they do and how they do it 628
Example usage of relationships 630
Using relationships 632
Setting up the workspace 633
Creating the ITSOItems relationship 634
Deploying the relationship 636
Populating the relationship data 637
Creating the cross-reference data mapping: Delivery days 645
Creating the interface map: Delivery days 651
Creating the cross-reference data mapping: Place order 652
Creating the interface map: Place order 654
Modifying the ITSOElectronics assembly diagram 656
Testing relationships 658

Appendix D. WebSphere Process Server dynamicity 661
Scenario implementation overview 662
Preparing the workspace 663
Creating the service implementation for Wholesaler C 663
Creating the Wholesaler C module 663
Copying the business objects and interfaces from WholesalerA 664
Creating the business logic using the assembly diagram 664
Creating the implementation for Wholesaler C mediator 671
Creating the Wholesaler C mediator module 672
Copying the Web service exports from WholesalerC 672

Creating the mediator module using the assembly diagram 673
Replacing WholesalerA with WholesalerC at run time 679
Adding the required modules to the test server 679
Changing the selector targets in the test server admin console 679
Testing that WholesalerA has been replaced by WholesalerC 683
Exporting the selector changes to WebSphere Integration Developer . . . 684

Appendix E. Additional material 689
Locating the Web material 689
Using the Web material 689
System requirements for downloading the Web material 690
How to use the Web material 690

Abbreviations and acronyms 691

Related publications 693
IBM Redbooks 693
Other publications 693
Online resources 693
How to get IBM Redbooks 695
Help from IBM 695

Index 697

Archived

Archived

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

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

© Copyright IBM Corp. 2006. All rights reserved.

Trademarks

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

- Cloudscape™
- CICS®
- Distributed Relational Database Architecture™
- Domino®
- DB2 Connect™
- DB2 Universal Database™
- DB2®
- DRDA®
- @server®
- @server®
- IBM®
- IMS™
- iSeries™
- Lotus®
- pSeries®
- Redbooks™
- Redbooks (logo) ™
- Tivoli®
- WebSphere®

The following terms are trademarks of other companies:

Enterprise JavaBeans, EJB, Java, JavaBeans, JavaServer, JavaServer Pages, JDBC, JSP, JVM, J2EE, J2SE, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, 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.

Archived

Preface

IBM® WebSphere® Process Server, and the accompanying IBM WebSphere Integration Developer development tool, provide a vast array of powerful features for building business process solutions. But how can you use those features for your organization? This IBM Redbook takes a scenario-based approach to demonstrate these product features and employs the IBM Patterns for e-business to illustrate proven business process patterns.

Part 1 of this redbook introduces the Patterns for e-business. The Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying On Demand Business applications. This redbook focuses on the Serial and Parallel Process patterns from the asset catalog.

Part 2 describes the business scenario used in this book. It also explains the key technologies that are relevant to the scenarios.

Part 3 guides you through the building of business process solutions in WebSphere Integration Developer for deployment to WebSphere Process Server. It includes the use of Web Services Business Process Execution Language (WS-BPEL) business processes, business state machines, and many other product features. Four solutions are described, each adhering to a separate pattern. Each solution describes the architectural considerations for designing the solution and step-by-step instructions for building the scenario.

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 (ITSO), Raleigh Center.

Martin Keen is a Senior IT Specialist at the ITSO, Raleigh Center. He writes extensively about WebSphere products, service-oriented architecture (SOA), and Patterns for e-business. He also teaches IBM classes worldwide about WebSphere, SOA, and business process management. Before joining the ITSO, Martin worked in the EMEA WebSphere Lab Services team in Hursley, United Kingdom (U.K.). Martin holds a bachelor degree in computer studies from Southampton Institute of Higher Education.

© Copyright IBM Corp. 2006. All rights reserved.

Andy Garratt is an Integration Consultant at IBM Software Services for WebSphere (ISSW) based in the U.K. He has fourteen years of experience in software architecture, design and development working on client sites. His areas of expertise include WebSphere integration software, SOA, business process integration and enterprise resource planning (ERP) systems. Before joining IBM, Andy worked for a number of IBM Business Partners implementing large ERP systems across most of Europe. Andy holds a degree in natural sciences from the University of Durham.

Ed Grossman is a Technical Enablement Architect based in Austin, Texas, although he spends a lot of time working with independent software vendors (ISVs) across the United States. He has over 20 years of experience in software architecture, design, and development. Prior to joining IBM two years ago, he held various technical roles at small companies and a few universities. His areas of expertise include Java™ 2 Platform, Enterprise Edition (J2EE™), business process integration, and the use of the WebSphere platform by software vendors. Ed has a master degree in computer science from the University of Washington.

Dennis Stilhoff is a Consulting IT Specialist at IBM Global Business Services in Copenhagen, Denmark. He is an IBM Certified IT Specialist and has 16 years of experience in application architecture, design, and development. His areas of expertise include J2EE, business process integration, WebSphere Process Server, SOA, and Web services. Dennis holds a degree in business administration and mathematics from the Copenhagen Business School, Denmark.

Torsten Wilms is an IT Specialist for IBM Software Group Germany. Since 2005, he has worked within the WebSphere Solution Center in the development laboratory in Boeblingen, Germany, where he provides expert knowledge about the newest versions of WebSphere Process Server. After obtaining his degree in Business Information Systems, he joined IBM in 2001 and started at the AMS World Wide Beta Test Site, which is part of the IBM SAP Customer Competence Center. He also has experience with SAP and IBM WebSphere integration scenarios. Torsten coauthored the redbook *WebSphere Business Integration for SAP*, SG24-6354, published in 2005.

Dan Zrobok is a Staff Software Developer from the IBM Toronto Lab. He has been a member of the WebSphere Studio Application Developer Integration Edition and WebSphere Integration Developer development team for five years. For version 6.0, he was a user interface developer on the Integration Test Client team. Dan holds a degree in software engineering from McMaster University.

The redbook team (left to right): Martin, Torsten, Dennis, Ed, Dan, and Andy

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

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

Sunita Chacko, Scenario Development, Toronto Lab
Peter Cinat, Software Developer, WebSphere Integration Developer
IBM Canada

Shane Claussen, WebSphere Business Integration Architect
Rashmi Hunt, Software Engineer
IBM Software Group, IBM US

Søren Kristiansen, IT Architect, IBM Sales & Distribution, FSS
IBM Denmark

Khirallah Birkler, BPTS, WebSphere SAP Technology Team
Thomas Kasemir, Manager, WebSphere Lab-based Services
Dr. Hans-Joachim Novak, IT Specialist,
Workflow and Process Choreographer Services
Gerd Watmann, Advisory IT Specialist, WebSphere Lab-based Services
IBM Germany

Archived

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an email to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HZ8 Building 662
  P.O. Box 12195
  Research Triangle Park, NC 27709-2195

Archived

Part 1

Patterns for e-business and business processes

Archived

© Copyright IBM Corp. 2006. All rights reserved.

Archived

1

Welcome to this redbook

This chapter introduces this redbook to you and provides guidelines for how to read it. It also lists the series of IBM Redbooks that cover the Patterns for e-business service-oriented architecture (SOA).

Archived

© Copyright IBM Corp. 2006. All rights reserved.

1.1 An introduction to this document

This IBM Redbook focuses on patterns for building business process solutions in WebSphere Integration Developer V6 for deployment to WebSphere Process Server V6. Specifically it explores the Serial Process and Parallel Process patterns (and their variations) from the Patterns for e-business.

This book was designed with IT architects, system administrators, and application programmers in mind. Throughout this book, we discuss a number of architectural patterns for building business process scenarios and highlight the design decisions in building these scenarios. We also provide step-by-step instructions on how to implement these scenarios in WebSphere Integration Developer and WebSphere Process Server.

The team that wrote this IBM Redbook assembled for six intense weeks in Raleigh, North Carolina, and then again remotely through the review process, to put together this resource. We hope you find this IBM Redbook to be a useful read.

1.2 Patterns for e-business SOA series of redbooks

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

The Patterns for e-business contain several patterns and product mappings for SOA solutions, from simple point-to-point solutions to complex Extended Enterprise solutions using an enterprise service bus (ESB). A series of Redbooks has been created to describe these SOA patterns and to provide product mapping implementation. This redbook is part of this SOA series.

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

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

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

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

  This book provides a more in-depth description of SOA and Web services technologies and introduces the SOA concept of the ESB. It expands the

Patterns for e-business SOA profile to provide ESB guidelines. This book also provides scenario chapters to show ESB implementations that are created in WebSphere Application Server V5.1 and WebSphere Business Integration Message Broker V5. An additional scenario chapter describes how WebSphere Business Integration Server Foundation V5.1 can interact with an ESB.

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

  This book extends the SOA Patterns for e-business introduced in *Patterns: Implementing an SOA using an Enterprise Service Bus*, SG24-6346, with product mappings for WebSphere Application Server V6.

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

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

- *Patterns: Extended Enterprise SOA and Web Services*, SG24-7135

  This book addresses SOA solutions that span enterprises. Extended Enterprise patterns are discussed, and step-by-step guides to building solutions are provided for WebSphere Application Server V6, WebSphere Partner Gateway V6, and WebSphere Business Integration Server V5.1.

1.3 How to read this redbook

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

Part 1, “Patterns for e-business and business processes”

This part introduces the Patterns for e-business, and specifically the patterns for designing business process solutions. It is primarily intended for IT architects.

- Chapter 1, “Welcome to this redbook” on page 3
- Chapter 2, “Introduction to the Patterns for e-business” on page 9

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

- Chapter 3, “Introduction to business processes” on page 25

  This chapter introduces business process automation and discusses implementation of the process in WebSphere Process Server. It provides a brief overview of SOA, a description of Web Services Business Process Execution Language (WS-BPEL, the SOA standard for business process description), and an overview of IBM’s proposed standard for integrating and composing services called *Service Component Architecture*.

- Chapter 4, “Application Integration and Extended Enterprise patterns” on page 47

  This chapter describes the Application Integration pattern and Extended Enterprise business pattern from the Patterns for e-business. Then it discusses Application patterns as high level patterns that help describe common interactions. It focuses on business process interactions for both intra- and inter-enterprise.

- Chapter 5, “Runtime patterns” on page 73

  Runtime patterns describe the logical architecture required to implement an Application pattern. This chapter describes the Serial and Parallel Process runtime patterns for both intra- and inter-enterprise solutions.

- Chapter 6, “Product descriptions” on page 93

  This chapter describes the IBM products discussed and implemented within this redbook to implement the business process scenarios.

- Chapter 7, “Product mappings” on page 101

  Product mappings are typical and proven implementations of a Runtime pattern, using IBM products. This chapter highlights product mappings for each Runtime pattern, as well as the WebSphere Process Server features in each of these product mappings. These product mappings are implemented in Part 3, “Scenario implementation” of this redbook.

Part 2, “Business scenario and guidelines”

This redbook provides four business process scenario implementations, all using WebSphere Process Server V6. Each of these scenarios uses a common business scenario case study and a collection of common technologies.

- Chapter 8, “Business scenarios used in this book” on page 111

  This chapter describes the business scenario used throughout Part 3, “Scenario implementation” on page 143, of this redbook. The business scenario describes a fictional company called *ITSO Electronics* that places orders for electrical parts from a variety of wholesalers. Wholesalers are located both within the ITSO Electronics organization and externally.

Archived

- Chapter 9, “Technology descriptions” on page 131

  This chapter introduces the technologies required to implement business processes in WebSphere Process Server.

Part 3, “Scenario implementation”

Four business process scenarios are implemented in this section, based on the Runtime patterns described in Chapter 5, “Runtime patterns” on page 73, and the product mappings described in Chapter 7, “Product mappings” on page 101.

Each scenario chapter is divided into three distinct parts:

- Design guidelines

  This section is primarily intended for architects. It describes the design alternatives that you should consider when designing a particular scenario.

- Development guidelines

  This section is primarily intended for application developers. It explains the application development steps in WebSphere Integration Developer that are required when implementing a particular scenario.

- Runtime guidelines

  This section is primarily intended for system administrators. It presents the steps to deploy, configure, and test the scenario in WebSphere Process Server.

This redbook contains the following scenario chapters:

Note: For those of you who have no previous experience using WebSphere Integration Developer, we recommend that you read Appendix A, “Building Wholesaler A and B service implementations” on page 501, before you proceed with the implementation of any of the scenarios in Part 3, “Scenario implementation” of this redbook.

- Chapter 10, “Serial Process scenario” on page 145

  This chapter describes how to build a simple business process solution in WebSphere Integration Developer and test in WebSphere Process Server. It includes the step-by-step development of a WS-BPEL process, business objects and interfaces, business rules, and visual snippets.

- Chapter 11, “Exposed Serial Process scenario” on page 235

  This chapter expands on the features used in the Serial Process scenario to include the use of a business state machine. It explains how to securely configure Web service calls to communicate with external enterprises.

Archived

- Chapter 12, “Parallel Process scenario” on page 331

  This chapter explains how to build a process with parallel paths of execution, using the top-down development model. It also demonstrates the use of the WebSphere Adapter for Java Database Connectivity (JDBC™) and transactional considerations.

- Chapter 13, “Parallel Workflow scenario” on page 393

  This chapter expands the Parallel Process scenario to introduce human interaction. It also demonstrates the use of compensation, faults, selectors, event handling, and Common Event Infrastructure (CEI) monitoring.

Appendixes

The appendixes provide step-by-step instructions for using other features of WebSphere Integration Developer and WebSphere Process Server within the context of the ITSO Electronics scenario.

- Appendix A, “Building Wholesaler A and B service implementations” on page 501

  This appendix describes how to build the first two wholesaler service implementations. It serves as an introduction to development in WebSphere Integration Developer. It provides detailed step-by-step instructions of common tasks such as the development of a simple WS-BPEL business process, visual and Java snippets, business rules, and Web services imports and exports.

- Appendix B, “Deploying and testing modules” on page 571

  This appendix describes how to deploy, test, and manage applications running in WebSphere Process Server. It also describes how to configure WebSphere Process Server for security and the JDBC J2EE Connector Architecture adapter.

- Appendix C, “Relationships and cross-referencing” on page 627

  This appendix shows how you can use the relationships feature of WebSphere Process Server to cross-reference data identifiers between different systems and services.

- Appendix D, “WebSphere Process Server dynamicity” on page 661

  This appendix explains how to use the dynamicity feature of WebSphere Process Server to dynamically replace one wholesaler with another without changing the WS-BPEL business process that invokes the wholesalers.

- Appendix E, “Additional material” on page 689

  This appendix direct you to the additional material, located on the Web, that accompanies this redbook.

Archived

2

Introduction to the Patterns for e-business

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

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

The IBM Patterns for e-business help facilitate this reuse of assets. Their purpose is to capture and publish e-business artifacts that have been used, tested, and proven to be successful. The information captured by them is presumed to fit the majority, or 80/20, situation. The IBM Patterns for e-business are further augmented with guidelines and related links.

© Copyright IBM Corp. 2006. All rights reserved.

2.1 The Patterns for e-business layered asset model

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

- Business patterns that identify the interaction between users, businesses, and data
- Integration patterns that tie multiple Business patterns together when a solution cannot be provided based on a single Business pattern
- Composite patterns that represent commonly occurring combinations of Business patterns and Integration patterns
- Application patterns that provide a conceptual layout that describe how the application components and data within a Business pattern or Integration pattern interact
- Runtime patterns that define the logical middleware structure that supports an Application pattern

  Runtime patterns depict the major middleware nodes, their roles, and the interfaces between these nodes.
- Product mappings that identify proven and tested software implementations for each Runtime pattern
- Best-practice guidelines for design, development, deployment, and management of e-business applications

Archived

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

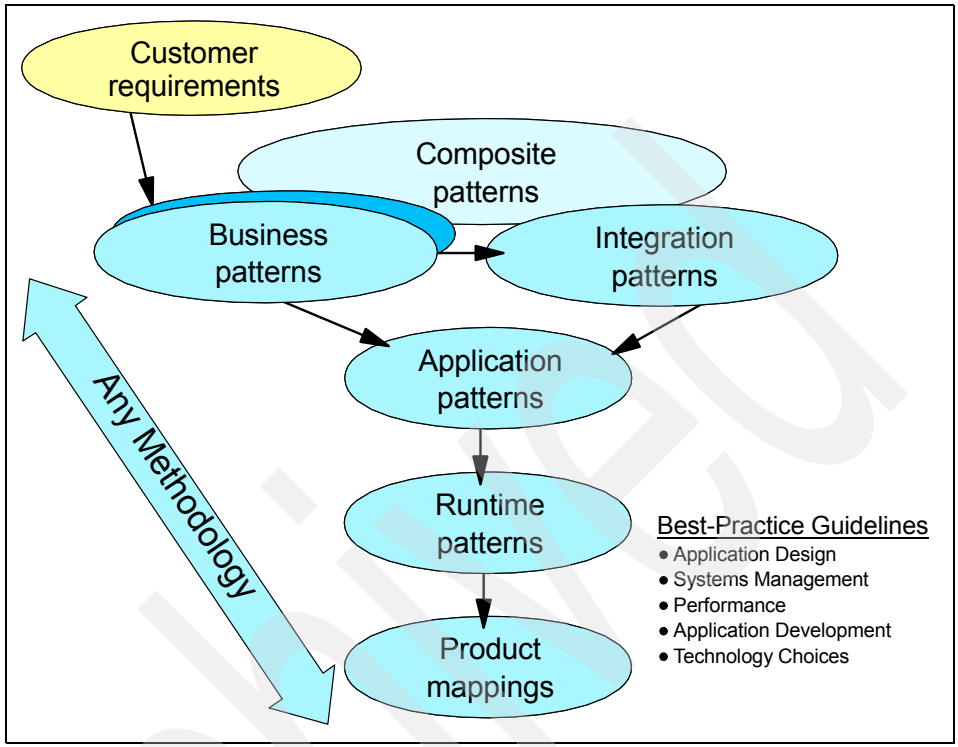


Figure 2-1 The Patterns for e-business layered asset model

Patterns for e-business Web site

The layers of patterns, along with their associated links and guidelines, allow the architect to start with a problem and a vision for the solution and then find a pattern that fits that vision. To navigate from top down from one level to another, a decision matrix is 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 need 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 to navigate through the layered Patterns assets to determine the most appropriate assets for a particular engagement. For easy reference, see the Patterns for e-business Web site at:

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

2.2 How to use the Patterns for e-business

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

Composite patterns appear in the hierarchy shown in Figure 2-1 above the Business patterns. However, Composite patterns are made up of a number of individual Business patterns and at least one Integration pattern. This section discusses how to use the layered structure of Patterns for e-business assets.

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

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

For example, suppose an insurance company wants to reduce the amount of time and money spent on call centers that handle customer inquiries. By allowing customers to view their policy information and request changes online, the company can cut back significantly on the resources that are spent handling this type of request by phone. The objective allows policy holders to view policy information that is stored in existing databases.

The Self-Service business pattern fits this scenario perfectly. You can use it in situations where users need direct access to business applications and data. The following sections discuss the available Business patterns.

Archived

Business patterns

A Business pattern describes the relationship between the users, the business organizations or applications, and the data to be accessed. Table 2-1 explains the four primary Business patterns.

Table 2-1 The four primary Business patterns

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

It is convenient when all problems fit well into these four slots, but reality says that things are often more complicated. The patterns presume that most problems, when broken down into their basic components, fit more than one of these patterns. When a problem requires multiple Business patterns, you can use Integration patterns.

Archived

Integration patterns

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

Table 2-2 Integration patterns

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

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

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

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

Custom design

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

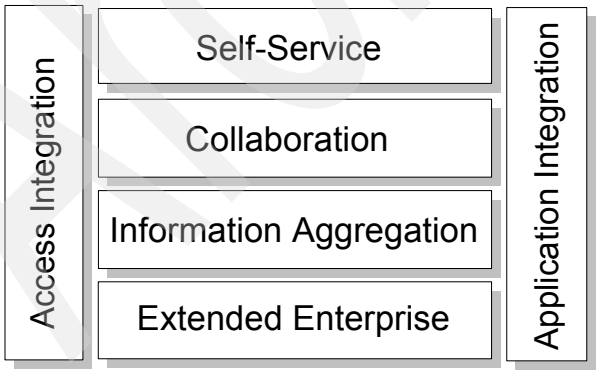


Figure 2-2 Patterns representing a Custom design

If you do not use any of the Business or Integration patterns in a Custom design, you can show the unused patterns as lighter blocks than those patterns that you

do use. For example, Figure 2-3 shows a Custom design that does not have a Collaboration or an Extended Enterprise business pattern for a business problem.

Figure 2-3 Custom design showing unused patterns

If a Custom design recurs many times across domains that have similar business problems, then it can also be a Composite pattern. For example, the Custom design in Figure 2-3 can also describe a Sell-Side Hub Composite pattern.

Composite patterns

Several common uses of Business and Integration patterns have been identified and formalized into Composite patterns. Table 2-3 shows the identified Composite patterns.

Table 2-3 Composite patterns

Composite Patterns	Description	Examples
Electronic Commerce	This pattern enables user-to-online buying.	http://www.macys.com http://www.amazon.com
Portal	This pattern is typically designed to aggregate multiple information sources and applications to provide uniform, seamless, and personalized access for its users.	• Enterprise intranet portal providing self-service functions such as payroll, benefits, and travel expenses • Collaboration providers who provide services such as e-mail or instant messaging
Account Access	Through this pattern, customers have around-the-clock account access to their account information.	• Online brokerage trading applications • Telephone company account manager functions • Bank, credit card, and insurance company online applications

Archived

Composite Patterns	Description	Examples
Trading Exchange	With this pattern, buyers and sellers trade goods and services on a public site.	• Buyer's side: Interaction between a buyer's procurement system and commerce functions of e-Marketplace • Seller's side: Interaction between the procurement functions of the e-Marketplace and its suppliers
Sell-Side Hub (supplier)	The seller owns the e-Marketplace and uses it as a vehicle to sell goods and services on the Web.	http://www.carmax.com (car purchase)
Buy-Side Hub (purchaser)	The buyer of the goods owns the e-Marketplace and uses it as a vehicle to leverage the buying or procurement budget in soliciting the best deals for goods and services from prospective sellers across the Web.	http://www.wwre.org (WorldWide Retail Exchange)

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

2.2.2 Selecting Application patterns

After you identify the Business pattern, the next step is to define the high-level logical components that make up the solution and how these components interact. This is known as the *Application pattern*. A Business pattern typically has multiple possible Application patterns. An Application pattern might have logical components that describe a presentation tier for interacting with users, an application tier, and a back-end application tier.

Archived

Application patterns break down the application into the most basic conceptual components that identify the goal of the application. In our example, the application falls into the Self-Service business pattern. The goal is to build a simple application that allows users to access back-end information. Figure 2-4 shows the Self-Service::Directly Integrated Single Channel application pattern, which fulfills this requirement.

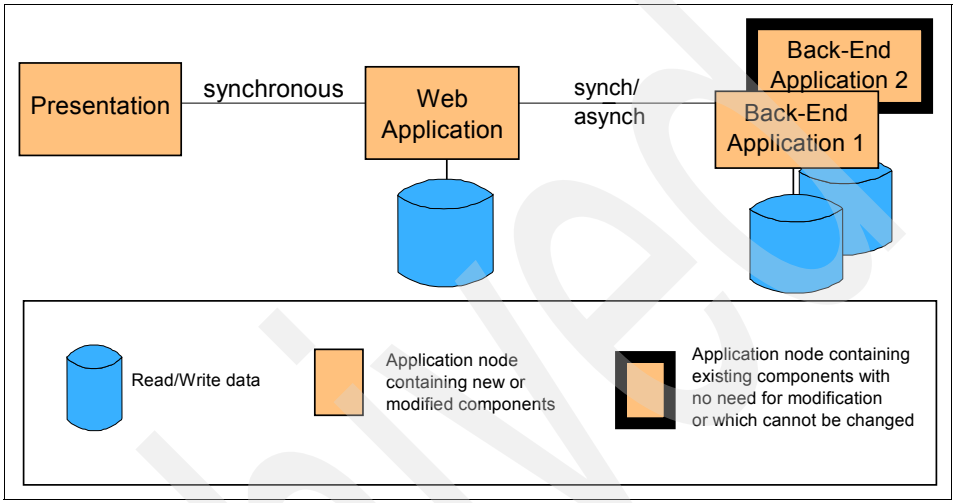


Figure 2-4 Self-Service::Directly Integrated Single Channel pattern

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

Archived

Suppose that the situation is more complicated. Let's say that the automobile policies and the homeowner policies are kept in two separate and dissimilar databases. The user request needs data from multiple, disparate back-end systems. In this case, there is a need to break the request down into multiple requests (decompose the request) to be sent to the two different back-end databases, then to gather the information that is sent back from the requests, and put this information into the form of a response (recompose). In this case, the Self-Service::Decomposition application pattern (as shown in Figure 2-5) is more appropriate.

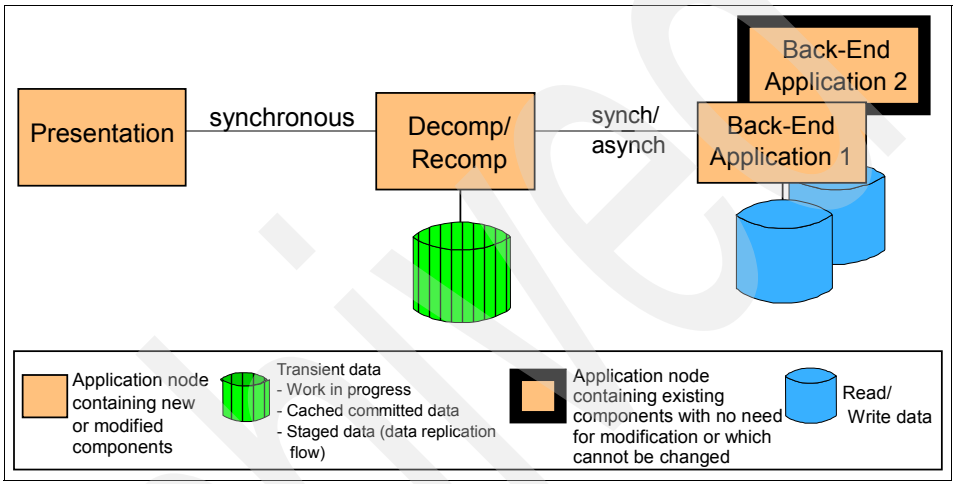


Figure 2-5 Self-Service::Decomposition pattern

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

2.2.3 Reviewing Runtime patterns

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

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

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

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

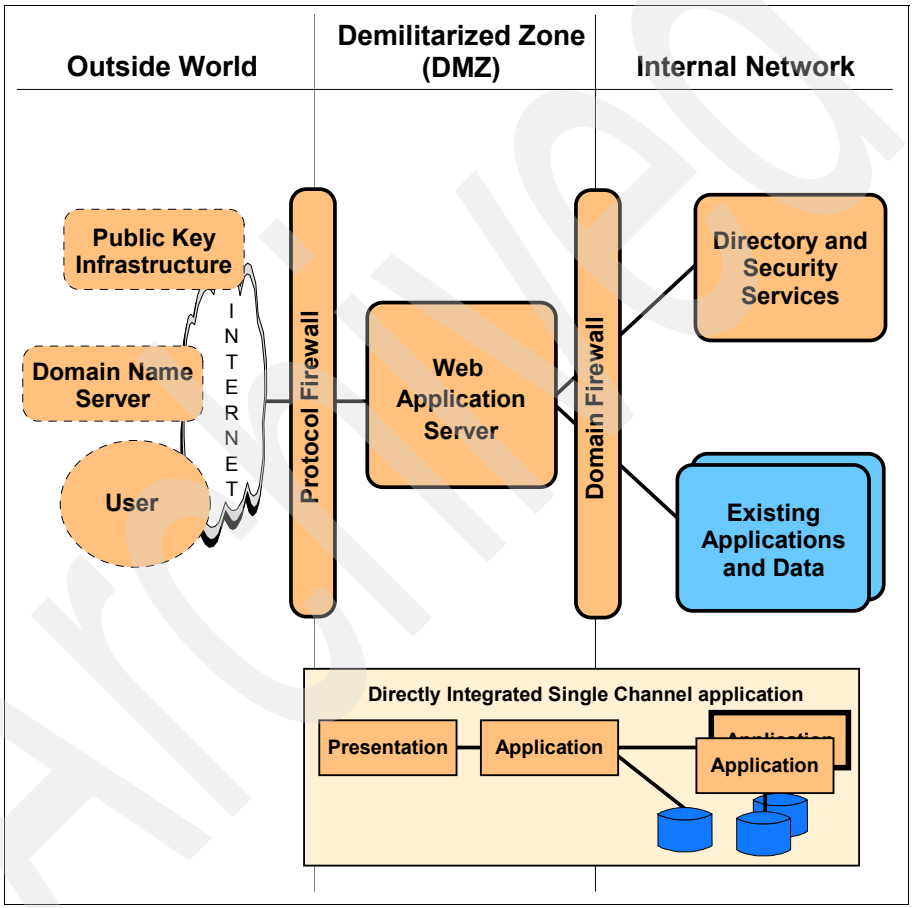


Figure 2-6 Directly Integrated Single Channel application pattern::Runtime pattern

By overlaying the Application pattern on the Runtime pattern, you can see the roles that each functional node fulfills in the application. The presentation and application tiers are implemented with a Web application server, which combines

the functions of an HTTP server and an application server. The Application pattern handles both static and dynamic Web pages.

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

A characteristic that makes this Runtime pattern different from others is the placement of the Web application server between the two firewalls. Figure 2-7 shows a variation of this pattern. It splits the Web application server into two functional nodes by separating the HTTP server function from the application server. The HTTP server (Web server redirector) provides static Web pages and redirects other requests to the application server. This pattern moves the application server function behind the second firewall, adding further security.

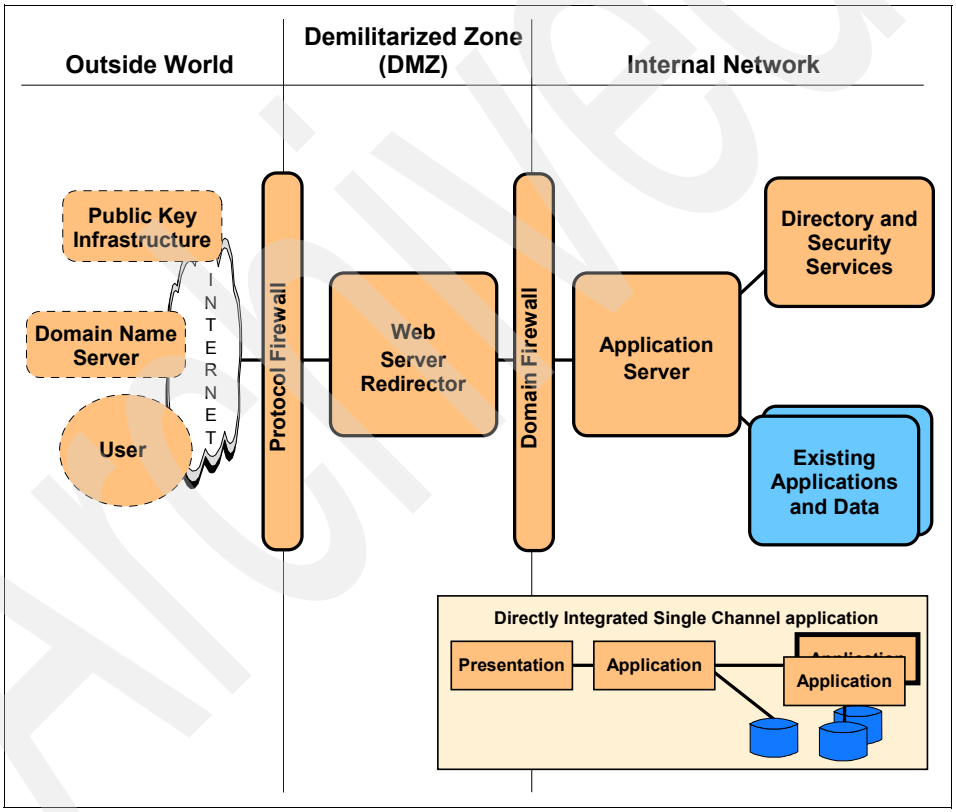


Figure 2-7 Directly Integrated Single Channel application pattern::Runtime pattern

These are just two examples of the possible Runtime patterns that are available. Each Application pattern hs one or more Runtime patterns defined. You can modify these Runtime patterns to suit the customer's needs. For example, the

customer might want to add a load-balancing function and multiple application servers.

2.2.4 Reviewing product mappings

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

For example, you can implement the runtime variation in Figure 2-7 using the product mapping depicted in Figure 2-8.

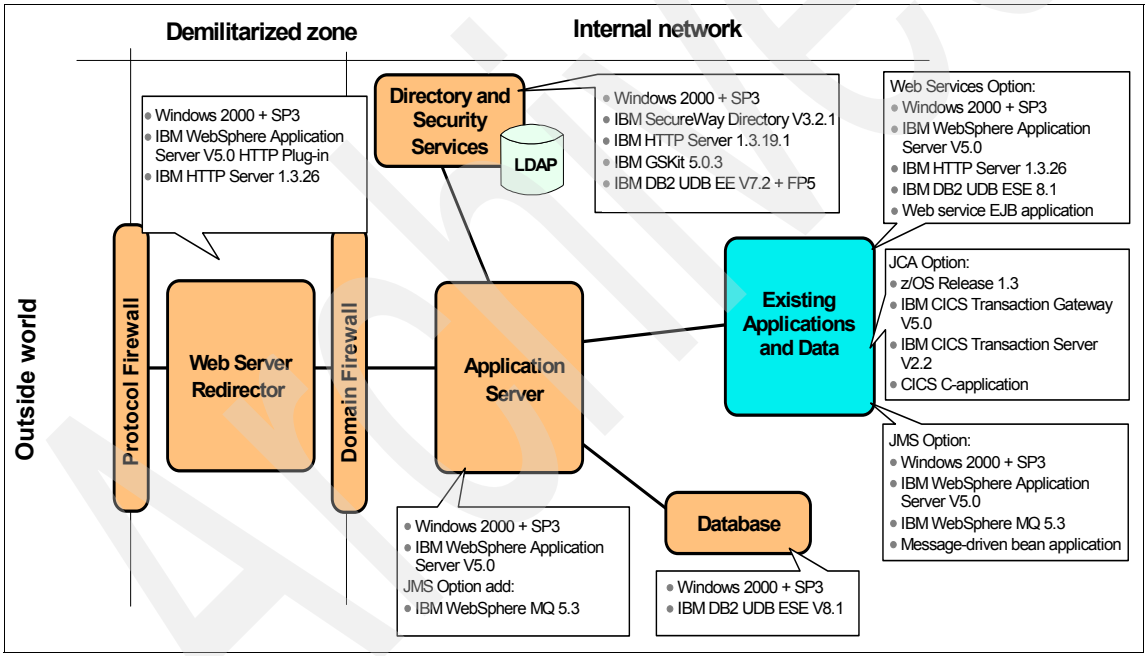


Figure 2-8 Directly Integrated Single Channel application pattern::Product mapping= Microsoft® Windows® 2000

For more information, see the Patterns for e-business Web site:

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

2.2.5 Reviewing guidelines and related links

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

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

For more information, see the Patterns for e-business Web site:

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

2.3 Patterns for e-business naming conventions

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

The capitalization convention is to use lowercase for “pattern” and uppercase for the first and most significant qualifier, for example “Business pattern”.

When referencing a specific type of pattern, the higher level qualifier (for example, business) is not capitalized, as in “Self-Service business pattern”.

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

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

Archived

In addition, when it is necessary to identify variations or product instances at the same level in the hierarchy, an equal sign (=) is used as in the following examples:

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

2.4 Summary

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

Archived

Archived

3

Introduction to business processes

Every enterprise has business processes that describe how it performs its business functions. Many times these processes are primarily implicit and are understood only by the people who perform the function. More successful businesses formally define their processes and work hard at optimizing them.

In this chapter, we introduce business processes and discuss their implementation in WebSphere Process Server. We begin by discussing why to formalize and automate business processes. We then provide a brief overview of service-oriented architecture (SOA), an approach to defining integration architectures based on the concept of a service. Following a description of Web Services Business Process Execution Language (WS-BPEL), the standard for business process description, we introduce *Service Component Architecture* (SCA), IBM's proposed standard for integrating and composing services in an SOA.

© Copyright IBM Corp. 2006. All rights reserved.

3.1 Overview of business process management

Business process management, as defined by the Gartner group, is the art of understanding, codifying, automating, and improving the way a company does business. Business process management used to be known as *business process reengineering*.

Most companies have unique business processes that differentiate them from their competitors. Examples of these processes include:

- Expense reports
- Purchase orders
- New accounts and credit authorization
- Project management

These processes must evolve over time as the business reacts to market conditions. Any solution that codifies and automates these processes must allow them to be quickly and easily modified in order to meet new business needs.

Furthermore, business process management reduces the distinction between work done by a human and a computer. A process-oriented approach includes steps performed by enterprise systems interspersed with tasks performed by people.

A more system-oriented view considers business process to be an additional tier in a multi-tier application environment. Business logic and business rules are extracted from the business logic tier and presented as a workflow, which shows graphically the different steps of a business process. At each step, business logic is executed, and business rules direct the subsequent flow. The application logic executed at each step is thus greatly simplified. It is executed only in specific cases. Business process logic, controlled by a graphical user interface, handles complex business cases and categorizes different business situations. This logic exists at the workflow level that controls the business process, and not in the application logic.

As a consequence, business logic and business rules have become explicit, visible, and rapidly changeable. They no longer exist solely in the province of the IT division, but are surfaced where they can be defined, monitored, and refined by business analysts. The enterprise can react more quickly to changes in its marketplace.

There are a number of reasons to focus on business process management. Every enterprise is faced with rapid changes in its environment. These changes are first recognized by the business analyst, not the IT department that runs the business applications. If a business analyst can update the business rules without the participation of the IT department, the enterprise will respond much

more quickly. This is accomplished by tools that separate the business process and its metrics from the IT infrastructure.

The tools that enable business process management also facilitate the addition of external users of previously internal applications. The Internet has opened new ways for customers and business partners to interact with your company. Most business processes are not entirely internal, and improving the efficiency of a process that interacts with customers or partners will reduce costs for all participants. For that reason, Web-driven applications can greatly increase customer satisfaction. Business partners also expect to use new flexible approaches such as business-to-business (B2B) integration and Internet market technologies. A conventional infrastructure, designed for internal use, is not flexible enough to cope with these new channels. Data connectivity and application integration by themselves are insufficient; they do not integrate with people.

Business managers, not the IT department, must be in control of the business objectives, which are reflected by the business processes. But before control, there is understanding. You need an integrated business view of your enterprise, from the process through to the resources and assets that participate in them. Formalizing business processes provides insight into business operations. The roles of employees and their relationships with suppliers and customers become clearer. Business process management tools generate useful data that helps the business analyst to understand and measure the effectiveness of each business process. This data exposes the activities that take the most time or use the most resources. Business process management tools help you decide where you can improve your business processes, what the costs are, and what benefits can be expected.

The underlying forces that push a company to business process management are technology, globalization, and competition. Technology has created a global marketplace, and it has given customers and suppliers the power to leave your business with a click of the mouse. Business process management brings processes, people, and information together to increase efficiency for the entire enterprise.

3.2 Overview of SOA

SOA defines integration architectures based on the concept of a *service*. Applications collaborate by invoking each others' services, and services are composed into larger sequences to implement business processes. SOA in fact can be seen as an IT architecture for business process management.

Drivers for SOA

The main driver for SOA is to define an architectural approach that assists in the flexible integration of IT systems. Organizations spend a considerable amount of time and money trying to achieve rapid, flexible integration of IT systems across all elements of the business cycle. The drivers behind this objective include:

- Increasing the speed at which businesses can implement new products and processes, can change existing ones, or can recombine them in new ways
- Reducing implementation and ownership costs of IT systems and the integration between them
- Enabling flexible pricing models by outsourcing more fine-grained elements of the business than were previously possible or by moving from fixed to variable pricing, based on transaction volumes
- Simplifying the integration work that is required by mergers and acquisitions
- Achieving better IT use and return on investment
- Achieving implementation of business processes at a level that is independent from the applications and platforms that are used to support the processes

SOA prescribes a set of design principles and an architectural approach to achieve this rapid flexible integration.

Definition of SOA

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

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

SOA brings the benefits of loose coupling and encapsulation to integration at an enterprise level. It applies successful concepts that are proven by Object-Oriented development, Component-Based Design, and enterprise

application integration (EAI) technology to an architectural approach for IT system integration.

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

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

3.2.1 Definition of a service

SOA is an architectural approach to defining integration architectures that are based on services. It is important to define what is meant by a *service* in this context in order to fully describe SOA and to understand what you can achieve by using it.

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

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

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

Reusable function

A service can be any business function. In an SOA, however, it is preferable that the function is genuinely reusable. In an SOA, the service can be used and reused by one or more systems that participate in the architecture. The intention of SOA is to achieve the reuse of services at:

- Run time

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

- Deployment time

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

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

Explicit implementation independent interfaces

The use of explicit interfaces to define and encapsulate service function is of particular importance in making services genuinely reusable. The interface should encapsulate only those aspects of process and behavior that are used in the interaction between the service consumer and the service provider. An explicit interface definition, or contract, is used to bind a service consumer and a service provider. It should specify only the mutual behavior that is required for the interaction and nothing about the implementation of the consumer or the provider.

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

Communication protocols that stress location transparency

Companies have a variety of choices when deciding how to connect applications. HTTP, HTTPS, Java Message Service (JMS), CORBA, and Simple Mail Transfer Protocol (SMTP) are all examples of protocols that can be used to connect applications. There are also many middleware products, for example WebSphere MQ, that provide application-to-application connectivity. Typically, even within a single company, a variety of techniques, products, and protocols are used to address different integration requirements. This variety of techniques can create problems when trying to extend the integration to connect to applications that do not use the same protocols.

SOA does not specify that any specific protocol should be used to provide access to a service. A key principle in SOA is that a service is not defined by the communication protocol that it uses but instead is protocol-independent so that different protocols can be used to access the same service.

Ideally, a service should be defined only once, through a service interface, and should have many implementations with different access protocols. This definition increases the reusability of any service definition. Also, services should be invoked, published, and discovered in a way that is abstracted away from the actual implementation using a single, standards-based form of interface. Thus, there is a complimentary nature between SOA and Web services.

3.2.2 Web services and SOA

An appropriate combination of both Web services technology and the SOA approach addresses many of the issues of building an SOA-enabled environment. That is not to say that Web services and SOA are intrinsically linked, because they can be implemented separately. In fact, many significant SOAs are proprietary or customized implementations that do not use Web services technologies.

However, existing SOA implementations have demonstrated the benefits of SOA, typically within a single enterprise, and the existing uses of Web services have demonstrated the benefits of the Web services technologies in integrating heterogeneous systems both within and among organizations. A custom approach gives an organization the problem of supporting heterogeneity; a proprietary approach gives it to one IT vendor. Adopting a standards-based approach, such as Web services, offers a solution to these issues.

There are logical links between Web services and SOA that suggest that they are complimentary:

- Web services provide an open-standard and machine-readable model for creating explicit, implementation-independent descriptions of service interfaces.
- Web services provide communication mechanisms that are location-transparent and interoperable.
- Web services are evolving, through WS-BPEL, document-style SOAP, Web Services Description Language (WSDL), and emerging technologies (such as WS-ResourceFramework), to support the technical implementation of well-designed services that encapsulate and model reusable function in a flexible manner.

Archived

3.2.3 The advantages of SOA

Use of SOA has the following advantages to achieving loosely coupled flexible integration of IT systems:

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

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

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

Currently, and for some time to come, many of the technologies that are used to implement SOAs are evolving rather than remaining mature and stable. Therefore, individual SOA solutions require carefully balanced decisions among customized, proprietary, and open-standard technologies. You must decide the characteristics and components of SOA to implement and the areas of the business function and process to apply them. You should balance these decisions between business benefits, technology maturity, and implementation or maintenance efforts.

3.2.4 SOA summary

SOA and Web services enable new opportunities for more flexible, rapid, and widespread integration in a model that is consistent with the exposure of business function as services. SOA and Web services, and in particular WS-BPEL, offer the choreography of those services into processes that can be modeled, executed, and monitored with such features as the following types:

- SOA defines concepts and general techniques for designing, encapsulating, and invoking reusable business functions through loosely bound service interactions. Most of the techniques have been proven individually in previous technologies or design styles. SOA unites them in an approach that is intended to bring encapsulation and reuse to the enterprise level.

- Web services provide an emerging set of open-standard technologies that can be combined with proven existing technologies to implement the concepts and techniques of SOA.
- Industry support for Web services standards, interoperability among different implementations of Web services, and the infrastructure technology that is required to support an SOA give technology customers increasingly mature and sophisticated technologies that are suitable for SOA implementation.

These techniques and technologies give you the tools that are required to implement flexible SOAs and to evolve toward an On Demand Business model.

3.3 WS-BPEL V2.0 standard

In this section, we discuss the key concepts covered by the WS-BPEL V2.0 specification. You can find further information about WS-BPEL on the Web at:

http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

WS-BPEL V2.0 is the current specification of the Business Process Execution Language. It is a standard Extensible Markup Language (XML) for expressing business processes consisting of functions defined through Web services interfaces. Previous versions were known as *Business Process Execution Language for Web Services* (BPEL4WS) or simply BPEL.

WS-BPEL provides a means of connecting Web services and for specifying how collections of Web services can be jointly used to provide more complex functionality in the form of a business process. A WS-BPEL process is a flow which describes the order and conditions in which those services are invoked. WebSphere Process Server uses WS-BPEL to implement a service integration platform based on open standards. This loosely coupled integration model allows flexible integration of heterogeneous systems within an enterprise or in a variety of domains, including business-to-business or (B2B) business-to-consumer (B2C) process models.

WS-BPEL provides a model and grammar for describing the behavior of a business process in terms of the interactions between the process and its partners. Each interaction with a partner occurs through Web service interfaces, and the WS-BPEL process includes the state and logic that coordinate service interactions to achieve a business goal. It also supports the processing of business exceptions and includes a mechanism to define what should happen when a fault occurs or a partner requests reversal.

WS-BPEL is expressive enough to express arbitrarily complex algorithms. The following sections describe the key elements of the language.

3.3.1 Partners

When modelling business processes, we need to describe the relationship between a business process and its partners.

A partner can be:

- A consumer of a service provided by a business process
- A provider of a service, which is used by a business process
- Services that a business process invokes and that also invoke the process

3.3.2 Endpoint references

In order for a business process to invoke a service from a partner, the partner must resolve to an actual service. The endpoint reference describes this binding between the service and the partner. This binding can be performed when the business process is modeled, at deployment time, or even at run time.

3.3.3 Activities

There are two types of activities, each of which is explained in the sections that follow:

- Basic activities
- Structured activities

Basic activities

Basic activities are primitive activities and do not contain other activities. They represent a step within a business process.

Receive

The *receive activity* allows a business process to wait for a message to arrive.

Reply

The *reply activity* returns a message to the partner that initiated the process. It is used in synchronous (receive/reply) operations.

Invoke

The *invoke activity* allows a business process to call a one-way or request-response service. Some things that can be invoked include:

- A Web service
- A Java class
- An Enterprise JavaBean (EJB™)
- Another process

Archived

Assign

The *assign activity* is used to update the values of variables with new data.

Throw

The *throw activity* is used to generate a fault from within a business process. It indicates that an error has occurred.

Wait

The *wait activity* pauses for a given time period or until a certain defined time has passed.

Empty

The *empty activity* performs no executable action. It is commonly used for modelling activities in a process that is still awaiting implementation.

Terminate

The *terminate activity* causes a process or activity to immediately terminate. The result of the terminate activity depends on where the activity is being used:

- If the terminate activity is within the process, then the process is forced to terminate, and all activities that are currently active are halted without any fault handling or compensation behavior.
- If terminate activity is within a structured activity and the activity has multiple control paths and has already chosen to execute one of them, then only the activities on that path are terminated.
- When used within a fault handler, the terminate activity stops all activities that are currently active within it.

Structured activities

Structured activities define the order in which a collection of activities takes place. Structured activities are comprised of basic activities and are used to express control patterns, data flow, handling of faults and external events, and coordination of message exchanges between process instances.

Structured activities include:

- *sequence*, *switch*, and *while*: Provide sequential control between activities
- *flow*: Provides concurrency and synchronization between activities
- *pick*: Provides a non-deterministic choice based on external events

Sequence

A *sequence activity* contains one or more activities which are performed sequentially.

Archived

Switch

The *switch activity* contains a condition to decide between two or more control paths.

While

The *while activity* contains other activities that are repeated until a given boolean condition no longer holds true.

Pick

The *pick activity* waits for one of a number of messages to arrive and then proceeds based upon the one that was received. Alternatively, it can wait for a time-out.

Flow

The *flow activity* is used to nest activities that will execute concurrently. It completes when all the nested activities have completed.

Link

A *link* synchronizes two activities to enforce a particular execution order.

3.3.4 Data handling

As business processes model stateful interactions, they require:

- State variables
- Expressions to control behavior
- Assignment to update state

WS-BPEL supports these features using WSDL messages and XML data types.

Expressions

WS-BPEL supports the following types of expressions in the appropriate contexts:

- Boolean-valued expressions
- Expressions of XML Schema type DateTime or *date*
- Expressions of *XML Schema type duration*
- Expressions of XML value type *string*, *number* or *Boolean*

In general, it is possible to query data from variables, to extract property values, and to evaluate the status of links from within expressions.

Archived

Variables

Variables are used to hold:

- Messages that are received from partners or are to be sent to a partner
- Data related to the internal state of the process

Variables can be associated with invoke, receive, and reply activities as input or output variables.

Variables can also be declared as:

- Global variables with business process scope
- Local variables with their scope being an activity or a group of activities

All variables are in an uninitialized state at the beginning of the scope to which they belong. They can be initialized by receiving a message or by assignment.

Assignment

The *assign activity* is used to copy data from one variable to another.

3.3.5 Correlation

Business process interactions involve stateful conversations between processes. These may in practice occur over a long period of time, possibly days or months. Message correlation is used to match returning consumers to long-running business processes. When a request is issued by a partner, it is necessary to identify if a new business process should be instantiated or whether the request should be directed to an existing process instance. Instead of using a special instance ID, WS-BPEL correlation reuses identifying information from the existing business messages.

WS-BPEL provides a mechanism to define a *correlation set*, which is a named group of properties shared by all messages in a correlated group. The correlation set is effectively the primary key for a business process instance.

3.3.6 Scope

The *scope element* is an optional element which provides a context for the behavior of each activity. A scope can provide:

- Fault handlers
- Event handlers
- A compensation handler
- Data variables
- Correlation sets

Each scope has a primary activity that defines its normal behavior. This can be a complex structured activity containing nested activities. The scope is shared by all nested activities.

Compensation handlers

Business processes may be long-running and may use asynchronous messages for communication. Errors can occur within the process or in the partner services being invoked. A single business process transaction may in practice have resulted in a number of local updates. If the overall business transaction is cancelled or fails, the partial work already completed must be undone.

Compensation handlers can be specified by the developer of the business process to specify the actions required to undo any completed units of work and to restore the data to its original state. Scopes are used to delineate the sections of a process that are to be reversed by a compensation handler.

A compensation handler is a completely self-contained action. After a given scope has completed, its compensation handler is ready to run. The compensation activity can be used only in the following parts of a business process:

- In a compensation handler of the scope that immediately encloses the scope requiring compensation
- In a fault handler of the scope that immediately encloses the scope requiring compensation

In these situations, a handler can be run *explicitly* by the execution of a compensate activity. *Implicit compensation* occurs when faults are being handled and propagated. Implicit compensation of a scope traverses all an activity's nested scope and runs their compensation handlers in reverse order of completion.

Fault handlers

The aim of fault handling is to undo the unsuccessful work of the scope in which the fault has occurred. When a fault handler completes, the activity it is associated with is considered to have completed unsuccessfully.

Fault handlers can be optionally defined for a scope using *catch activities*. A fault handler can be associated with a particular type of fault and will run if that fault occurs.

A fault handler can:

- Catch a fault on an activity or process and attempt to deal with it
- Use a throw activity to signal an internal fault
- Use a terminate activity to stop execution of the process or activity

Event handlers

Each scope, as well as the business process as a whole, can be associated with a set of event handlers. There are two types of events:

- Message events, which are incoming messages
- Alarms that go off after user-set times

Event handlers are part of the normal behavior of a scope, unlike compensation and fault handlers. The event handlers associated with a scope are enabled when that scope commences. They are disabled when the normal processing of the scope is complete. Any event handlers which have already been dispatched are allowed to complete. The scope as a whole is not considered to be complete until all event handlers have finished processing.

3.4 Service Component Architecture

Since WS-BPEL is a technology specifically designed for application composition, we need a way to encapsulate discrete chunks of business functionality and make them available for consumption. In WebSphere Process Server, this requirement is satisfied by SCA. SCA is a fundamental technology within WebSphere Process Server. It provides a model to define interfaces, implementations, and references in a technology neutral way. It enables you to then bind these elements to whichever technology specific implementations are most appropriate.

SCA separates application business logic and the implementation details. The ability to separate business logic from infrastructure logic reduces the IT resources needed to build an enterprise application. It gives developers more time to work on solving a particular business problem rather than focusing on the details of which implementation technology to use.

3.4.1 Anatomy of SCA

SCA is the service-oriented component model. It provides an abstraction that covers stateless session EJBs, Web services, Plain Old Java Objects (POJOs), WS-BPEL processes, database access, Enterprise Information System (EIS) access, and so on. SCA separates business logic from infrastructure logic so that application programmers can focus on the business problem. SCA covers both the usage of services and the development of services. It provides a uniform model for application programmers and for tools.

SCA is a universal model for business services that publish or operate on business data. Service Data Objects (SDO) provide the universal model for business data.

Figure 3-1 shows the main terms of an SCA component:

- Interface
- Implementation
- Reference

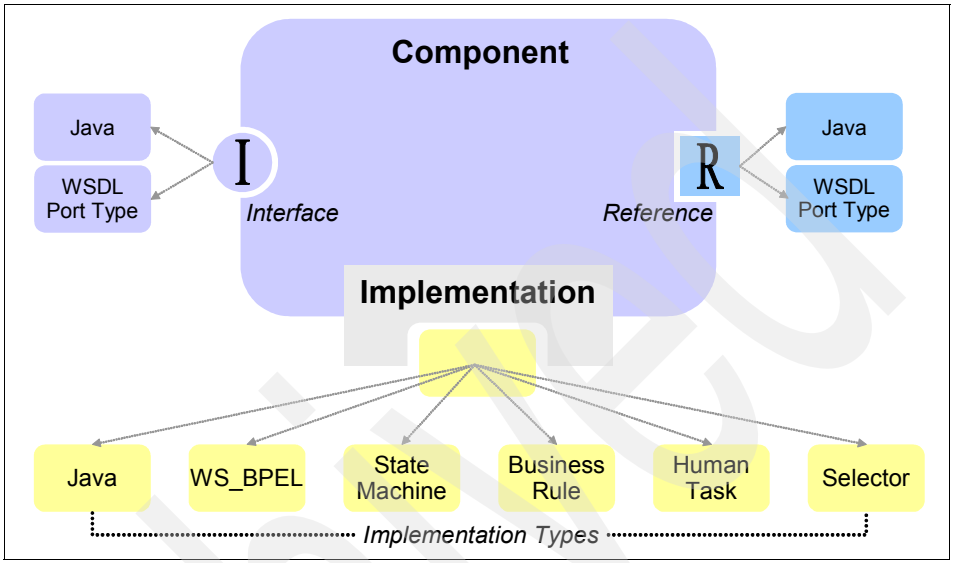


Figure 3-1 Service component overview

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.

A component exposes business-level interfaces to its application business logic 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. An import and export also has interfaces so that the published service can be invoked.

All components have interfaces of the WSDL type. Only Java components support Java-type interfaces. If a component, import or export, has more than one interface, all interfaces must be the same type.

A component can be called *synchronously* or *asynchronously*, which 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.

A service can be implemented in a range of languages (for example Java, WS-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 create SCA applications by component assembly.

Figure 3-2 shows a service component and a number of references. When a component wants to use the services of another component, it must have a partner reference or simply a reference. We can consider an in-line reference, which means that the referenced service component is defined within the same scope of the referencing component. Both components are defined within the same module.

Applications that are not defined as SCA components (for example, JavaServer™ Pages™ (JSPs)) can still invoke SCA components, 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.

Archived

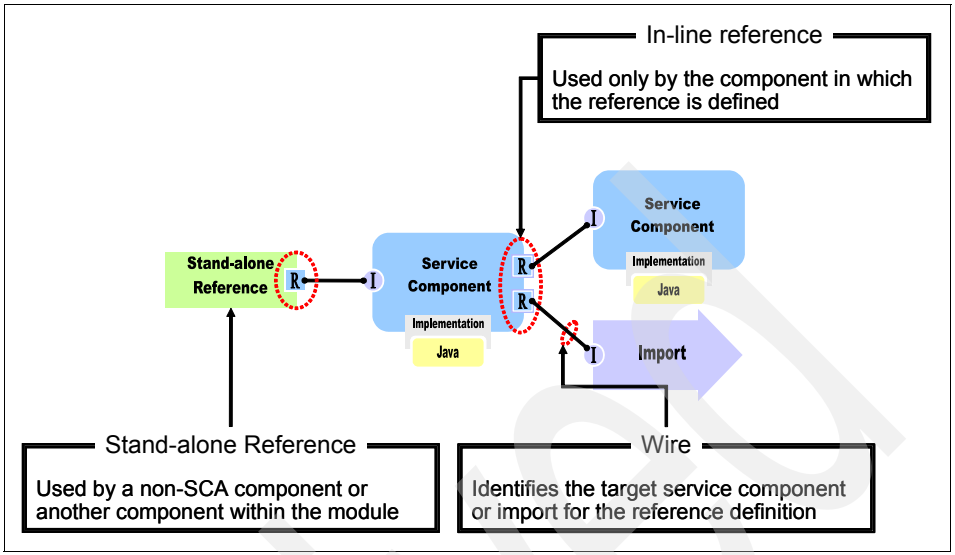


Figure 3-2 Service component and references

Components are assembled in a module (Figure 3-3), which is a basic unit of deployment in the WebSphere Process Server.

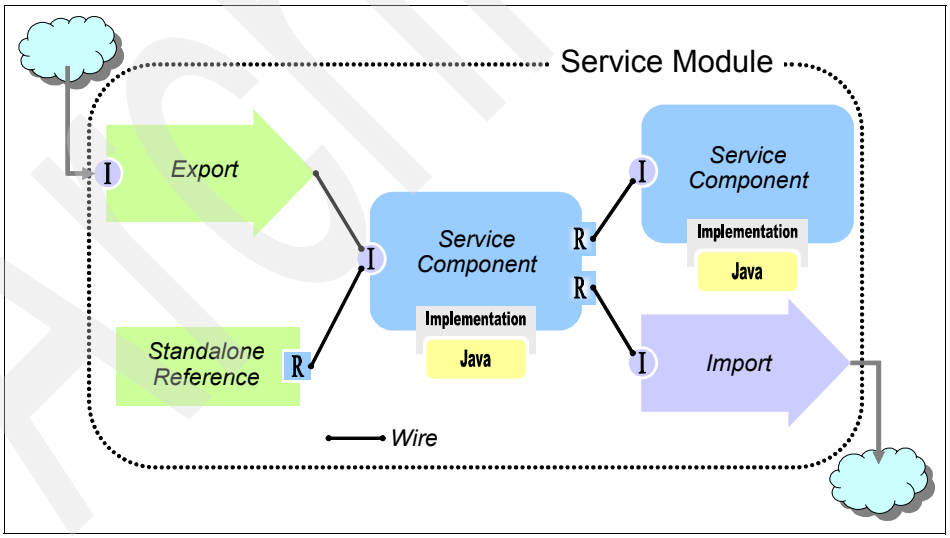


Figure 3-3 Service module overview

The module assembly contains a diagram of the integrated business application, consisting of components and the wires that connect them. You use an Assembly

Editor to visually compose the integrated application with elements that you drag from the palette or from the tree in the Business Integration view of WebSphere Process Server.

The implementations of components that are used in a module assembly might reside within the module. 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 that have their interfaces and dragging the exports into the required 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.

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. An 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* (Figure 3-4).

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.

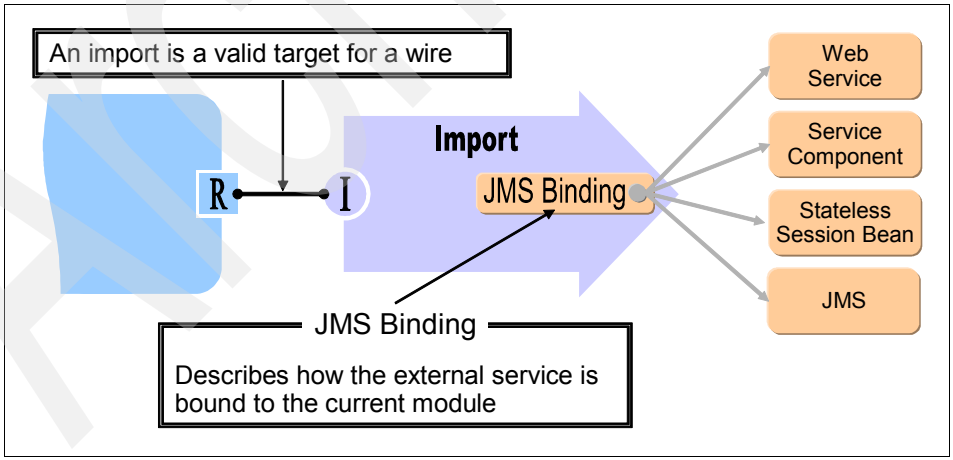


Figure 3-4 Service component and import

An *export* is a published interface from a component (Figure 3-5) 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.

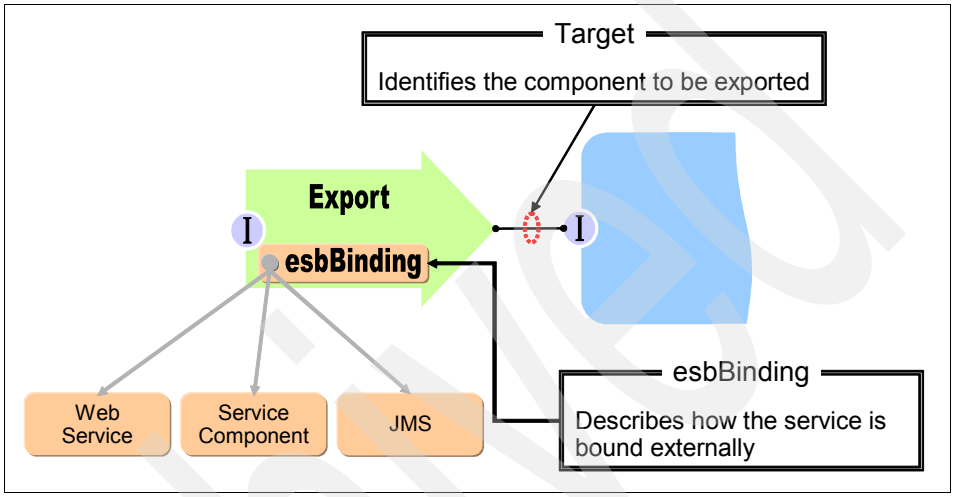


Figure 3-5 Service component and export

Archived

The service component details are stored in an XML file (Figure 3-6) using a new definition language, *Service Component Definition Language* (SCDL).

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:java="http://www.ibm.com/xmlns/prod/websphere/scdl/java/6.0.0"
xmlns:ns1="http://HelloWorld/HelloWorldInterface"
xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/6.0.0"
xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/6.0.0"
displayName="HelloWorld" name="HelloWorld">
  <interfaces>
    <interface xsi:type="wsdl:WSDLPortType" portType="ns1:HelloWorldInterface">
      <method name="sendMessage"/>
    </interface>
  </interfaces>
  <references>
    <reference name="HelloWorldInterfacePartner">
      <interface xsi:type="wsdl:WSDLPortType" portType="ns1:HelloWorldInterface"/>
    </reference>
  </references>
  <implementation xsi:type="java:JavaImplementation" class="sample.HelloWorldImpl"/>
</scdl:component>
```

Figure 3-6 Sample SCDL file

Archived

Archived

4

Application Integration and Extended Enterprise patterns

Within the Patterns for e-business, two types of Business patterns are appropriate for many of the uses characterized as enterprise application integration (EAI). Integration within a single enterprise is addressed by the *Application Integration business pattern*. The *Extended Enterprise business pattern* addresses the interactions and collaborations between business processes in separate enterprises.

The Application Integration pattern captures commonly observed solution alternatives in the domain of EAI. It comprises patterns that capture best practices around back-end integration of applications and data, process automation, and workflow implementations involving human interactions. 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. The Application Integration pattern does not include front-end integration such as the composition of a portal or single sign-on across multiple applications; they are captured by the Access Integration pattern.

© Copyright IBM Corp. 2006. All rights reserved.

The Extended Enterprise business pattern is used when the business processes to be integrated span separate enterprises. Like the Application Integration business pattern, it does not cover applications that are directly invoked using a user interface by business partners across organizational boundaries.

This chapter provides an overview of all of the Application patterns comprising the Application Integration and Extended Enterprise business patterns, with emphasis on the patterns that are implemented in the subsequent chapters of this redbook.

4.1 Application Integration pattern

Application integration requirements typically call for the seamless execution of multiple applications and access to their respective data in order to automate business function. Application integration can be divided into two complementary approaches:

- *Process-focused integration*: The integration of the functional flow of processing between the applications
- *Data-focused integration*: The logical integration of the information used by applications

Business and IT drivers that result in the selection of this Integration pattern are:

- The business processes need to be integrated with existing business systems and information.
- The business activity needs to aggregate, organize, and present information from various sources within the organization.

This redbook focuses on ITSO Electronics, a company that wants to integrate its retail and wholesale departments. Since the process-focused Application Integration patterns address this problem, we focus on those patterns in subsequent sections.

Archived

4.1.1 Process-focused Application Integration patterns

Process-focused Application Integration patterns are used to combine multiple business processes or business systems. The result is a new business offering or a consolidated view of some business entity. These patterns are particularly useful in tying together different platforms and technologies. They tend to be more difficult to design and implement than data-focused integration, and often require complex middleware.

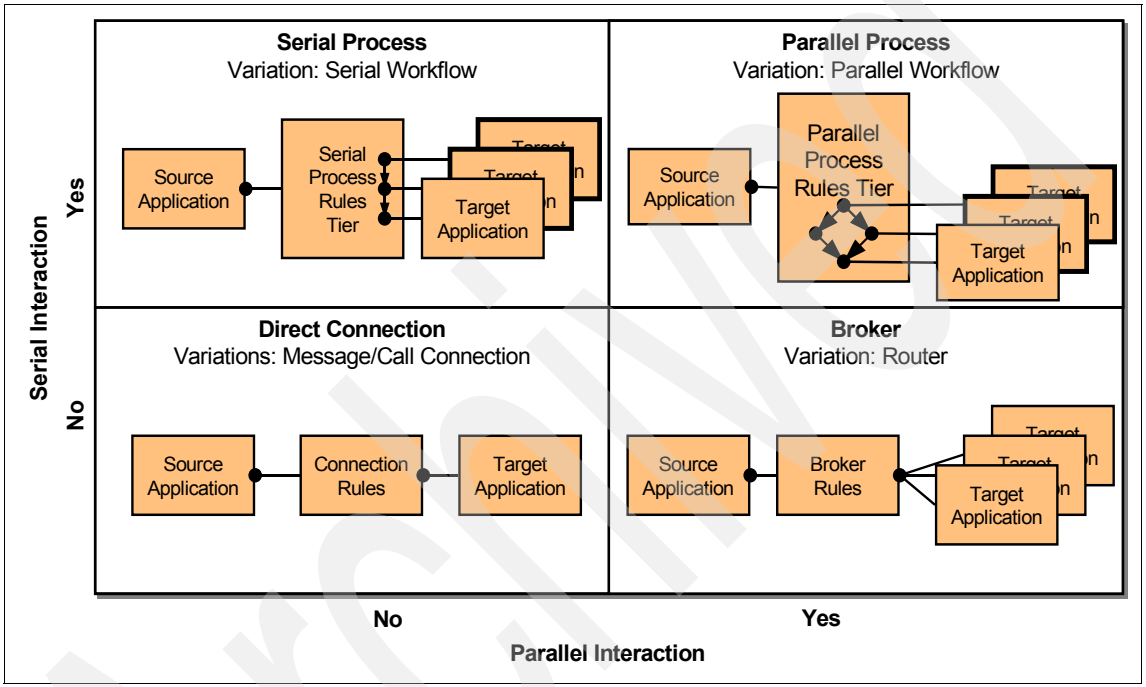


Figure 4-1 Process-focused application integration patterns

The Process-focused Application Integration patterns are presented in the following list in order of increasing flexibility and sophistication. As the Application patterns build on each other, their capabilities and reliance on middleware increase and they require less application development effort.

- Direct Connection application pattern

  The Direct Connection application pattern allows a pair of applications within the organization to directly communicate with each other. The Direct Connection application pattern has two variations:

- Message Connection variation

  This variation applies to solutions in which the business process does not require a response from the target application. It supports one-way message flows.

- Call Connection variation

  This variation applies to solutions in which the business process requires a response from the target application. It supports request/reply message flows.

- Broker application pattern

  The Broker application pattern allows a single interaction from the source application to be distributed to multiple target applications, reducing the proliferation of point-to-point connections. With the Router variation, each interaction is forwarded to, at most, one of the target applications, rather than all of them.

- Serial Process application pattern

  The Serial Process application pattern, described in 4.1.2, “Serial Process application pattern” on page 53, adds sequencing to the one-to-N topology of the Broker application pattern. It enables the orchestration of a serial business process.

  Its Serial Workflow variation, specified in 4.1.4, “Serial Workflow variation” on page 58, adds support for including interaction with people as steps of the serial business process.

- Parallel Process application pattern

  The Parallel Process application pattern, described in 4.1.3, “Parallel Process pattern” on page 56, adds concurrency to the Serial Process application pattern. The process orchestration may include paths that lead to parallel invocation of target application services.

  Its Parallel Workflow variation, specified in 4.1.5, “Parallel Workflow variation” on page 60, adds support for including interaction with people as steps of the parallel business process.

Archived

Business and IT drivers

Table 4-1 summarizes the business for the Process-focused Application Integration patterns and their variations.

Table 4-1 Business drivers

Business drivers	Direct Connection Message variation	Direct Connection Call variation	Broker: Router variation	Broker	Serial Process	Serial Workflow variation	Parallel Process	Parallel Workflow variation
Improve the organizational efficiency	✓	✓	✓	✓	✓	✓	✓	✓
Reduce the latency of business events	✓	✓	✓	✓	✓	✓	✓	✓
Support a structured exchange within the organization	✓	✓	✓	✓	✓	✓	✓	✓
Support real-time one-way message flows	✓		✓	✓	✓	✓	✓	✓
Support real-time request/reply message flows		✓	✓	✓	✓	✓	✓	✓
Support dynamic routing of messages to one of many target applications			✓	✓	✓	✓	✓	✓
Support dynamic distribution of messages to multiple target applications				✓	✓	✓	✓	✓
Support automated coordination of business process flow					✓	✓	✓	✓
Reduce cycle time through parallel execution of portions of a process flow							✓	✓
Support human interaction and intervention within the process flow						✓		✓

Archived

Table 4-2 summarizes the IT drivers for the Process-focused Application Integration patterns and their variations.

Table 4-2 IT drivers

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

Archived

4.1.2 Serial Process application pattern

The Serial Process application pattern, shown in Figure 4-2, supports the sequential execution of business services hosted by a number of target applications. The source application initiates a serial business process.

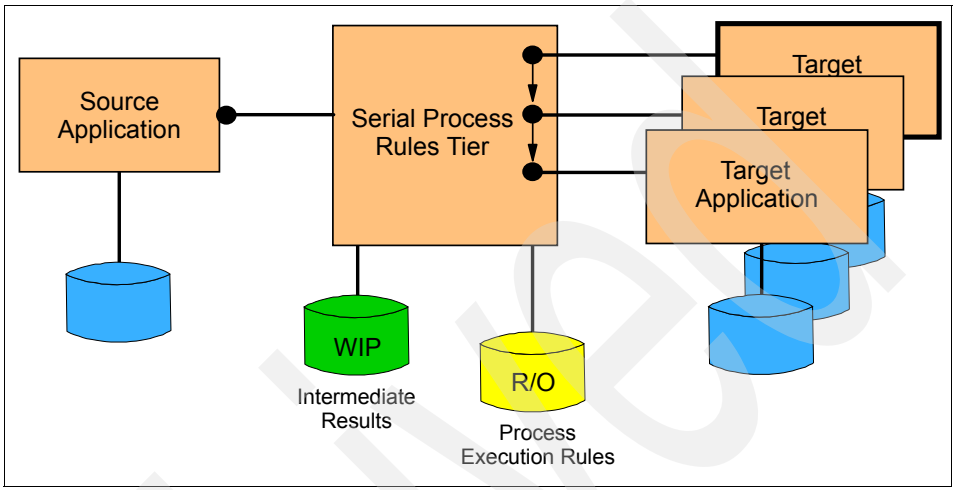


Figure 4-2 Serial Process application pattern

Business and IT drivers

The primary business driver for this pattern is composing a set of business services into an end-to-end business process.

From an IT perspective, the key driver is improving the flexibility and responsiveness of IT by externalizing the process flow logic from individual applications.

Solution

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

- The Source Application tier represents one or more applications that are interested in interacting with the target applications.
- The Serial Process Rules tier supports most of the services provided by the broker tier in the Broker application pattern, including routing of requests, protocol conversion, message broadcasting, and message decomposition/ recomposition. It also supports the separation of business process flow logic from individual application logic.

  The process logic is governed by the serial process rules that define execution rules for each target application, together with control flow and data

Archived

flow rules. It may also include any necessary adapter rules. The combination of these process execution rules are stored in read-only databases.

This externalization of process flow logic is essential for the implementation of a flexible and responsive IT environment that can respond quickly to changing business needs. It also makes it possible to compose new end-to-end processes by combining different business services provided by different applications. Finally, this tier uses a work-in-progress (WIP) database to store the intermediate results from the execution of different process steps.

- The Target Application tier represents new, modified existing, or unmodified existing applications that implement the necessary business services.

Guidelines for use

The flexibility and responsiveness provided by this Application pattern depend on externalizing process execution logic from individual applications. Applications designed with a service-oriented architecture (SOA) approach, which have well-defined and coarse-grained business services, are better suited for participation in this pattern. A service may participate in more than one end-to-end process.

Typically, traditional applications are not designed with this approach in mind. Moreover, many have significant amounts of process logic embedded within them. These constraints in existing environments may pose challenges to fully implementing this Application pattern. A good starting point is to wrap traditional and packaged applications into business services.

Composing process flows by tying together different applications may introduce the need for compensating transaction support. This is especially true when participating applications do not leverage XA-compliant transaction processing engines. You may need to design compensating transaction pairs for every affected transaction. You may also need to modify participating traditional and packaged target applications to introduce compensating transactions.

Finally, pay particular attention to the business process management capabilities supported by the business process design tools and the process execution engines when you select middleware products. The goal is to enable business users to compose business processes and make necessary changes with minimal involvement from IT professionals. The business processes thus defined must be easily exported into a process execution engine. More sophisticated business process management tools allow for the definition of metrics during process design and support monitoring of the metrics in the process execution engine.

Benefits

The Serial Process application pattern improves the flexibility and responsiveness of an organization by implementing end-to-end process flows and by externalizing process logic from individual applications.

- It integrates multiple, diverse applications.
- It minimizes the impact to existing applications.
- It relieves the source application from being aware of the target application.
- It can provide transformation services that allow the source and target to use different communication protocols.
- It minimizes the impact of changes in location of the target application.
- It provides a foundation for automated support for business process management, enabling monitoring and measurement of the effectiveness of business processes.

Limitations

This Application pattern supports straight-through processing without human interaction. If human interaction is needed to complete certain process steps, consider the Workflow variation of this Application pattern as described in 4.1.4, “Serial Workflow variation” on page 58.

It also does not support the parallel execution of multiple tasks. Under such circumstances, consider the Parallel Process application pattern discussed in 4.1.3, “Parallel Process pattern” on page 56.

Archived

4.1.3 Parallel Process pattern

The Parallel Process application pattern, shown in Figure 4-3, adds concurrency to the Serial Process application pattern. The process orchestration may include paths that lead to parallel invocation of target application services.

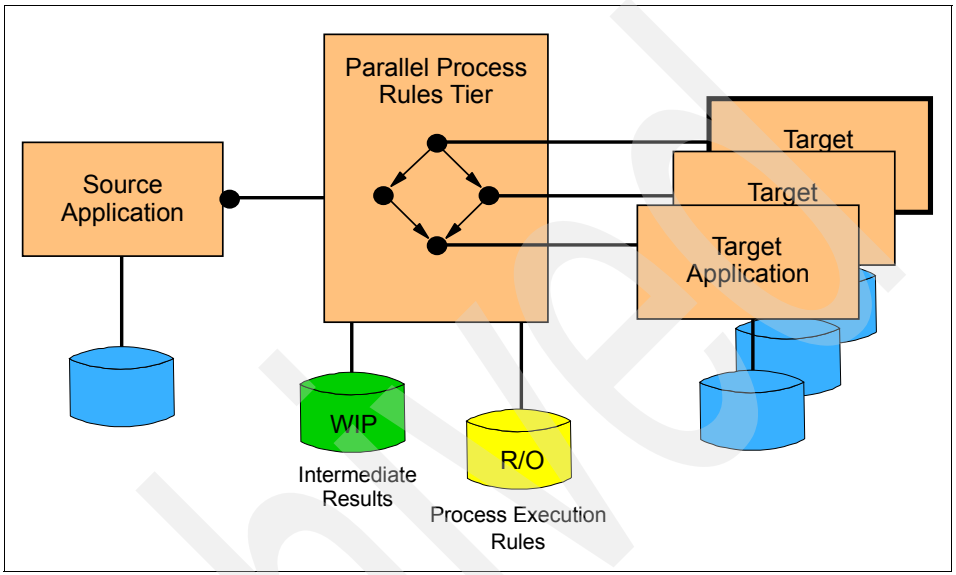


Figure 4-3 Parallel Process application pattern

Business and IT drivers

All the business and IT drivers listed under the Serial Process application pattern apply to this Application pattern as well. The additional driver is the need to reduce cycle time by executing parts of the process in parallel.

Solution

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

- The *Source Application tier* is the same as for the Serial Process application pattern.
- The *Parallel Process Rules tier* supports all the services provided by the Serial Process Rules tier within the Serial Process application pattern.

  In addition, the interaction initiated by the source application may invoke services on multiple target applications in parallel. This parallelism requires that fork and join conditions be defined. The runtime engine must be able to initiate parallel threads of control, ensure these threads join upon completion, and manage them as a unit (for example to allow cancellation of the process or to report its status).

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

Guidelines for use

The following guidelines apply to this variation in addition to the guidelines that are documented under the Serial Process application pattern.

The selected runtime engine must support parallel processes. Use this as a driver for your middleware selection decision.

Judicious use of parallelism is a powerful tool for reducing the cycle time of a process in the right circumstances. You must ensure that all of the error scenarios are carefully analyzed and that the impact of these scenarios upon the end-user experience is thoroughly understood. The number of error scenarios and processing complexity increases exponentially with the degree of parallelism. The best practice is to start with a Serial Process and introduce limited parallelism only where there is a clear and worthwhile benefit.

Benefits

In addition to providing all the benefits provided by the Serial Process application pattern, this pattern provides a foundation for the reduction of cycle times by implementing parallel processes.

Limitations

Parallel processes are more complex to design, test, and operate than Serial Processes. In addition, this Application pattern is for processing without human interaction. If human interaction is needed to complete certain process steps, consider the Workflow variation of this Application pattern, described in 4.1.5, “Parallel Workflow variation” on page 60.

Archived

4.1.4 Serial Workflow variation

The Serial Workflow variation of the Serial Process application pattern, shown in Figure 4-4, extends the basic Serial Process by including interaction with people as steps in the business process.

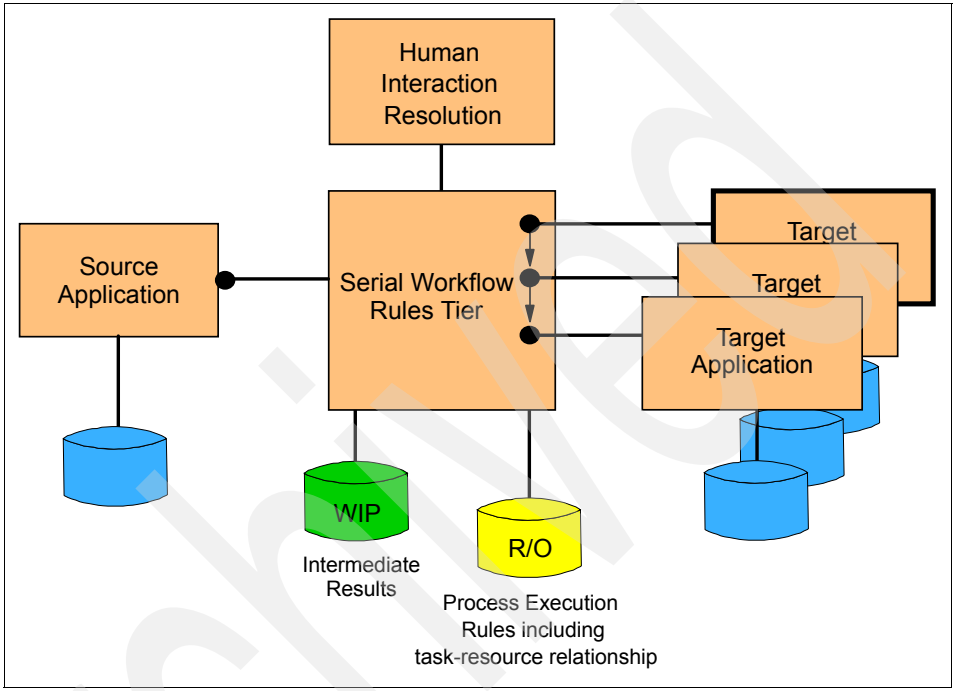


Figure 4-4 Serial Workflow variation

Business and IT drivers

All the business and IT drivers for the Serial Process application pattern apply to this variation as well. In addition the need exists to support human interaction and intervention within the process flow. Support for long-running transactions is often a prerequisite for the automation of process flows involving human interaction, and thus is another IT driver.

Archived

Solution

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

- The *Source Application tier* is the same as for the Serial Process application pattern.
- In addition to all the services provided by the Serial Process Rules tier within the Serial Process application pattern, the *Serial Workflow Rules tier* supports routing certain tasks to human actors for completion. The rules are augmented with relationships that define the resources that are capable of performing specific tasks. People, departments, and target applications can all be resources capable of executing a particular task.

  This tier resolves task-resource relationships during the execution of a process. If human interaction is needed, the task is added to a worklist to be completed by a human. The process is typically suspended until the completion of the task.

  This tier provides support for long-running transactions, using a WIP database to store intermediate results while the process runs.
- The *Target Application tier* is the same as for the Serial Process application pattern.

Guidelines for use

The guidelines for the Serial Process application pattern all hold for this variation. In addition, people-based exception handling should be implemented within the process. If an automated task reaches an error condition, human actors must be able to intervene and handle the exceptions.

Benefits

In addition to the benefits of the Serial Process application pattern, further flexibility is introduced by the externalization of task-resource resolution rules, which allow people to execute steps in the process.

Limitations

The Serial Workflow application pattern does not support the parallel execution of multiple tasks. Under such circumstances, consider the Parallel Process application pattern and Parallel Workflow variation discussed in 4.1.3, “Parallel Process pattern” on page 56, and 4.1.5, “Parallel Workflow variation” on page 60.

Archived

4.1.5 Parallel Workflow variation

The Parallel Workflow variation of the Parallel Process application pattern, shown in Figure 4-5, extends the basic parallel process orchestration capability by supporting human interaction for completing certain process steps.

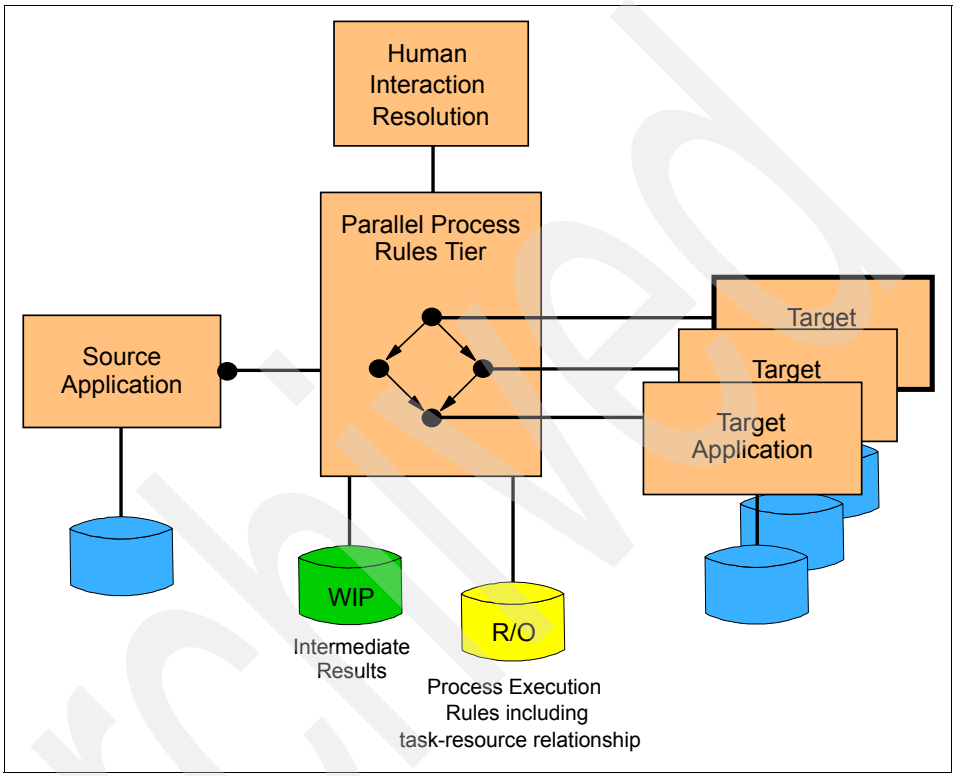


Figure 4-5 Parallel Workflow variation

Business and IT drivers

All of the business and IT drivers listed under the Parallel Process application pattern apply to this variation as well. The additional business driver is the need to support human interaction and intervention within the process flow. Support for long-running transactions is often a prerequisite for the automation of process flows involving human interaction, and thus is another IT driver.

Archived

Solution

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

- The *Source Application tier* is the same as for the Parallel Process application pattern.
- The *Parallel Workflow Rules tier* supports all the services provided by the Parallel Process Rules tier within the Parallel Process application pattern. In addition, it supports routing certain tasks to people for completion. The rules are augmented with relationships that define the resources that are capable of performing specific tasks. People, departments, and target applications can all be resources capable of executing a particular task.

  This tier resolves task-resource relationships during the execution of a process. If human interaction is needed, the task is added to a worklist to be completed by a human. The process is typically suspended until the completion of the task.

  This tier provides support for long-running transactions, using a WIP database to store intermediate results while the process runs.
- The *Target Application tier* is the same as for the Parallel Process application pattern.

Guidelines for use

The guidelines for the Parallel Process application pattern all hold for this variation. In addition, people-based exception handling should be implemented within the process. If an automated task reaches an error condition, human actors must be able to intervene and handle the exceptions.

Benefits

In addition to the benefits of the Parallel Process application pattern, further flexibility is introduced by the externalization of task-resource resolution rules, which allow people to execute steps in the process.

Limitations

Only a few middleware products, including WebSphere Process Server, support all the capabilities needed to realize this Application pattern. Do not try to implement this Application pattern using middleware products that do not support the necessary capabilities.

Archived

4.2 Extended Enterprise pattern

The Application Integration patterns automate business functionality within an enterprise. When the same requirements need to be fulfilled between separate enterprises, you can use the Extended Enterprise business pattern.

Like the Application Integration business pattern, this pattern addresses requirements to execute services from multiple applications. There is no corresponding data integration component; distinct enterprises generally do not collaborate at the data integration level. Likewise, it does not cover applications that are directly invoked using a user interface.

The Application patterns for Extended Enterprise are essentially the Application Integration patterns implemented across organizational boundaries. They differ mainly in aspects of the quality of service expressed in the Runtime patterns.

Business and IT drivers that result in the selection of this Integration pattern are:

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

4.2.1 Extended Enterprise application patterns

The specific business functionality supported by applications that automate the Extended Enterprise business pattern varies from one industry to another. A survey of such applications in multiple industries, however, reveals certain common approaches that have been successful.

We present the Extended Enterprise Application patterns in order of increasing flexibility and sophistication. As the Application patterns build on each other, their capabilities and reliance on middleware increase, and they require less application development effort. Select the Application pattern, as shown in Figure 4-6, that best fits your requirements.

Archived

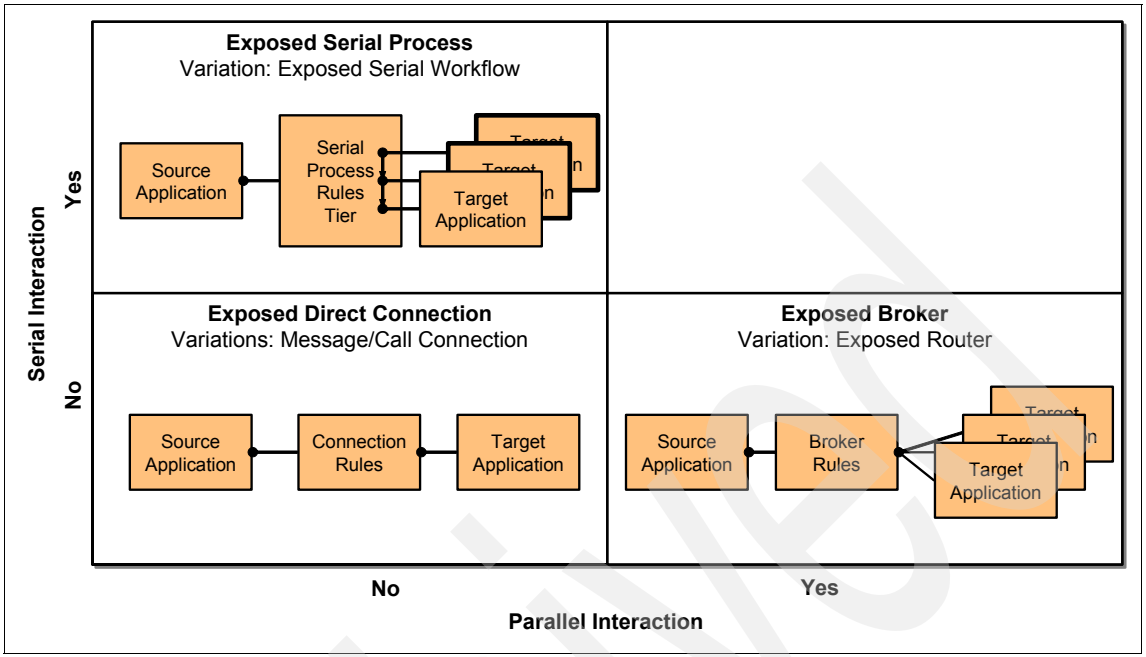


Figure 4-6 The Extended Enterprise application patterns

The Extended Enterprise application patterns are:

- Exposed Direct Connection application pattern

  The Exposed Direct Connection application pattern allows a pair of applications within the organization to directly communicate with each other across organization boundaries. The Exposed Direct Connection application pattern has two variations:

  – Message Connection variation

    This variation applies to solutions in which the business process does not require a response from the target application. It supports one-way message flows and lends itself to asynchronous protocols.

  – Call Connection variation

    This variation applies to solutions in which the business process requires a response from the target application. It supports request/reply message flows and fits best with synchronous protocols.

Archived

- Exposed Broker application pattern

  The Exposed Broker application pattern allows a single interaction from the source application to be distributed to multiple target applications across organization boundaries, reducing the proliferation of point-to-point connections. With the Router variation, each interaction is forwarded to at most one of the target applications, rather than all of them.

- Exposed Serial Process application pattern

  The Exposed Serial Process application pattern, described in detail in 4.2.2, “Exposed Serial Process application pattern” on page 67, adds sequencing to the one-to-N topology of the Broker application pattern. It enables the orchestration of a serial business process across enterprise boundaries.

  Its Serial Workflow variation, specified in 4.2.3, “Exposed Serial Workflow pattern” on page 69, adds support for including interaction with people as steps of the serial business process.

> **Note:** The Exposed Parallel Process application pattern is another possibility, but it is not currently being observed in the Extended Enterprise domain. We expect it to appear as the technology evolves.

Archived

Business and IT drivers

Table 4-3 summarizes the business drivers for the Extended Enterprise application patterns and their variations.

Table 4-3 Business drivers

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

Archived

Table 4-4 summarizes the IT drivers for the Extended Enterprise application patterns and their variations.

Table 4-4 IT drivers

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

Archived

4.2.2 Exposed Serial Process application pattern

The Exposed Serial Process application pattern, shown in Figure 4-7, extends the one-to-*N* topology provided by the Exposed Broker application pattern with sequential execution of business services hosted by a number of target applications. The source application initiates a serial business process across enterprise boundaries.

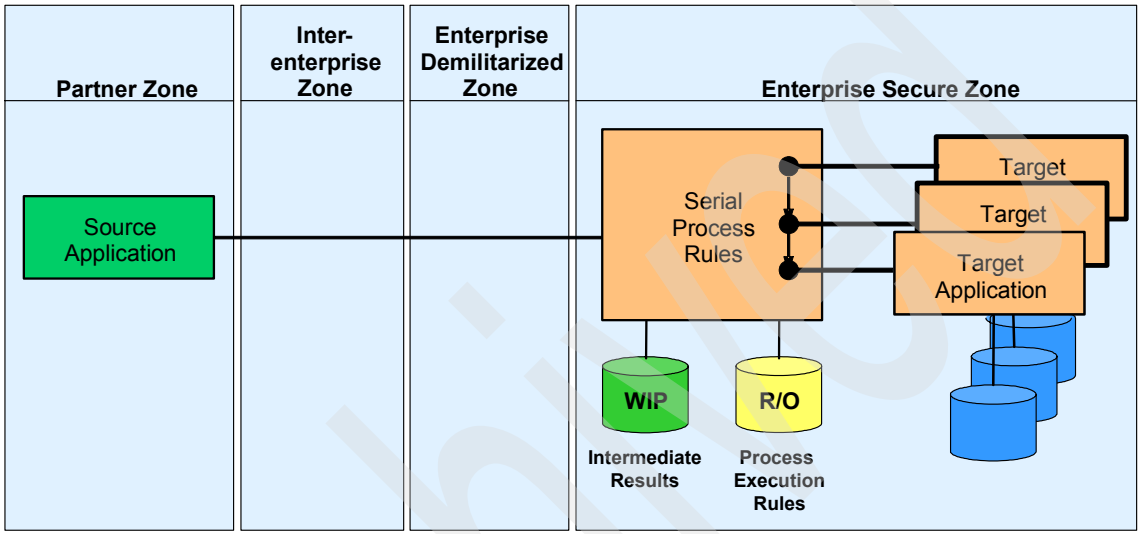


Figure 4-7 Exposed Serial Process application pattern

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

Business and IT drivers

The primary business driver for selecting this Application pattern is composing a business process flow between partners. From an IT perspective, the key driver is improving the flexibility and responsiveness of IT by externalizing the process flow logic from the application logic.

Archived

Solution

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

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

  The process logic is governed by Serial Process Rules that define execution rules for each target application, together with control flow and data flow rules. It may also include any necessary adapter rules. The combination of these process execution rules is stored in read-only databases. This externalization of process flow logic is essential for the implementation of a flexible and responsive IT environment that can respond quickly to changing business needs. It also makes it possible to compose new end-to-end processes by combining different business services provided by different applications.

  Finally, this tier uses a WIP database to store the intermediate results from the execution of different process steps.
- The *Target Application tier* represents new, modified existing, or unmodified existing applications that implement the necessary business services.

Guidelines for use

Consider all of the guidelines for the Serial Process application pattern in 4.1.2, “Serial Process application pattern” on page 53.

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

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

Benefits

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

Limitations

The Exposed Serial Process application pattern is ideally suited for straight-through processing where human interactions are not necessary to complete an end-to-end process. If support for human interactions is needed to complete certain process steps, consider the Workflow variation of this Application pattern.

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

4.2.3 Exposed Serial Workflow pattern

The Workflow variation of the Exposed Serial Process application pattern, shown in Figure 4-8, extends the basic Serial Process orchestration capability by supporting human interaction for completing certain process steps.

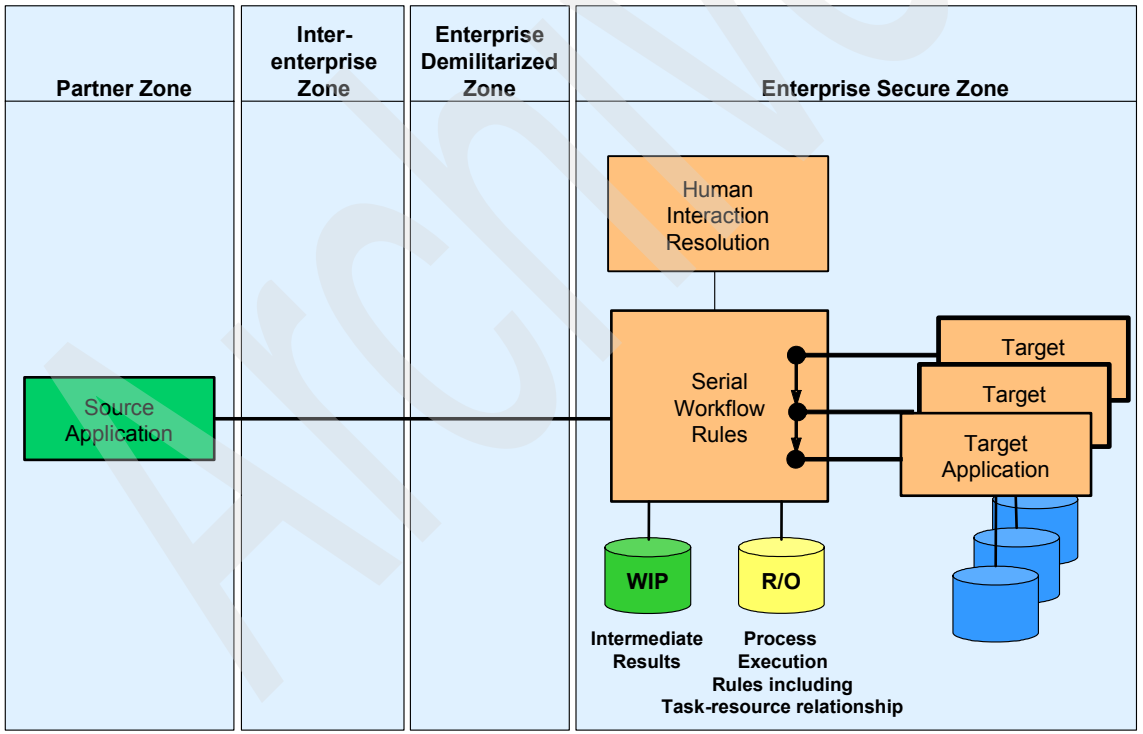


Figure 4-8 Exposed Serial Workflow variation

Archived

Business and IT drivers

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

Solution

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

- The *Source Application tier* is the same as for the Exposed Serial Process application pattern.
- The *Serial Workflow Rules tier* supports all the services provided by the Serial Process Rules tier within the Exposed Serial Process application pattern. In addition, it supports certain tasks within the process to be routed to a person or people for completion. The rules are augmented with relationships that define the resources that are capable of performing specific tasks. People, departments, and target applications can all be resources capable of executing a particular task.

  This tier resolves task-resource relationships during the execution of a process. If human interaction is needed, the task is added to a worklist to be completed by a human. The process is typically suspended until the completion of the task.

  This tier provides support for long-running transactions, using a WIP database to store intermediate results while the process runs.
- The *Target Application tier* is the same as for the Exposed Serial Process application pattern.

Guidelines for use

The following guidelines apply to this variation in addition to the guidelines that are documented in the Exposed Serial Process application pattern. In addition, people-based exception handling should be implemented within the process. If an automated task reaches an error condition, human actors must be able to intervene and handle the exceptions.

Archived

Benefits

In addition to the benefits of the Exposed Serial Process application pattern, further flexibility is introduced by the externalization of task-resource resolution rules, which allow people to perform steps in the process.

Limitations

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

Archived

Archived

5

Runtime patterns

This chapter describes the Application Integration runtime patterns for the Serial and Parallel Process patterns, together with their workflow variations. It also addresses the Extended Enterprise runtime patterns for the Serial Process pattern and Serial Workflow variation.

Following the process of pattern selection from the Patterns for e-business layered asset model, we arrive at the Runtime pattern selection as shown in Figure 5-1.

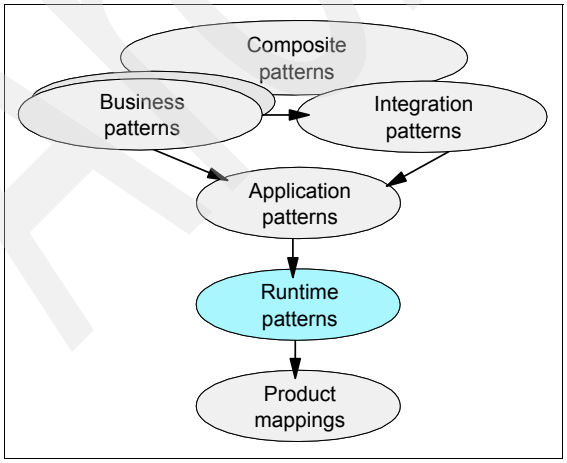


Figure 5-1 Choosing a pattern: Runtime patterns

© Copyright IBM Corp. 2006. All rights reserved.

5.1 Application Integration runtime patterns

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

Each function is associated with a runtime node. In reality these functions, or nodes, can exist on separate physical machines or coexist on the same machine. In the Runtime pattern, this is not relevant. The focus is on the logical nodes required and their placement in the overall network structure.

We can overlay the Application pattern over the Runtime pattern to identify where business logic and data are deployed on nodes. The Runtime patterns that are illustrated give some typical examples of possible solutions.

For reference, the Application Integration patterns are shown in Figure 5-2.

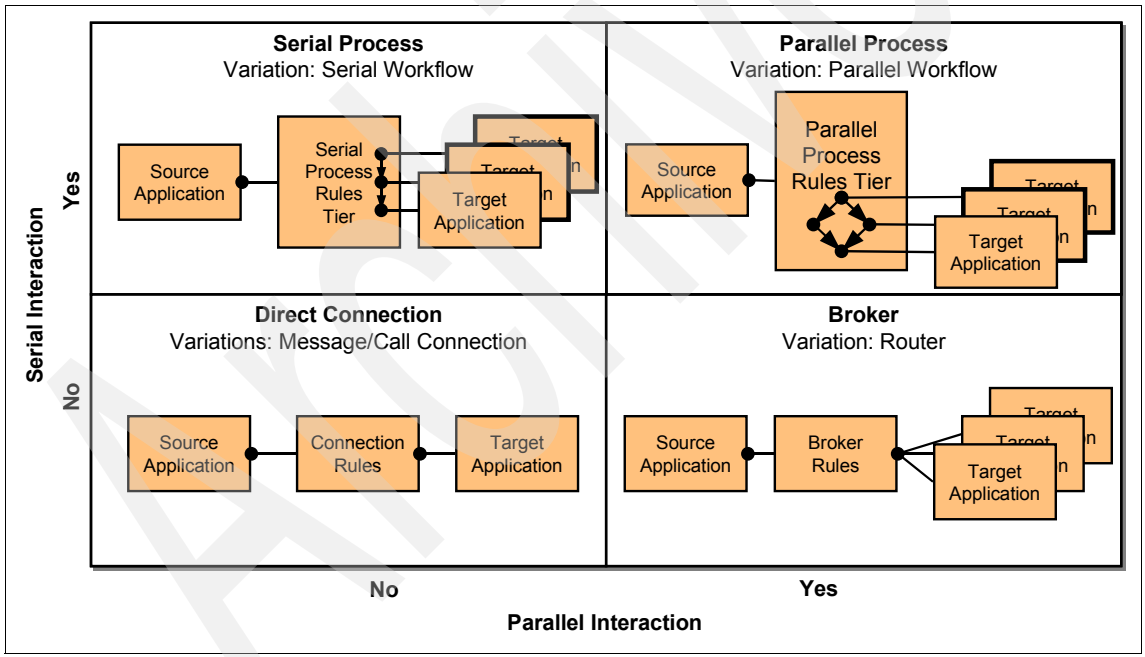


Figure 5-2 Application Integration patterns

This redbook covers only Runtime patterns for Serial and Parallel Process application patterns and their variations.

Each Runtime pattern has two profiles associated with it:

- Generic profile
- Service-oriented architecture (SOA) profile

The SOA profile category of the Runtime pattern applies specifically to SOA solutions, using an enterprise service bus (ESB). The generic profile can be used for any run time and is the profile we use for the example implementations in this redbook.

In this chapter, we discuss both generic and SOA Runtime patterns for the following Application patterns:

- “Serial Process pattern” on page 80
- “Parallel Process pattern” on page 84
- “Serial Workflow variation” on page 82
- “Parallel Workflow variation” on page 86
- “Exposed Serial Process pattern” on page 89
- “Exposed Serial Workflow pattern” on page 90

We do not implement all of the patterns in full in this redbook. For reasons of space, we use only the Serial and Parallel processes together with the Exposed Serial Process and the Parallel Workflow variations as design and implementation examples. Because this redbook does not cover ESB, all of the implementations use the generic runtime profile. The SOA profile is not implemented, because it requires the inclusion of an ESB, which is beyond the scope of this book. However all solutions in this book use SOA principles and are relevant to (non-ESB) SOA implementations.

Note: You may be wondering where the *Exposed Parallel* and *Exposed Parallel Workflow* patterns are. At the time of writing this redbook, these patterns had not been used extensively enough in the real world to justify inclusion in this redbook. This does not necessarily mean that they are invalid, or even anti-patterns; their use is not prohibited.

5.1.1 Node types

Each Runtime pattern is made up of a collection of nodes of different types, each performing a specific function. As each node type appears in multiple Runtime patterns, we discuss them together here.

Archived

Process Manager node

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

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

Connector node

A connector node is used to provide connectivity between:

- The App Server/Services node and the Process Manager

  This allows a source application to communicate with a Process Manager. Process Managers typically support multiple connector types.

- The Process Manager and an App Server/Services node

  This allows a Process Manager to communicate with a target application. Process Managers typically offer multiple connector types.

- The Protocol Firewall and Domain Firewall nodes

  This allows the flow of information across the Enterprise Demilitarized Zone (DMZ).

Directory and Security services node

The Directory and Security services node supplies authentication and authorization services. It also holds the user ID and password and related privileges. This node typically leverages Lightweight Directory Access Protocol (LDAP)-based directories.

Repository node

The Repository node provides a persistent data storage and retrieval service in support of the execution of the process flow. It holds the Process Execution Rules and the Intermediate Results from the execution of certain activities within the context of an end-to-end process flow. The implementation of this node may involve several persistent data technologies (such as a database management system (DBMS) and flat file) for the different data types. In some cases, non-persistent storage is used to store the Immediate Results.

Archived

ESB node

The ESB node is a key enabler for an SOA because it provides the capability to route and transport service requests from the service consumer to the correct service provider. The ESB node controls routing within the scope of a service namespace, indicated symbolically by the ellipse on the ESB node representation.

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.

Business Service Choreography node

Business Service Choreography (BSC) allows for the development and execution of business process flow logic, which is centrally controlled and outside application logic. This facilitates the implementation of changes to the business process and the monitoring and analysis of business process execution. The business process governs the sequence and control of service invocations. It also enables tracking by leveraging the process execution rules stored in the associated database.

This node contains the business processes and logic execution engine, together with definitions of those processes and logic. The node receives requests from service consumers and calls service providers through the ESB. These allows for the separation of business logic definition and execution from the implementation of the services used by the processes. This node also holds the state of individual processes and tracks the sequence of each of the process flows.

The BSC node can execute processes that consume services from any application to which the ESB has access. In doing so, it enables enterprise-wide composite applications to be built and the full power of SOA to be unleashed.

Using the ESB, the services used by any process can be replaced or updated without affecting the process logic, which enables the enterprise to be truly dynamic.

Exposed ESB Gateway

An Exposed ESB Gateway node makes the services of one organization available to others, and vice versa, in a controlled and secure manner. This action might require capabilities, such as partner provisioning and management, which are distinct from ESB capabilities. However, the intent of this component is different from the intent of the ESB, which is to provide a service infrastructure

within an organization. For both these reasons, the Exposed ESB Gateway is likely to be integrated to, but not be a part of, the enterprise service bus.

The connection between the App Server/Services node in the partner zone and the network infrastructure in the inter-enterprise zone could be an HTTP server, an ESB, an Exposed ESB Gateway, or a firewall. Therefore, depending on security requirements, the Exposed ESB Gateway node can be inside or outside of the Enterprise DMZ.

Business Service Directory node

The Business Service Directory node holds the directory of all the business services available through the ESB. An example implementation of this may use Universal Description, Discovery, and Integration (UDDI) to hold details of a Web service. A business service is not necessarily exposed as a Web service. The Business Service Directory, together with the use of an ESB, should allow for this this.

App Server/Services node

The App Server/Services nodes represent applications running in a managed application server environment. They can be providers, with the ability to provide a service, or consumers, who have the ability to request a service.

Workflow Manager node

The Workflow Manager node contains the workflow execution engine that coordinates the process flow across systems and people. It provides the capability for model-driven business process automation and enables tracking by leveraging the process execution rules stored in the associated database. These processes can span multiple applications and people across organizational boundaries within an enterprise.

This node maintains its state and tracks sequencing through the process flow. In doing so, it often leverages the associated database to store intermediate results. It is also responsible for invoking target applications as necessary through associated connectors. Finally, when a particular task requires human interaction, this node creates a work item and identifies a particular person or a department responsible for executing that task and adds the work item to its worklist.

Staff Worklist Adapter node

A specialized adapter is responsible for presenting the work items to be executed by a particular person or a department. It is the primary interface through which humans interact within the end-to-end workflow.

Network Infrastructure node

The Network Infrastructure node provides the network connectivity between the enterprise and the partners. Most commonly in today's business-to-business (B2B) world, this is provided by the Internet. However, many enterprises still communicate using leased lines, value-added networks (VANs), Integrated Services Digital Network (ISDN) and other network infrastructures.

Protocol Firewall node

A *firewall* is a hardware or 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. They can also block some virus attacks, as long as the viruses come from the Internet.

A firewall can separate two or more parts of a local network to control data exchange between departments. Components of firewalls include filters or screens, each of which controls transmission of certain classes of traffic.

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

The Protocol Firewall node is an enterprise's first line of defense against attack from the outside world and typically screens the enterprise's routers. It allows traffic through into the DMZ, where it must still pass the Domain Firewall before it can gain access to applications.

Domain Firewall node

The Domain Firewall node protects the Enterprise Secure Zone and is the second line of defense against attack. It protects the application gateways from traffic that has reached as far as the DMZ. It is typically implemented as a dedicated server node for additional security.

Rules directory

The rules directory contains the rules that are generally used to control the mode of operation of an interaction, depending on external factors. Examples of such rules are:

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

Exposed Process Manager node

The Exposed Process Manager node contains the process flow execution engine. It provides the capability for model-driven business process automation. It also enables tracking by leveraging the process execution rules stored in the associated database. These processes can span multiple applications and organizational boundaries within an enterprise. The node maintains its state and tracks sequencing through the process flow. In doing so, it often leverages the associated repository to store intermediate results.

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

5.1.2 Serial Process pattern

This section discusses the Runtime pattern for the Serial Process application pattern. The pattern has two profiles, the generic profile and the SOA profile.

Generic profile

The generic profile Runtime pattern shown in Figure 5-3 represents a basic topology for the implementation of the Serial Process application pattern. It can be further enhanced by clustering the key nodes to improve availability characteristics.

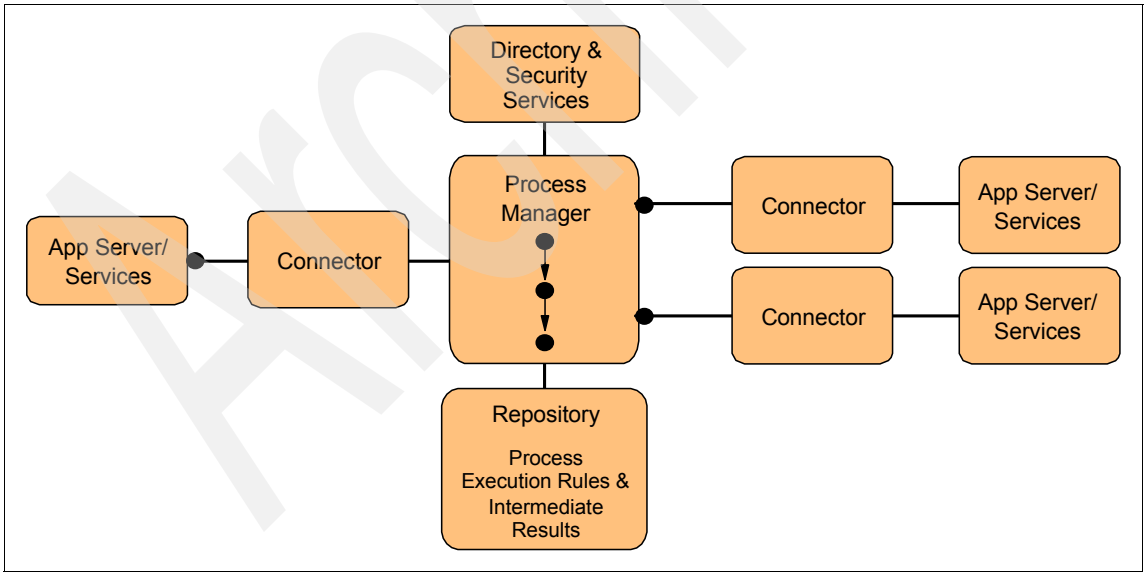


Figure 5-3 Serial Process runtime pattern: Generic profile

Archived

SOA profile

Figure 5-4 shows the SOA profile. As in the generic profile, each of the nodes can be clustered to improve availability. Notice how the ESB is used to link all of the other nodes and allows the Business Service Choreography node to be separated.

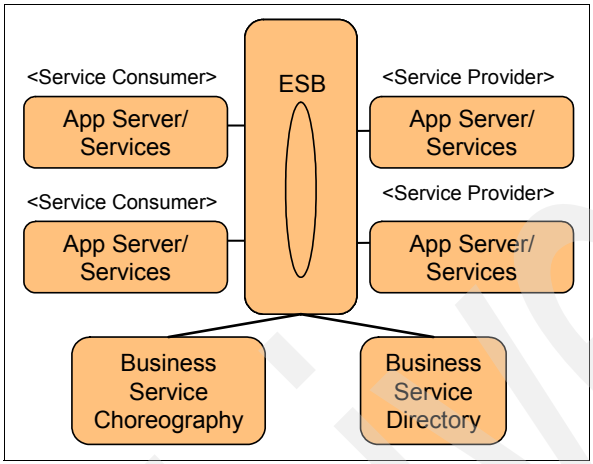


Figure 5-4 Serial Process runtime pattern: SOA profile

Archived

5.1.3 Serial Workflow variation

The Serial Workflow runtime variation is similar to the Serial Process runtime pattern except that we need to deploy nodes to handle human interaction.

Generic profile

The Runtime pattern shown in Figure 5-5 represents the basic topology for the implementation of the Serial Workflow variation. It is similar to the Serial Process generic profile Runtime pattern, but has the addition of the Staff Worklist Adapter, which is accessed by the Workflow Manager.

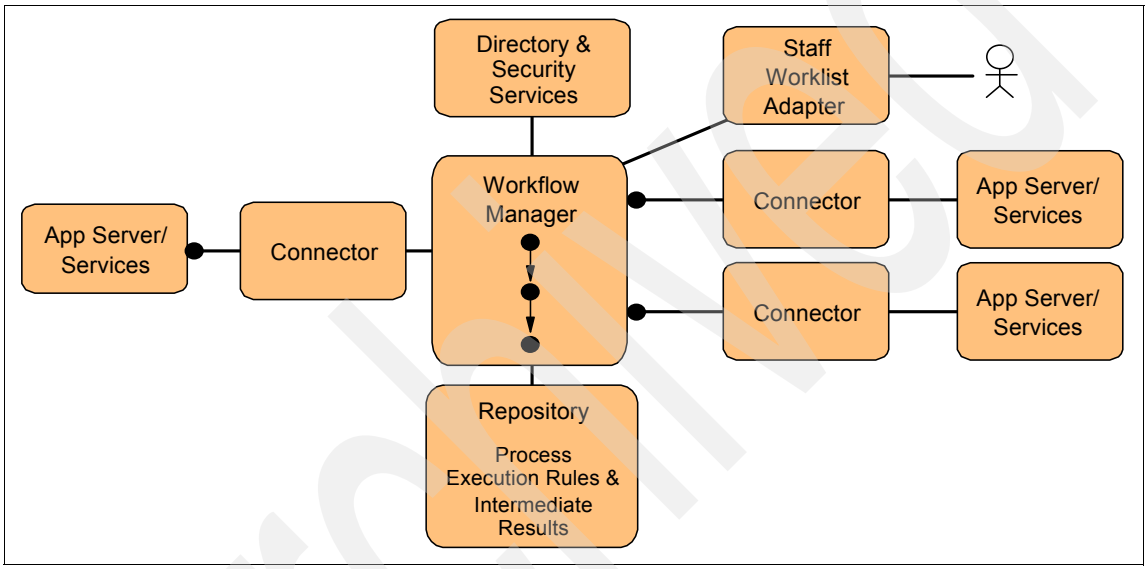


Figure 5-5 Serial Workflow runtime pattern: Generic profile

Archived

SOA profile

The SOA profile shown in Figure 5-6 is the same as the SOA profile Runtime pattern for the Serial Process with the addition of the Staff Worklist Adapter to handle human interaction.

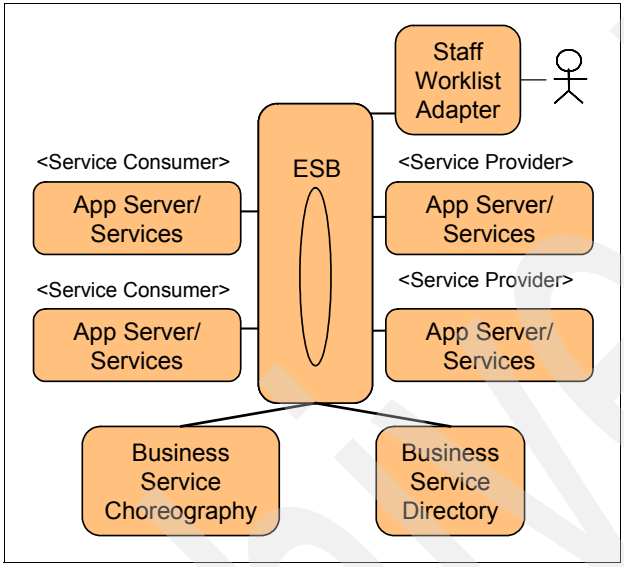


Figure 5-6 Serial Workflow runtime pattern: SOA profile

Archived

5.1.4 Parallel Process pattern

As for the Serial Process, the Parallel Process runtime pattern has both a generic profile and an SOA profile.

Generic profile

The generic profile Runtime pattern shown in Figure 5-7 is similar to the generic profile Runtime pattern for the Serial Process, with the exception of the parallel capabilities of the Process Manager node.

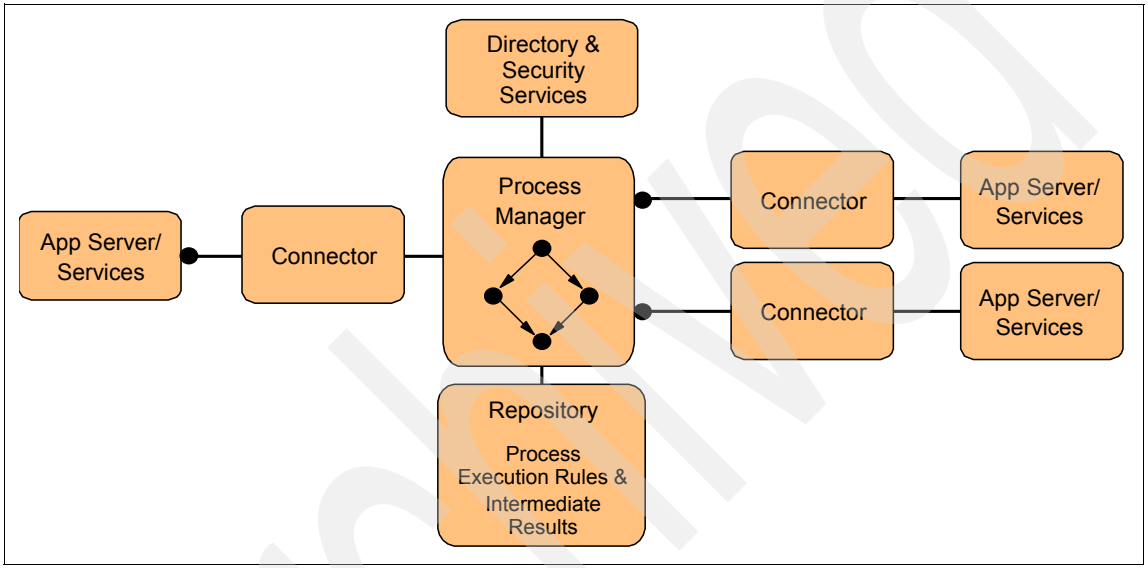


Figure 5-7 Parallel Process runtime pattern: Generic profile

Archived

SOA profile

The SOA profile for the Parallel Process pattern shown in Figure 5-8 is identical to the one for the Serial Process pattern. The Business Service Choreography node handles both the Serial and Parallel Process execution.

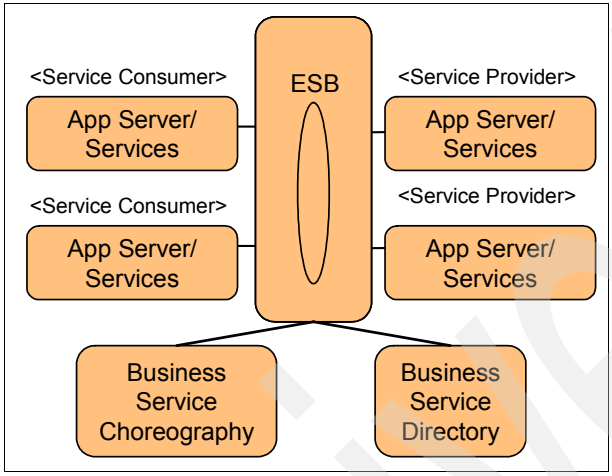


Figure 5-8 Parallel Process runtime pattern: SOA profile

Archived

5.1.5 Parallel Workflow variation

The Parallel Workflow variation combines nodes from the Parallel Process and the Serial Workflow run times. Notice again how the SOA profile is identical to the Serial Workflow SOA profile.

Generic profile

The generic profile Runtime pattern shown in Figure 5-9 represents a basic topology for the implementation of the Parallel Workflow variation. It can be further enhanced by clustering the key nodes to improve availability characteristics.

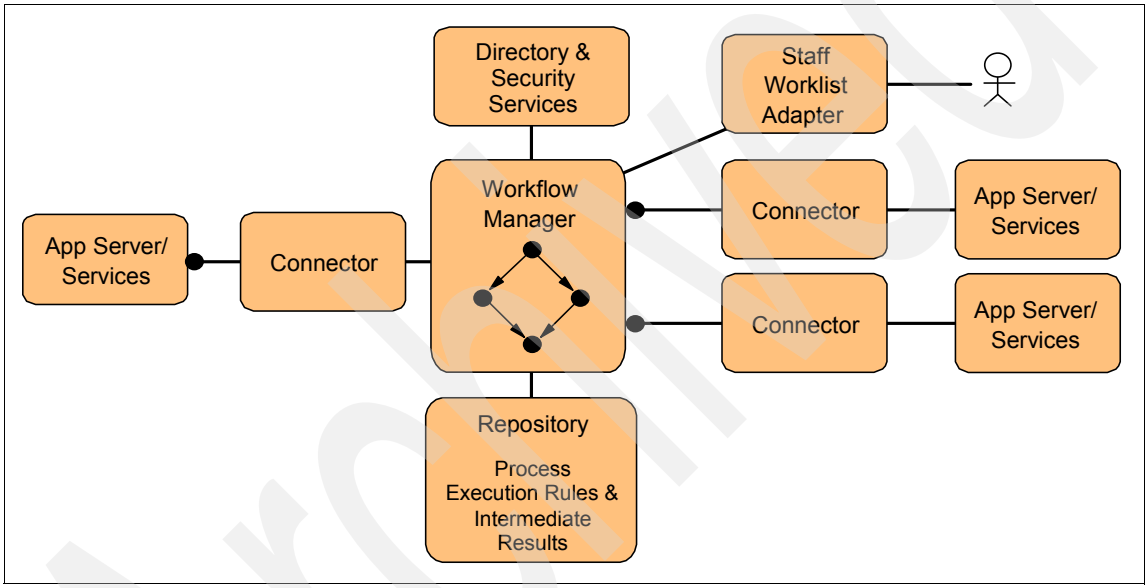


Figure 5-9 Parallel Workflow runtime pattern: Generic profile

Archived

SOA profile

The SOA profile run time for the Parallel Workflow variation, shown in Figure 5-10, is identical to the one for the SOA profile for the Serial Workflow pattern.

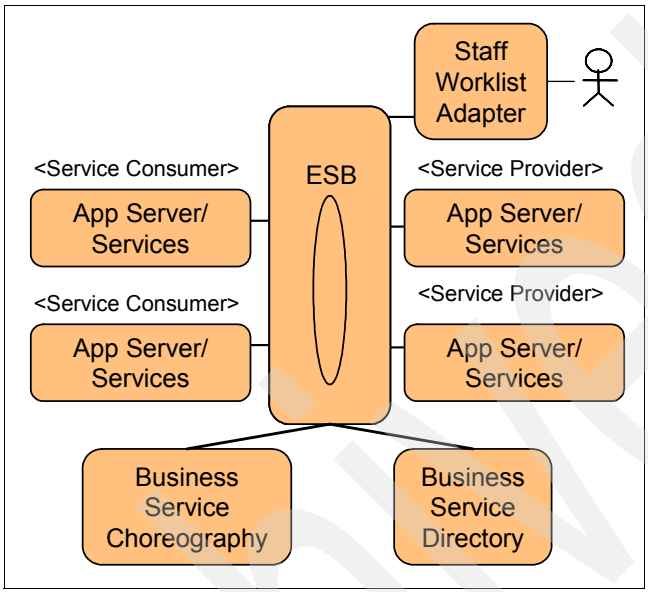


Figure 5-10 Parallel Workflow runtime pattern: SOA profile

5.2 Extended Enterprise runtime patterns

The Extended Enterprise runtime patterns are a logical extension of the Application Integration runtime patterns, allowing the patterns to extend business processes to external partners. As with the Application Integration runtime patterns, each Extended Enterprise runtime pattern has both a generic profile and an SOA profile. We show both of these profiles for each Runtime pattern.

From a business process perspective, where a service is provided is less important than the functionality that is provided by the service. From a runtime perspective, the difference between internal and external services is important.

When considering Extended Enterprise runtime patterns, consider the concept of zones, because security is a key component of an Extended Enterprise. In the Runtime pattern profile illustrations, four zones are defined as explained in the following sections.

Archived

Enterprise Secure Zone

The Enterprise Secure Zone is the safe zone of the non-Extended Enterprise patterns. Security is less of a factor within this zone because it is completely within the control of the enterprise. Hopefully, it will not be a target for a malicious attack from within the zone, although this cannot be discounted. This zone is protected by the domain firewall, discussed in “Domain Firewall node” on page 79.

This zone contains all of the nodes and systems that perform the functions of the enterprise. The concept of a secure zone means that the systems can be optimized for their own particular requirements, without a huge emphasis on external security.

Enterprise Demilitarized Zone

Named after the real-world article, and logically not too dissimilar, the DMZ provides a second line of defense between the Enterprise Secure Zone and the Inter-enterprise Zone. This zone is protected by the protocol firewall, discussed in “Protocol Firewall node” on page 79.

The systems in this zone are normally network traffic and security focussed. If this zone is compromised, the systems within it should fail without impacting those in the Enterprise Secure Zone.

Inter-enterprise Zone

The Inter-enterprise Zone is the zone outside the firewalls and where the enterprise connects to the outside world. Any systems deployed in this zone are effectively open to attack and must be capable of defending themselves. This zone is often the responsibility of a third party, either a VAN provider or an Internet service provider (ISP) in the case of the Internet.

Partner Zone

The Partner Zone is furthest from the enterprise but cannot be dismissed as beyond the sphere of concern of those who are implementing the Runtime patterns. Each partner typically has their own zone structure behind their own firewalls. The firewalls must be negotiated before you can gain access to their systems.

While you many not be responsible for this zone, knowledge of it is vital to enable you to use partner services. You must fulfill all of the Quality of Service (QoS) requirements specified by your partners, such as encryption and integrity, before they let your requests through their firewalls.

Archived

5.2.1 Exposed Serial Process pattern

The Exposed Serial Process pattern is similar to the Serial Process pattern, within the Enterprise Secure Zone. As before, each Runtime pattern has two profiles, the generic profile and the SOA profile.

Generic profile

Figure 5-11 shows the generic profile, with the Exposed Process Manager in place. Notice how all of the process implementation components are located in the Enterprise Secure Zone.

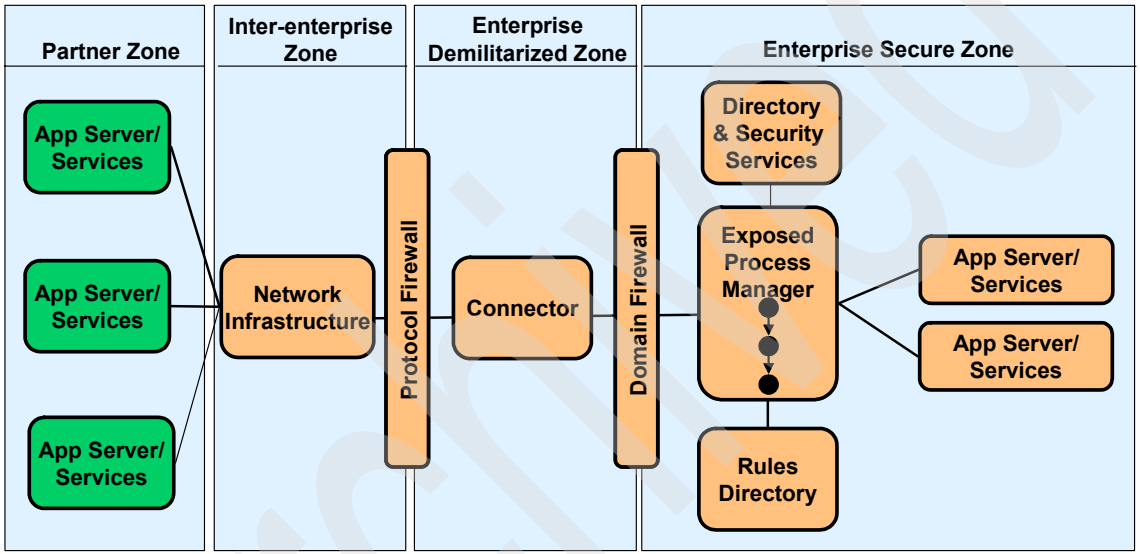


Figure 5-11 Exposed Serial Process runtime pattern: Generic profile

Archived

SOA profile

This profile shown in Figure 5-12 is identical to the generic profile, up to the Enterprise Secure Zone. At this point, the key differentiator to the SOA profile is used, the ESB. The Exposed ESB Gateway is used in conjunction with the Directory and Security Services to either allow the external request to be placed on the ESB as a service consumer or to allow an ESB request to consume a partner service to be sent out through the firewall.

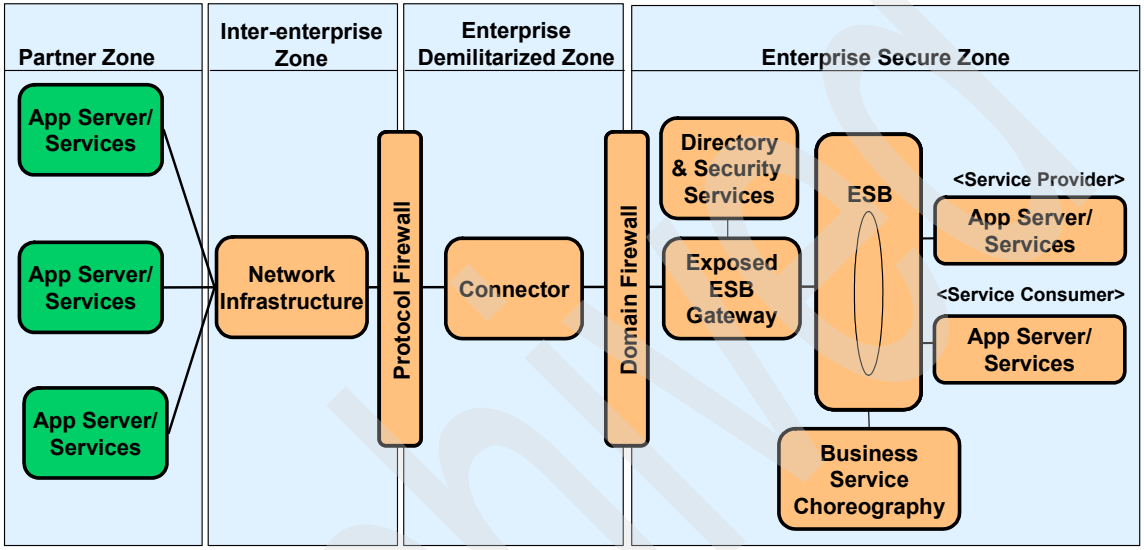


Figure 5-12 Exposed Serial Process runtime pattern: SOA profile

5.2.2 Exposed Serial Workflow pattern

The Exposed Serial Workflow pattern is similar to the Exposed Serial Process, but with the addition of the Staff Worklist Adapter.

Generic profile

Figure 5-13 shows the generic profile. Notice the addition of the Staff Worklist Adapter.

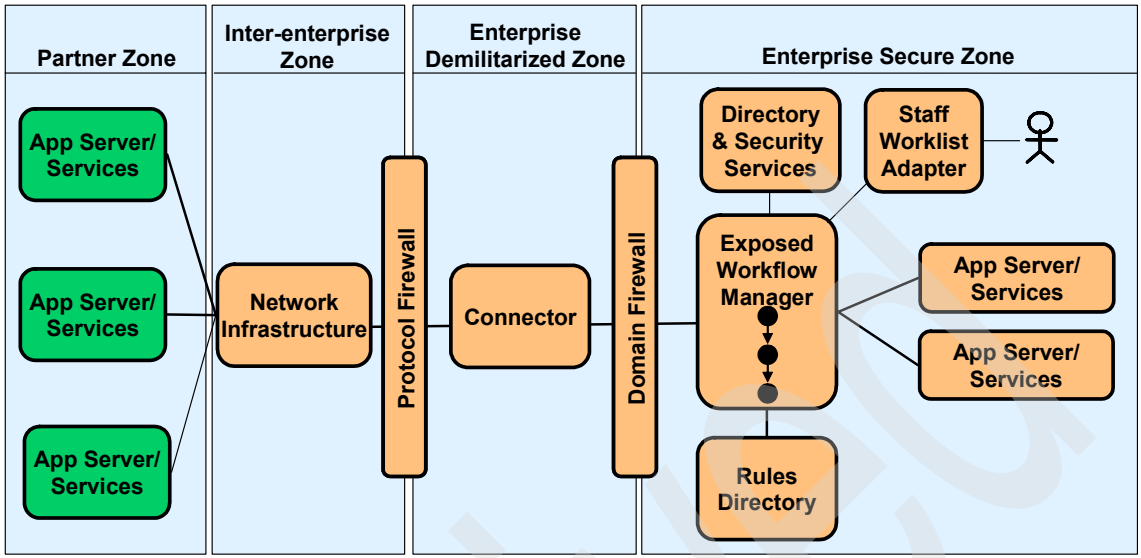


Figure 5-13 Exposed Serial Workflow runtime pattern: Generic profile

SOA profile

Figure 5-14 shows the SOA profile, again with the Exposed ESB Gateway.

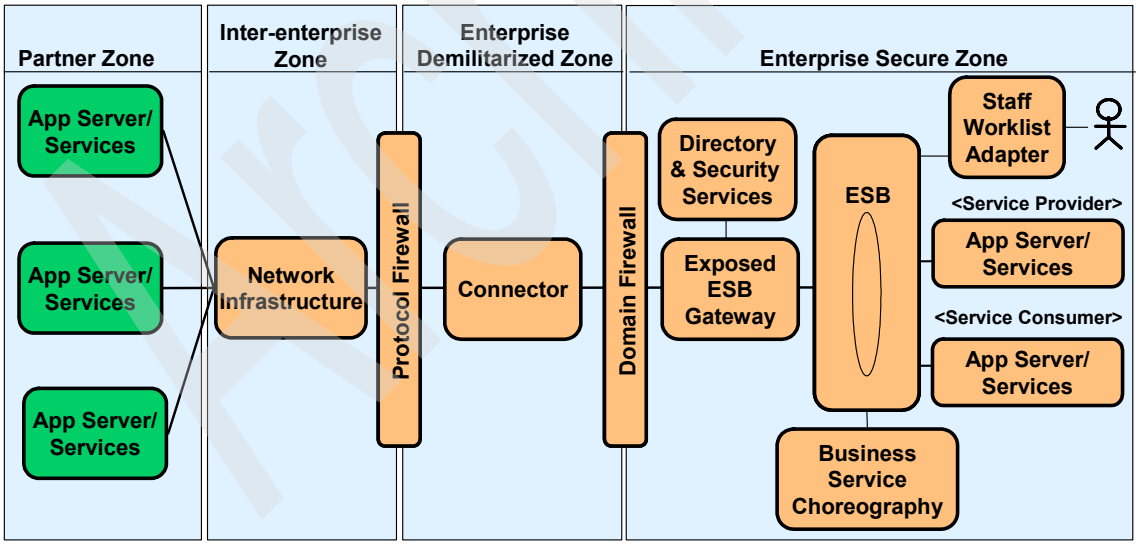


Figure 5-14 Exposed Serial Workflow runtime pattern: SOA profile

Archived

6

Product descriptions

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

- IBM WebSphere Process Server V6.0
- IBM DB2 Universal Database Enterprise Server Edition V8.2
- IBM Cloudscape
- IBM WebSphere Adapter for JDBC V6.0
- IBM WebSphere MQ V5.3
- IBM WebSphere Integration Developer V6.0

© Copyright IBM Corp. 2006. All rights reserved.

6.1 Runtime products

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

6.1.1 IBM WebSphere Process Server V6.0

IBM WebSphere Process Server V6.0 combines integration capabilities with a composite application platform. This combination results in an integration platform with a fully converged, standards-based business process engine, using the full power of WebSphere Application Server V6.0.

IBM WebSphere Process Server is a service-oriented architecture (SOA) integration platform built on a uniform invocation programming model and a uniform data representation model. The base runtime infrastructure for WebSphere Process Server is WebSphere Application Server. The Service Component Architecture (SCA) and business objects that are part of the SOA core provide uniform invocation and data-representation programming models. The SOA core includes Common Event Infrastructure for generating events for the monitoring and management of WebSphere Process Server. Supporting services provide the foundational business object and transformation framework for WebSphere Process Server. Service components represent the functional components required to build composite applications.

The combination of a powerful foundation (WebSphere Application Server and the SOA core) and service components in WebSphere Process Server allows quick development and deployment of sophisticated composite applications (Figure 6-1).

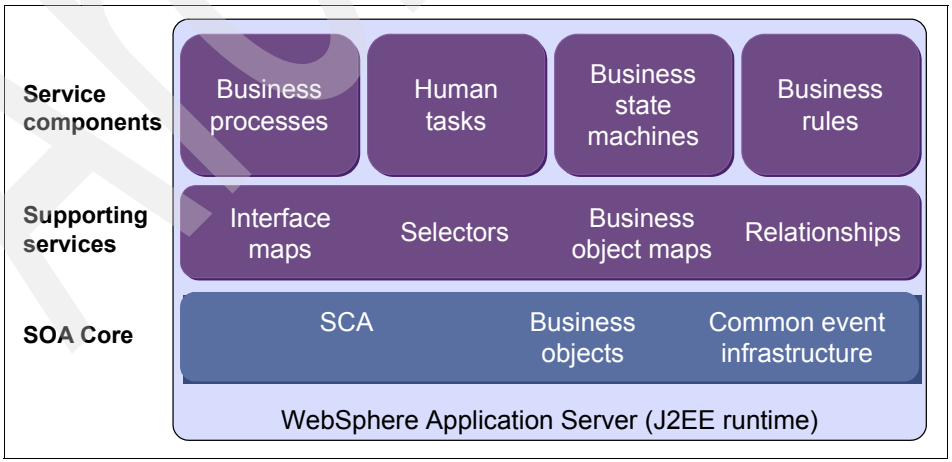


Figure 6-1 Architectural model for WebSphere Process Server

Everything in WebSphere Process Server is a component. These components have an interface and can be wired together to form a module. This enables the changing of any part of an application without affecting the other parts. For example, a human task can be replaced with a business rule without the need to modify the business process.

Components can interact with existing applications, using the following programming constructs:

- Java Beans
- Enterprise JavaBeans™
- Web services
- Java Message Service (JMS) messages

In addition, components can interact with other applications on enterprise information systems (EIS) with IBM WebSphere Adapters Version 6.0 and WebSphere Business Integration Adapters, based on WebSphere Business Integration Framework, Version 2.6.

WebSphere Process Server includes the following features:

- Web Services Business Process Execution Language (WS-BPEL) is used to choreograph the flow of business processes. Business process integration services are built on BPEL4WS version 1.1 and add major capabilities of the WS-BPEL version 2.0 specification.
- Business process integration semantics include compensation and recovery.
- Human task services allow role-based task assignment, invocation, and escalation.
- Business rules make business processes more flexible.
- Business process transformation services include mapping, mediation, and assembly.
- Business state machines enrich business transactions.
- A simplified programming model eliminates programming complexities.
- Support for Common Event Infrastructure (CEI) and Common Base Events (CBE) enhances tracking, auditing, and monitoring of business processes.
- A common administration console simplifies administration.
- Business models can be transferred from WebSphere Business Modeler to WebSphere Integration Developer.

You can find more information about the IBM WebSphere Process Server on the Web at:

http://www.ibm.com/software/integration/wps

Archived

6.1.2 IBM DB2 Universal Database Enterprise Server Edition V8.2

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

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

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

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

  These new capabilities simplify the development and deployment of DB2 applications and allow application developers to take advantage of the openness, performance, and scalability of DB2, without regard to the back-end database or the chosen application architecture.
- Integration of industry proven high availability disaster recovery technology, allowing line-of-business managers and the enterprise to benefit because applications face less risk of downtime

For more information, refer to the IBM DB2 Universal Database Web site:

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

6.1.3 IBM Cloudscape

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

- Rapid application development through the Java-based RDBMS that is built from the ground up for the embedded environment

  This platform independent, small footprint database integrates tightly with any Java based solution, allowing shortened development cycles.

- Support for Java technology standards

  Single application versions can be created that run on any standard Java Virtual Machine (JVM™).

- Does not require database administration or resource management and is invisible to non-technical users, eliminating the need for database administration at each client installation site

  IBM Cloudscape can also be deployed anywhere, for example, from notebook or desktop applications to robust server solutions.

- Tuned for high performance as well as efficient use of resources, with a straightforward migration path to various IBM DB2 versions

- Supports international characters and formats as well as a rich set of RDBMS features that are based on SQL-92E, including row locking, triggers, and stored procedures

- Available access to IBM Cloudscape from inside Java programs using Java Database Connectivity (JDBC) and the ability to embed the IBM Cloudscape database inside Java applications on the server

The Cloudscape Network server comes as part of the IBM Cloudscape package. It provides multiuser connectivity to IBM Cloudscape databases within a single system or over a network using Standard Distributed Relational Database Architecture™ (DRDA®) protocol.

For more information, see the IBM Cloudscape Web site:

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

6.1.4 IBM WebSphere Adapter for JDBC V6.0

The IBM WebSphere Adapter for JDBC enables object-oriented integration with enterprise applications through the data tier. This adapter can be used with applications which provide specific interface logic within the data tier or those that do not provide programmable or service interfaces.

The WebSphere Adapter for JDBC delivers an adapter based on the J2EE Connector Architecture V1.5. It implements the Service Data Objects (SDO) specification for exchanging data and the Enterprise Metadata Discovery (EMD) specification for wizard-driven configuration from database tables, views, and stored procedures. The adapter allows integration with application built on any database supported by a JDBC driver that adheres to the JDBC 2.0 or later standard.

The adapter supports access to relational data including tables, views, and stored procedures. It provides comprehensive support for create, update, delete,

and retrieve operations, among others. The adapter also provides a polling event mechanism to detect and deliver events from any application based on a relational database.

This product is the follow-on for IBM WebSphere Business Integration Adapter for JDBC. The packaging for IBM WebSphere Adapter for JDBC includes IBM WebSphere Business Integration Adapter for JDBC and IBM WebSphere Business Integration Adapter Framework.

Refer to the IBM WebSphere Adapters Web site for more information:

http://www.ibm.com/software/integration/wbiadapters

6.1.5 IBM WebSphere MQ V5.3

IBM WebSphere MQ provides assured once-only delivery of messages across more than 35 industry platforms using a variety of communications protocols.

The transportation of message data through a network is made possible through the use of a network of WebSphere MQ queue managers. Each queue manager hosts local queues that are containers used to store messages. Through remote queue definitions and message channels, data can be transported to its destination queue manager.

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

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

WebSphere MQ supports a variety of application programming interfaces (APIs), such as Message Queue Interface (MQI), Application Messaging Interface (AMI), and JMS. These APIs provide support for several programming languages as well as the point-to-point and publish/subscribe communication models. In addition to support for application programming, WebSphere MQ provides a number of connectors and gateways to a variety of other products, such as Microsoft Exchange, Lotus® Domino®, SAP/R3, CICS®, and IMS™.

To learn more, refer to the IBM WebSphere MQ Web site:

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

6.2 Development products

This section describes the product that is discussed and used throughout this book for development.

6.2.1 IBM WebSphere Integration Developer V6.0

IBM WebSphere Integration Developer V6.0 software is Eclipse technology-based tooling designed to focus on developer productivity. It provides easy-to-use authoring tools that enable you to rapidly assemble, build, and debug composite business-integration applications on a composite application-development framework.

With WebSphere Integration Developer, you can author SOA-based services and choreograph them into business processes that you can deploy on IBM WebSphere Process Server. WebSphere Integration Developer offers a role-based development experience that specifically targets the integration developer on a single and integrated Eclipse platform.

You can expand the capabilities of WebSphere Integration Developer to include business analysts and Java 2 Platform, Enterprise Edition (J2EE), developers by adding other IBM application-development tools. WebSphere Integration Developer is based on open-standards-based technologies, such as Web Services Description Language (WSDL), XML Schema Definition (XSD), and Web Service-Business Process Execution Language (WS-BPEL). This technology foundation enables you to model, implement, and deploy complex composite applications without extensive knowledge of the underlying implementation.

Archived

Together, WebSphere Process Server and WebSphere Integration Developer provide comprehensive services to enable the development of composite integration applications. These service components include:

- Business processes
- Human tasks
- Business-state machines
- Business rules
- Interface maps
- Business object maps
- Relationships
- Selectors
- Java objects
- Imports and exports for Web services, adapters, JMS and Enterprise JavaBeans (EJB)

For more information, see the IBM WebSphere Integration Developer Web site:

http://www.ibm.com/software/integration/wid

Archived

7

Product mappings

This chapter describes how we chose and laid out the products that we used to implement all of the patterns described in this redbook. To understand the product mappings in this chapter, first review the Runtime patterns in Chapter 5, “Runtime patterns” on page 73. Also, for an overview of each of the products described in this chapter, consult Chapter 6, “Product descriptions” on page 93.

The next step after choosing a Runtime pattern is to determine the products and platforms to be used. Choose your platform based on the requirements you gathered during the analysis phase, and consider such issues as:

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

Select a platform that fits the customer’s environment and ensures availability and performance so that the solution can grow along with the On Demand Business.

This chapter provides an overview of the product mappings that we chose based on the previously listed factors and on the scenarios that we chose to illustrate the Patterns for e-business that we selected.

© Copyright IBM Corp. 2006. All rights reserved.

7.1 Application Integration product mappings

The product mappings documented in this section focus on the key nodes (the Process Manager and Workflow Manager nodes) in the appropriate Runtime patterns. They illustrate the products and configurations that we chose. Keep in mind that other choices are possible.

Note: We developed these product mappings from our sample scenario on the Windows 2000 operating system. Several other platform options are available because IBM WebSphere products run on a wide range of platforms, for example Windows 2000, Linux®, IBM @server pSeries®, and IBM @server iSeries™.

7.1.1 Serial Process product mapping

This section presents product mappings for the Serial Process pattern using WebSphere Process Server. Figure 7-1 illustrates the topography.

In this product mapping, the source application running on WebSphere Process Server uses Web services to invoke the automated process instance implemented by the Process Manager node. Since instructions for finding the service are in the Web Services Description Language (WSDL) files managed by WebSphere Process Server, it is also responsible for Directory and Security Services. Although the repository likely runs on DB2 Universal Database in production, in our implementation, we used the Cloudscape database provided with WebSphere Process Server.

We also implemented our target applications on WebSphere Process Server. The Process Manager uses the Service Component Architecture to invoke Web services provided by the target applications running on WebSphere Process Server. We implemented the target applications as processes and used the Service Component Architecture to expose them as Web services. We could have chosen WebSphere Application Server to implement the target application services, or for that matter, any application server that provides Web services, such as Microsoft .NET.

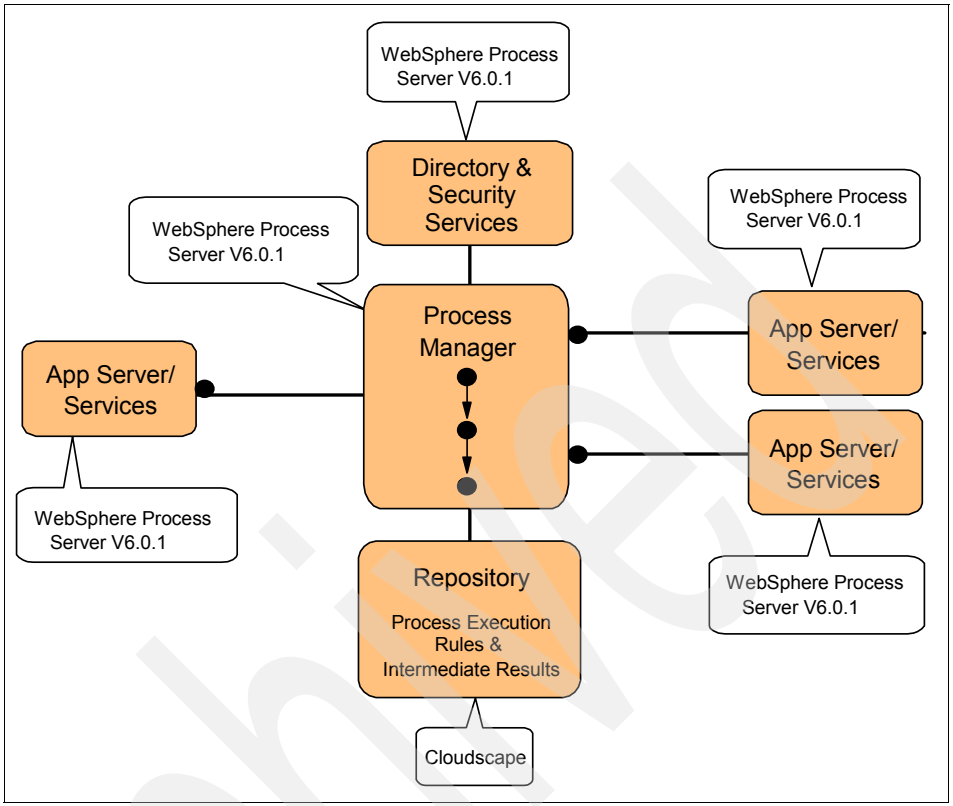


Figure 7-1 Serial Process: WebSphere Process Server product mapping

This product mapping is implemented in Chapter 10, “Serial Process scenario” on page 145.

Archived

7.1.2 Parallel Process product mapping

This section presents product mappings for the Parallel Process pattern using WebSphere Process Server. Figure 7-2 illustrates the topography. Notice the remarkable similarity between Figure 7-2 and Figure 7-1. The topologies are nearly identical; the only difference is that we implement a business process with parallel paths in the Process Manager. See 7.1.1, “Serial Process product mapping” on page 102, for a description of each node.

Product wise, the only other difference is the use of DB2 Universal Database as the Repository node. WebSphere Process Server will not truely execute multiple threads of execution in parallel if a Cloudscape database is used.

Finally, notice that one of the App Server/Services nodes is now implemented by DB2 Universal Database. Connectivity is through a Java Database Connectivity (JDBC) adapter, which in this case is also part of WebSphere Process Server.

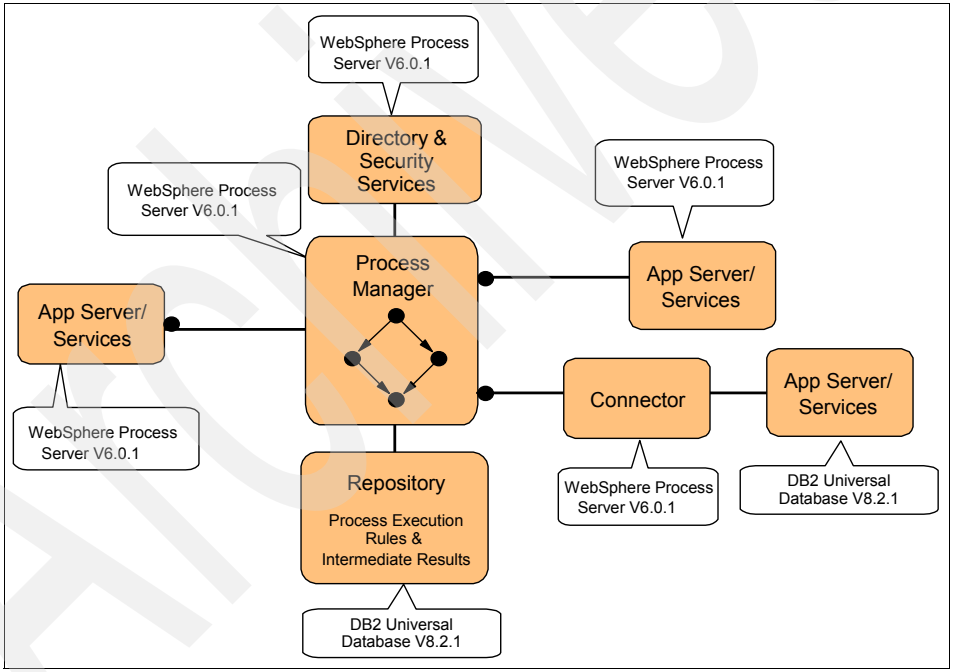


Figure 7-2 Parallel Process: WebSphere Process Server product mapping

This product mapping is implemented in Chapter 12, “Parallel Process scenario” on page 331.

7.1.3 Parallel Workflow product mapping

This section presents product mappings for the Workflow variation of the Parallel Process pattern using WebSphere Process Server. Figure 7-3 illustrates the topography. This product mapping adds a few nodes to the mapping for the Parallel Process to represent the human interaction. The Process Manager node is replaced with the Workflow Manager node, although this node is still implemented by WebSphere Process Server.

The Workflow Manager invokes processes that require human interaction. The human interactions within the process flow are resolved and handled by the Staff Worklist Adapter, which in this case is the Business Process Explorer included in WebSphere Process Server. Another good option for the Staff Worklist Adapter is the Process Portal component of WebSphere Portal. WebSphere Process Server includes plug-ins for different user directories to store the users and roles necessary for human interaction. We used the Windows User Registry. Another popular choice, and one more likely to be used in production, is a Lightweight Directory Access Protocol (LDAP) directory such as IBM Tivoli® Directory Server.

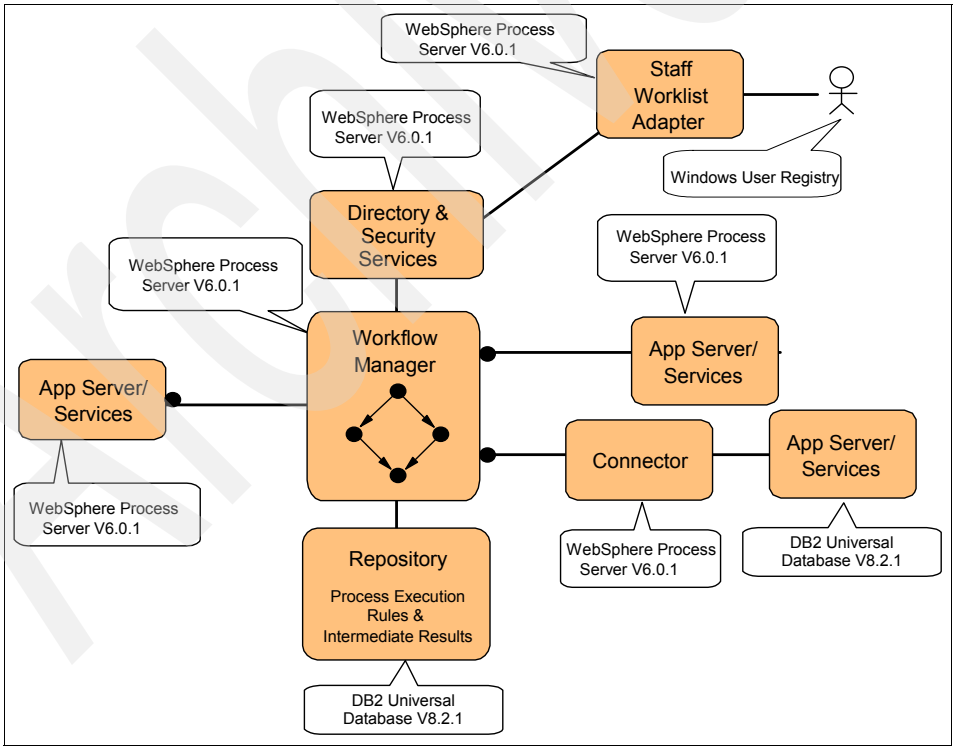


Figure 7-3 Parallel Workflow variation: WebSphere Process Server product mapping

This product mapping is implemented in Chapter 13, “Parallel Workflow scenario” on page 393.

7.2 Extended Enterprise product mappings

The product mapping documented in this section focuses on the key nodes in the Exposed Serial Process pattern. It illustrates the products and configurations that we chose. Keep in mind that other options are possible.

7.2.1 Exposed Serial Process product mapping

In this product mapping for the Exposed Serial Process runtime pattern, shown in Figure 7-4, the Exposed Process Manager node is implemented using WebSphere Process Server.

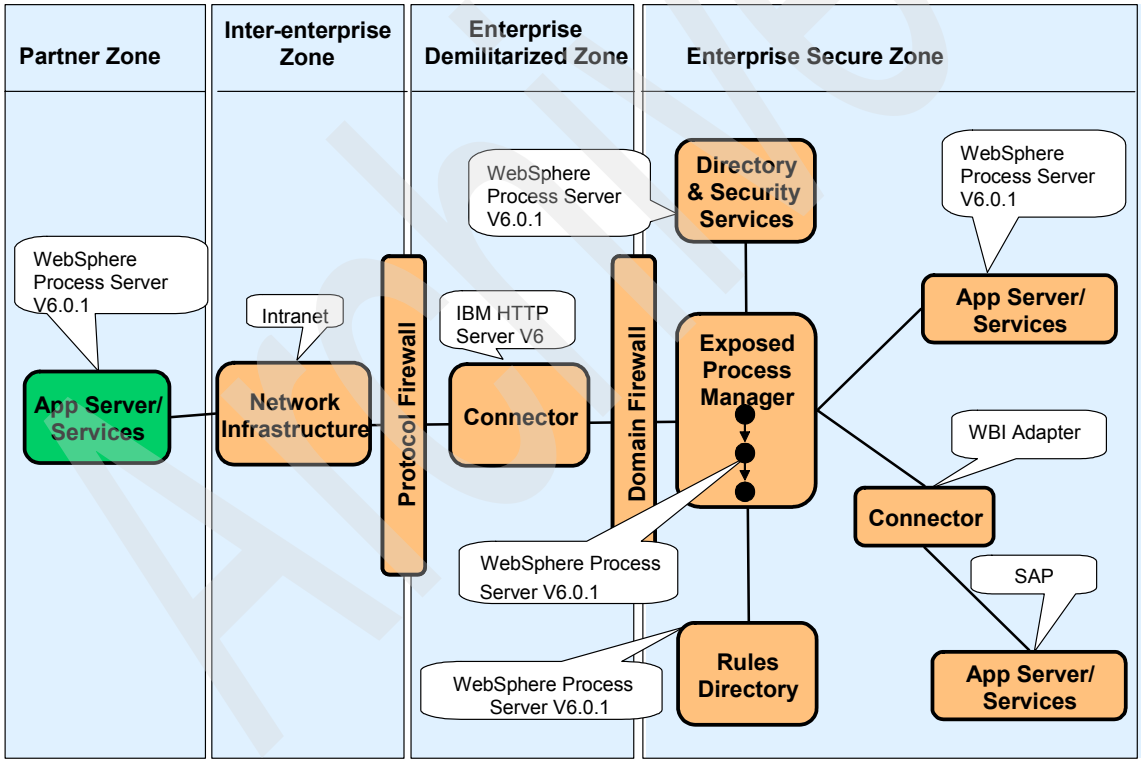


Figure 7-4 Exposed Serial Process: WebSphere Process Server product mapping

We illustrate two choices for implementing services in the Enterprise Secure Zone. As in the previous sections, we implemented one of our target applications on WebSphere Process Server. The second target application illustrates a connection through a WebSphere Business Integration Adapter to an enterprise application, in this case SAP. As in the previous cases, the directories are also hosted on WebSphere Process Server.

An application service at the partner site invokes the automated process instance implemented by the Process Manager node over the Internet using SOAP/HTTPS. The IBM HTTP Server acts as a connector by exposing SOAP/HTTPS to partner organizations while allowing the Process Manager to process standard SOAP over HTTP (SOAP/HTTP) calls.

In the Directory and Security Services node, the service integration bus within WebSphere Process Server is configured to secure all transactions to the external Partner Zone using WS-Security integrity and confidentiality.

This product mapping is implemented in Chapter 11, “Exposed Serial Process scenario” on page 235.

Archived

Archived

Part 2

Business scenario and guidelines

Archived

© Copyright IBM Corp. 2006. All rights reserved.

Archived

8

Business scenarios used in this book

In this chapter, we introduce ITSO Electronics, the imaginary customer in our simplified business scenarios. As ITSO Electronics's business grows, its requirements become more complex. Coincidentally, each new phase of the business requires the application of a different Patterns for e-business pattern. These business scenarios are simplified so that we can implement them in this redbook. They show you how the Patterns for e-business can be applied to the sorts of problems that growing enterprises often experience.

Each of the business scenarios described in this chapter are implemented in Part 3, "Scenario implementation" on page 143, of this redbook.

© Copyright IBM Corp. 2006. All rights reserved.

8.1 Customer overview

This section provides some background information about ITSO Electronics, including:

- Business profile and goals
- Existing environment
- Non-functional requirements or quality of service requirements

8.1.1 Business profile

ITSO Electronics is a fictional retail electronics store that specializes in both consumer and business goods. Founded 30 years ago, the company has grown from a small local storefront to a large, regional department store featuring televisions, computer equipment, stereo equipment, and household electronics. The company has a large wholesale business as well, supplying computer equipment, fax machines, copiers, and other business electronics to merchants throughout the region.

Part of ITSO Electronics's growth has come from acquisition, and it currently maintains two separate and overlapping wholesale divisions. Its retail department can use either wholesaler to fulfill orders. The retail operation also uses external wholesalers for items that cannot be provided internally.

Business goals

By integrating their retail ordering and wholesale inventory processes, ITSO Electronics plans to:

- Reduce costs by reducing the staff workload associated with placing stock replenishment orders with the wholesale department
- Increase customer satisfaction by decreasing the likelihood that an item will be out of stock

Because the process is not automated, the retail department only replenishes stock on a monthly basis and often runs out of popular items. ITSO Electronics will move to an on demand environment, where inventory will be kept at the wholesalers and orders will be fulfilled directly by the wholesalers.

Archived

8.1.2 Existing environment

The existing environment at ITSO Electronics should be considered from a business perspective as well as an IT perspective.

Business perspective

From the business perspective, the existing environment includes:

- The wholesale ordering process shown in Figure 8-1

  This process uses the wholesale department's inventory business process, which employs paper-based forms and manual processes.

- The internal retail department, which places monthly orders with each wholesale department in order to replenish its stock

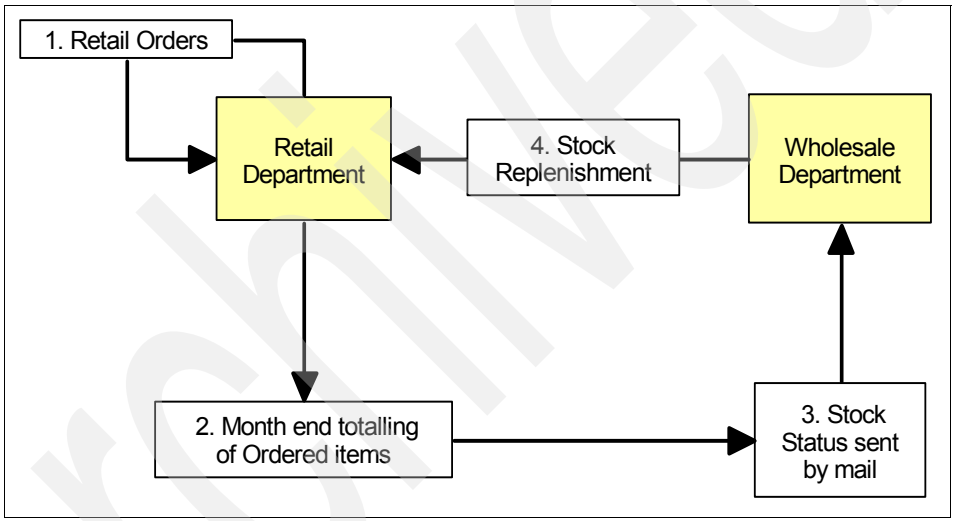


Figure 8-1 ITSO Electronics: Wholesale ordering process flow

Archived

IT perspective

The existing IT environment, shown in Figure 8-2, includes:

- Two wholesale inventory systems

  These systems implement the core business processes of each wholesale department.

- A retail ordering system

  The system used by retail staff recently upgraded to a Self-Service, browser-based Java 2 Platform, Enterprise Edition (J2EE), application.

- External wholesalers that own heterogeneous IT infrastructures

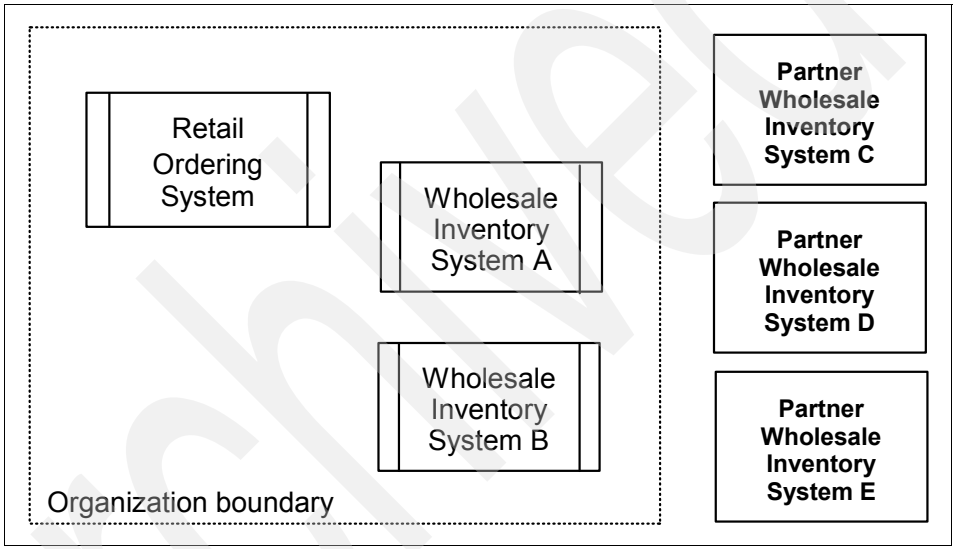


Figure 8-2 ITSO Electronics: Current IT infrastructure

8.1.3 Non-functional requirements

ITSO Electronics requires that all solutions provide a standard quality of service. The following specific criteria must be met:

- Availability
  - Solutions meet both the defined unplanned- and planned-downtime requirements.
  - Meaningful messages are provided to system users during downtime.
- Operability
  - Solutions provide suitable logs and traces.

Archived

- Federation
  - The responsibilities of the stakeholders are clearly defined and agreed to by all parties.
- Performance
  - Solutions meet the defined throughput and response times.
  - Solutions scale to provide for future growth.
- Security
  - Sensitive systems and data are protected from unauthorized access.
  - Nonrepudiation of the end user for all commercial transactions is provided.
- Standards compliance
  - Appropriate standards are identified and applied.
- Transactionality
  - Transaction support is required for our business scenario where stock orders span multiple wholesale systems.

It is beyond the scope of this redbook to define such requirements in real and measurable terms for our sample scenarios. In a real-world implementation, you must ensure that the delivered solution meets these requirements.

8.2 Scenario overview

ITSO Electronics wants to integrate its retail and wholesale departments. Currently, both organizations have proven IT infrastructures, but they have no interconnectivity. ITSO Electronics wants to focus on the inventory and order replenishment process.

Currently, the sold items are tallied at the end of the month by the retail ordering process and delivered to the wholesale organization by internal mail. This creates a lag in the inventory replenishment process and causes many out-of-stock situations. The business goals are twofold: to minimize the loss of sales due to items being out of stock, and at the same time, to minimize the inventory costs at the retailer.

Archived

Selecting a Business or Integration pattern

In 4.1, “Application Integration pattern” on page 48, the following drivers are listed for selecting the Application Integration pattern:

- The business processes need to be integrated with existing business systems and information.
- The business activity needs to aggregate, organize, and present information from various sources within the organization.

Both drivers apply to ITSO Electronics. The business processes of the retail department and the wholesale department need to be integrated by combining the existing retail business system with the existing wholesale business system. One valid use of this pattern is to replenish the retail inventory from the wholesale inventory whenever the retail inventory sunk to a certain level. This effectively prevents items from going out of stock as long as they can be replenished faster than they sold.

However, ITSO Electronics has elected to use the Application Integration pattern to implement an on demand retail operation where all requests from the retail ordering system are fulfilled directly from wholesale inventory. In essence, the retail operation will not carry any inventory.

Figure 8-3 shows the Application Integration pattern applied to our scenario.

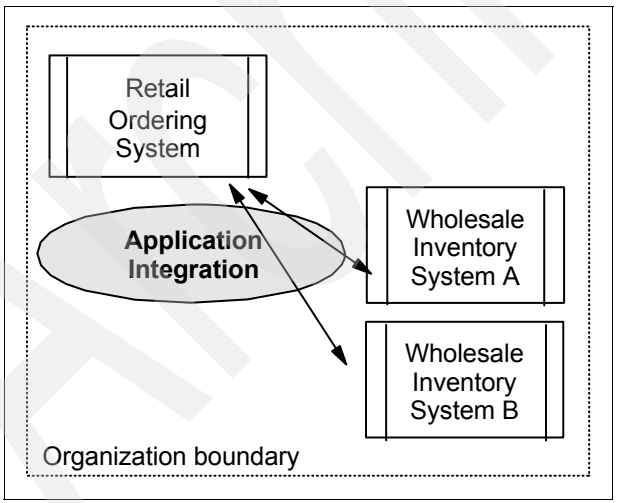


Figure 8-3 ITSO Electronics overview diagram

8.2.1 Scenario 1: Internal ordering on demand

In the first stage of the internal implementation, ITSO Electronics integrates its retail system and its wholesale systems. This allows the internal retail ordering system to forward orders to the internal wholesale system that is best able to fulfill the order. The automated process implements the business rule that determines which wholesale location to order from, eliminating the need for a retail group employee to choose it manually. The integrated system meets the business requirements for order confirmation.

For scenario 1, we can identify two *actors*:

- The retail system
- The wholesale system

We can also identify a *use case*, Place Order.

Actors

Table 8-1 provides details about the retail system actor.

Table 8-1 Retail system actor details

Actor name	Retail system
Brief description	The retail system implements the retail ordering business process.
Status	Primary
Relationships	
Associations to use cases	001 Place Order

Table 8-2 provides details about the wholesale system actor.

Table 8-2 Wholesale system actor details

Actor name	Wholesale system
Brief description	The wholesale system implements the order fulfillment business process.
Status	Primary
Relationships	
Associations to use cases	001 Place Order

Archived

Use case 001: Place Order

Table 8-3 provides details about the Place Order use case.

Table 8-3 Use case 001: Place Order

Use case name	001 Place Order
Subject area	Wholesale ordering
Business event	An item sold by the retail division needs to be fulfilled from the wholesale inventory.
Actors	Retail system, wholesale systems
Use case overview	The retail system places an order with a wholesale system for an item.
Preconditions	The retail system supplies a part number and the quantity to be ordered.
Termination outcome 1	The retail inventory system records the confirmation number of the order, the wholesale system where the order was placed, and the expected delivery date.
Notes	

Figure 8-4 illustrates the use-case model of scenario 1.

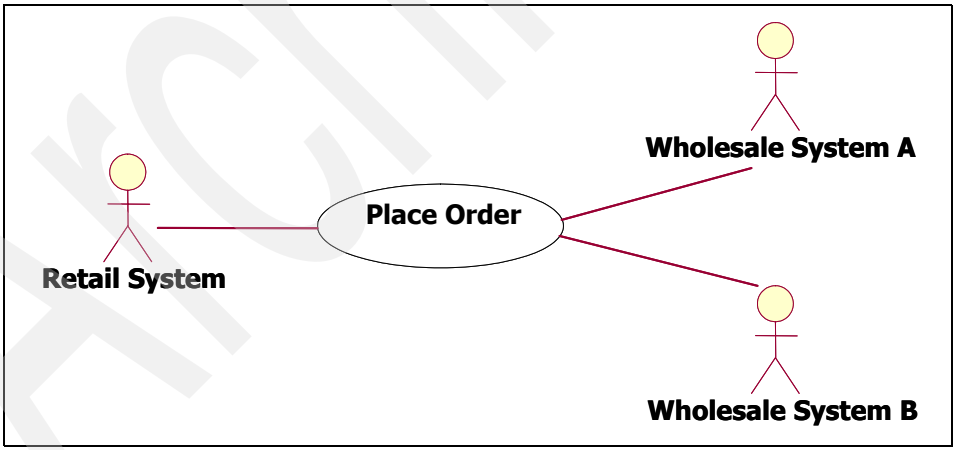


Figure 8-4 Scenario 1 use-case model

Archived

Selecting an Application pattern

In Table 4-1 on page 51 and Table 4-2 on page 52, the following business and IT drivers are listed for selecting the Application Integration::Serial Process application pattern:

- Improve the organizational efficiency
- Reduce the latency of business events
- Support a structured exchange within the organization
- Support real-time one-way message flows
- Support real-time request/reply message flows
- Support dynamic routing of messages to one of many target applications
- Support dynamic distribution of messages to multiple target applications
- Support automated coordination of business process flow
- Minimize total cost of ownership (TCO)
- Leverage existing skills
- Leverage the existing investment
- Enable back-end application integration
- Minimize application complexity
- Minimize enterprise complexity
- Improve maintainability
- Improve flexibility by externalizing process logic from application logic

Scenario 1 explicitly or implicitly incorporates most of these drivers. You can implement this scenario using either the Serial or Parallel Process application pattern; for the sake of simplicity, we illustrate this scenario using the Serial Process application pattern. Figure 8-5 shows how we implement this scenario. If we are integrating a process that includes human interaction, we choose the Workflow variation for the Serial or Parallel Process application pattern instead.

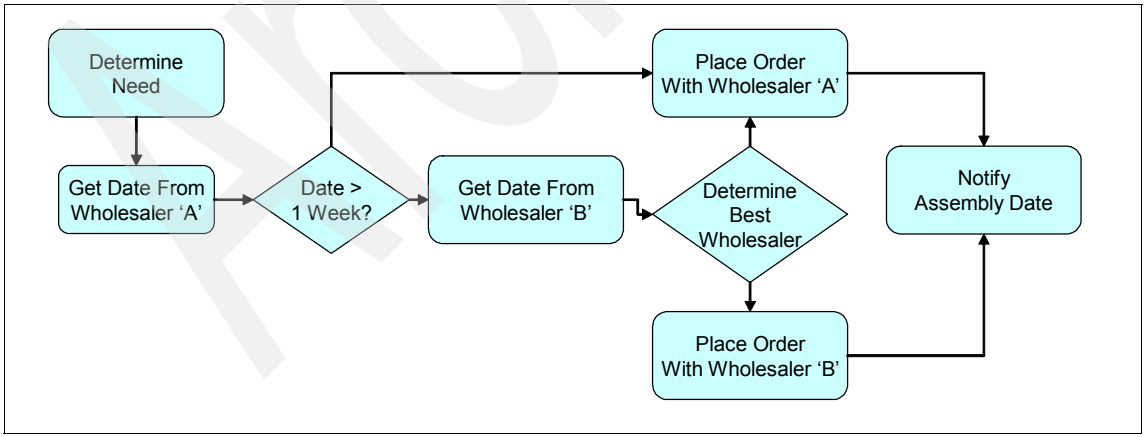


Figure 8-5 Serial Process scenario implementation

Archived

See Chapter 10, “Serial Process scenario” on page 145, which describes how we apply the Serial Process Application pattern to scenario 1 using WebSphere Process Server.

8.2.2 Scenario 2: Internal ordering on demand in parallel

Soon after the successful implementation of scenario 1, the retail crew at ITSO Electronics observes that because they queried each wholesaler consecutively, a delay in receiving an answer from Wholesaler A means a delay in the entire process, even if Wholesaler B provides a quick response. When the crew reports this to IT, they decide to modify the process so that both wholesalers are queried at the same time. Management also indicates that the written records that were used to transfer information provide a good source of historical information, and that the automated system does not keep similar records. IT responds by adding an audit trail to the requirements.

Furthermore, since scenario 1 was so successful and implementation was far ahead of schedule (due to the incredible productivity gains realized by using WebSphere Process Server), the IT department decides to further streamline the process by integrating their accounting systems into it. Their business rules require simultaneous entries into the general ledger and accounts payable, and they decide to implement a transaction for this purpose.

Scenario 2 includes two additional *actors:*

- General ledger system
- Accounts payable system

Actors

Table 8-4 provides details about the general ledger actor.

Table 8-4 General ledger actor details

Actor name	General ledger
Brief description	This accounting system automates the general ledger, which refers to all accounts of a business.
Status	Primary
Relationships	
Associations to use cases	002 Place Order with Parallel Queries

Table 8-5 provides details about the accounts payable actor.

Table 8-5 Accounts payable actor details

Actor name	Accounts payable
Brief description	This accounting system automates the accounts payable, which is the amount owed to creditors.
Status	Primary
Relationships	
Associations to use cases	002 Place Order with Parallel Queries

Use case 002: Place Order with Parallel Queries

Table 8-6 provides details about the Place Order with Parallel Queries use case.

Table 8-6 Use case 002: Place Order with Parallel Queries

Use case name	002 Place Order with Parallel Queries
Subject area	Wholesale ordering
Business event	An item sold by the retail division needs to be fulfilled from the wholesale inventory, and accounts should be updated.
Actors	Retail system, wholesale systems, general ledger, accounts payable
Use case overview	The retail system places an order with a wholesale system for an item. After the order is placed, accounting systems are updated.
Preconditions	The retail system supplies a part number and the quantity to be ordered.
Termination outcome 1	The retail inventory system records the confirmation number of the order, the wholesale system where the order was placed, and the expected delivery date. The accounting systems record the monetary transaction.
Notes	

Archived

Figure 8-6 illustrates the use-case model for scenario 2.

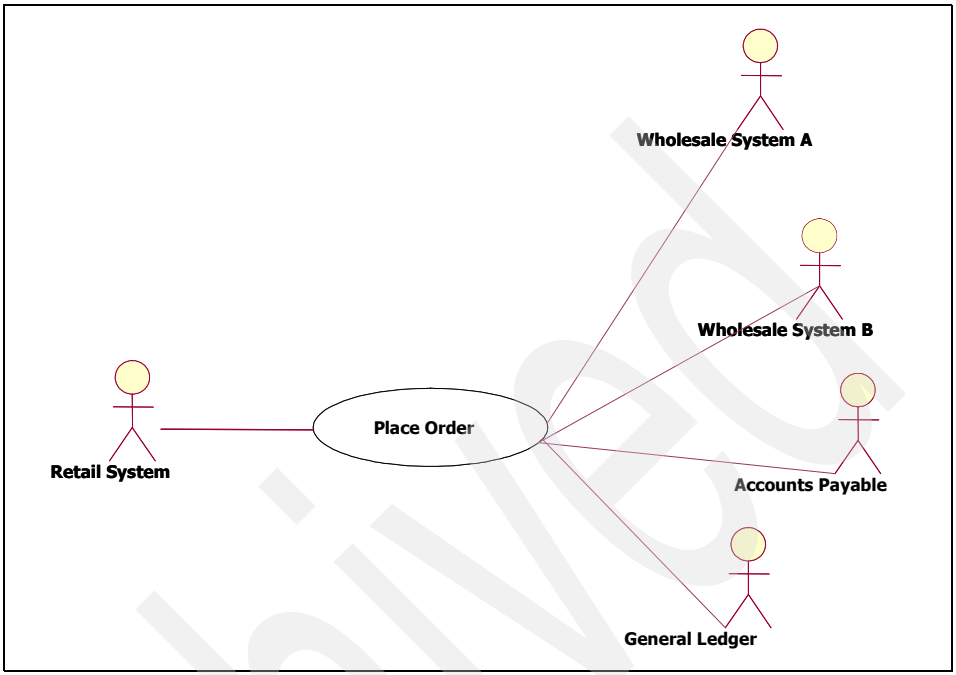


Figure 8-6 Scenario 2 use-case model

Archived

Selecting an Application pattern

Table 4-1 on page 51 and Table 4-2 on page 52 list the business and IT drivers for selecting the Application Integration::Parallel Process application pattern. In addition to the drivers listed in scenario 1, the driver to *reduce cycle time through parallel execution of portions of a process flow* applies.

This scenario specifically requires a parallel process implementation, as shown in Figure 8-7.

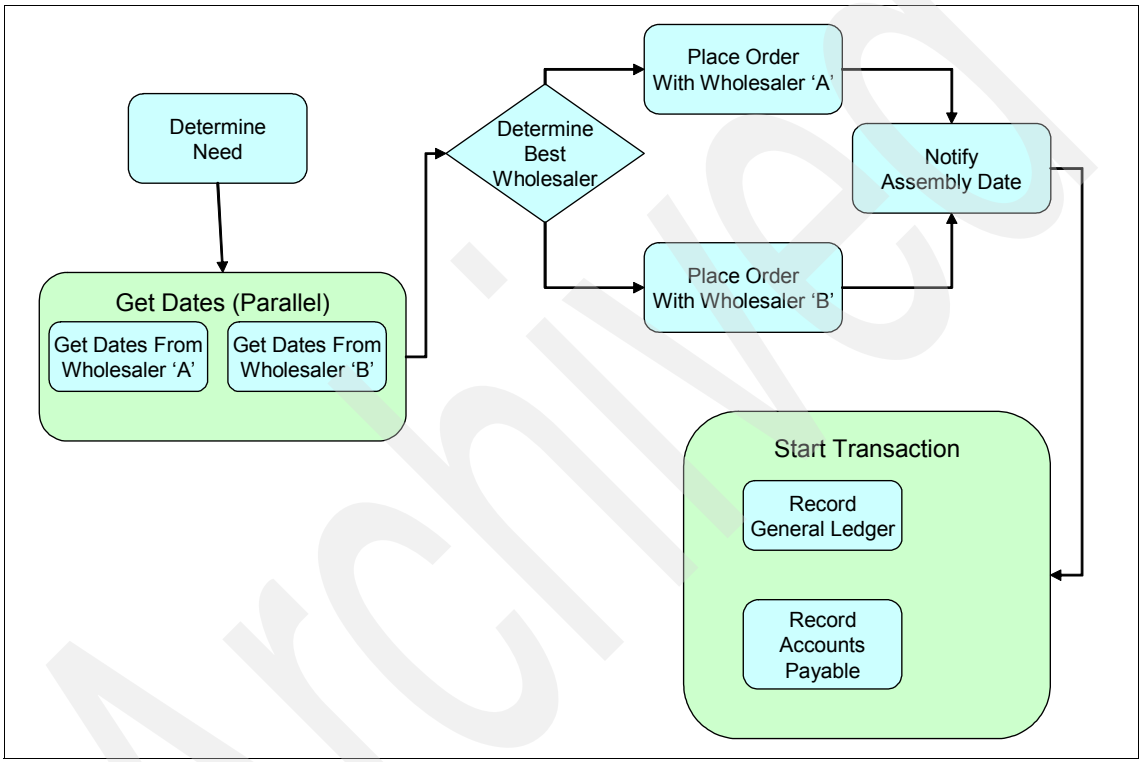


Figure 8-7 Parallel Process scenario implementation

Refer to Chapter 12, "Parallel Process scenario" on page 331, for a description of this scenario.

Archived

8.2.3 Scenario 3: Ordering on demand with multiple wholesalers

After the successful implementation of scenarios one and two, the increase in revenue has resulted in monetary rewards for employees in the ITSO Electronics IT department. While the staff is celebrating, change continues rapidly. Also ITSO Electronics's two wholesaler departments have just merged, with Wholesaler B migrating all its processes to Wholesaler A's systems. With this change, the business managers have become concerned that a single wholesaler may not be able to supply every product in a timely manner.

This could lead to frustrated customers and cancelled orders. Traditionally, ITSO Electronics has maintained relationships with a number of external wholesalers. The next challenge is to incorporate those wholesalers into the order process. The new process includes a rule that if the internal wholesaler cannot deliver a product in a "timely" manner, third parties are given the opportunity to do so. Furthermore, the definition of *timely* is expected to change with business conditions, so the rule describing timeliness must be changeable by a business analyst without requiring IT department intervention. Finally, since business partners can and will change regularly, the process must accommodate the available partners when it is executed.

For scenario 3, we identify an additional actor, External wholesaler. We can also identify an additional *use case*, Place Order with Internal or External Wholesalers.

Actors

Table 8-7 provides details about the external wholesaler.

Table 8-7 External wholesaler actor details

Actor name	External wholesaler
Brief description	An ITSO Electronics business partner who can supply wholesale products and has agreed to participate in ITSO Electronics's order process
Status	Primary
Relationships	
Associations to use cases	003 Place Order with Internal or External Wholesalers

Archived

Use case 003: Place Order with Internal or External Wholesalers

Table 8-8 provides details about the Place Order with Multiple Wholesalers use case.

Table 8-8 Use case 003: Place order with internal or external wholesalers

Use case name	003 Place Order with Internal or External Wholesalers
Subject area	Wholesale ordering
Business event	An item sold by the retail division needs to be fulfilled from either internal or external wholesale inventory. If the internal wholesaler cannot fulfill an order in a timely manner, the order is forwarded to external wholesalers.
Actors	Retail system, internal wholesale systems, external wholesale systems
Use case overview	The retail system places an order for an item with the internal wholesale systems. If they cannot fulfill the order, the process queries external wholesalers until it finds one that is capable of fulfilling it.
Preconditions	The retail system supplies a part number and the quantity to be ordered.
Termination outcome 1	The retail inventory system records the confirmation number of the order, the wholesale system where the order was placed, and the expected delivery date. If no wholesaler can fulfill the order, the process notifies the retail system.
Notes	

Archived

Figure 8-8 illustrates the use-case model of scenario 3.

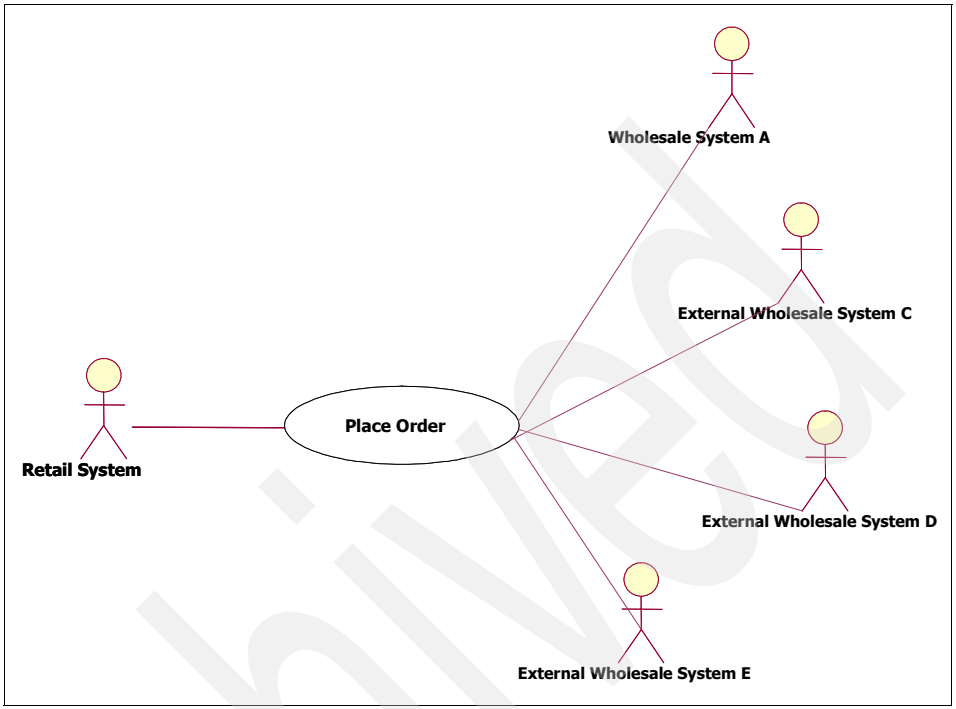


Figure 8-8 Scenario 3 use-case model

Selecting an Application pattern

Table 4-3 on page 65 and Table 4-4 on page 66 list the business and IT drivers for selecting the Extended Enterprise::Exposed Serial Process application pattern. In addition to the drivers listed in scenario 1, the driver to *link business processes across separate enterprises* applies.

Since there are no longer two internal wholesalers, the IT department decides to revert to an Exposed Serial Process as they assess the effects of opening up the process to third parties. To fulfill the flexibility requirements, they design a business state machine to query all available partners until one fulfills their request. Since external partners cannot be trusted in the same way as internal ones, they design the business state machine to properly handle cancellation by a partner. Partner interactions are secured with WS-Security as well.

Archived

Figure 8-9 illustrates the resulting implementation.

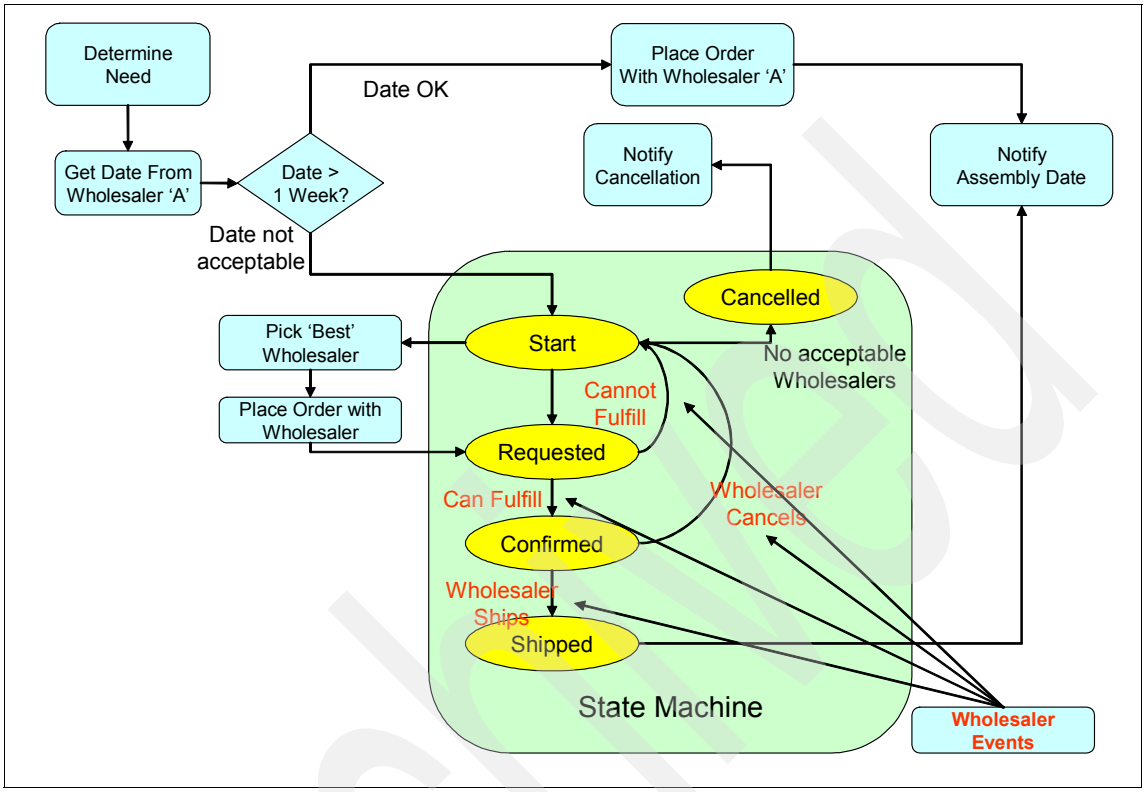


Figure 8-9 Exposed Serial Process with a business state machine scenario implementation

Explore the details of this process and its implementation in Chapter 11, "Exposed Serial Process scenario" on page 235.

8.2.4 Scenario 4: Ordering on demand with approval workflow

One of ITSO Electronics's most profitable services is building custom home entertainment systems for customers with more money than time. The orders for these systems comprise multiple parts, and thus are often partially fulfilled by each wholesaler. Because these orders are so profitable, management has dictated that delivery date promises to the customer must be kept at all costs. In addition, concern for an order's timeliness mandates that fulfillment be through the internal wholesalers.

To comply with this requirement regarding delivery dates, an ITSO Electronics manager must approve every order before a date is specified to the customer. Orders that are not approved are sent back to the retail department for rework,

and the corresponding orders to wholesalers are cancelled. Due to a generous vacation policy, managers can be out of the office for several weeks at a time, so there is also an escalation policy if an approval is not quickly acted upon.

For scenario 4, we identify an additional actor, Approver. We can also identify an additional *use case*, Place Order with Approval Workflow.

Actors

Table 8-9 provides details about the Approver actor.

Table 8-9 Approver actor details

Actor name	Approver
Brief description	A manager who is authorized to approve or reject orders
Status	Primary
Relationships	
Associations to use cases	004 Place Order with Approval Workflow

Use case 004: Place Order with Approval Workflow

Table 8-10 provides details about the Place Order with Approval Workflow use case.

Table 8-10 Use case 004: Place Order with Approval Workflow

Use case name	004 Place Order with Approval Workflow
Subject area	Wholesale ordering
Business event	The components of an item sold by the retail division need to be fulfilled from the wholesale inventory.
Actors	Retail system, wholesale systems, approver
Use case overview	The retail system places an order for a system comprising multiple parts with its wholesale systems. The approver must sign off on the order. If it is not approved, the orders to the wholesalers must be cancelled and the retail system notified.
Preconditions	The retail system supplies a set of part numbers and the quantity of each.

Archived

Use case name	004 Place Order with Approval Workflow
Termination outcome 1	The retail inventory system records the confirmation number of the order, the wholesale system where the order was placed, and the expected delivery date, or it records a nonapproval.
Notes	

Figure 8-10 illustrates the use-case model for scenario 4.

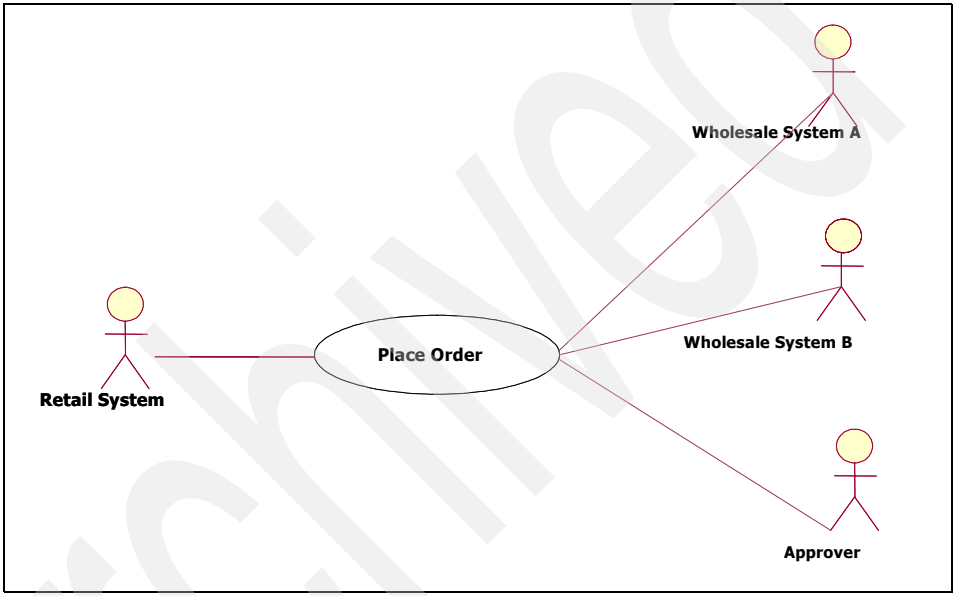


Figure 8-10 Scenario 4 use-case model

Archived

Selecting an Application pattern

Table 4-1 on page 51 and Table 4-2 on page 52 list the business and IT drivers for selecting the Workflow variation of the Application Integration::Parallel Process application pattern. In addition to the drivers listed in scenario 1, the following drivers are defined:

- Support for human interaction and intervention within the process flow
- Support for long-running transactions

The IT department thinks it's strange to place the wholesaler order before the retailer is approved, but being programmers they also view it as a golden opportunity to exercise more WebSphere Process Server functionality. The implementation will be similar to that of scenario 2, with order approval represented as a human task. They also decide to add a fault handler and compensation, which will allow them to enact the cancellation scenario. Figure 8-11 illustrates the resulting implementation.

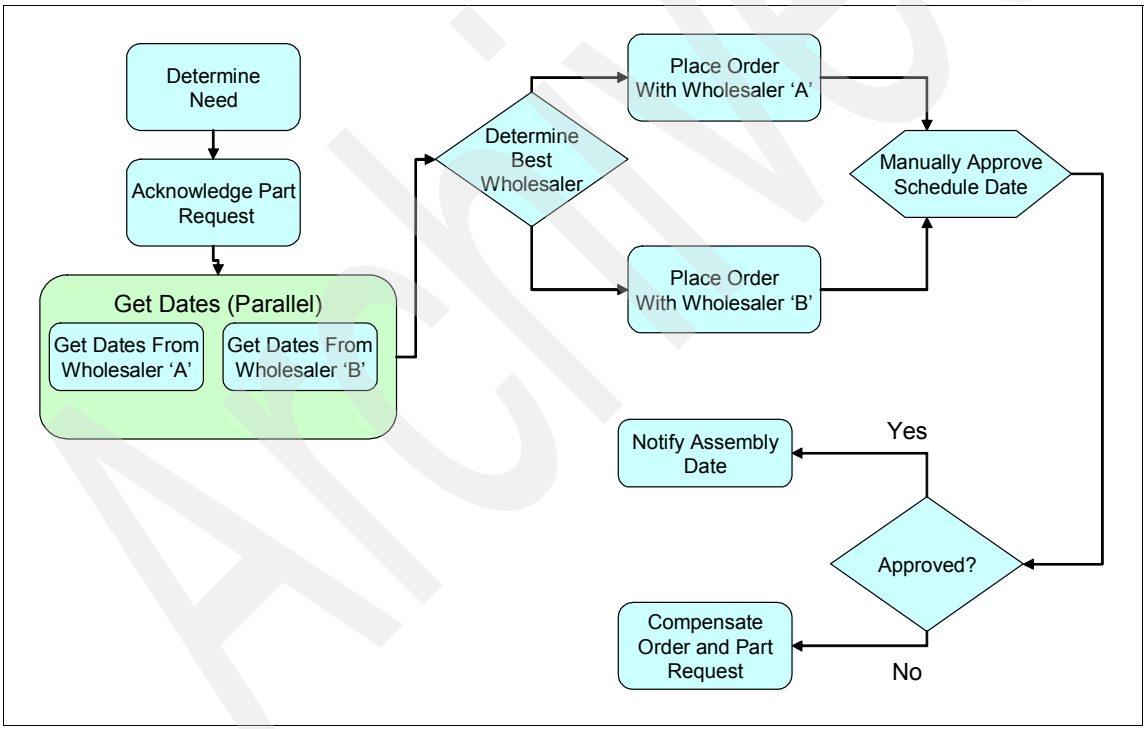


Figure 8-11 Parallel process with workflow scenario implementation

Refer to Chapter 13, "Parallel Workflow scenario" on page 393, to learn more about the details of this scenario.

9

Technology descriptions

This chapter discusses the essential technologies that this book uses to implement the scenarios in this redbook. It describes the following technologies:

- Service Data Objects (SDO)
- J2EE Connector architecture (JCA)
- Web services
- Java 2 Platform, Enterprise Edition (J2EE)
- Transport protocols

© Copyright IBM Corp. 2006. All rights reserved.

9.1 Service Component Architecture

Service Component Architecture (SCA) is a set of specifications which describe a model for building applications and systems using a service-oriented architecture (SOA). SCA extends and complements prior approaches to implementing services, and SCA builds on open standards such as Web services.

For more details about SCA, see 3.4, “Service Component Architecture” on page 39. Also refer to the following Web site:

http://www.ibm.com/developerworks/library/specification/ws-sca/

9.2 Service Data Objects

Service Data Objects (SDO) is designed to simplify and unify the way in which applications handle data. Using Service Data Objects, application programmers can uniformly access and manipulate data from heterogeneous data sources, including relational databases, Extensible Markup Language (XML) data sources, Web services, and enterprise information systems (EIS).

SDO is based on the concept of disconnected data graphs. A *data graph* is a collection of tree-structured or graph-structured data objects. Under the disconnected data graph architecture, a client retrieves a data graph from a data source, mutates the data graph, and can then apply the data graph changes back to the data source.

The task of connecting applications to data sources is performed by *data mediator services*. Client applications query a data mediator service and get a data graph in response. Client applications send an updated data graph to a data mediator service to have the updates applied to the original data source. This architecture allows applications to deal principally with data graphs and data objects.

SDO enables both a static (or strongly typed) programming model and a dynamic (or loosely typed) programming model. This enables a simple programming model without sacrificing the dynamic model needed by tools and frameworks.

SDO also provides a metadata application programming interface (API), which allows applications, tools, and frameworks to introspect the data model for a data graph. The SDO metadata API unifies data-source-specific metadata APIs to enable applications to handle data from heterogeneous data sources in a uniform way.

SDO is intended to be language-neutral and available in a range of programming languages.

To learn more information about SDO, see the following Web site:

http://www.ibm.com/developerworks/webservices/library/specification/ws-sdo/

9.3 J2EE Connector Architecture

The J2EE Connector Architecture defines a set of standards about how a resource adapter provides connectivity to an EIS and how the system contracts with an application server. The set of contracts between the resource adapter and the application server provides services to the resource adapter and maintains a pluggable mechanism when running in the application server.

There are currently two versions of the J2EE Connector Architecture specification. The J2EE Connector Architecture V1.0 specification only includes outbound communication and does not include life cycle management and work management. The J2EE Connector Architecture V1.5 specification includes inbound communication and the following system contracts:

- Transaction management

  This contract involves the coordination of the transaction manager of the application servers and the EIS transaction resource managers to provide transactional functionality through the resource adapter.

- Connection management

  This contract involves connection pooling and the framework for creating and managing physical connections to the EIS.

- Life cycle management

  This contract involves the startup and shutdown functions of the resource adapter, and provides timely and consistent contracts for these procedures.

- Work management

  This contract allows the resource adapter to schedule and submit work tasks with the application server. These work tasks can facilitate processes that involve monitoring tasks.

- Security

  This contract provides the resource adapter with features to enable security using application server mechanisms.

Components that want to access information from the EIS can do so in two ways. They can use the Common Client Interface (CCI) to make requests on the EIS,

Archived

which is referred to as an *outbound communication*. The component can also allow the EIS to make inbound calls that send information from the EIS to the component.

The CCI provides an API that the resource adapter can implement and extend, which allows clients to make remote function calls on the EIS. A client component obtains a connection to the EIS using a connection factory.

The client uses an interaction to execute functions on the EIS. The interaction modes indicate how the request is handled by the resource adapter and is specified on the InteractionSpec by stating one of the following interaction modes:

- SYNC_SEND

  The execution only sends an input to the EIS via the resource adapter. No output is expected.

- SYNC_RECEIVE

  The execution receives a synchronous output record from the EIS.

- SYNC_SEND_RECEIVE

  The execution sends an input to the EIS, and a synchronous output record is expected.

The EIS can also make calls into the application component using Enterprise JavaBeans (EJB) that are deployed on the application server. Inbound communication is available only when the adapter is running in a J2EE application server.

9.4 Web services

Web services is a recent re-invention of concepts that have been around for some time. It introduces many new advantages and capabilities. In a sense, none of the function that Web services provides is new; CORBA has provided much of this function for many years.

Web services, however, builds upon existing open Web technologies, such as XML, Uniform Resource Locator (URL), and Hypertext Transfer Protocol (HTTP). Web services is defined in several different standards, such as SOAP (an XML-based messaging protocol) and Web Services Description Language (WSDL), which build upon general Web and other Web services standards. These standards are defined by the World Wide Web Consortium (W3C), the Organization for the Advancement of Structured Information Standards (OASIS), and Web Services Interoperability Organization (WS-I).

Archived

Basic Web services support provides three simple usage models:

- One-way usage scenario

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

- Synchronous request/response usage scenario

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

- Basic callback usage scenario

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

Other Web service standards are built upon these basic standards and invocation models to provide higher level functions and qualities of service. Examples of these standards are WS-Transaction, WS-Security, and WS-ResourceFramework.

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

The interactions work as follows:

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

> **Note:** Although this model is regularly discussed, the service registry is often removed from the cycle in real implementations in the interests of simplicity and lack of trust of the services in the service registry. This has the drawback that if the service provider is relocated, the service consumer needs to be changed to refer to the new location of the service provider.

Archived

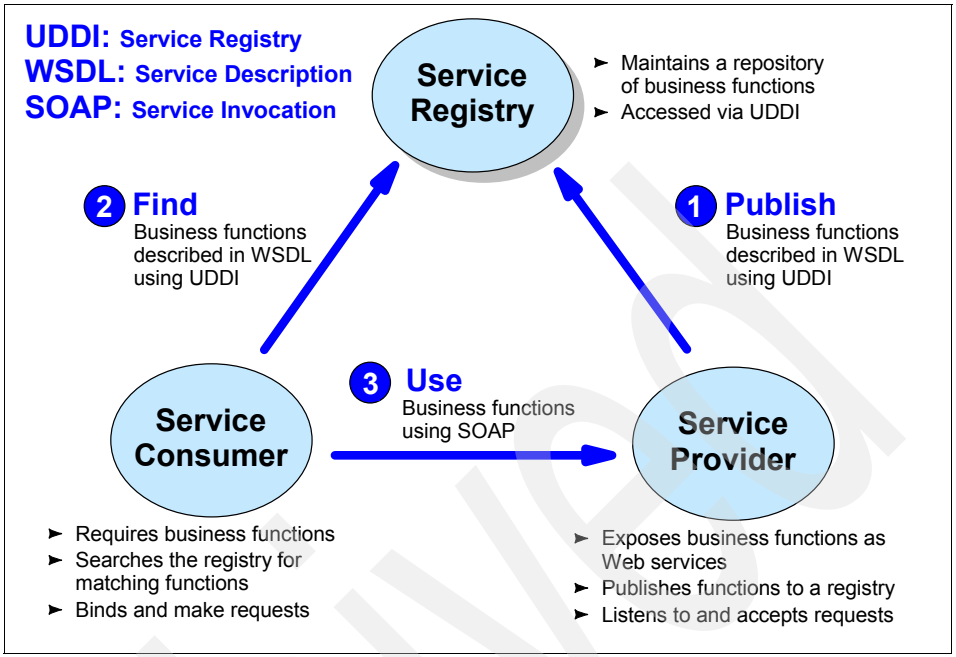


Figure 9-1 Basic Web service interaction model

9.4.1 XML Schema Definition

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

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

XSD has several advantages over earlier XML schema languages, such as document type definition (DTD) or Simple Object XML. For example, XSD is more direct. In contrast to the earlier languages, it is written in XML, which means that it does not require intermediary processing by a parser. Other

benefits include self-documentation, automatic schema creation, and the ability to be queried through XML Transformations (XSLT). Despite the advantages of XSD, it has some detractors who claim, for example, that the language is unnecessarily complex.

9.4.2 Web Services Description Language

WSDL is an XML-based interface definition language that separates function from implementation and enables design by contract as recommended by SOA. WSDL descriptions contain:

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

It is common for these aspects to be defined in three separate WSDL files, each importing the others.

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

The WSDL specification is extensible and provides additional aspects of service interactions to be specified, such as security and transactionality.

9.4.3 SOAP

SOAP is an XML-based format for constructing messages in a transport-independent way and a standard on how the message should be handled. SOAP messages consist of an envelope that contains a header and a body. It also defines a mechanism for indicating and communicating problems that occurred while processing the message, which are known as *SOAP faults*.

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

Archived

The body section contains the content of the SOAP message. When used by Web services, the SOAP body contains XML-formatted data. This data is specified in the WSDL that describes the Web service.

When talking about SOAP, it is common to talk about SOAP in combination with the transport protocol that is used to communicate the SOAP message. For example, SOAP that is transported using HTTP is referred to as *SOAP over HTTP*.

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

The SOAP version 1.1 and 1.2 specifications are available from the W3C at:

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

9.4.4 Universal Description, Discovery, IntegrationI

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

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

- *The role of private UDDI nodes in Web services, Part 1: Six species of UDDI*

  http://www.ibm.com/developerworks/webservices/library/ws-rpu1.html

- *The role of private UDDI nodes, Part 2: Private nodes and operator nodes*

  http://www.ibm.com/developerworks/webservices/library/ws-rpu2.html

9.4.5 Web Services Business Process Execution Language

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

9.4.6 WS-Security

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

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

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

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

Archived

For further information, see:

- *Security in a Web Services World: A Proposed Architecture and Roadmap*

  http://www-128.ibm.com/developerworks/library/specification/ws-secmap/

- *Web Services Security: Moving up the stack*

  http://www.ibm.com/developerworks/webservices/library/ws-secroad/

9.5 J2EE

Java technology (currently in version 2) is an object-oriented programming language and a platform originally developed by Sun™ Microsystems. The Java platform consists of the Java API and the Java Virtual Machine (JVM), an interpreter between the programming language and the underlying software and hardware architectures. The Java API is a large collection of ready-made software components to ease the development and deployment of applets and applications, including robust, secure, and interoperable enterprise applications.

J2EE is the enterprise version of Java that simplifies the construction and deployment of multitier enterprise applications. It bases these applications on standardized modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming.

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

You can track the JCP on the Web at:

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

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

9.5.1 Java Message Service

JMS is an API that adds a provider framework that enables the development of portable, message-based applications for the Java platform. It defines a common set of messaging concepts and programming strategies that are supported by all JMS technology-compliant messaging systems.

9.5.2 Web services for J2EE

Web services for J2EE leverages J2EE technologies by defining the needed mechanism to standardize a deployment model for Web services. This standardization wants to achieve interoperability across different, compliant J2EE platforms, transforming the migration among them into a routine process ensuring that vendors interoperate.

Web services for J2EE defines the concepts, interfaces, file formats, and responsibilities to support the development and runtime models for Web services. Web services for J2EE-compliant Web service providers certify that their services can be redeployed in other compliant servers. Web services for J2EE enables developers, assemblers, and deployers to configure Web services through XML-based deployment descriptors.

9.5.3 Java API for XML-based RPC

Java API for XML-based RPC (JAX-RPC) facilitates distributed computing in a Web services environment. JAX-RPC-based Java applications can easily communicate with non-Java-based technologies in the RPC style fashion.

A JAX-RPC server application's entry point is also known as an *endpoint*. A Web service endpoint is described using a WSDL document. JAX-RPC is about Web services interoperability across heterogeneous platforms and languages. This makes JAX-RPC a key technology for Web services-based integration.

Archived

9.6 Transport protocols

The fundamental transport mechanisms between Web components are HTTP and HTTPS.

9.6.1 HTTP

HTTP is a request/response protocol between clients and servers. It is also the defacto communication protocol of the World Wide Web.

An HTTP server listening on that port waits for an HTTP client to send a request string, such as “GET / HTTP/1.1” followed by an optional body of arbitrary data. Upon receiving the request string (and message, if any), the server sends back a response string, such as “200 OK”, and a message of its own, the body of which is perhaps the requested file, an error message, or some other requested information.

An HTTP client, such as a Web browser, typically initiates a request by establishing a TCP/IP connection to a particular port on a remote host (typically port 80).

Both requests and responses have headers, which contain useful information. Some headers are optional, while others (such as Host) are required by the HTTP/1.1 protocol.

9.6.2 HTTPS

HTTPS is the secure version of HTTP. Instead of using plain text socket communication, HTTPS encrypts the session data using either a version of the Secure Socket Layer (SSL) protocol or the Transport Layer Security (TLS) protocol. Using these protocols ensures reasonable protection from eavesdroppers and man-in-the-middle attacks. The default TCP/IP port of HTTPS is 443.

The level of protection depends on the correctness of the implementation by the Web browser and the server software, as well as the actual cryptographic algorithms that are supported.

Archived

Part 3

Scenario implementation

Archived

© Copyright IBM Corp. 2006. All rights reserved.

Archived

10

Serial Process scenario

This chapter details the construction of a process using the Serial Process pattern. It describes the following items:

- Business scenario: For the ITSO Electronics business problem
- Design guidelines: For building a process conforming to the Serial Process pattern
- WebSphere Process Server guidelines: For implementing the process using WebSphere Process Server

This chapter illustrates the use of the following WebSphere Process Server features:

- Web Services Business Process Execution Language (WS-BPEL) business processes
- Business objects and interfaces
- Business rules
- Interface maps
- Data maps
- Visual snippets
- Java snippets

© Copyright IBM Corp. 2006. All rights reserved.

10.1 Business scenario

ITSO Electronics wants to integrate its retail and wholesale departments in order to implement an on demand retail operation where all requests from the retail ordering system are fulfilled directly from wholesale inventory. The retail department can order from either of two wholesale departments, Wholesaler A and Wholesaler B. Figure 10-1 shows how these components are integrated.

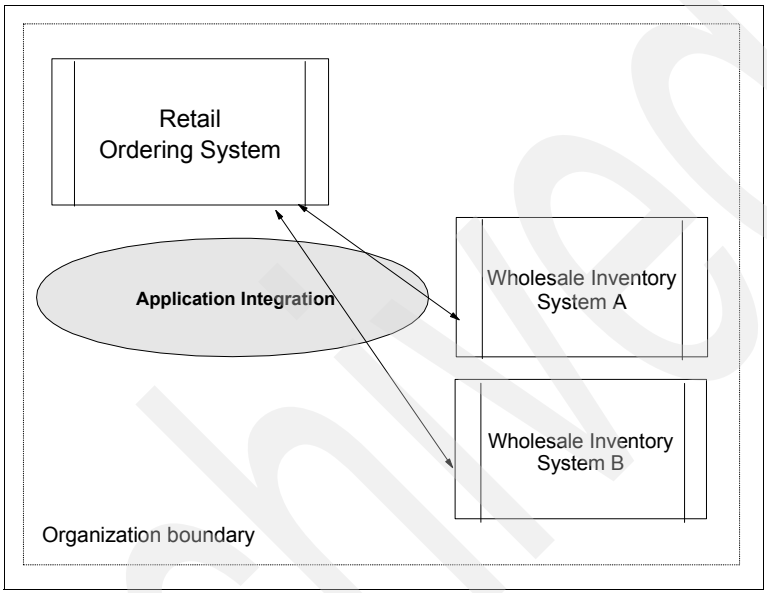


Figure 10-1 ITSO Electronics integration overview

Once the integration is accomplished, an order clerk at ITSO Electronics uses a browser-based application to enter the part number and the quantity for an order. A response, including the wholesaler that was used, the confirmation number, and the expected delivery date, should be available within a few seconds.

Since Wholesaler A is the preferred supplier for ITSO Electronics, the order is placed with Wholesaler A if it can fulfill the order within a period of seven days. However, if Wholesaler A's expected delivery date is greater than seven days, then Wholesaler B is contacted for its delivery date. Ultimately, the order is placed with the wholesaler that has the shortest delivery date.

Archived

Here are some additional scenario elements:

- The wholesalers offer identical prices. The cost of the part does not factor into ITSO Electronics's decision as to which supplier it chooses.
- Competitive pressures are increasing, and management is expecting the seven-day order fulfillment window to shrink, especially during high volume shopping periods. Management wants the ability to change the number of days without the intervention of the IT department.
- ITSO Electronics needs a record of the orders that are placed. This record must include who placed the order, the part number and quantity being ordered, and the supplier with whom the order was placed.

Figure 10-2 illustrates how this scenario is implemented at ITSO Electronics.

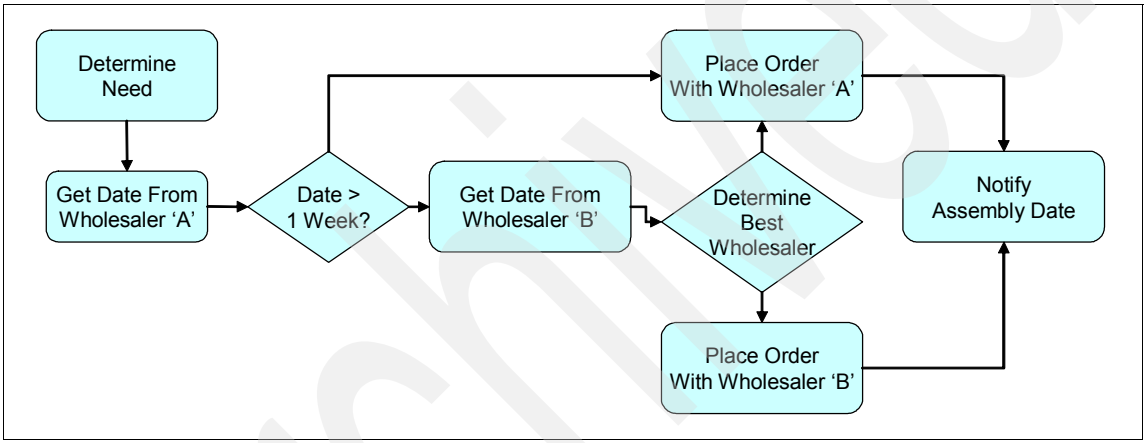


Figure 10-2 The ITSO Electronics Serial Process

10.2 Design guidelines

This section begins with a brief discussion of the terms that we use to describe the different kinds of business processes. Then it outlines the following steps to move from the scenario as described in the previous section to a concrete design with a specific set of products.

1. Analyze the business requirements. We decompose the scenario into a set of business requirements.
2. Select a pattern. We determine which Runtime pattern from the Patterns for e-business will solve this business problem.
3. Analyze the design options. We describe the design considerations in building the solution.

Archived

10.2.1 Vocabulary

The vocabulary used to describe the business processes has not been standardized. You will find the same words are given different, specific meanings by different authors. We define our terms here.

Patterns for e-business uses the term *process* to generically describe a business process. A process may or may not contain human interaction, but generally you can presume that if it includes human interaction, it is designated by the more specific term *workflow*.

The term *workflow* is used to indicate that human interaction is required to complete some of the activities described in the business process. Since a workflow is a specific type of business process, the term *business process* is often reserved for those processes without any human interaction.

A *long-running business process* is interruptible, and each step of the process runs in its own physical transaction. *Interruptible* means that it can be affected by external events, which may include a message from another business process, a response to an asynchronous invocation, or the completion of a human task. Long-running business processes persist process instance data within the process manager's database while the process is being executed.

A *short-running business process* contains all automated steps and completes in a short period of time. A short-running process runs in one transaction and does not persist any process instance data.

Since a short-running process consists of only synchronous services and short-running subprocesses, it cannot contain:

- Human tasks
- Wait activities
- Multiple receive activities
- Asynchronous invoke activities

10.2.2 Analysis of the business requirements

The scenario maps directly to a set of business requirements. The business requirements are:

- Integrate the retail system with both wholesale systems, so that retail orders flow electronically to each wholesale system.
- Establish a process that sends an order to Wholesaler A and determines the expected delivery date. The process uses a rule to decide whether that date is acceptable. If it is acceptable, the process immediately sends the order to Wholesaler A. If it is not, it asks Wholesaler B for its delivery date. In that

case, once both dates are received, award the order to the wholesaler who can deliver first.

- Provide a way for business people to change the criteria for deciding whether to get the additional date from Wholesaler B.
- Provide all relevant information (the wholesaler chosen and delivery date) to the retail system.
- Complete the process within a few seconds.

10.2.3 Pattern selection

This section describes the process that we used to navigate the Patterns for e-business to determine the patterns and product mapping that apply to this scenario. Figure 10-3 illustrates the step-by-step approach that we used to navigate the Patterns for e-business asset catalog.

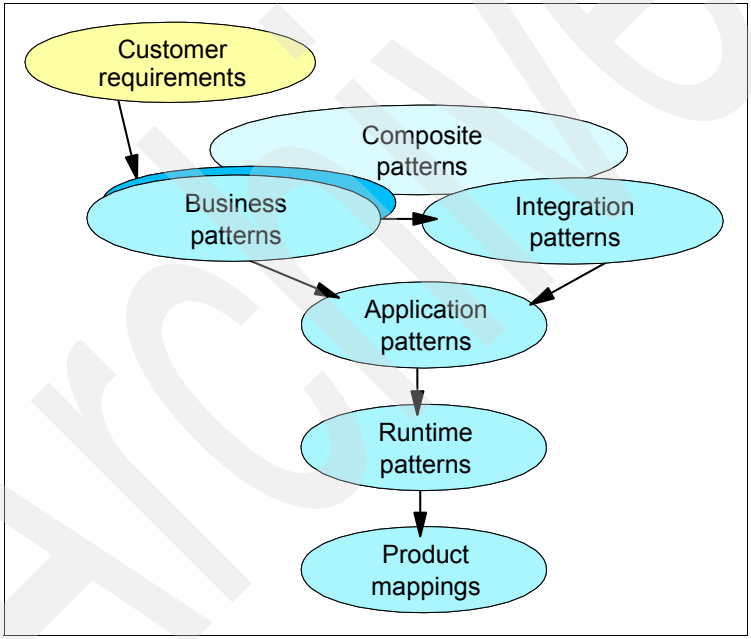


Figure 10-3 Patterns for e-business layered asset model

Business pattern

We are looking to integrate two applications within an enterprise, which is addressed by the Application Integration business pattern. This pattern is described as bringing together multiple applications and information sources. We recommend using this pattern when multiple applications must be combined to

Archived

automate a new business function. This describes our scenario and requirements.

Application pattern

Now that we have chosen a Business pattern, the next step is to choose from among the Application patterns associated with it. These patterns are broadly categorized as either *Process Integration* or *Data Integration*. Since we are implementing a new process (rather than integrating only data), we focus on the Process Integration patterns.

The Process Integration patterns are shown in Figure 10-4 and are categorized along two axes. The vertical axis shows whether there are multiple distinct interactions with one or more targets, while the horizontal axis describes whether those interactions happen sequentially or in parallel.

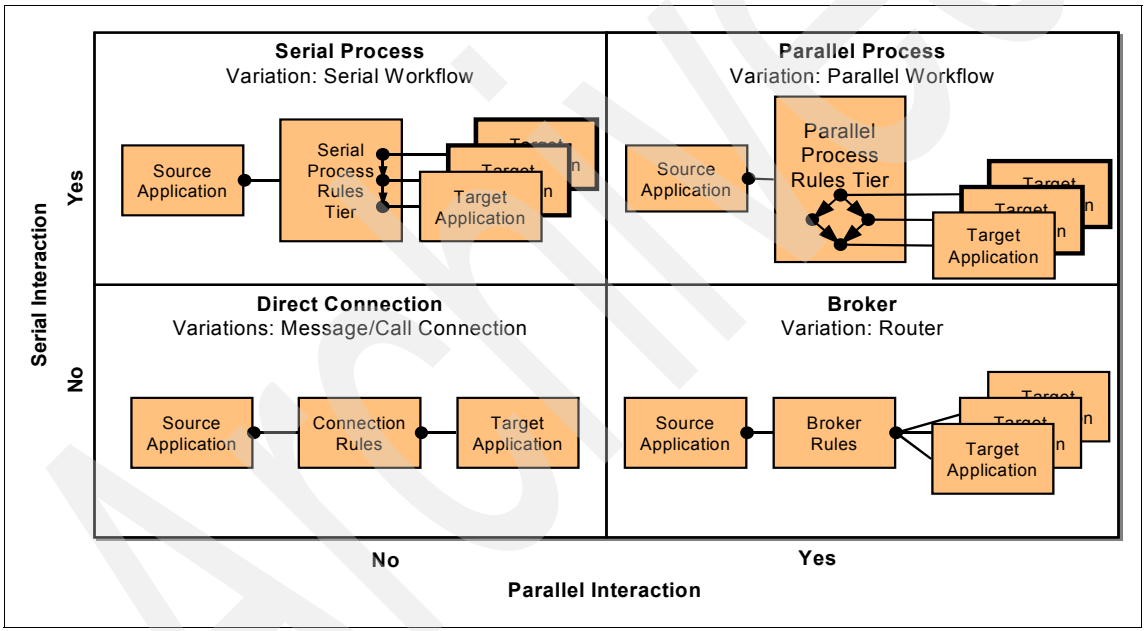


Figure 10-4 Process Integration patterns

This scenario includes four distinct interactions with two targets. Let's look in the top row. We wait for Wholesaler A's answer before we talk to Wholesaler B, and we wait for that answer before we decide to whom we will send the order. Therefore, this is not a case of parallel interaction, and we choose the Serial Process from the top left quadrant of the table. The pattern has a variation if human intervention is required. In this case, it is not, so the final result is the Serial Process pattern.

Runtime pattern

Figure 10-5 illustrates the Runtime pattern for the Serial Process application pattern. This Runtime pattern has two variants: the generic profile and the service-oriented archive (SOA) profile. Although our architecture is service-oriented, the SOA profile is intended for use with an Enterprise Service Bus. We choose the generic profile for this scenario because we do not have an enterprise service bus (ESB).

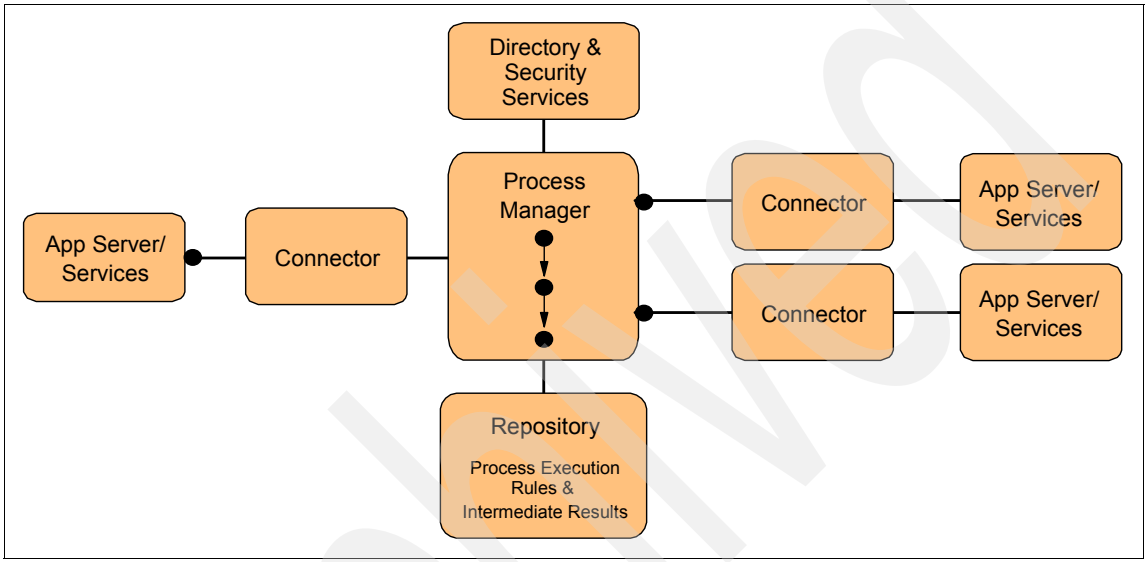


Figure 10-5 Serial Process runtime pattern: Generic profile

Archived

Product mappings

Having selected the generic profile of the Serial Process runtime pattern, we can apply product mappings to each of the nodes, as shown in Figure 10-6.

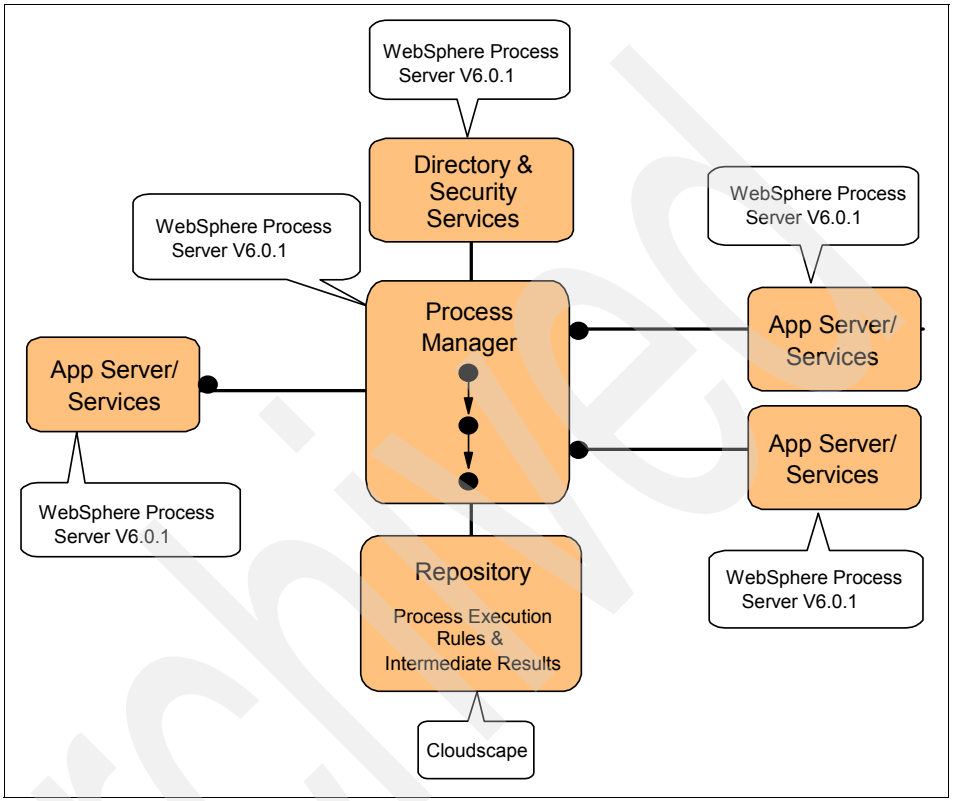


Figure 10-6 Serial Process: WebSphere Process Server product mapping

The main product of choice for implementing this pattern is WebSphere Process Server V6.0.1. It is used both as the Process Manager and to host the source and target applications in the App Server/Services nodes. The App Server/Services are Web services that are accessed by using SOAP over HTTP (SOAP/HTTP). Because WebSphere Process Server natively supports communication with Web services, we do not need Connector nodes between the App Server/Services and Process Manager nodes.

Although not shown in the product mapping, the development tool for use with WebSphere Process Server V6.0.1 is WebSphere Integration Developer V6.0.1.

Archived

The Repository used in this product mappings is IBM Cloudscape, which is provided with WebSphere Process Server. In production, the Repository is more likely to run on a production-ready database such as DB2 Universal Database.

The Process Manager uses the Service Component Architecture (SCA) to invoke Web services provided by the target applications running on WebSphere Process Server. We implemented the target applications as processes and used the Service Component Architecture to expose them as Web services. We could have chosen WebSphere Application Server to implement the target application services, or for that matter any application server that provides Web services, such as Microsoft .NET.

10.2.4 Analysis of the design options

The array of design considerations when planning any type of process implementation is large and daunting. This section describes how we applied our favorite considerations to the requirements listed in 10.2.2, “Analysis of the business requirements” on page 148. Unlike the previous sections, this one is not product agnostic. In addition to describing design choices and trade-offs, we discuss how particular design elements are handled by WebSphere Process Server. Here’s our top 10 list of design options to consider:

1. Short-running or long-running process
2. Synchronous versus asynchronous process invocation
3. Business rules
4. Interface and business object mapping
5. Security
6. Human interaction
7. Transactions
8. Events and state
9. Audit trail
10. Compensation

Each of these design options is discussed in the chapter that describes the first scenario in which it is required. This section discusses the first four items on the list.

Archived

Short-running or long-running process

The characteristics of short and long-running processes are explained in 10.2.1, “Vocabulary” on page 148. Thus, the choice of one or the other is generally dictated by other characteristics of the process:

- Will the process listen for external events? Do any of its steps require human interaction? Then it is by necessity a long-running process.
- Are all of its steps automated? Is it expected to run quickly and return an answer? In that case, a short-running process may be considered.

One question that is frequently asked is: If you have a process with no external events and no human interaction, why use a short-running process rather than simply coding in a programming language? With WebSphere Process Server, the answer involves a combination of flexibility, transparency, and features. Processes are easier than code for a less technical person to read and modify. Also, using a process makes it easier to take advantage of other WebSphere Process Server features such as business rules or data mapping.

Applying the guidelines to ITSO Electronics

We chose to implement this scenario as a short-running process. This decision was based on the following factors:

- The process should complete in a short period of time.
- The process contains no activities requiring human interaction.
- The user invoking this process requires an immediate response as to which wholesaler will fulfill the order.

Synchronous versus asynchronous process invocation

There are two basic process invocation methods:

- Synchronous invocation

  A synchronous invocation of a process means that a response is not returned from the process manager until the process completes. When the response is received, the process is considered complete.

- Asynchronous invocation

  An asynchronous invocation of a process responds immediately, but this response only indicates whether the process instance was started successfully.

Archived

Choosing an invocation method

A synchronous invocation should typically be chosen when all of the following circumstances are true:

- The process is short running.
- The calling application receives a response in a reasonable amount of time.
- The user who has invoked the process expects to wait for a response.

An asynchronous invocation should typically be chosen when either of the following circumstances are true:

- The process is long running.
- Human interaction is required within the process.

In general, if a process is likely to take more than a few seconds to complete, it should be called asynchronously. In fact, it is almost always a logical error to synchronously invoke a long-running process. The invoking entity is, by definition, in memory and disappears at the end of a session, while the long-running process can span many sessions before it completes and returns. The result is that the invoker is no longer available to return to.

Applying the guidelines to ITSO Electronics

We decided to invoke this short-running process using a synchronous invocation. This decision was based on the following factors:

- The process is short running.
- The user invoking this process requires an immediate response as to which wholesaler will fulfill the order and plans to wait for it.

Business rules

A *business rule* is anything that captures and implements business policies and practices. A rule can enforce business policy, make a decision, or infer new data from existing data.

In a business process, there are typically points where the flow of the process can traverse one or more control paths depending on some sort of decision. The decision to be made is most likely a business decision; in this scenario, the first decision to be made is whether Wholesaler A's delivery date is acceptable. Each of these decision points can be thought of as an instance of a business rule.

Sometimes these rules are quite static, as in the level of authority needed to override a credit decision. Other times, business conditions might dictate that the rule be fairly flexible. Ideally, the business wants the ability to change a rule without programming, so that they do not need to involve a programmer in business decisions.

Archived

Traditionally, complex business rules have been implemented in a *rules engine*, which separates the routing rules from the application code and keeps the routing rules external to the process. Rules engines provide a user interface to allow qualified users to make changes without programming.

WebSphere Process Server includes a business rules manager that allows templates for if-then rules and decision tables to be defined. A template defines the parts of a deployed business rule that can be modified by an authorized Web user and by exactly how much.

By using business rules, business users with the appropriate access rights can change the content of these business rules without recompiling the business process.

When to use a business rule

Every decision point is a candidate for a business rule, but that does not mean every decision point should be implemented as a business rule. Sometimes it is best to leave the decision logic in the body of the process. If the decision is simple, if it is not likely to change, or if the decision is simply not based on business considerations, a business rule is not necessarily appropriate. Choose a business rule in the following situations:

- There is a requirement for the business rule to change at run time.
- The rule is complex enough that abstracting it into its own component is a good design decision.
- It is expected that additional rules from the same template could be added. For example, if a rule is based on dividing customers into Gold, Silver, and Bronze categories, using a business rule template means that a Platinum category can be added in the future without any programming.

Applying the guidelines to ITSO Electronics

We use one business rule to determine whether to query Wholesaler B for a shipping date. We use another rule to compare Wholesaler A's date to Wholesaler B's date.

Interface and business object mapping

Remember that we are using an Application Integration pattern. One of the fundamental truths of any integration is that the things being integrated, even if they serve similar functions, have different interfaces and operate on different data types. One of the things you always need to consider is how to map the interfaces and data between the different components comprising your solution.

You need to consider which pattern to use for integration. Since the integration mechanism is consolidated into a single central process, it makes sense to follow a similar process for data. WebSphere Process Server supports a pattern in which the process defines a set of generic business objects that represent the data essential to executing the process. Each external application has its own data representation, and translation between the two happens in a separate mapping component.

The next decision is how to implement the mapping. Simple mapping from the internal business object to the application programming interface (API) needed to invoke another application can use the WS-BPEL assign operator to copy data from fields in the one object to corresponding fields in the other. Anything more complex, such as splitting an address into street name and street number, either uses programming or a mapping tool. Since WebSphere Process Server includes a graphical mapping tool, this is an easy decision.

Even with the generic business object pattern and a mapping tool, map design is difficult. Without help, you may not understand the applications you are mapping well enough to design a comprehensive generic object and map it to all of the application-specific objects.

We recommend that you gather the experts on each application together in a room and have them ensure that all necessary data is there and that the correlation between applications is understood and documented. You need to identify the source and target business objects and the transformations that are required to get the correct data into the target business objects. You must identify the data required by the target system, the location of that data in the source system, and the transformations themselves. Common transformations are provided by the mapping tool; you may need to code more unusual ones.

When to use a map

You use maps whenever the data types of the integration source differ from those of the target. Although you almost always want to use a map to move data from a source object to a target object, there are a few instances where it makes more sense to use the WS-BPEL assign operation instead, such as:

- When you only need to move a few fields, and those fields are directly assignable from source to target
- When you need to conform strictly to the WS-BPEL standard

Applying the guidelines to ITSO Electronics

We use maps to transform the retail system data into our process's generic business object and when we make calls from the process to the wholesalers' systems.

10.3 Development options

In this section, we discuss some of the implementation decisions that need to be made when using WebSphere Process Server and WebSphere Integration Developer. Not all of the scenarios that we implement in this redbook are implemented in the same way. This section gives you some idea of the different approaches that you can use.

Business objects versus business graphs

Depending on the type of application you are developing and the systems that are being integrated, you either always or never use business graphs.

Business objects represent business data, whether is it coming from an adapter or created from simple input from the user. There are no additional services provided by the business object. They are the lightest form of transmitting complex data in a module, and all invocations in SCA require their use.

Business graphs are a special type of business object. They provide services on top of a business object. These services include a change summary history, event summary, and the ability to specify verbs. Business graphs should not contain any business logic or data. They are strictly wrappers used to specify information needed at run time. If business graphs are chosen, the recommended pattern is for a business graph to contain only business objects as children. Business graphs should avoid including other business graphs, but this is not an enforced restriction.

The WebSphere Adapter framework uses business graphs because it can use the change summary logging to calculate the deltas to apply to a backend system. If your solution does not require these services, then it is better to use business objects directly. You should have a strong use case to populate your interfaces with business graphs.

Top-down versus bottom-up design

In the *top-down design*, we define the interfaces, references, component types, and wires in the Assembly Editor before we create any of the actual implementations. This allows us to visualize the entire solution at once to ensure that the data flows as expected. The added benefit in WebSphere Integration Developer is that the Integration Test Client can be used to test the solution before all the implementations are complete. The best time to use this approach is when creating a completely new solution from scratch because the designer can create interfaces and business objects as required.

The opposite pattern of top-down is bottom-up. In a *bottom-up design*, we define the implementations of our components first. Once they are complete, then we

drag them to the Assembly Editor. This pattern is more rigid than the top-down pattern due to the fact that the implementations have already been created. If a design problem is found during integration, it is more difficult to fix. This pattern is primarily used when artifacts have already been created from existing systems, when applications must be re-used, or both.

Application Specific Business Object and Generic Business Object versus direct Web Services Description Language invocation

In the scenario implementations used in this redbook, two styles of Web Services Description Language (WSDL) invocation are used. In this chapter, maps are used to convert data from an application specific business object (ASBO) into a generic business object (GBO) via interface and business object maps. This is done as an abstraction from the application specific business object, which could change as new wholesalers are added, and a generic business object, which remains constant. Using this pattern allows us to isolate the parts of our module that require changing when these points of our process change. We are able to keep the core business logic the same as new wholesalers enter and exit.

In other scenarios is this redbook, such as the Parallel Process scenario, we simply invoke the wholesalers directory in the business process. We use WS-BPEL assign nodes to convert data into the proper request/response variables. This is a less elegant solution but one that allows for a quicker implementation. We have traded the ability to have quick changes for quick implementation.

In the enterprise level solution, we recommend that you follow the ASBO/GBO pattern. This helps to isolate change as the technical details are modified while allowing business logic to remain untouched. In our example, the Parallel Process requires significant modification should a wholesaler change. If the ASBO/GBO option is selected, only a few map modifications are required.

The number of modules

A single module should be created per business service to be created. A module should contain a logical grouping of business functionality. Modules support multiple imports and exports, but calls to exports can be expensive and should be minimized.

While it is possible to model your complete solution using multiple modules linked together via exports, it may result in a slower performing system. Each call to an export results in a message being placed on a Java Message Service (JMS) queue and then future consumption by the service. If an export is not used, the data is simply passed by reference.

When to use modules and when to use libraries

A *module* is the basic unit of deployment to the runtime environment. Modules are projects that contain all the artifacts (either directly or from a referenced library project) necessary to create a business service. All resources contained in a module are private and cannot be shared with other modules or library projects. The module project always contains all the artifacts related to the Assembly Editor.

Library projects should be used when interfaces, business objects, business object maps, roles, relationships or Web service ports need to be shared so that several modules can use them. In our runtime design, we used a library project to hold all the common data associated with the Wholesalers since they provide a common re-usable interface. Library projects cannot be deployed to WebSphere Process Server on their own. They must be used by a module.

When developing a solution, some thought needs to go in to deciding which parts of the solution will potentially be common to a broad range of services and which parts are internal implementation details and should be kept hidden.

WebSphere Adapters versus WebSphere Business Integration Adapters

WebSphere Adapters are J2EE Connector Architecture V1.5 compliant. As such, they fully support the Enterprise Service Discovery wizard which can automatically create interfaces and business objects to access the adapter via a simple SCA Import or Export. This results in a system that is quick to author because most of the heavy lifting of creating business objects and interfaces to represent your adapter data is already done for you.

WebSphere Business Integration Adapters provide the same type of services as WebSphere Adapters, but they execute in a separate operating system process outside the WebSphere Process Server run time. WebSphere Adapters execute within the context of the WebSphere run time. Therefore WebSphere Adapters have the potential for better performance with WebSphere Process Server than WebSphere Business Integration Adapters.

Inline human tasks versus SCA human tasks

An *inline human task* is a human task contained inside a business process. An *SCA human task* is a human task specified as a component in the Assembly Editor. Inline human tasks and SCA human tasks behave the same at the fundamental level. They both specify the execution of a human task. The subtle difference between the two is that an inline task can access the business process's context to dynamically set information. For example the variable *%wf.process.starter%* can be used in an inline human task to assign tasks to the

instantiator of the process. If the SCA human task is used, there is no business context and these dynamic assigns cannot be used.

We recommend that, unless you require dynamic assigning, use SCA human tasks. This allows easier potential future replacement of the human task with an automated task.

Autobuild and autodeploy: Off or on

Autobuild should always remain on when developing in WebSphere Integration Developer. There are generated artifacts that only get created when certain files are changed. If autobuild is turned off, it is possible that this generation will not occur, leaving your workspace in an undeployable state.

Should this situation occur, you have two courses of action to choose from:

- Attempt a clean build by clicking Project → Clean.
- Export your source modules and libraries into a project interchange file and re-import them into a new workspace.

Enabling or disabling autopublish is a personal preference. The option can be found in the server configuration by double clicking the entry in the Servers view. If autopublish is turned on while there is considerable authoring work left to do, it is probably better to turn it off or shut down the test environment.

Visual snippets versus Java snippets

If a developer is familiar with the Java language and syntax, then we recommend Java snippets as the fastest method of development. If the developer is only familiar with general programming ideas, then visual snippets help to hide the implementation details of Java.

Archived

10.4 Development guidelines

> **Note:** In this section, we use *WebSphere Integration Developer V6.0.1*. If you are inexperienced with WebSphere Integration Developer, we recommend that you first complete Appendix A, “Building Wholesaler A and B service implementations” on page 501, before commencing these development guidelines. This appendix provides a good introduction to WebSphere Integration Developer.

This section explains how to build the Serial Process scenario using WebSphere Integration Developer. Because we use a large number of components, throughout the chapter, we guide you through the development and highlight where individual development units fit.

The following five projects are implemented as illustrated in Figure 10-7:

- ITSOElectronics

  The ITSOElectronics module is the main module in this scenario. It implements business logic, interfaces, and mediations of a distributor. As a distributor, it consumes the services provided by the Wholesalers A and B. We build the following components in this module:

  - WS-BPEL business process
  - Business rule
  - Interface
  - Interface maps
  - Data object
  - Data map

- ITSOLibrary

  This module contains a common set of components used throughout the scenarios in this redbook:

  - Interface
  - Data object
  - Data map

- OrderAck

  The OrderAck module contains the implementation of ITSO Electronics’s order acknowledgement. It is implemented in a Java snippet. We build the following components in this module:

  - Java snippet
  - Interface
  - Data object

Archived

- WholesalerA

  The WholesalerA module contains the business logic of Wholesaler A. The business logic is implemented as a BPEL process. It contains:

  – WS-BPEL business process
  – Interface
  – Data object

- WholesalerB

  The WholesalerB module contains the business logic of Wholesaler B. The business logic is implemented in a Java snippet. It contains:

  – Java snippet
  – Interface
  – Data object

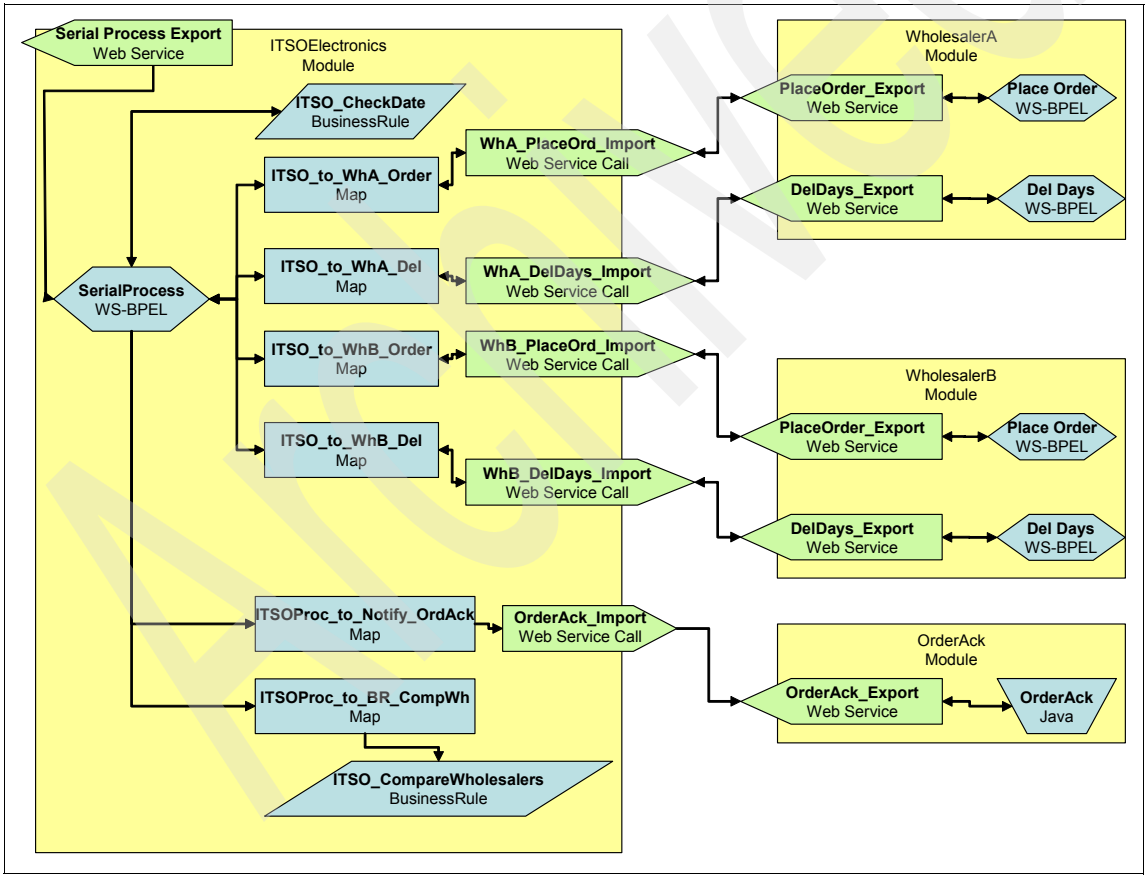


Figure 10-7 Overview of the scenario showing the interlinked components and modules

Archived

10.4.1 Development plan

We build the components in the following order. This is not a hard and fast order of construction, but allows for reuse of the components in other scenarios.

1. Set up the workspace.

   Refer to Appendix A, “Building Wholesaler A and B service implementations” on page 501, to create the WholesalerA and WholesalerB modules.

2. Build the business logic. See 10.4.3, “Creating the business logic for ITSO Electronics” on page 171.

   a. Build the business process using WS-BPEL. See “Creating the WS-BPEL process” on page 182.

   b. Build the business rules called by the WS-BPEL business process. See “Creating the business rules” on page 194.

   c. Use the assembly diagram to link the WS_BPEL business process and the business rules. See “Creating the assembly diagram” on page 213.

3. Create the supporting services for the ITSO Electronics business logic. See 10.4.4, “Creating supporting services for the business process” on page 215.

   a. Create the Order Acknowledgment service. See “Building an order notification service” on page 216.

   b. Update the assembly diagram to integrate the Wholesaler with the business logic and to link the WS-BPEL process to the Order Acknowledgment service. See “Adding the supporting services to the business process” on page 219.

10.4.2 Setting up the workspace

This chapter describes the initial set up of the WebSphere Integration Developer workspace. We create the modules and libraries projects for the ITSOElectronics and the Wholesalers. By the end of this section, we will have a skeleton of projects which we will fill during this chapter.

Archived

First, start the WebSphere Integration Developer and select a directory for the workspace.

1. Start WebSphere Integration Developer. Select a Workspace directory, for example c:\workspaces\SerialProcess (Figure 10-8).

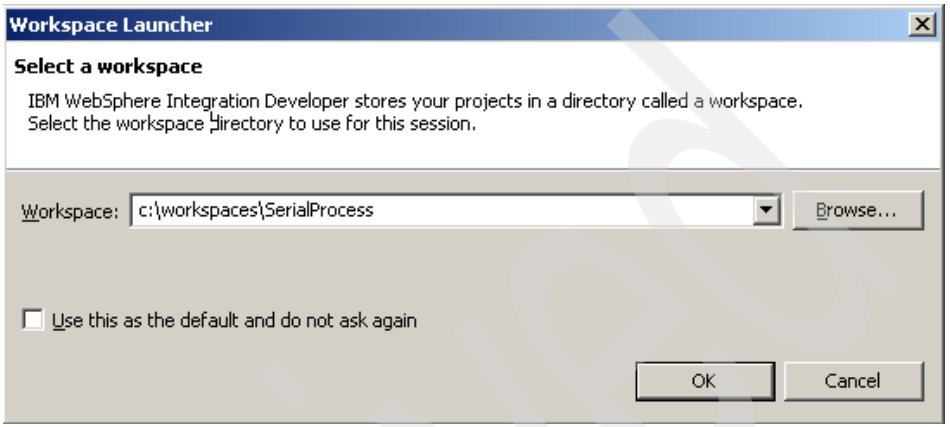


Figure 10-8 Selecting a workspace directory

2. Open the Business Integration perspective if it is not yet open. Select **Window** → **Open Perspective** → **Business Integration**.

WebSphere Integration Developer uses modules and libraries to organize and separate the SCA components. For each logical unit or functionality, one module can be created. In our case, we build separate modules for each Wholesaler, because they represent different parts of a company and run in a real environment on different WebSphere Process Servers. We decide also to create our own module for the Order Notification functionality, because it can be reused from other modules than ITSO Electronics.

> **Note:** Each module project is equivalent to one enterprise archive (EAR) and one Enterprise JavaBean (EJB). Therefore compare the reuse of components against the amount of EJBs that will run on your WebSphere Process Server.

We decide to place common artifacts in a shareable library project. A library can be referenced in modules and its content is accessible to the module. A library is deployed in a module EAR, if the module references it. If several modules reference it, the library is deployed in each module.

Complete the following steps to create the library and module projects that are required for this scenario.

1. Create the ITSOLibrary library project.
   a. Select **File** → **New** → **Project**.
   b. In the New Project panel (Figure 10-9), select **Library** and click **Next**.

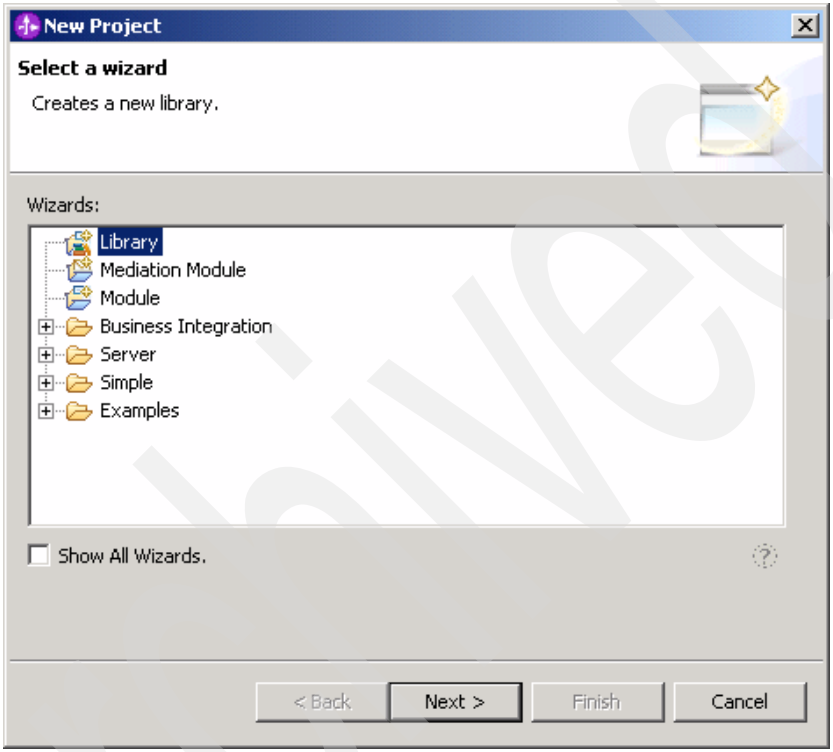


Figure 10-9 New Project panel

Archived

c. In the New Library panel (Figure 10-10), complete these steps:

i. In the Library Name field, type `ITSOLibrary`.
ii. For the Library Location, ensure the **Use default** option is selected.
iii. Click **Finish**.

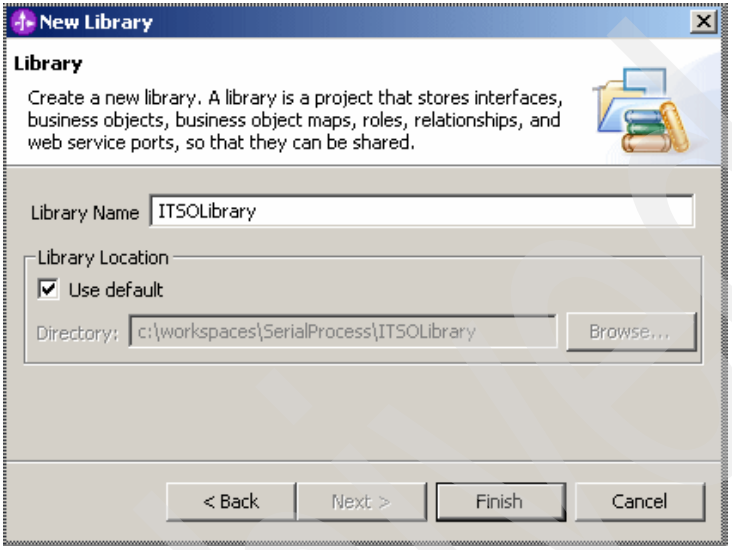


Figure 10-10 New Library panel

2. Create the ITSOElectronics module project.

a. Select **File** → **New** → **Project**.
b. In the New Project panel, select **Module** and click **Next**.
c. In the next panel, complete these steps:

i. In the Module Name field, type `ITSOElectronics`.
ii. For Module Location, ensure that **Use default** is selected.
iii. Click **Finish**.

3. Create another module project named `OrderAck`, by repeating step 2.

4. The creation of the WholesalerA and WholesalerB modules is described in Appendix A, “Building Wholesaler A and B service implementations” on page 501. You can either complete the steps in this appendix to build these wholesalers, or you can import these modules into your workspace. To import these modules, follow these steps:

a. Select **File** → **Import**.

b. In the Import wizard, select **Project Interchange** and click **Next**.

c. Browse to the location of WholesalerA.zip. It is in the Wholesalers\WholesalerA directory of the additional material supplied with

this redbook. For information about how to obtain this additional material, see Appendix E, “Additional material” on page 689.

d. Verify the **WholesalerA** project and click **Finish**. The WholesalerA module imports into your workspace.

e. Repeat this process to import WholesalerB.zip, which is in the Wholesalers\WholesalerA directory of the additional material supplied with this redbook.

The workspace must be displayed like the example in Figure 10-11.

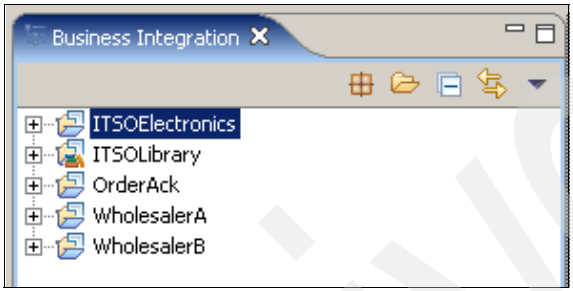


Figure 10-11 Workspace with five projects created

In the following steps, we add references to the modules ITSOElectronics and OrderAck, which point to the library project ITSOLibrary. We do this because this library contains common artifacts required by the two modules.

1. Right-click the **ITSOElectronics** module project and click **Open Dependency Editor**.
2. Click **Add** to add a new library dependency.

Archived

3. In the Library Selection window (Figure 10-12), select the Library project **ITSOLibrary** and click **OK**.

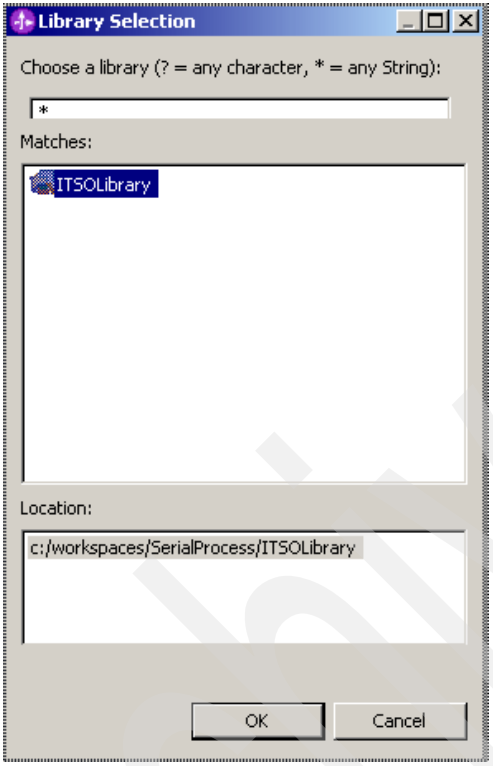


Figure 10-12 Library Selection window

Archived

4. In the Dependency Editor (see Figure 10-13), click **ITSOLibrary** to highlight it, and ensure that the **Deploy with Module** check box is selected.

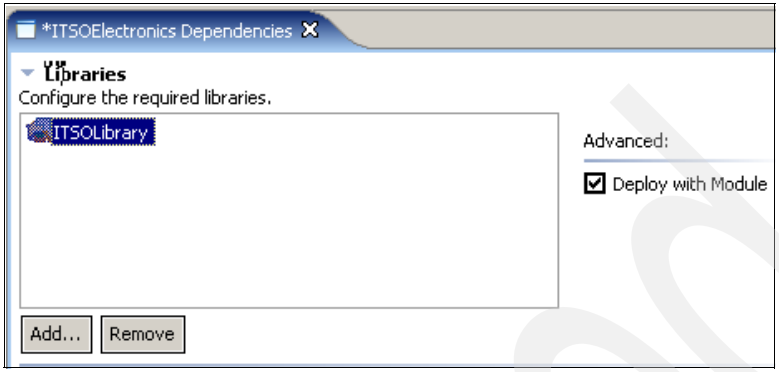


Figure 10-13 Dependency Editor

5. Click **File** → **Save** to save the settings in the Dependency Editor.
6. Open the Dependency Editor for the OrderAck Module project and add the ITSOLibrary in the same way. Again make sure that the **Deploy with Module** check box is selected.

The workspace is set up now and the artifacts can now be created.

Archived

10.4.3 Creating the business logic for ITSO Electronics

In this section, the business logic of ITSO Electronics is implemented. Figure 10-14 highlights the components that are created in this section.

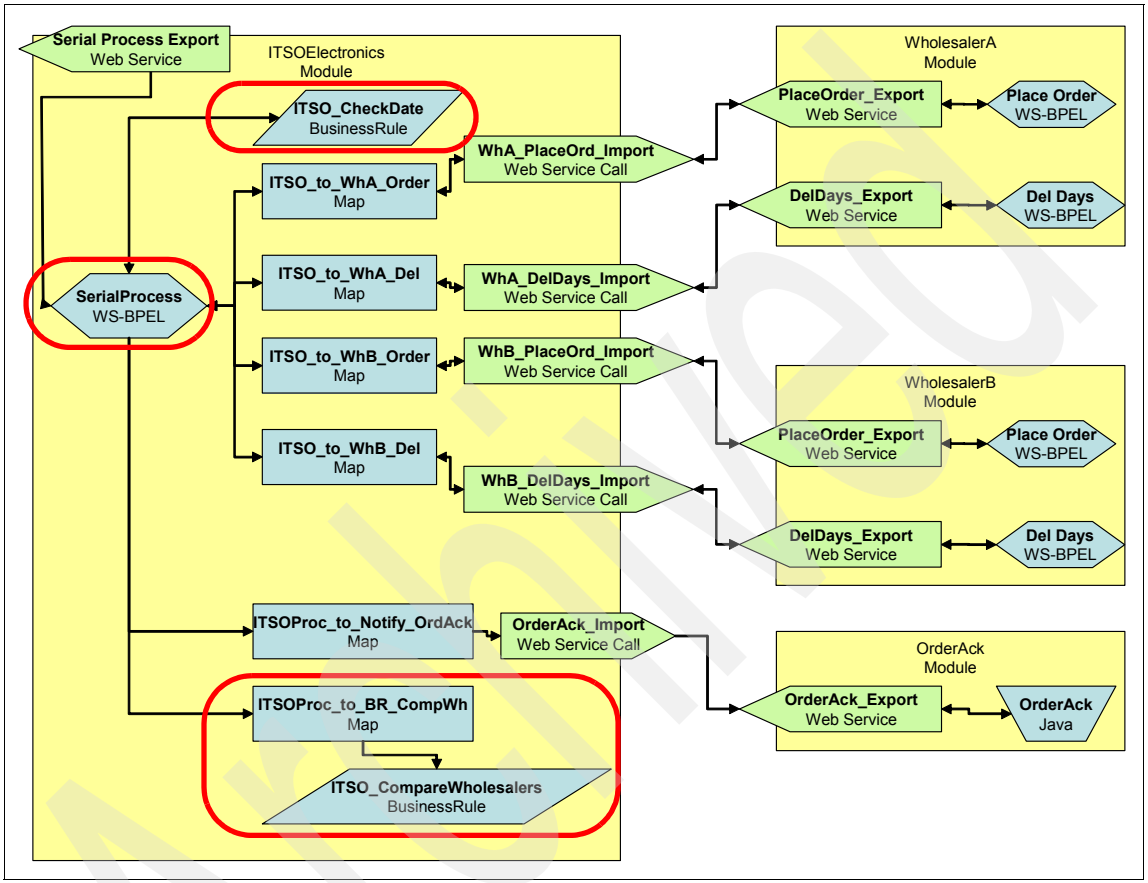


Figure 10-14 Overview of the scenario showing the interlinked components and modules

The business logic is implemented using WS-BPEL and business rules. The flow of the business logic is shown in Figure 10-2 on page 147.

ITSOElectronics consumes external services (get delivery days, place order, cancel order) that are provided by Wholesalers A and B using Web service technology. It also consumes internal SCA services provided by the SCA component OrderAck.

In this section, we describe all of the artifacts needed for this business process and the steps needed to construct them. We are concerned only with building the

process and the interfaces that are used to call the services that the process consumes. We build the process using data structures and interfaces that are independent of the services that we will call. We use the mapping functionality of WebSphere Process Server to map our interfaces to the interfaces of our service providers. This allows us to easily change service providers without changing our business process logic.

In the following sections, we create the interfaces and business objects listed in Table 10-1. Each of the inputs and outputs is represented by a simple data type or a business object type.

Table 10-1 ITSOElectronics interfaces

Description	Name	Operations	Input	Output
Order Request	ITSOOrdReq_I	requestOrder	order Data Type: ITSOProcessOrder	N/A
Order Notification	ITSOOrdNotify_I	notifyDeliveryDate	order Data Type: ITSOProcessOrder	N/A
Delivery Days	ITSODelDays_I	getDeliveryDays	getDeliveryDays WhReq Data Type: ITSOProcessOrder	getDeliveryDays WhRes Data Type: ITSOProcessOrder
Place Order	ITSOPlaceOrd_I	placeOrder	orderReq Data Type: ITSOProcessOrder	orderConf Data Type: ITSOProcessOrder
Date Check	ITSODateCheck_I	checkDeliveryDate	deliveryDays Data Type: ITSOProcessOrder	daysOK Data Type: boolean
Pick Best Wholesaler	ITSOPickBestWh_I	pickBestWh	whResponses Data Type: ITSOMultiWh Responses	bestWhResponse Data Type: ITSOMultiWh Responses
Compare Wholesalers	ITSOCompareWh_I	compare	wholesalers Data Type: ITSOTwoWh Responses	bestWholesaler Data Type: ITSOOrdWh Response

Note: Keep the interface names short because WSDL files are created based on the interface names.

Archived

Creating the business objects

This section explains how to build the required business objects. All SCA components communicate with each other using business objects. *Business objects* are containers for application data that represent business functions or elements, such as a customer or an order.

In this business process, we use the main business object ITSOProcessOrder to store all the information about the order as it moves through the process. The ITSOProcessOrder business object contains two child business objects called *ITSOrdPart* and *ITSOOrdWhResponse*. We implement child business objects because we want to demonstrate how to build hierarchical business objects. The ITSOOrdWhResponse business object will be reused in another parent business object (the ITSOMultiWhResponses business object).

1. Create the business object ITSOProcessOrder in the module ITSOElectronics:
   a. Right-click **ITSOElectronics** in the Business Integration view and select **New** → **Business Object**.
   b. In the New Business Object window (Figure 10-15), complete these tasks:
      i. For Name, type `ITSOProcessOrder`.
      ii. For Namespace, keep the default `http://ITSOElectronics`.
      iii. Click **Finish**.

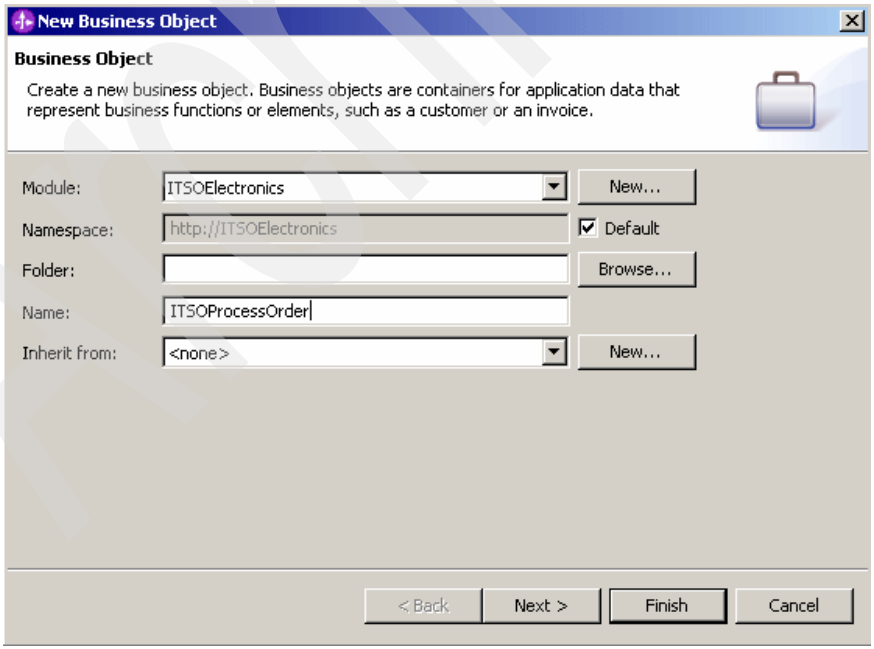


Figure 10-15 New Business Object panel

2. The Business Object Editor opens. Add the attributes as shown in Figure 10-16.

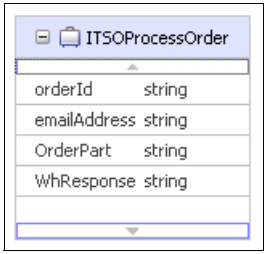


Figure 10-16 Attributes of the business object ITSOProcessOrder

3. For the attribute OrderPart, click the type **String** and select **New** from the pop-up menu.
4. In the New Business Object window (Figure 10-17), enter a name of `ITSOOrdPart` and click **Finish**. This associates the attribute OrderPart with the newly created ITSOOrdPart business object.

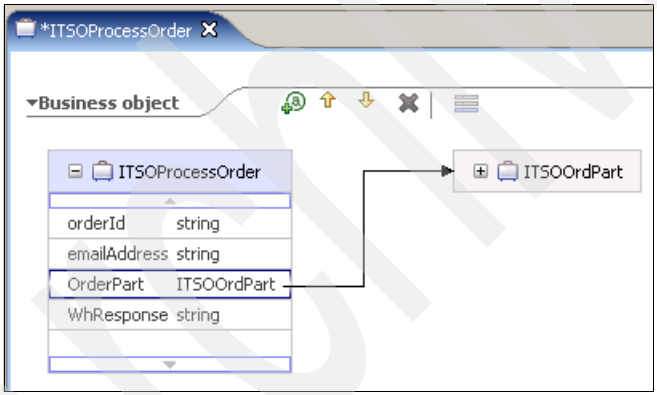


Figure 10-17 Attributes of the business object ITSOProcessOrder

5. Double-click the **ITSOOrdPart** business object to open it in its own Business Object Editor.

Archived

6. Add the attributes shown in Figure 10-18. Remember to change the data type of the qty attribute to `int`.

Figure 10-18 ITSOOrdPart business object

7. Press Ctrl+S to save the ITSOOrdPart business object, and close the ITSOOrdPart Business Object Editor.
8. In the ITSOProcessOrder Business Object Editor, expand **ITSOOrdPart**. The ITSOProcessOrder business object should now be displayed as shown in Figure 10-19.

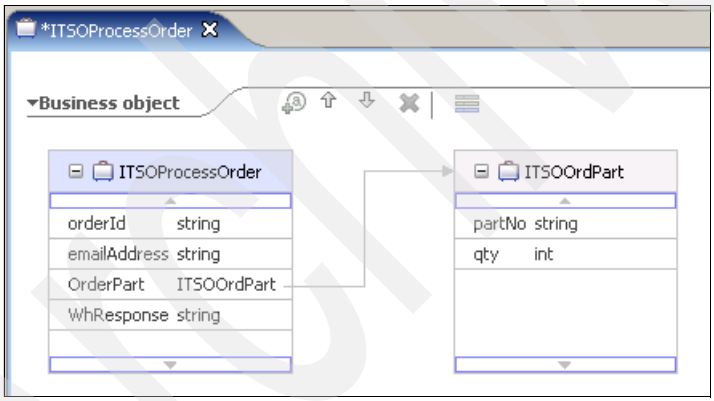


Figure 10-19 Business object ITSOProcessOrder

9. Click the type **String** of the attribute WhResponse and select **New** from the pop-up menu.
10. In the New Business Object panel, in the Name field, type `ITSOOrdWhResponse` and click **Finish**.

Archived

11. Double-click the **ITSOOrdWhResponse** business object to open it in the Business Object Editor. Then add the attributes shown in Figure 10-20.

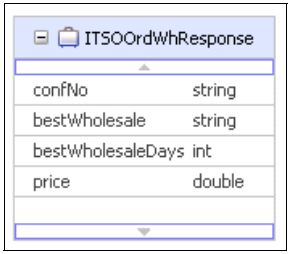


Figure 10-20 Business object ITSOOrdWhResponse

12. Press Ctrl+S to save the ITSOOrdWhResponse business object, and then close the Business Object Editor.

    The ITSOProcessOrder business object should now be displayed like the example in Figure 10-21.

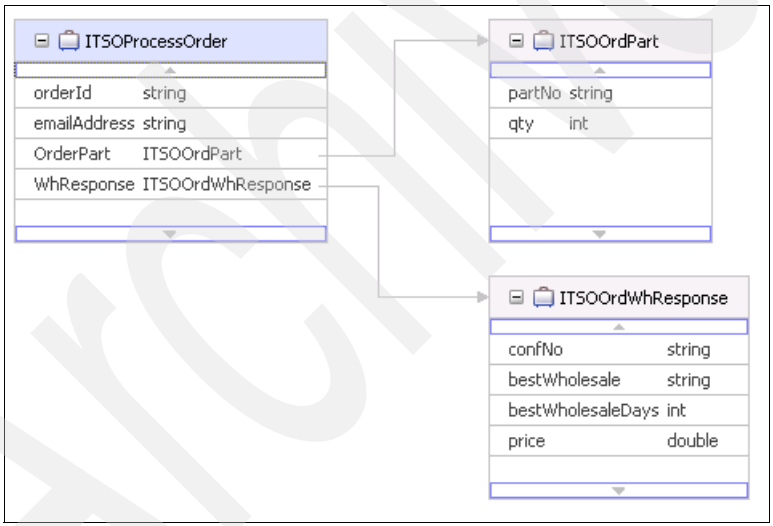


Figure 10-21 Business object ITSOProcessOrder with all child business objects

13. Press Ctrl+S to save the ITSOProcessOrder business object and close the Business Object Editor.

14. Close the ITSOProcessOrder business object.

Archived

Next we create the ITSOMultiWhResponses business object. This business object stores the delivery days of the different wholesalers. It contains an array of the data type ITSOOrdWhResponse. Using this business object, we can determine the best wholesaler.

1. Create a new business object in the ITSOElectronics module and name it `ITSOMultiWhResponses`.
2. Create an attribute named `whResponses`, and select **ITSOOrdWhResponse** as the data type (Figure 10-22).

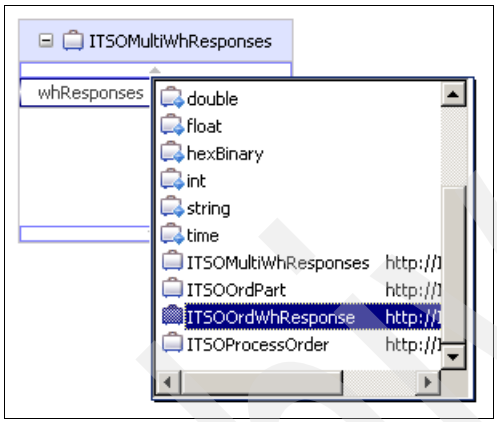


Figure 10-22 Selecting the data type ITSOOrdWhResponse

3. Select the **whResponses** attribute, and then click the **Properties** tab and the **Description** subtab.

Archived

4. Select the **Array** check box. Make sure that the square brackets appear next to the type of the attribute whResponse (Figure 10-23).

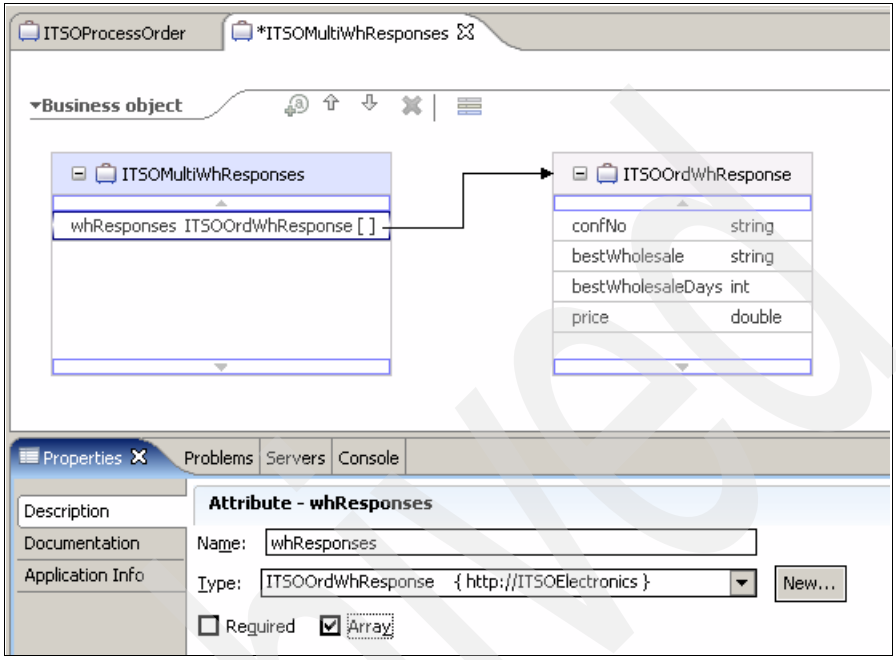


Figure 10-23 Data type and arrays

5. Press Ctrl+S to save the ITSOMultiWhResponses business object, and close the Business Object Editor.

Finally create the business object ITSOTwoWhResponses. This business object is used to compare the delivery days of the different Wholesalers in a business rule. We do not use the ITSOMultiWhResponses object in a business rule, because you cannot easily access array data types using the Business Rule Editor.

1. Create a business object named `ITSOTwoWhResponses` in the ITSOElectronics module.
2. Create an attribute and name it `wh1Response`. Select a data type of **ITSOOrdWhResponse**.

3. Create an attribute and name it `wh2Response`. Select a data type of **ITSOOrdWhResponse** (Figure 10-24).

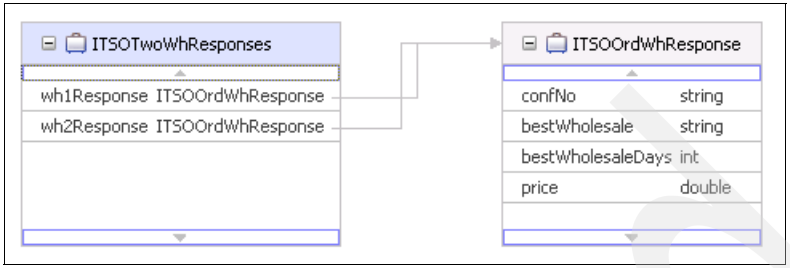


Figure 10-24 Business object ITSOTwoWhResponses

Creating the interfaces

An interface provides the means to interact with a component. It is created independently of the internal implementation of the component. Create the following interfaces in the ITSOElectronics module:

> **Note:** To create an interface, right-click the module where you want the interface to be created and click **New** → **Interface**.

- ITSOOrdReq_I as displayed in Figure 10-25

  This interface is used to test the Serial Process using the test client of WebSphere Integration Developer. You might use it to expose the Serial Process as a Web service.

  > **Note:** When creating interfaces with one-way operations, as is the case with this operation (it has one input and no outputs), create this operation with the Add One Way Operation icon.

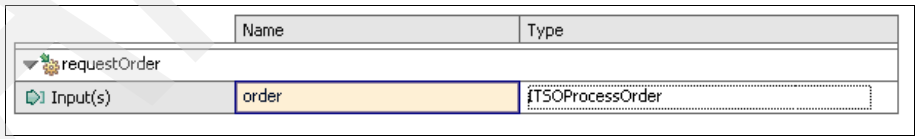


Figure 10-25 The ITSOOrdReq_I interface

- ITSOOrdNotify_I as displayed in Figure 10-26

  The ITSOProcessOrder business object is sent through this interface to the OrderAck component.

	Name	Type
notifyDeliveryDate		
Input(s)	order	ITSOProcessOrder

Figure 10-26 The ITSOOrdNotify_I interface

- ITSODelDays_I as displayed in Figure 10-27

  The ITSOProcessOrder business object is sent through this interface to the wholesalers to get the delivery days of an item.

	Name	Type
getDeliveryDays		
Input(s)	getDeliveryDaysWhReq	ITSOProcessOrder
Output(s)	getDeliveryDaysWhRes	ITSOProcessOrder

Figure 10-27 The ITSODelDays_I interface

- ITSOPlaceOrd_I as displayed in Figure 10-28

  The ITSOProcessOrder business object is sent through this interface to the wholesalers to place an order.

	Name	Type
placeOrder		
Input(s)	orderReq	ITSOProcessOrder
Output(s)	orderConf	ITSOProcessOrder

Figure 10-28 The ITSOPlaceOrd_I interface

Archived

- ITSODateCheck_I as displayed in Figure 10-29

  This is the interface to the business rule to check if the amount of delivery days of Wholesaler A is acceptable.

	Name	Type
checkDeliveryDate		
Input(s)	deliveryDays	ITSOProcessOrder
Output(s)	daysOK	boolean

Figure 10-29 The ITSODateCheck_I interface

- ITSOPickBestWh_I as displayed in Figure 10-30

  The ITSOProcessOrder business object is sent through this interface to the business rule to compare wholesalers.

	Name	Type
pickBestWh		
Input(s)	whResponses	ITSOMultiWhResponses
Output(s)	bestWhResponse	ITSOMultiWhResponses

Figure 10-30 The ITSOPickBestWh_I interface

- ITSOCompareWh_I as displayed in Figure 10-31

  This is the interface of the business rule ITSO_CompareWholesalers.

	Name	Type
compare		
Input(s)	wholesalers	ITSOTwoWhResponses
Output(s)	bestWholesaler	ITSOOrdWhResponse

Figure 10-31 The ITSOCompareWh_I interface

Archived

Make sure the ITSOElectronics module looks like the example in Figure 10-32.

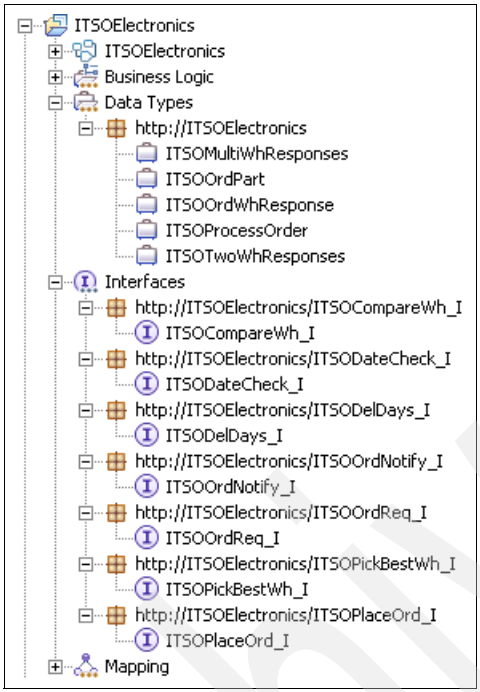


Figure 10-32 Artifacts for the ITSOElectronics module

Creating the WS-BPEL process

In this section, the WS-BPEL part of the business logic of ITSOElectronics is implemented using the Business Process Editor. The Business Process Editor is a powerful enterprise workflow tool that supports both business processes and human tasks. In this scenario, we implement a serial business process.

Refer to Figure 10-2 on page 147 for an overview of the business process.

Creating the business process

Before using any WS-BPEL activities, we have to create interface partners and reference partners for the WS-BPEL process. *Interface partners* are inbound interfaces of the business process. *Reference partners* are outbound interfaces of the business process. Our business process has only one interface partner. In this section, the business process and the interface partner are created.

Archived

Create a business process called *SerialProcess*:

1. Right-click the **ITSOElectronics module** and select **New** → **Business Process**.
2. In the New Business Process panel, for Name, type `SerialProcess` and click **Next**.
3. In the Select an Interface panel (Figure 10-33), complete these tasks:
   a. Select the **Select an existing Interface** option
   b. Click **Browse** to select the **ITSOOrdReq_I interface**.
   c. For Operation, select **requestOrder operation**. This is the interface partner of the business process.
   d. Click **Finish**.

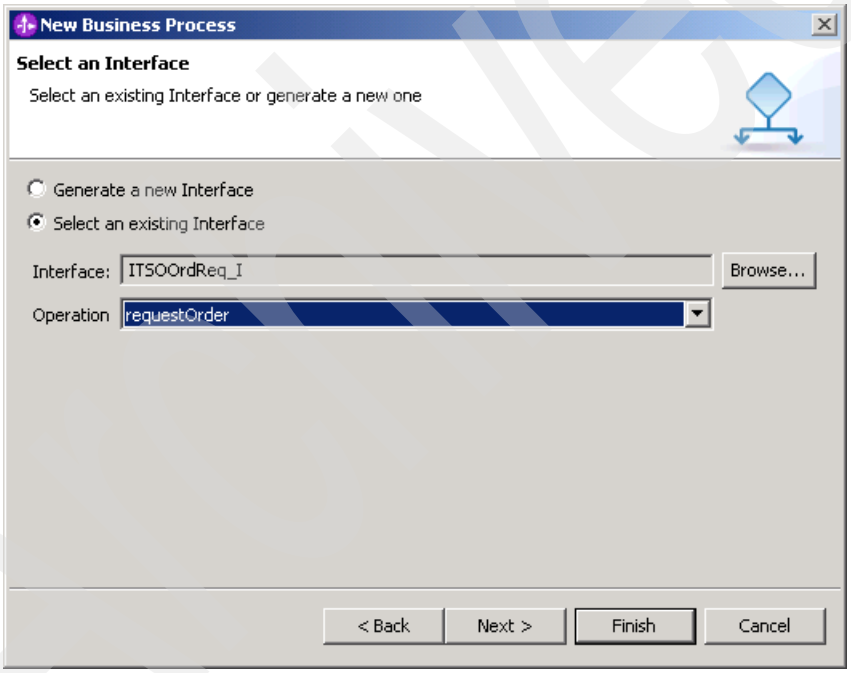


Figure 10-33 New Business Process wizard: Select an Interface panel

The business process opens in the Business Process Editor.

Note: The business process has a receive activity, but no reply activity. This is because its triggering operation is a one-way operation.

Creating the reference partners

To create the reference partners:

1. Drag the following interfaces to the SerialProcess Business Process Editor canvas to add reference partners:
   - ITSOOrdNotify_I
   - ITSODateCheck_I
   - ITSOPickBestWh_I
2. Drag the following interfaces twice each to the SerialProcess Business Process Editor canvas to add reference partners. We need two reference partners of each interface because each interface will be used by Wholesaler A and for Wholesaler B:
   - ITSODelDays_I
   - ITSOPlaceOrd_I

   Important: Do *not* drag the ITSOOrdReq_I interface to the canvas, because it is the triggering interface and therefore it is already available.

3. Rename the reference partners as indicated in Table 10-2.

Table 10-2 SerialProcess reference partner renaming

Rename from	Rename to
ITSOOrdNotify_I	OrderNotifyPartner
ITSODateCheck_I	DateCheckPartner
ITSOPickBestWh_I	PickBestWhPartner
ITSODelDays_I	WhADeliveryPartner
ITSODelDays_I	WhBDeliveryPartner
ITSOPlaceOrd_I	WhAOrderPartner
ITSOPlaceOrd_I	WhBOrderPartner

Note: We rename the reference partners to avoid dependencies to interfaces.

Tip: You can check the interface of the referencePartner by highlighting the reference partner and selecting the Properties tab and the Details subtab.

Figure 10-34 shows the Reference Partners before and after renaming.

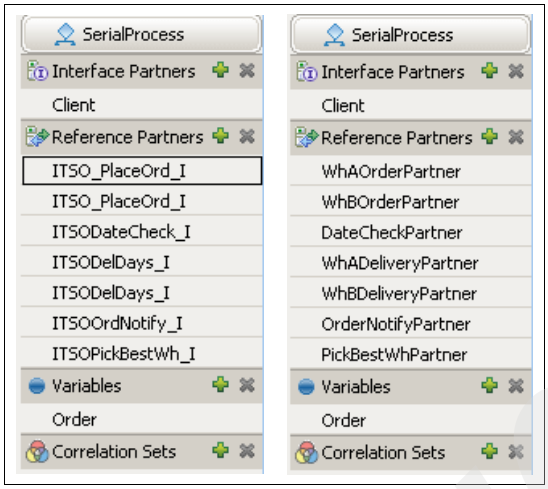


Figure 10-34 Reference partners before and after renaming

Creating the variables in the Business Process Editor

Before using any WS-BPEL activities, create variables which are assigned to data types. Variables help to hold data while we run through the process.

1. In the Business Process Editor, rename the Order variable to `currentOrder`.
2. Add the variable `isDateOK`.
   a. Click the plus sign (**+**) next to **Variables** and rename Variable to `isDateOK`.
   b. Select the **Properties** tab and the **Details** subtab.
   c. Click **Browse** (Figure 10-35).
   d. Select **boolean** and click **OK**.

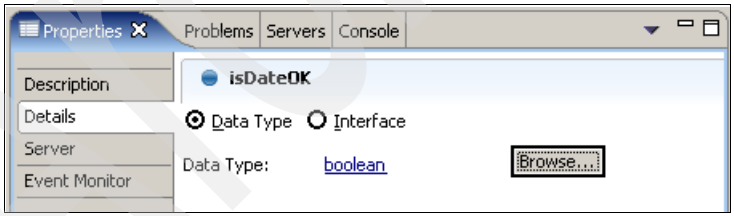


Figure 10-35 Adding a variable and selecting the data type

3. Add the variables listed in Table 10-3.

Table 10-3 Variables to be added to the SerialProcess

Variable	Type
multipleWholesalerDates	ITSOMultiWhResponses
wholesalerResponse	ITSOProcessOrder

4. Press Ctrl+S to save the business process. Do not close the Business Process Editor.

Adding the activities to the Business Process Editor

Now we start to add WS-BPEL activities to the Business Process Editor canvas. We build the business process described in Figure 10-36 and in Table 10-4.

Figure 10-36 SerialProcess business process

Archived

Table 10-4 SerialProcess activity names and types

Name	Type
Receive Order Request	Receive
Get Date From Wholesaler A	Invoke
Check if the Date is OK using Business Rule	Invoke
Is Date OK?	Choice
True	Choice case
False	Choice case
Best Wholesaler is A	Assign
Store date from Wholesaler A in Responses List	Snippet
Get Date from Wholesaler B	Invoke
Add Date from Wholesaler B to List of Responses	Snippet
Pick Best Wholesaler from List using Business Rule	Invoke
Update Order with Best Wholesaler	Snippet
Choose Best Wholesaler	Choice
Wholesaler A	Choice case
Wholesaler B	Choice case
Place Order with Wholesaler A	Invoke
Place Order with Wholesaler B	Invoke
Copy Wholesaler Confirmation into Order	Assign
Notify Order Details	Invoke

Renaming the existing activity

The business process already contains one activity, a receive activity. We provide it with a more descriptive name:

1. Select the receive activity in the Business Process Editor.
2. Rename the receive activity to `Receive Order Request`.

Archived

Adding the invoke activities

Add the invoke activities to the business process.

1. Drag an **invoke activity** to the canvas and rename it to `Get Date From Wholesaler A`. The canvas should look like the example in Figure 10-37.

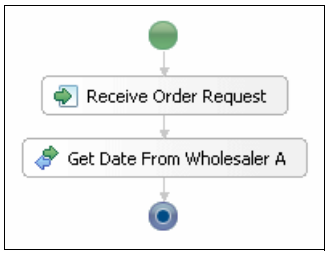


Figure 10-37 Two first activities

2. Click **Get Date from Wholesaler A** to highlight it.
3. Click the **Properties** page and the **Details** subtab.
4. Click **Browse** and select **WhADeliveryPartner**.
5. Verify that the operation is set to `getDeliveryDays` as shown in Figure 10-38.

> **Note:** The getDeliveryDays is the only operation of WhADeliveryPartner. Therefore, it is set as the default.

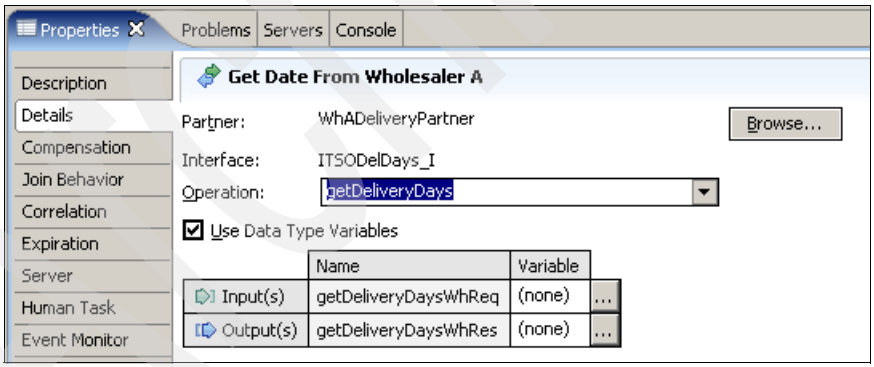


Figure 10-38 Setting the partner, interface, and operation

6. In the Input(s) row, click the **...** button to open the Select Variable window.
7. In the Select Variable for getDeliveryDaysWhReq window, select the **currentOrder** variable and click **OK**.
8. In the Output(s) row, click the **...** button.

9. In the Select Variable for getDeliveryDaysWhRes window, select the **wholesalerResponse** variable, and click **OK**.
10. Repeat the previous steps for the remaining invoke activities. Refer to Table 10-5 to set the Partner, Interface, Operation, and Variables for these activities.

Table 10-5 The SerialProcess invoke activities

Name	Partner	Interface	Operation	Input	Output
Check if the Date is OK using Business Rule	DayCheck Partner	ITSODate Check_I	checkDelivery Date	wholesaler Response	isDateOK
Get Date from Wholesaler B	WhBDelivery Partner	ITSODel Days_I	getDelivery Days	currentOrder	wholesaler Response
Pick Best Wholesaler from List using Business Rule	PickBestWh Partner	ITSOPickBest Wh_I	pickBestWh	Multiple Wholesaler Dates	Multiple Wholesaler Dates
Place Order with Wholesaler A	WhAOrder Partner	ITSO_Place Ord_I	placeOrder	currentOrder	wholesaler Response
Place Order with Wholesaler B	WhBOrder Partner	ITSOPlace Ord_I	placeOrder	currentOrder	wholesaler Response
Notify Order Details	OrderNotify Partner	ITSOOrd Notify_I	notifyDelivery Date	currentOrder	N/A

Archived

11. Save the business process. The invoke activities should look like the example in Figure 10-39.

Figure 10-39 Invoke activities

Adding the assign activities

In this section, we add the assign activities to the business process. Refer to Table 10-6 to set the Order, From Type, From Value, To Type, and To Value of the two assign activities.

Table 10-6 The SerialProcess assign activities

Name	Order	From type	From	To type	To
Best Wholesaler is A	1	Fixed value	Wholesaler A	Variable	currentOrder. WhResponse.best Wholesale
	2	Variable	wholesalerResponse.Wh Response.bestWholesale Days	Variable	currentOrder. WhResponse.best WholesaleDays
Copy Wholesaler Confirmation into Order	1	Variable	wholesalerResponse.Wh Response.confNo	Variable	currentOrder.Wh Response.confNo

Archived

Adding the choice activities

Add the choice activities to the business process.

1. Add a choice activity to the business process:
   a. On the Process Editor palette, click the **Choice** icon ( ) and click the **arrow** before the last assign activity to add the new choice activity.
   b. Click the **Properties** page. Click the **Description** tab. In the Display Name field, type `IsDateOK`.
   c. Click **Case**.
   d. Click the **Properties** page. Click the **Description** tab. In the Display Name field, type `true`.
   e. Click the **Details** tab. In the Expression language field, select **Same as Process (Java)**.
   f. Click the Expression Type **Visual** radio button.
   g. To build the true visual snippet, complete the following tasks in the Visual Snippet Editor:
      i. Click the **true** expression and click **isDateOK** from the helper menu.
      ii. From the helper menu, click **==**.
      iii. From the helper menu, click **true**.

      Figure 10-40 shows the completed true case visual snippet.

Figure 10-40 IsDateOK choice case true

2. Press Ctrl+S to save the business process.
3. Right-click **IsDateOK** and click **Add Case**.
4. Add and define a case element in the result of a false expression as follows:
   a. Click **Case**.
   b. Click the **Properties** page. Click the **Description** tab. In the Display Name field, type `false`.
   c. Click the **Details** tab. In the Expression language field, select **Same as Process (Java)**.
   d. For Expression Type, select the **Visual** radio button.

Archived

e. To build the false visual snippet, complete the following steps in the Visual Snippet Editor:

   i. Click the **true** expression and click **isDateOK** from the helper menu.
   ii. From the helper menu, click **==**.
   iii. From the helper menu, click **false**.

   Figure 10-41 shows the completed false case visual snippet.

Figure 10-41 IsDateOK choice case false

5. Press Ctrl+S to save the business process.
6. Repeat the previous steps for the Choose Best Wholesaler choice activity. The Choose Best Wholesaler choice activity also has two cases: Wholesaler A and Wholesaler B. Complete the following steps to create the respective case:

   a. Refer to Figure 10-42 to create the Wholesaler A case.

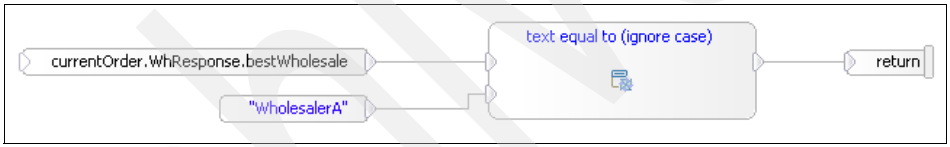


Figure 10-42 Choose Best Wholesaler choice case Wholesaler A

   b. Refer to Figure 10-43 to create the Wholesaler B case.

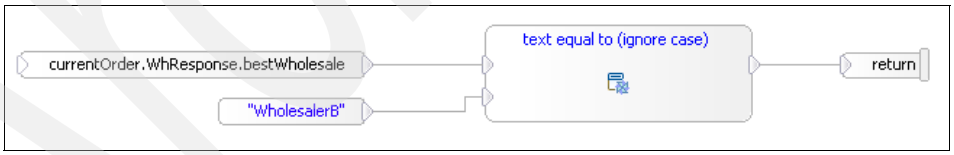


Figure 10-43 Choose Best Wholesaler choice case Wholesaler B

> **Note:** To access the `text equal to (ignore case)` snippet, in the Visual Snippet Editor, click **Standard Visual Snippets** → **text**.

Archived

Adding the visual snippet activities

Add the visual snippet activities to the business process.

1. Refer to Figure 10-44 to create the snippet activity *Store Date from Wholesaler A in Responses List*. You may find the following hints to be useful:
   - `create ITSOMultiWhResponses` is under the menu path **Standard** → **SCA services** → **create specific BO snippet**.
   - To add the variables, such as multipleWholesalerDates, right-click and use **Add Existing**.
   - To select child objects, such as mutlipleWholesalerDates.whResponses, right-click the parent snippet and use the visual assist menu to select the required child.
   - `add item to list` is under the menu path **Standard** → **list** → **add item to list snippet**.
   - "WholesalerA" is a snippet of type Expression. Make sure the quotation marks appear.
   - Remember to link from source to target and from left to right. For example, click **"WholesalerA"** and link it to **wholesalerResponse.WhResponse.bestWholesale**, and not vice versa.

> **Remember:** In the snippet, use `create` *`specific BO`*, and not `create` *`BO`*, when creating business objects, such as `create ITSOMultiWhResponses`.

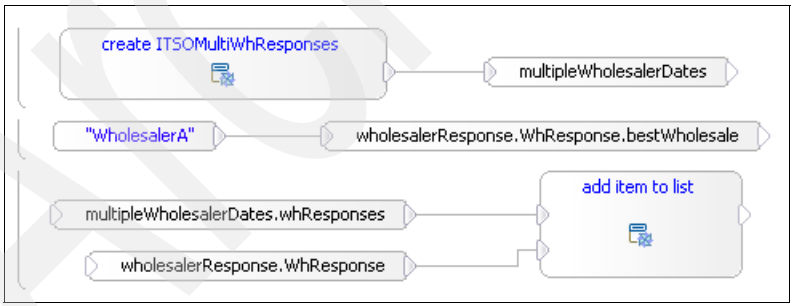


Figure 10-44 Store Date from Wholesaler A in Responses List snippet activity

Archived

2. Refer to Figure 10-45 to create the snippet activity *Add Date from Wholesaler B to List of Responses*.

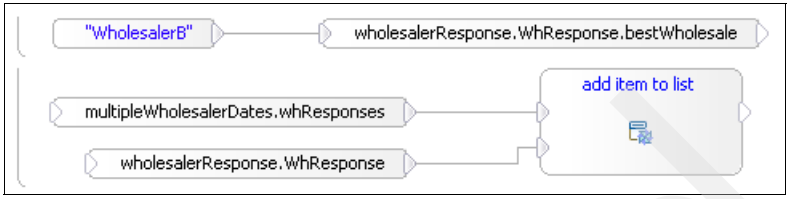


Figure 10-45 Add Date from Wholesaler B to List of Responses snippet activity

3. Refer to Figure 10-46 to create the snippet activity *Update Order with Best Wholesaler*.

Figure 10-46 Update Order with Best Wholesaler snippet activity

4. Press Ctrl+S to save the SerialProcess business process.

Rearranging the activities

Arrange the activities of the business process.

1. Rearrange the SerialProcess activities as shown in Figure 10-36 on page 186.
2. Press Ctrl+S to save the SerialProcess business process.
3. Close the Business Process Editor.

Creating the business rules

A *business rule* is anything that imposes structure upon, or controls the behavior of, a business practice. A *rule* can enforce business policy, establish common guidelines within an organization, or control access in a business environment. A business rule is used to help abstract the client from implementation.

In this section, we build the following business rules:

- CheckDate
- PickBestWholesaler

Use business rules to officiate over frequently changing business practices that can come from within a business or mandated from outside a business. Business rules allow Business Analysts to change business rules on demand without

Archived

restarting the project or the application server using a Business Rule Editor Web tool.

Creating the business rule ITSO_CheckDate

The business rule ITSO_CheckDate checks if the delivery date from Wholesaler A is an acceptable delivery date. If the date is acceptable, then it returns *true*; otherwise, it returns *false*. You can change the maximum number of acceptable delivery days without restarting the application using the Business Rule Editor Web tool:

1. Right-click the **ITSOElectronics** module and click **New** → **Rule Group**.
2. In the New Rule Group wizard (Figure 10-47) that opens, in the Name field, type `ITSO_CheckDate`. Click **Next**.

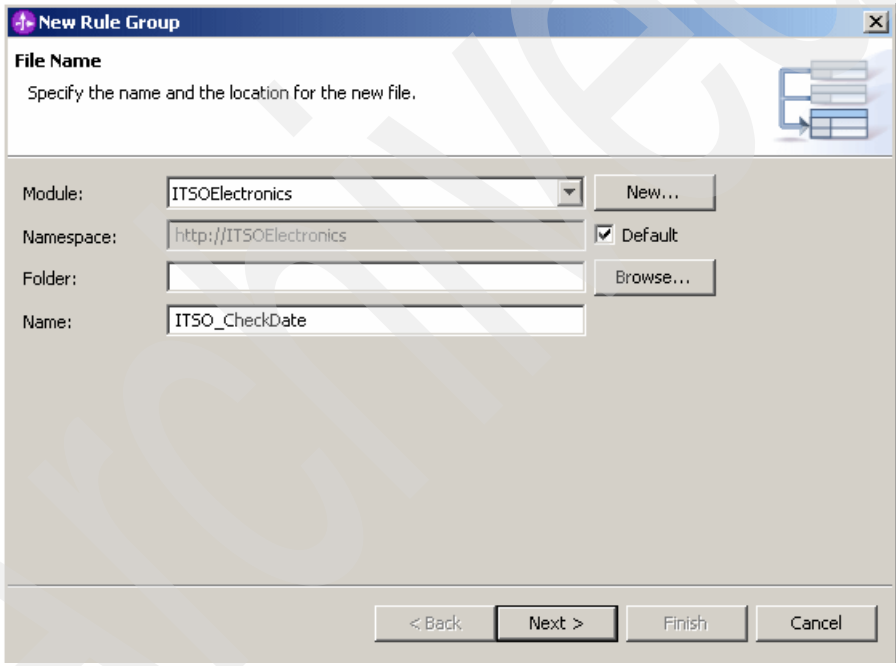


Figure 10-47 New Rule Group wizard: File Name panel

Archived

3. In the Select an Interface panel (Figure 10-48), select the **ITSODateCheck_I** interface, and click **Finish**.

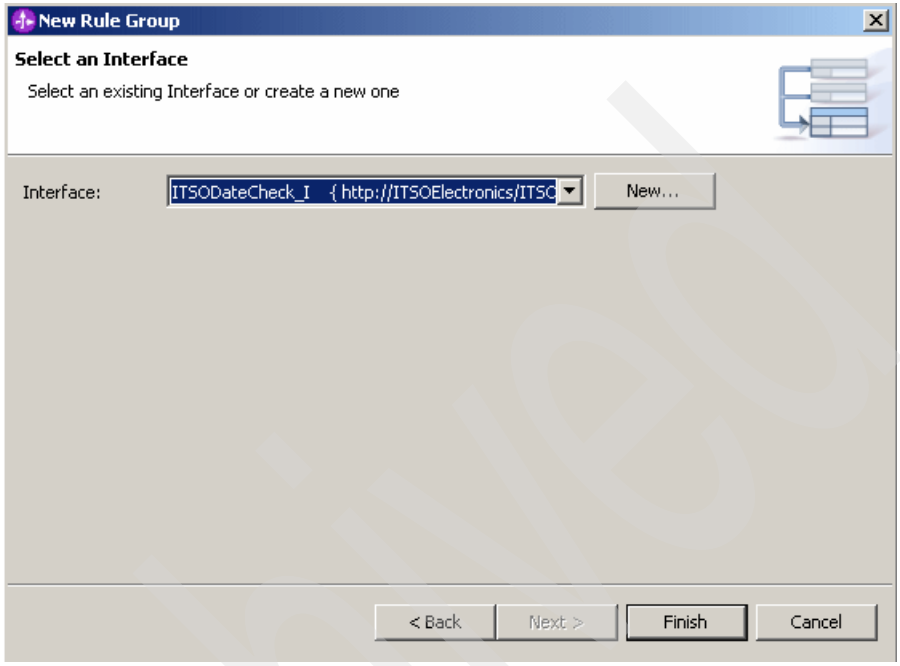


Figure 10-48 New Rule Group wizard: Select an Interface panel

4. In the ITSO_CheckDate Rule Group Editor that opens, complete the following steps:
   a. Click the **checkDeliveryDate** operation.
   b. In the Active Destinations section, click **Enter Destination**.
   c. From the helper menu, click **New Ruleset**.
5. In the New Rule Set window that opens, in the Name field, type `dateCheck_RS`. Then click **Finish**.
6. In the dateCheck_RS Rule Set Editor that opens, complete these steps:
   a. In the **Variables** section, click the **+** sign.
   b. Rename var1 to `numberOfDays`.
   c. Click **Select Type** and select **int**. Figure 10-49 illustrates the changes.

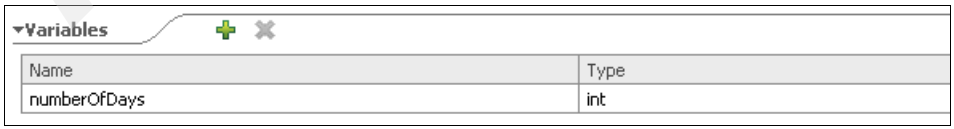


Figure 10-49 Adding a variable

7. In the Template section, click the **Add If Then Template** icon (), and complete the following steps:

   a. Click the **+** sign in the **Parameters** row.
   b. In the Name field, rename `param1` to `numDaysParam`.
   c. For Type, select **int** as displayed in Figure 10-50.

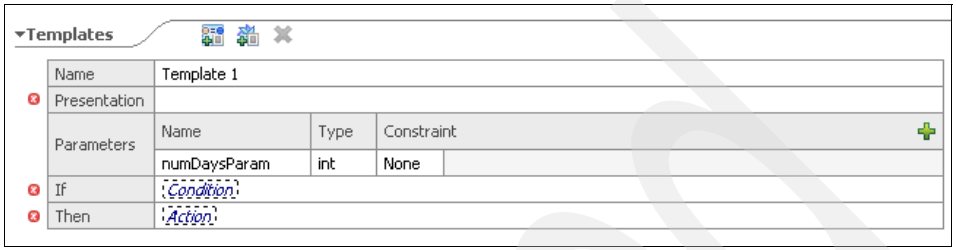


Figure 10-50 Adding a parameter

   d. In the If row, click **Condition** and select **var1 is greater than var2**.
   e. Click the first **Select Variable** and select **numberOfDays**.
   f. Click the second **Select Variable** and select **numDaysParam**.
   g. Click **Action** and select **Set var1 to var2**.
   h. Click the first **Select Variable** and select **daysOK**.
   i. Click the second **Select Variable** and select **false**. Refer to Figure 10-51.

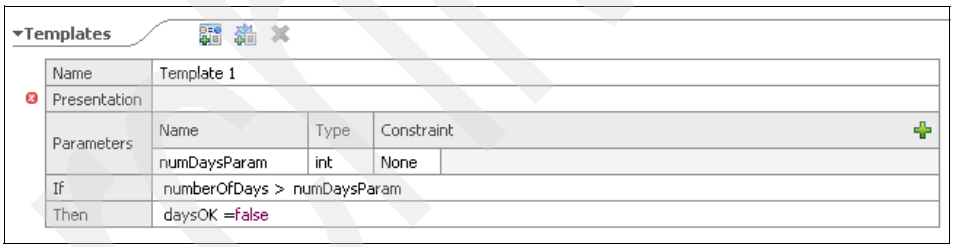


Figure 10-51 Adding an if-then rule

   j. Click the **Presentation** row, and type `Select Number of Days:`. Then press Ctrl+Space bar to open the visual assist and select **numDaysParam**.

Archived

The template section is now complete and should look like the example in Figure 10-52.

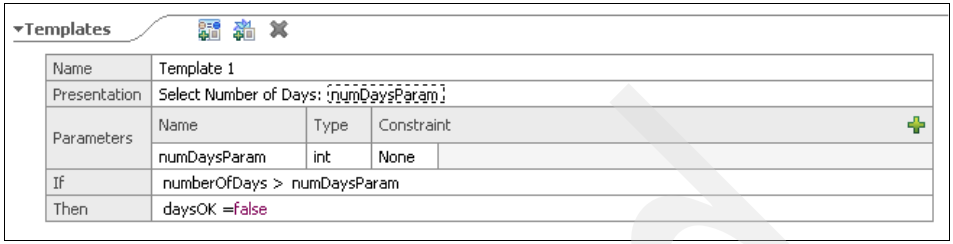


Figure 10-52 The completed dateCheck_RS Template section

8. In the Rule section, click the **Add Action Rule** icon ( ) twice to add two new rules.

   a. In Rule1, set the Action to `daysOK=true`.

   > **Important:** If you are using Visual Assist, make sure that you select = and not ==.

   b. In Rule2, set Action to `numberOfDays=deliveryDays.WhResponse.bestWholesaleDays` as in Figure 10-53.

Figure 10-53 The dateCheck_RS if-then rules

9. Based on the template, we create a new rule. Click the **Add Template Rule** icon ( ) and select **Template 1**.

Archived

10. In the Presentation row of Rule3, click **Enter Value** and type 7. Figure 10-54 shows the completed dateCheck_RS rule set.

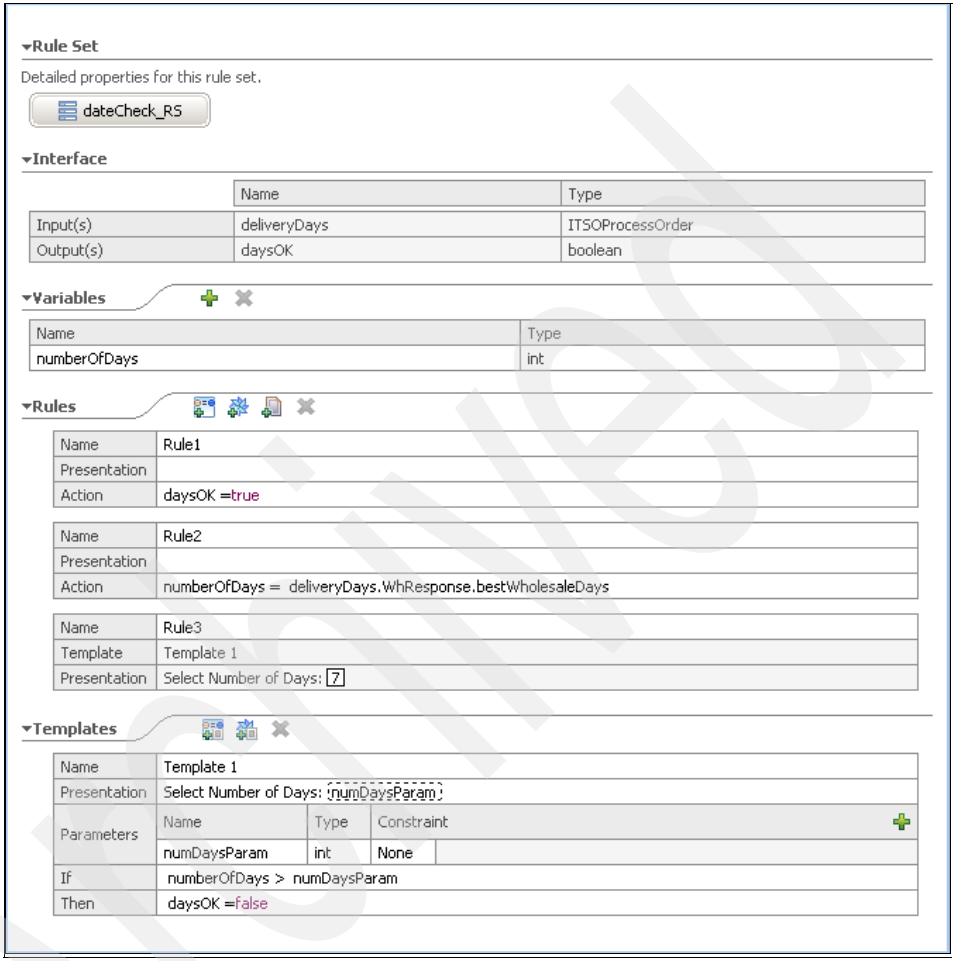


Figure 10-54 The completed dateCheck_RS business rule set

Archived

11. Press Ctrl+S to save the dateCheck_RS rule set, and then close the Rule Set Editor. The ITSO_CheckDate rule group should now look like the example in Figure 10-55.

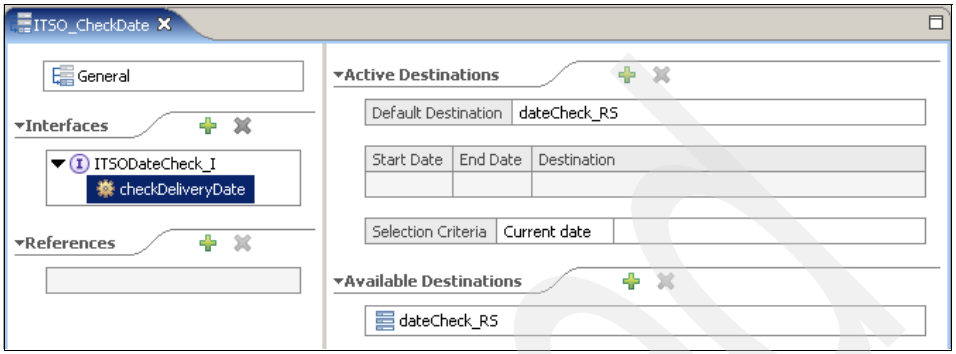


Figure 10-55 The completed ITSO_CheckDate rule group

12. Press Ctrl+S to save the ITSO_CheckDate rule group, and then close the Rule Group Editor.

Creating the business rule ITSO_CompareWholesalers

The business rule ITSO_CompareWholesalers, shown in Figure 10-56 on page 201, compares Wholesaler A and Wholesaler B and decides which is the best wholesaler.

1. Right-click the **ITSOElectronics** module and click **New** → **Rule Group**.
2. In the New Rule Group wizard that opens, in the Name field, type `ITSO_CompareWholesalers`, and click **Next**.
3. In the Select an Interface window, select the **ITSOCompareWh_I** interface, and click **Finish**.
4. From the ITSO_CompareWholesalers Rule Group Editor that opens, complete the following steps:
   a. Click the **compare** operation.
   b. In the Active Destinations section, click **Enter Destination**.
   c. From the helper menu, click **New Ruleset**.
5. In the New Rule Set window that opens, in the Name field, type `compareWh_RS`. Then click **Finish**.
6. From the compareWh_RS Ruleset Editor that opens, add the following variables:
   – `wholesaler1Days` with type `int`
   – `wholesaler2Days` with type `int`

Archived

7. Add four rules to the rule set as follows:

   a. In the Ruleset Editor, add `Rule1` and `Rule2` as two action rules. Add `Rule3` and `Rule4` as two if-then rules.

   b. In Rule1, set the Action to `wholesaler1Days=wholesalers.wh1Response.bestWholesaleDays`.

   c. In Rule2, set the Action to `wholesaler2Days=wholesalers.wh2Response.bestWholesaleDays`.

   d. In Rule3, set *If* to `wholesaler1Days>wholesaler2Days`. Set *Then* to `bestWholesaler=wholesalers.wh2Response`.

   e. In Rule4 set *If* to `wholesaler1Days<=wholesaler2Days`. Set *Then* to `bestWholesaler=wholesalers.wh1Response` (Figure 10-56).

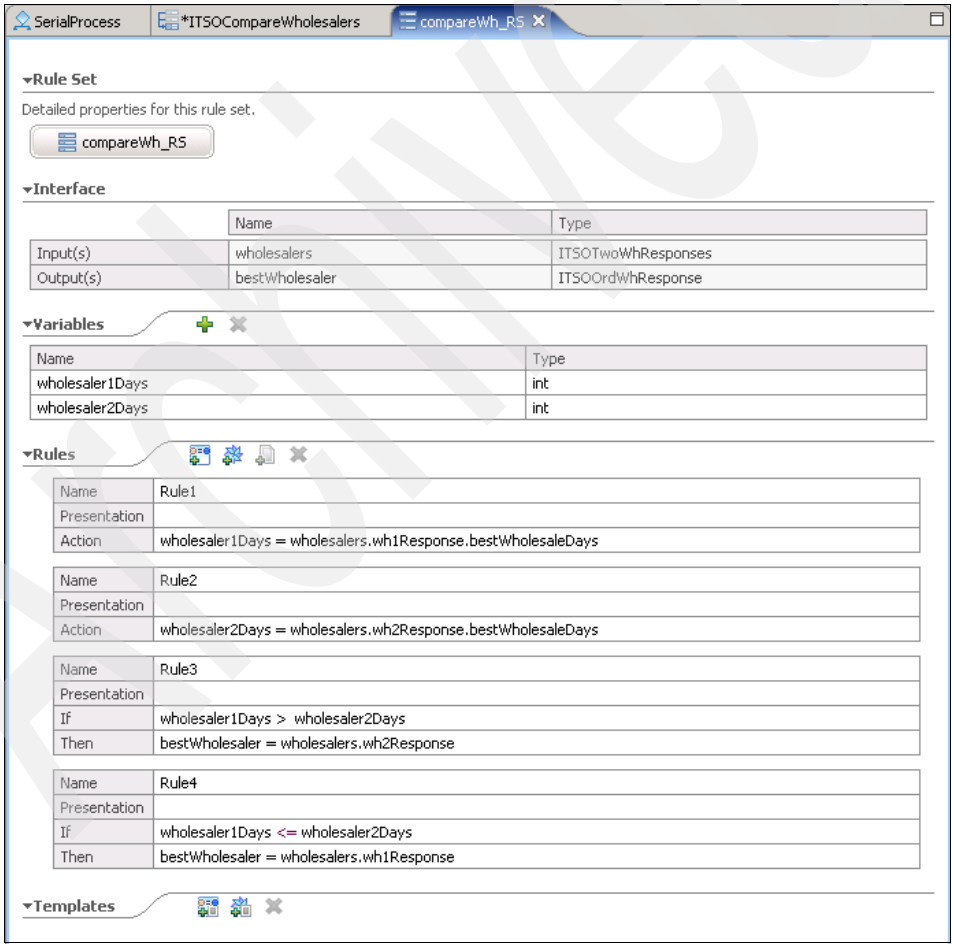


Figure 10-56 The completed compareWh_RS business rule set

Archived

8. Press Ctrl+S to save the compareWh_RS rule set, and close the Ruleset Editor. The ITSO_CompareWholesalers rule group should now look like the example in Figure 10-57.

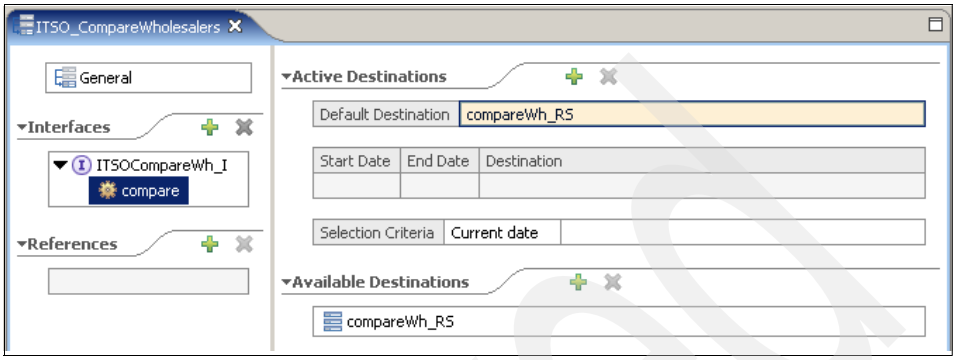


Figure 10-57 The completed ITSO_CompareWholesalers rule group

9. Press Ctrl+S to save the ITSO_CompareWholesalers rule group, and close the Rule Group Editor.

Creating data maps for mapping business objects

Create two data maps for mapping business objects used in the WS-BPEL business process to business objects used in the ITSO_CompareWholesalers business rule. In this section, we create the following data maps:

- ITSOProc_to_BR_WhResp that maps the business objects ITSOMultiWhResponse to ITSOTwoWhResponse
- ITSOBR_to_Proc_WhResp that serves as the reverse map of the data flow and maps the business objects ITSOOrdWhResponse to IITSOMultiWhResponse

Archived

Follow these steps:

1. Right-click the **ITSOElectronics** module and select **New** → **Business Object Map**. The New Business Object Map wizard opens.
2. In the Business Object Map panel (Figure 10-58), in the Name field, type `ITSOProc_to_BR_WhResp` and click **Next**.

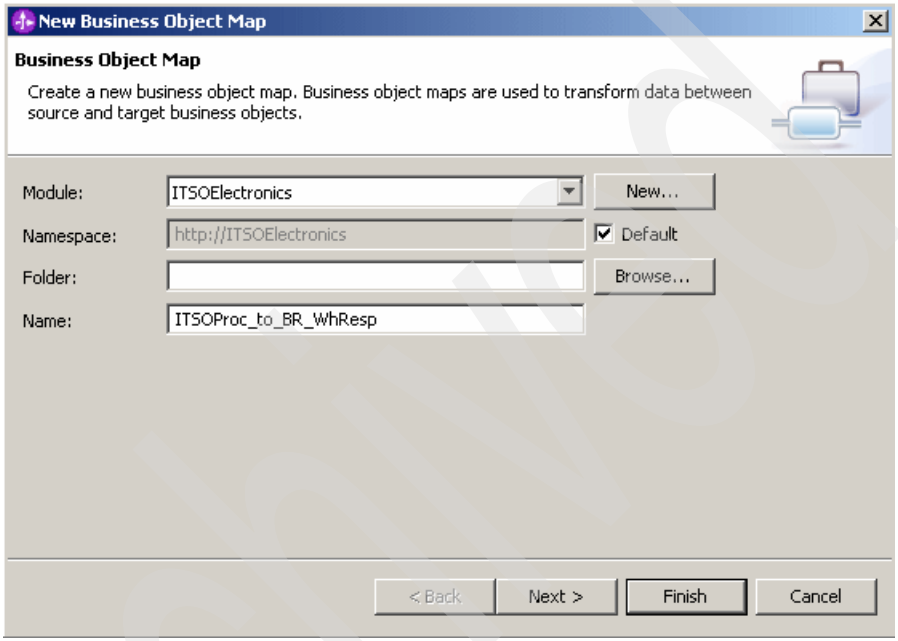


Figure 10-58 Business Object Map panel

Archived

3. In the next panel (Figure 10-59), for Inputs, select **ITSOMultiWhResponses**. For Outputs, select **ITSOTwoWhResponses** and click **Finish**.

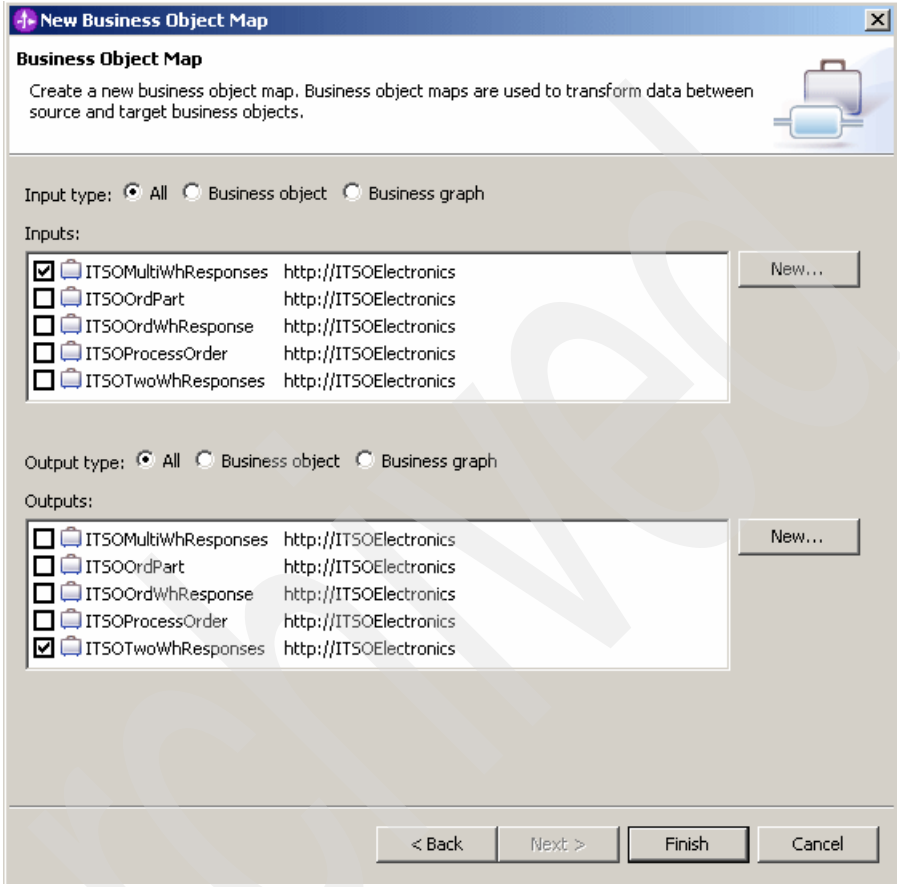


Figure 10-59 Business Object Map panel: Specifying the Inputs and Outputs

4. In the Business Object Mapping Editor panel that opens, complete these tasks:
   a. In the ITSOMultiWhResponses object, drag **whResponses** to **wh1Response** in the ITSOTwoWgResponses object. This creates a link between these two objects called a *submap*.
   b. In the ITSOMultiWhResponses object, drag **whResponses** to **wh2Response** in the ITSOTwoWhResponses object (Figure 10-60).

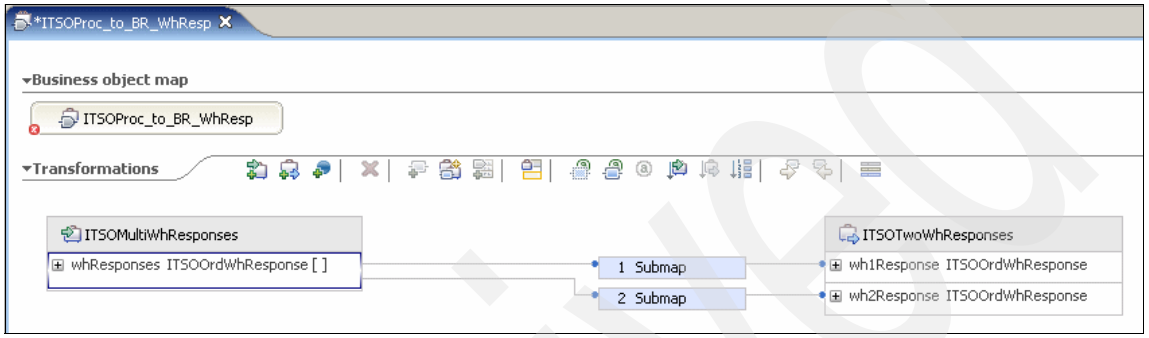


Figure 10-60 Dragging the objects

5. We want to visually create a custom mapping. Therefore, click **1 Submap** and select **Custom**.
6. Click **2 Submap** and select **Custom**.
7. Click **1 Custom** and select the **Properties** tab. Then select the **Details** tab.
8. The Visual Editor appears. Create the visual snippet shown in Figure 10-61. This snippet takes the first item from the ITSOMultiWhResponses_whResponses array and copies it to ITSOTwoWhResponses_wh1Response.

Figure 10-61 The 1 Custom visual snippet

> **Note:** The *get item at index* snippet is under Standard → list → get item at index.

9. Press Ctrl+S to save the data map.

Archived

10. Click **2 Custom** and select the **Properties** tab. Then select the **Details** tab.
11. In the Visual Editor, create the visual snippet shown in Figure 10-62.

Figure 10-62 The 2 Custom visual snippet

12. Press Ctrl+S to save the data map, and close the Business Object Mapping Editor.

Create another business object map ITSOBR_to_Proc_WhResp that serves as the reverse map of the data flow.

1. Set the name to `ITSOBR_to_Proc_WhResp`.
2. For Input, select **ITSOOrdWhResponse**.
3. For Output, select **ITSOMultiWhResponses**.
4. Map ITSOrdWhResponse to the ITSOMultiWhResponses object as shown in Figure 10-63 and change Submap to **Custom**.

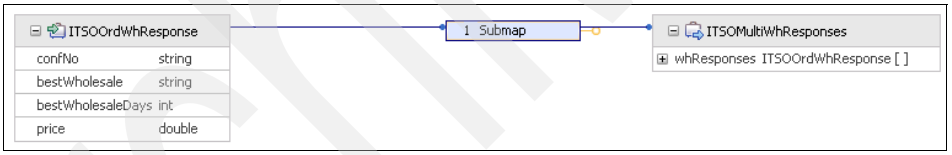


Figure 10-63 Map ITSOrdWhResponse to ITSOMultiWhResponse

Archived

5. Create the visual snippet for the custom mapping as shown in Figure 10-65 on page 208:
   a. In the Visual Editor, click the **Java** icon ( ).
   b. In the Select a Java Visual Snippet window (Figure 10-64), complete these tasks:
      i. In the Matching types box, select **Vector**.
      ii. In the Select a Visual Snippet box, select **Vector()**.
      iii. Click **OK**.

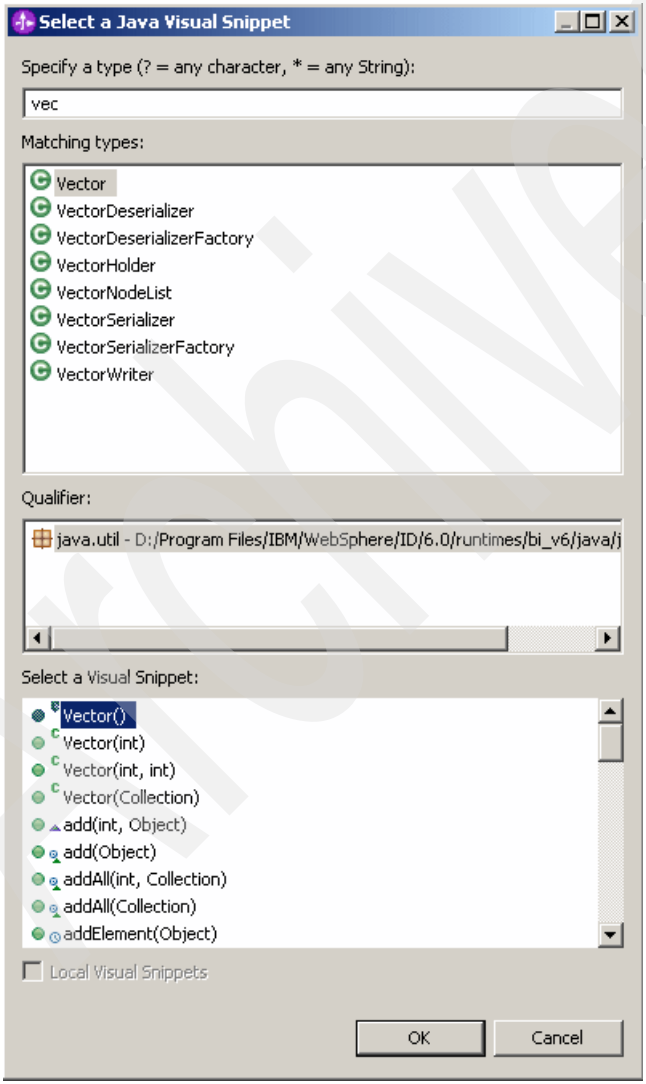


Figure 10-64 Selecting Vector()

Archived

c. Click the canvas to create the snippet.

d. Complete the visual snippet as shown in Figure 10-65.

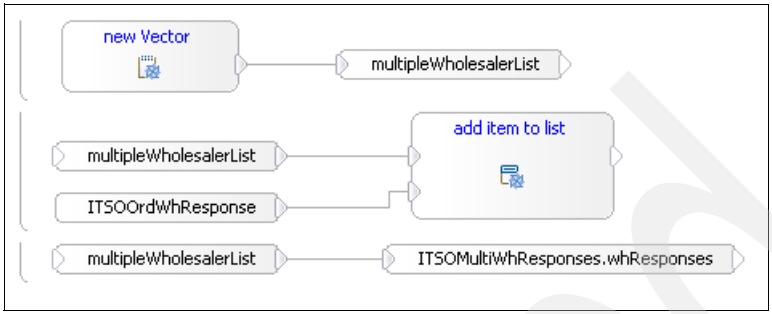


Figure 10-65 The 1 Custom visual snippet

6. Press Ctrl+S to save the business object map, and then close the Business Object Mapping Editor.

Archived

Creating an interface map

In this section, we build an interface map. The WS-BPEL business process and the business rule have two different interfaces. Because the two components communicate with each other, the interfaces must be mapped using an interface map. The interface map that we implement uses the two business object maps created in “Creating data maps for mapping business objects” on page 202.

1. Right-click the **ITSOElectronics** module and select **New** → **Interface Map**.
2. In the New Interface Map window (Figure 10-66), in the Name field, type `ITSOProc_to_BR_CompWh` and click **Next**.

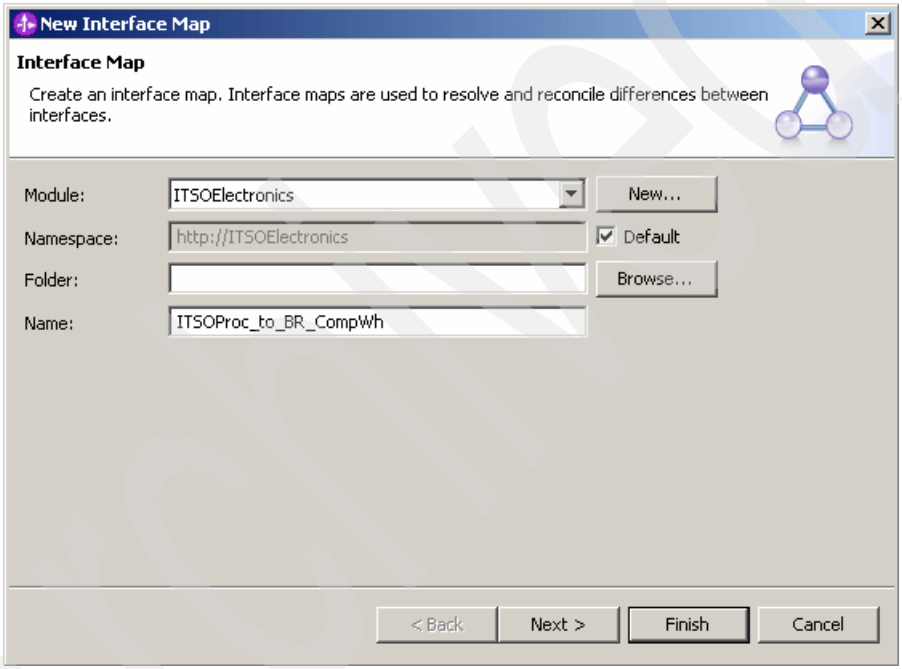


Figure 10-66 New Interface Map wizard: Creating an interface map

Archived

3. In the next panel (Figure 10-67), for Source interface, select **ITSOPickBestWh_I**. For Target interface, select **ITSOCompareWh_I**. Click **Finish**. The Interface Mapping Editor opens.

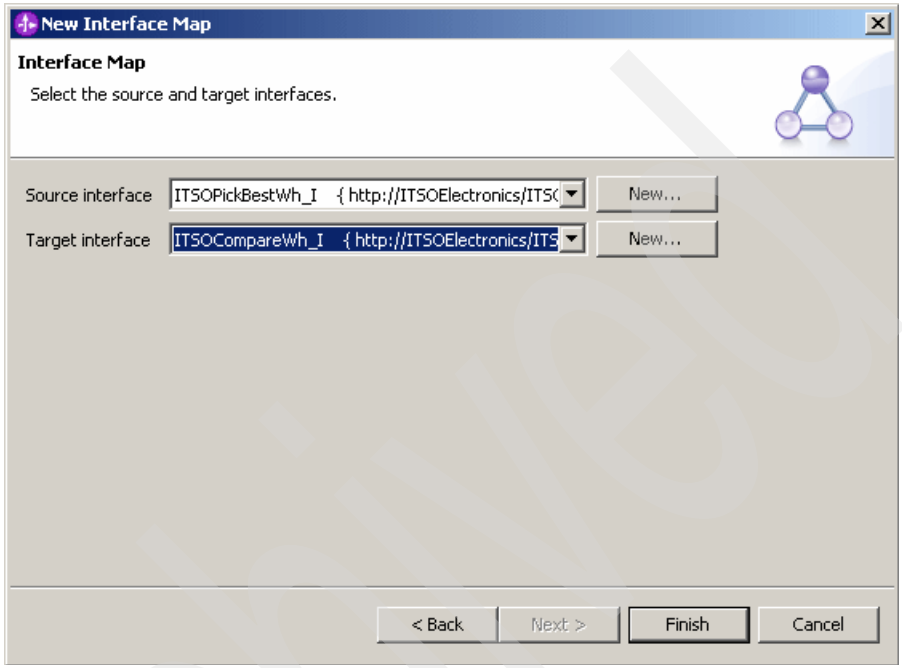


Figure 10-67 New Interface Map wizard: Selecting the source and target interfaces

Archived

4. Drag **pickBestWh** to **compare** to create a connection as shown in Figure 10-68.

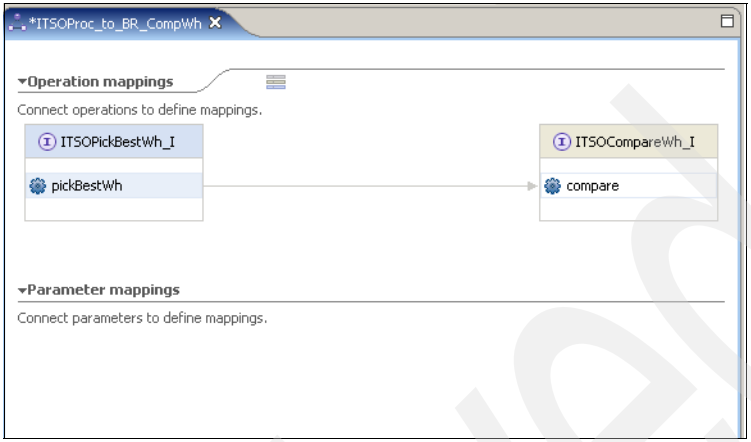


Figure 10-68 Interface map

5. Click the **line** between pickBestWh and compare to select it. This displays the parameter mappings.
6. Drag **whResponses** to **wholesalers**. Then drag **bestWholesaler** to **bestWhResponse** (Figure 10-69).

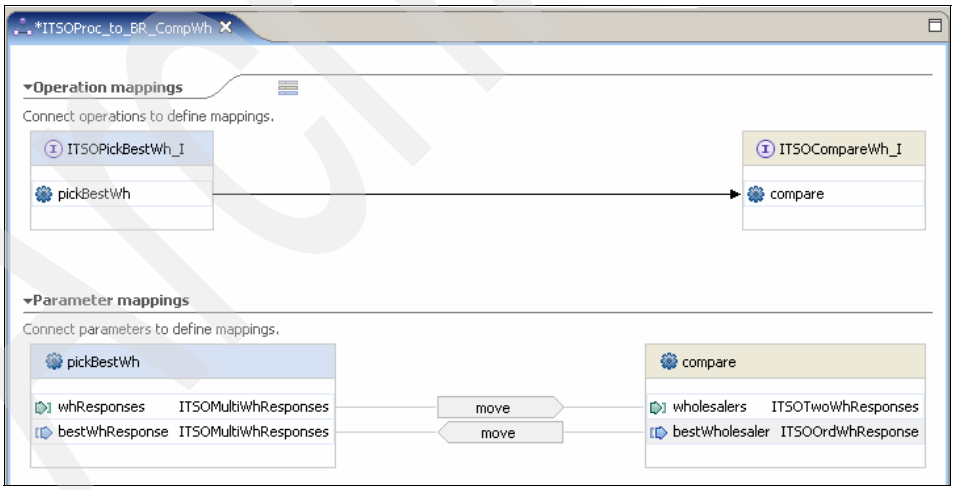


Figure 10-69 Mapping in the interface map

7. Click the first **move** and, on both the **Properties** tab and the **General** tab, change the Parameter Mapping Type to **map**. Click the second **move**, and again on the same tabs, change Parameter Mapping Type to **map**.

Archived

8. Click the **map** between whResponses and wholesalers.
9. Select the **Properties** tab and the **Details** subtab. Under Parameter Mapping, for Business Object Map, select **ITSOProc_to_WhResp** as shown in Figure 10-70.

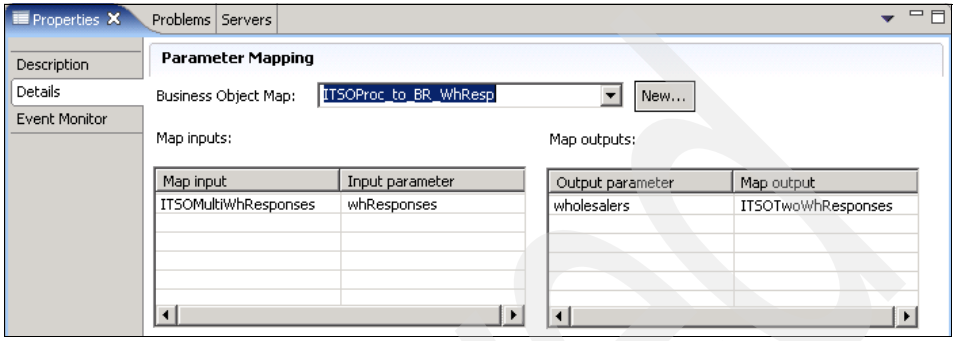


Figure 10-70 Selecting ITSOProc_to_WhResp as the business object map

10. Click the **map** between bestWholesaler and bestWhResponse.
11. Select the **Properties** tab and the **Details** subtab. Under Parameter Mapping, for Business Object Map, select **ITSOBR_to_Proc_WhResp** as shown in Figure 10-71.

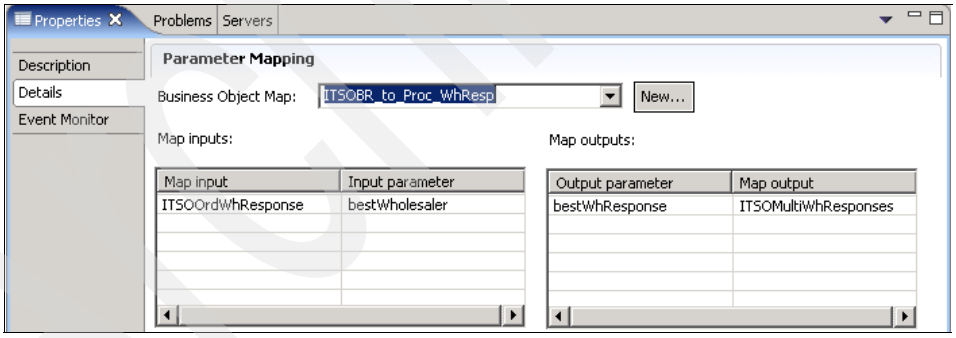


Figure 10-71 Selecting ITSOBR_to_Proc_WhResp as the business object map

Archived

12. Press Ctrl+S to save the map, and then close the Interface Mapping Editor. The artifacts should be now available as shown in Figure 10-72.

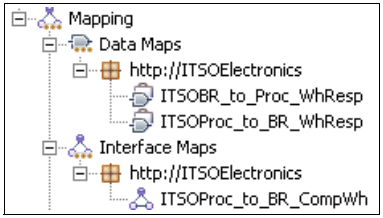


Figure 10-72 Interface map artifacts

Creating the assembly diagram

In this section, we start to create the assembly diagram. Then later we test the available components using the Integration Test Client of WebSphere Integration Developer. We do not finish it in this section, because not all required components are built yet.

1. Open the ITSOElectronics assembly diagram.
2. Drag the following components into the assembly diagram as shown in Figure 10-73:
   a. SerialProcess
   b. ITSO_CheckDate
   c. ITSO_CompareWholesalers
   d. ITSOProc_to_BR_CompWh

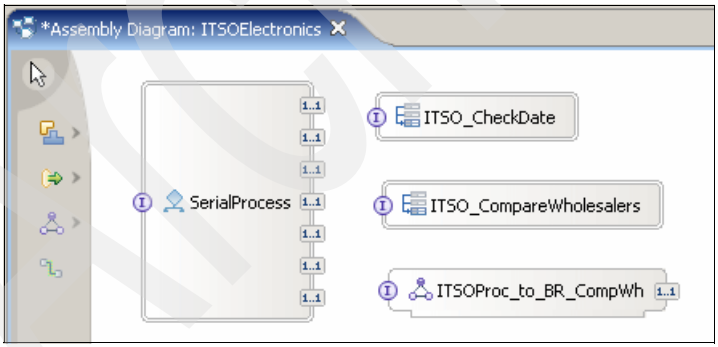


Figure 10-73 Assembly diagram

Archived

3. In the assembly diagram canvas, right-click **SerialProcess** and select **Wire to Existing**. The components are wired together as shown in Figure 10-74.

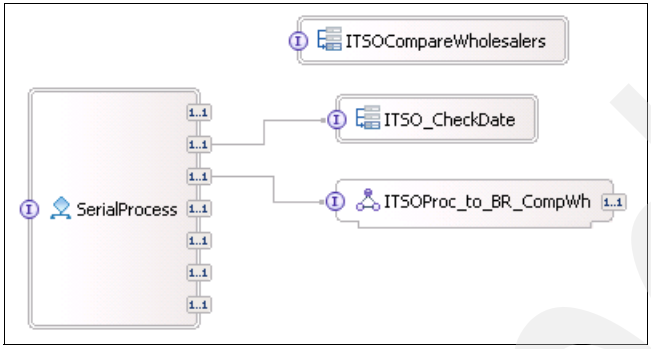


Figure 10-74 Wired SerialProcess

4. In the assembly diagram canvas, right-click **ITSOProc_to_BR_CompWh** and select **Wire to Existing**. The components are wired together as shown in Figure 10-75.

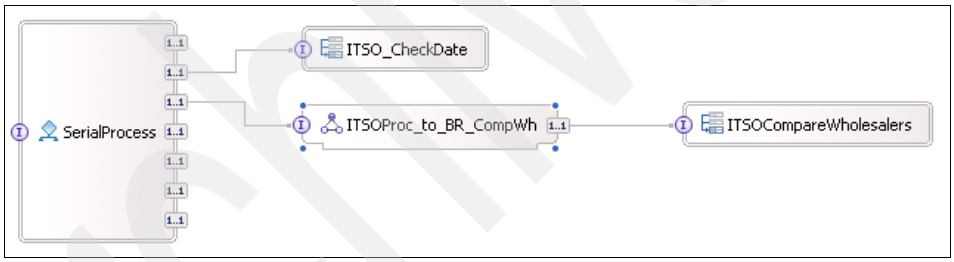


Figure 10-75 Wired ITSOProc_toBR_CompWh

5. Press Ctrl+S to save the assembly diagram.

You can test the Serial Process including the business rules. For details about the testing, see 10.5, “Runtime guidelines” on page 229.

Archived

10.4.4 Creating supporting services for the business process

In this section, we build an order notification service and map the WholesalerA and WholesalerB services to our business process SerialProcess. Figure 10-76 shows the components that we implement in this section.

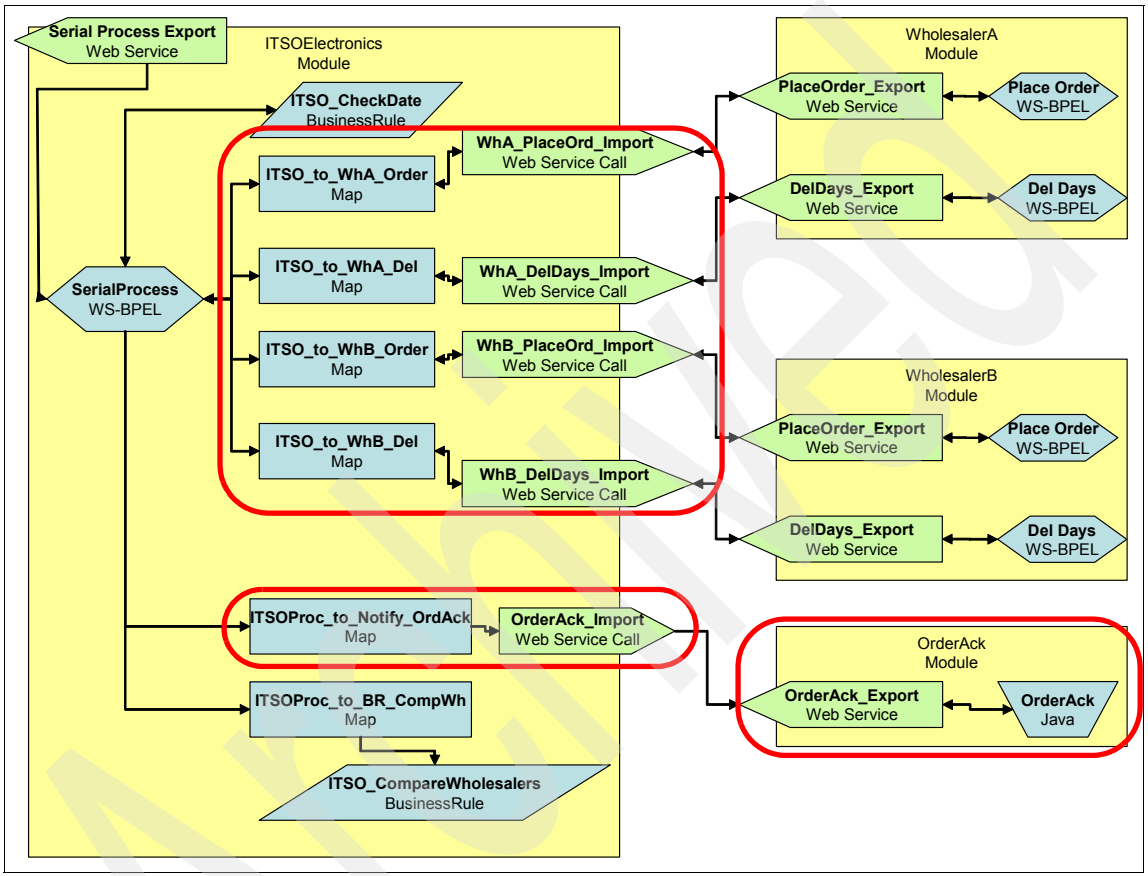


Figure 10-76 Overview of the scenario showing the interlinked components and modules

Archived

Building an order notification service

To build the order notification service:

1. In the ITSOLibrary module, build the business object NotifyDetails. The NotifyDetails business object is used as input to the Notify_I interface.

   a. Use the default namespace `http://ITSOLibrary`.
   b. Specify the attributes shown in Figure 10-77.

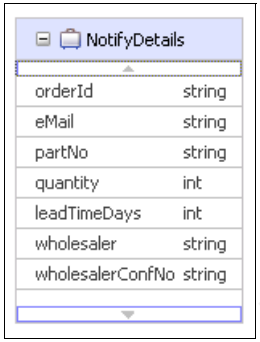


Figure 10-77 The NotifyDetails business object

2. In the ITSOLibrary, build the interface Notify_I. The Notify_I interface is used as the interface to the order notification service.

   a. Use the default namespace `http://ITSOLibrary/Notify_I`.
   b. Use the one-way operation shown in Figure 10-78.

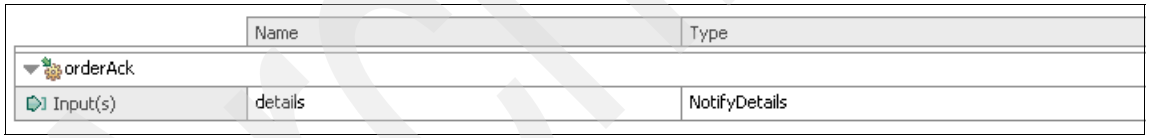


Figure 10-78 The Notify_I interface

Archived

3. In the ITSOElectronics module, build the business object map ITSOProc_to_Notify_Order as shown in Figure 10-79.

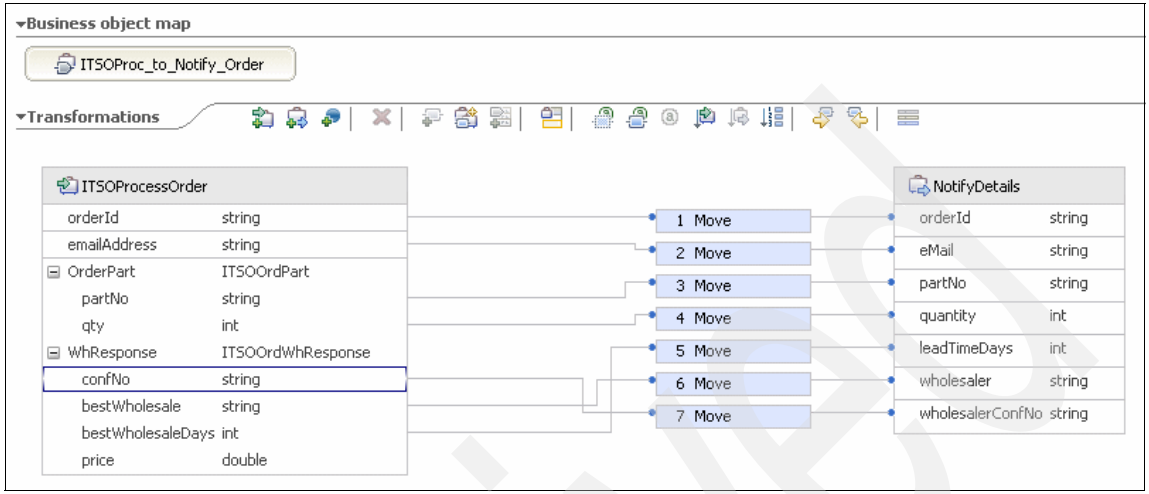


Figure 10-79 Data map ITSOProc_to_Notify_Order

4. Implement the OrderAck service:
   a. Open the OrderAck assembly diagram in the OrderAck module.
   b. Create a new Java Component and name it `OrderAck`.
   c. Add the interface **Notify_I** to the Java component.
   d. Export the Java component as an SCA Binding (right-click the component and select **Export** → **SCA Binding**). An Export component is added to the assembly diagram.
   e. Create an implementation for the Java component. Right-click **OrderAck** and select **Generate Implementation**.
   f. Create a new package called `com.ibm.itso.orderAck`.
   g. Select the new package and click **OK** to generate the default implementation.

Archived

h. Replace the method `public void orderAck (DataObject details)` with the code in Example 10-1.

Example 10-1 OrderAck Java implementation

```
public void orderAck(DataObject details) {
   //This is a simple implementation to display the fields of the
   //OrderNotify Data Object in StdOut

   System.out.println("*****************************************************************");
   System.out.println("Subject:"+details.getString("wholesalerConfNo"));
   System.out.println("Thank you for your order no: "+details.getString("orderId"));
   System.out.println("Wholesaler "+details.getString("wholesaler")+" will deliver "
+details.getInt("quantity")+"items of part number " +details.getString("partNo")+ " in "
+details.getString("leadTimeDays")+" days.");
   System.out.println("If there are changes, we will email you at "
+details.getString("eMail"));
   System.out.println("*****************************************************************");
}
```

i. Save the Java code and close the Java Editor.

j. Close the Java editor. The assembly diagram should look like the example in Figure 10-80.

Figure 10-80 Assembly diagram of OrderAck

k. Press Ctrl+S to save the assembly diagram and close the Assembly Editor.

The Notify Order service is now complete.

Archived

Adding the supporting services to the business process

Assemble the three supporting services WholesalerA, WholesalerB, and OrderAck in the ITSOElectronics assembly diagram so the SerialProcess WS-BPEL business process can call these services. The interfaces to services are not the same as for the SerialProcess. Therefore we have to implement interface maps.

1. In the ITSOElectronics module, build the interface map ITSOProc_to_Notify_OrdAck for the OrderAck interface as shown in Figure 10-90.

   a. For Source interface, specify `ITSOOrdNotify_I`.
   b. For Target interface, specify `Notify_I`.

   Note: Build the interface map in the ITSOElectronics module, not in ITSOLibrary.

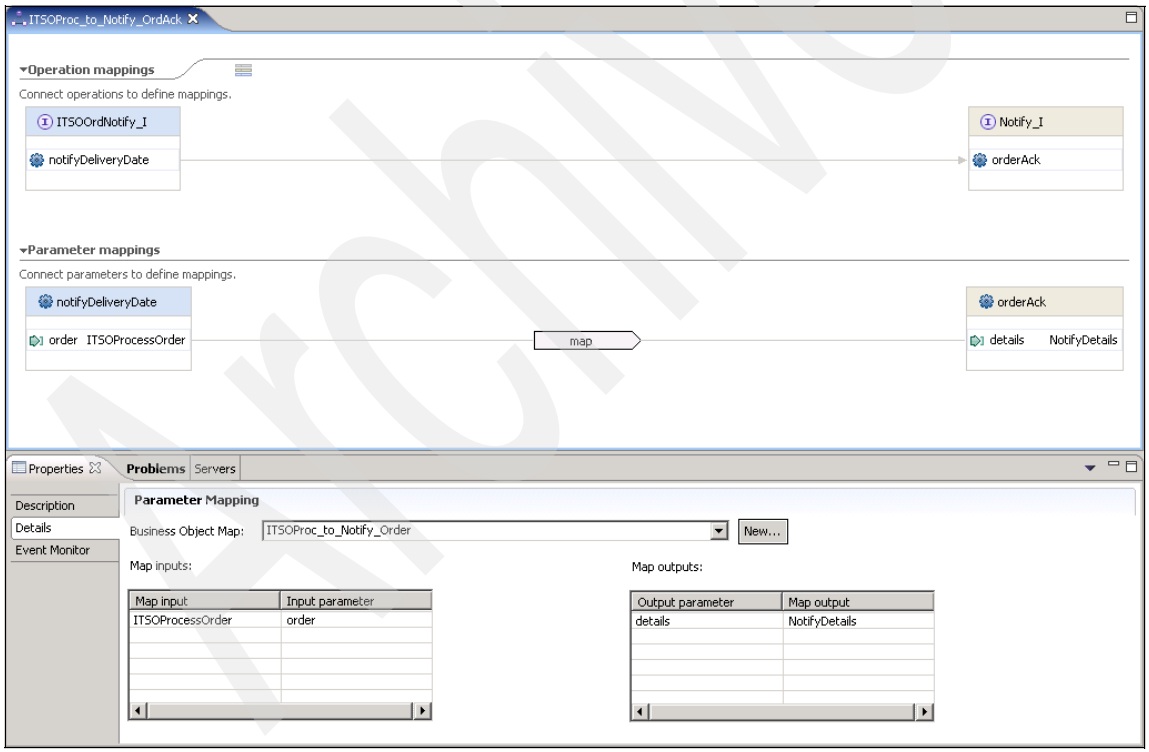


Figure 10-81 Interface map ITSOProc_to_Notify_OrdAck

Archived

2. In the ITSOElectronics module, we now need to build maps from the SerialProcess interfaces to the WholesalerA interfaces. Copy the WholesalerA interfaces and data types into the ITSOLibrary, so that they are visible to the maps that we create in the ITSOElectronics module.

   a. From the WholesalerA module, copy the following data types to ITSOLibrary:

      - WholesalerCannotFulfillOrder
      - WholesalerDeliveryDaysRequest
      - WholesalerDeliveryDaysResponse
      - WholesalerPlaceOrderRequest
      - WholesalerPlaceOrderResponse

   b. From the WholesalerA module, copy the following interfaces to ITSOLibrary:

      - WhDelReqWeb_I
      - WhPlaceOrdWeb_I

   c. Build the business object map ITSOOrder_to_DelDays in the ITSOElectronics module, as shown in Figure 10-82.

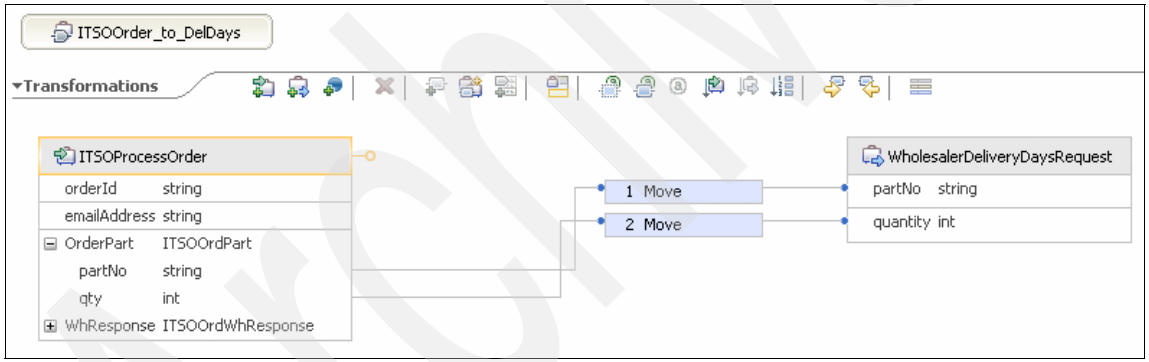


Figure 10-82 Data map ITSOOrder_to_DelDays

Archived

d. Build the business object map ITSODelDays_to_Order in the ITSOElectronics module, as shown in Figure 10-83.

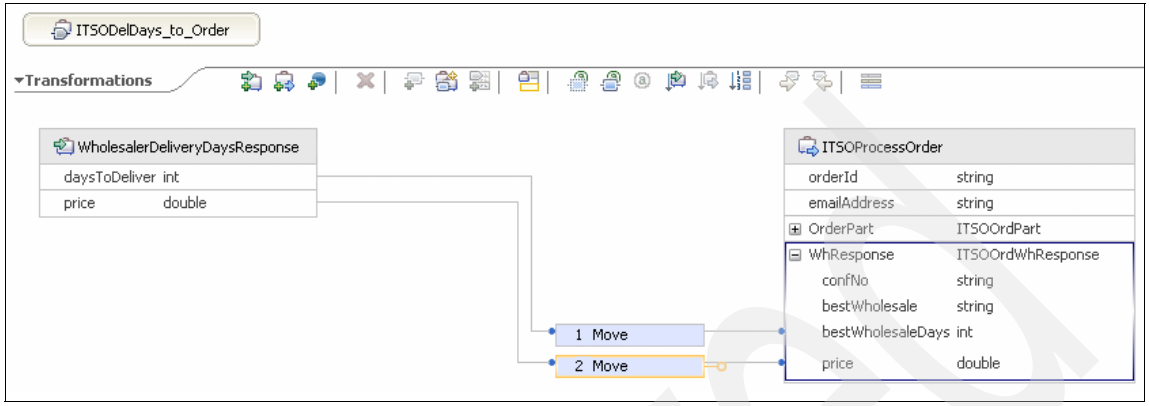


Figure 10-83 Data map ITSODelDays_to_Order

e. Build the business object map ITSOOrder_to_PlaceOrder in the ITSOElectronics module, as shown in Figure 10-84.

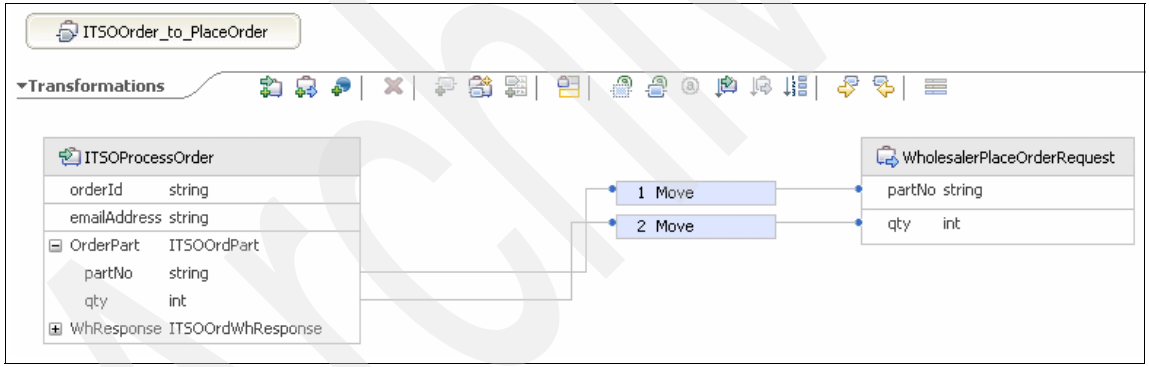


Figure 10-84 Data map ITSOOrder_to_PlaceOrder

Archived

f. Build the business object map ITSOPlaceOrder_to_Order in the ITSOElectronics module, as shown in Figure 10-85.

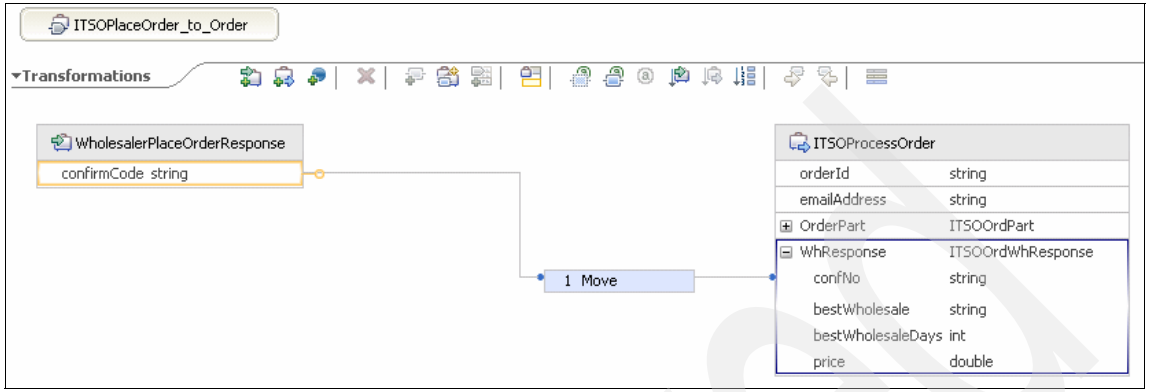


Figure 10-85 Data map ITSOPlaceOrder_to_Order

g. Build the interface map ITSO_to_WhA_Del in the ITSOElectronics module, as shown in Figure 10-86.

- The map that maps getDeliveryDaysWhReq to getDeliveryDaysWholsalersRequest uses the data map ITSOOrder_to_DelDays.
- The map that maps getDeliveryDaysWholsalersResponse to getDeliveryDaysWhRes uses the data map ITSODelDays_to_Order.

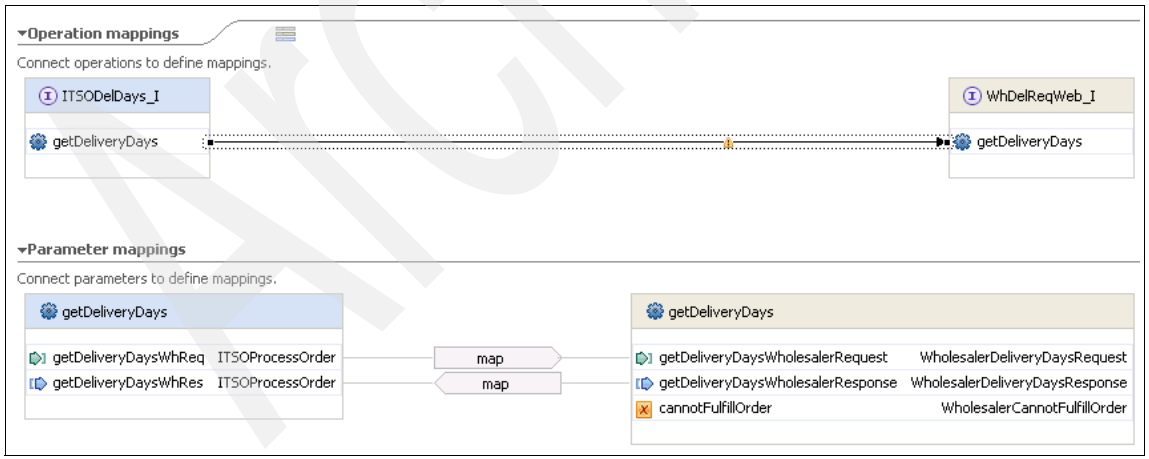


Figure 10-86 Interface map ITSO_to_WhA_Del

Archived

h. Build the interface map ITSO_to_WhA_Order in the ITSOElectronics module, as shown in Figure 10-87.

- The map that maps orderReq to placeOrderRequest uses the data map ITSOOrder_to_PlaceOrder.
- The map that maps placeOrderResponse to orderConf uses the data map ITSOPlaceOrder_to_Order.

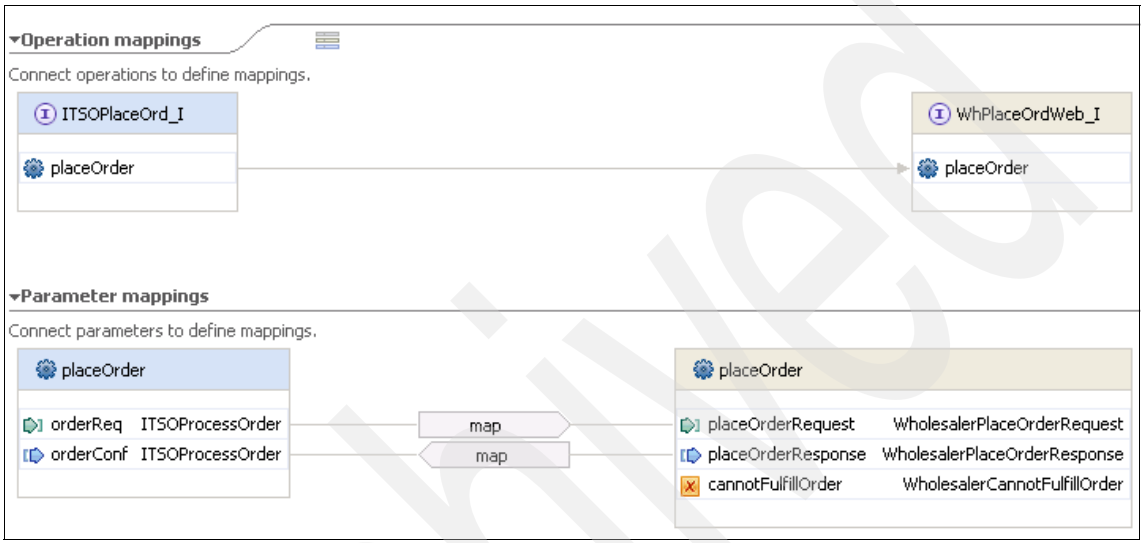


Figure 10-87 Interface map ITSO_to_WhA_Order

Archived

3. In the ITSOElectronics module, we need to build interface maps from the SerialProcess interfaces to the WholesalerB interfaces. We can reuse the existing WholesalerA data types and data maps. We can also reuse the interface maps. However, we decide to keep the interface maps separate, so we are more flexible in replacing the interfaces of the wholesalers.

   a. Build the interface map ITSO_to_WhB_Del in the ITSOElectronics module, as shown in Figure 10-88.

      - The map that maps getDeliveryDaysWhReq to getDeliveryDaysWholsalersRequest uses the data map ITSOOrder_to_DelDays.
      - The map that maps getDeliveryDaysWholsalersResponse to getDeliveryDaysWhRes uses the data map ITSODelDays_to_Order.

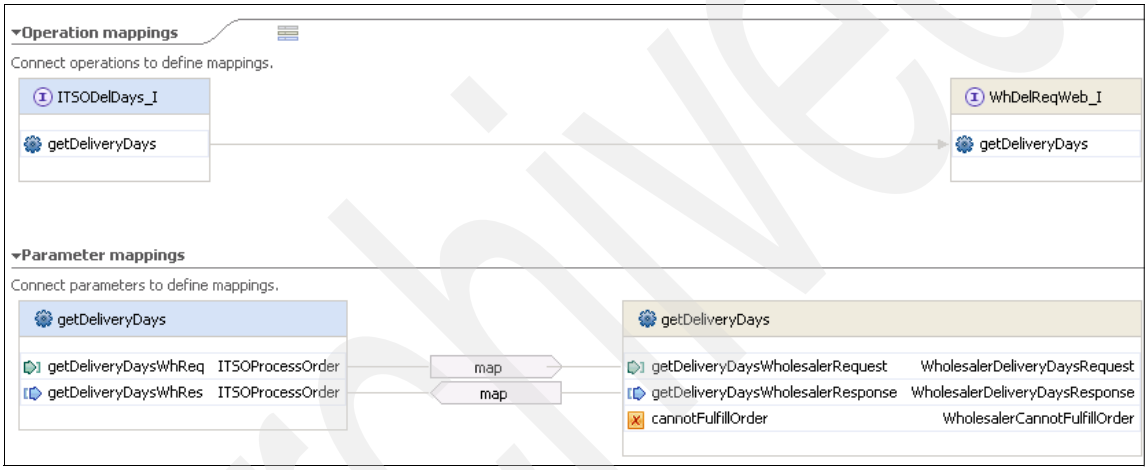


Figure 10-88 Interface map ITSO_to_WhB_Del

Archived

b. Build the interface map ITSO_to_WhB_Order in the ITSOElectronics module, as shown in Figure 10-89.

- The map that maps orderReq to placeOrderRequest uses the data map ITSOOrder_to_PlaceOrder.
- The map that maps placeOrderResponse to orderConf uses the data map ITSOPlaceOrder_to_Order.

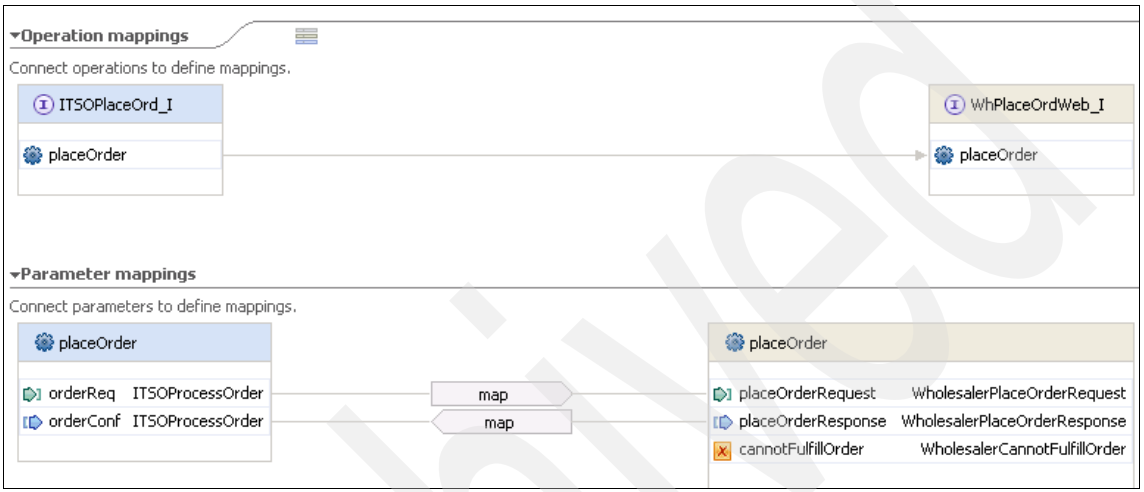


Figure 10-89 Interface map ITSO_to_WhB_Order

4. From Wholesaler A, copy the following Web service ports to the ITSOLibrary. These WSDLs are required for the bindings in ITSOElectronics:
   - WholesalerA_GetDeliveryDateExport_WhDelReqWeb_IHttpPort
   - WholesalerA_PlaceOrderExport_WhPlaceOrdWeb_IHttpPort
5. From Wholesaler B, copy the following Web service ports to the ITSOLibrary:
   - WhBExpDelv_WhDelReqWeb_IHttpPort
   - WhBExpPlaceOrd_WhPlaceOrdWeb_IHttpPort
6. Open the assembly diagram of the ITSOElectronics module.
7. Drag the following interface maps into the assembly diagram:
   - ITSO_to_WhA_Del
   - ITSO_to_WhA_Order
   - ITSO_to_WhB_Del
   - ITSO_to_WhB_Order
   - ITSOProc_to_Notify_OrdAck
8. Right-click **SerialProcess** and select **Wire to existing**.

Archived

9. In the Advanced Wiring window, complete the following actions:
   a. Click **WhADeliveryPartner** and select **ITSO_to_WhA_Del**.
   b. Click **WhAOrderPartner** and select **ITSO_to_WhA_Order**.
   c. Click **WhBDeliveryPartner** and select **ITSO_to_WhB_Del**.
   d. Click **WhBOrderPartner** and select **ITSO_to_WhB_Order**.
   e. Click **OK**. The assembly diagram should be connected as shown in Figure 10-90.

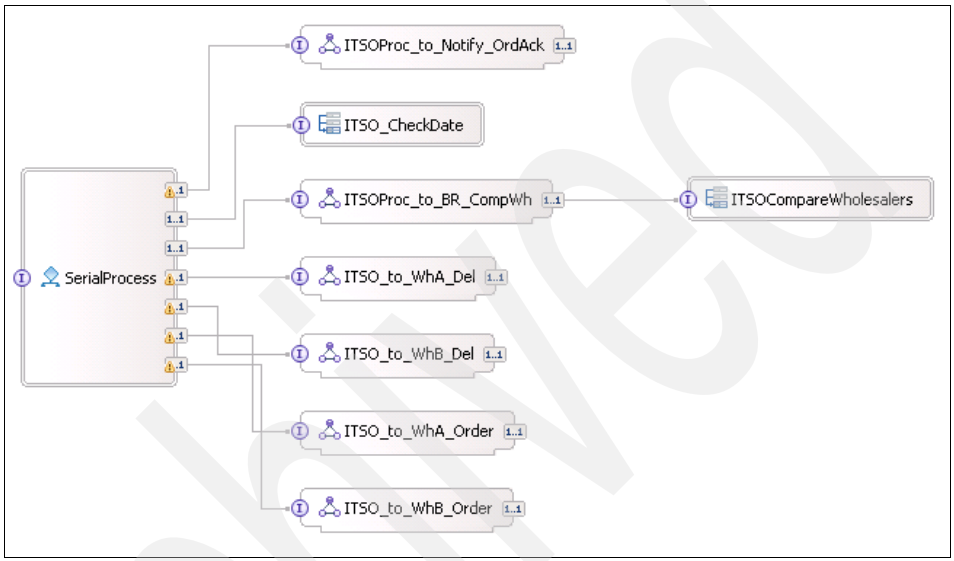


Figure 10-90 Assembly diagram with interface maps

10. Right-click **ITSO_to_WhA_Order** and select **Wire References to New → Imports**. This should connect ITSO_to_WhA_Order with a new import component called *WhPlaceOrdWeb_I*.
11. Right-click the **WhPlaceOrdWeb_I** import component and select **Generate Binding → Web Service Binding**.
12. Click the **Properties** tab and then the **Binding** subtab. Click **Browse...** to assign a Web service to this import component.
13. In the Select a WSDL file with binding/service window, from the Choose a file box, select **ITSOLibrary → WholesalerA_PlaceOrderExport_WhPlaceOrdWeb_IHttp_Service.wsdl**. Then click **OK**.

Archived

Figure 10-91 shows how the Binding page looks now.

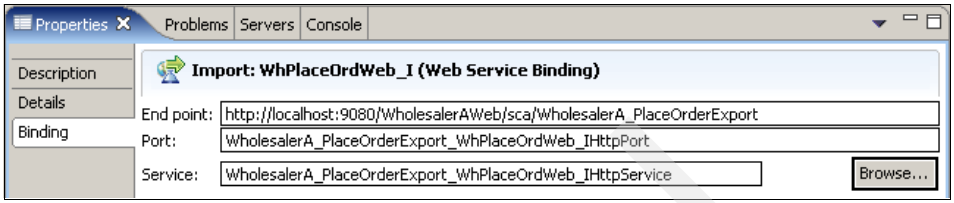


Figure 10-91 Defining the binding

> **Note:** By default, the endpoint for a Web services binding uses a port of 9080. You need to align this port number with the port number assigned to WC_defaulthost of your WebSphere Process Server test server profile. To check this value, open the administrative console for the test server, and click **Servers** → **Application Servers** → **server1** and expand **Ports**.
>
> For example, if WC_defaulthost is set to port 9081, the Endpoint for this Web service binding needs to be:
>
> `http://localhost:9081/WholeslerAWeb/sca/WholesalerA_PlaceOrderExport`
>
> Remember to apply this port number to all Web service endpoints defined in your assembly diagram.

14. In the same way as explained in the previous steps, add bindings for the components listed in Table 10-7.

Table 10-7 Components for which to add bindings

Map	Import	Port	WSDL file
ITSO_to_WhA_Del	WhDelReq Web_I	WholesalerA_GetDeliveryDate Export_WhDelReqWeb_IHttpPort	ITSOLibrary → WholesalerA_ GetDeliveryDateExport_WhDel ReqWeb_IHttp_Service.wsdl
ITSO_to_WhB_Order	WhPlaceOrd Web_I1	WhBExpPlaceOrd_WhPlaceOrd Web_IHttpPort	ITSOLibrary → WhB ExpPlaceOrd_WhPlaceOrd Web_IHttp_Service.wsdl
ITSO_to_WhB_Del	WhDelReq Web_I1	WhBExpDelv_WhDelReq Web_IHttpPort	ITSOLibrary → WhB ExpDelv_WhDelReq Web_IHttp_Service.wsdl

> **Note:** Use the WSDL binding files in the ITSOLibrary folder. Do not use the WSDL files in the WholesalerA or WholesalerB folders.

15. Add an SCA binding for the ITSOProc_to_Notify_OrdAck component:

   a. Right-click the **ITSOProc _to_Notify_OrdAck** component and select **Wire References to New → Import**.

   b. Right-click the new import component **Notify_I** and select **Generate Binding → SCA Binding**.

   c. Click the **Properties** tab and the **Binding** subtab. Then click **Browse**.

   d. In the SCA Export Selection panel (Figure 10-92), for Export name, select **OrderAckExport**.

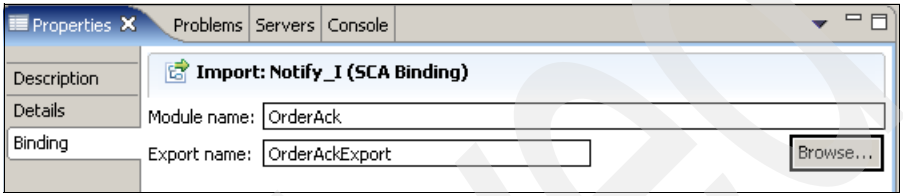


Figure 10-92 SCA Export Selection panel

The final assembly diagram should look like the example in Figure 10-93.

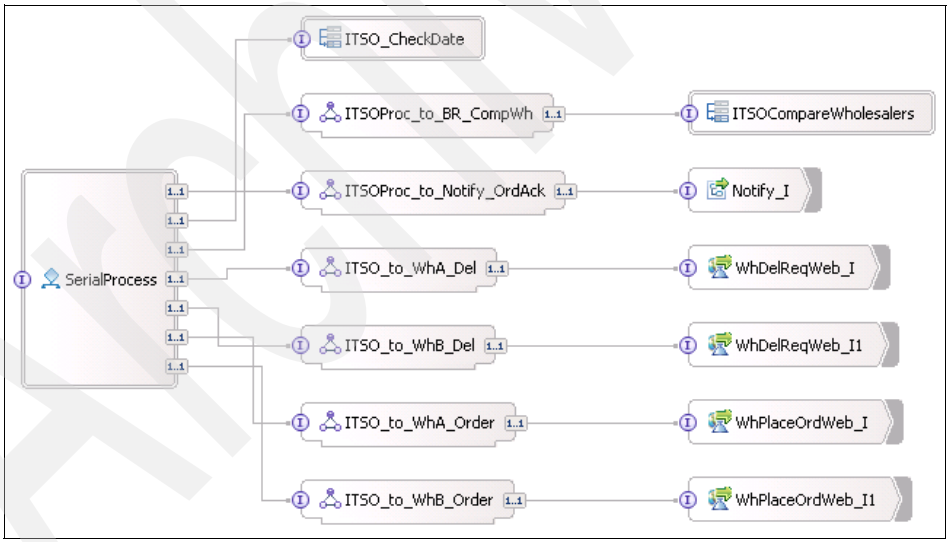


Figure 10-93 Completed assembly diagram

16. Press Ctrl+S to save the assembly diagram.

10.5 Runtime guidelines

In this section, we explain how to deploy authored module projects to the WebSphere Process Server Test Environment. This process involves starting the server, using the wizards to publish, and invoking our solution using the Integration Test Client. We use the Integration Test Client to test the business process. The Integration Test Client is able to test any SCA component, regardless of whether it is implemented. We can choose the component, interface and operation to invoke.

The following steps explain how to test the entire process. If you encounter a problem, narrow the problem down by testing each component individually.

1. Add the projects that you created to the test environment:
   a. In the Servers view, right-click **WebSphere Process Server v6.0** and click **Start**. Wait for the server to have a status of *Started* before continuing.
   b. In the Servers view, right-click **WebSphere Process Server v6.0** and click **Add and remove projects**.
   c. In the Add and Remove Project window (Figure 10-94), click **Add All** to add all projects to the test environment server. Then click **Finish**. The projects are now deployed to the server.

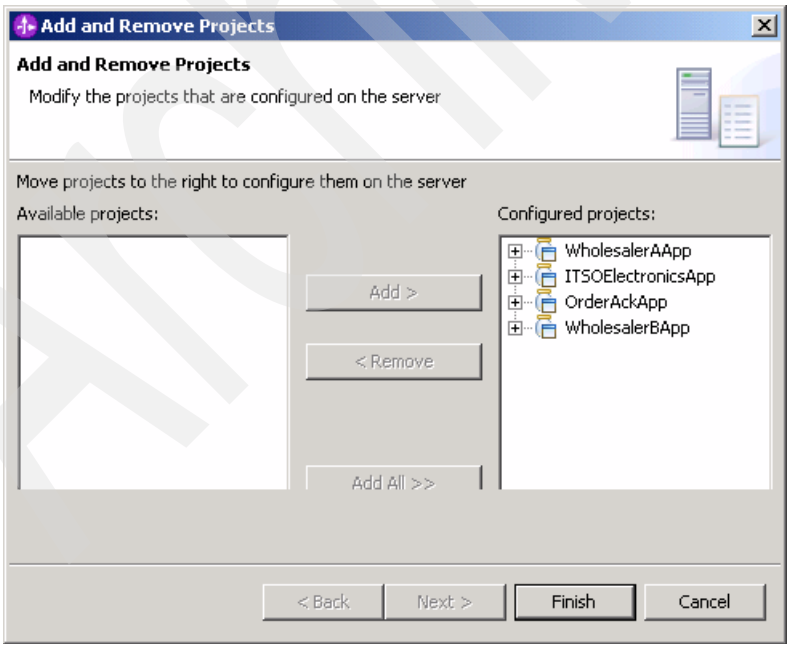


Figure 10-94 Add and Remove Projects window

2. Test the SerialProcess business process using the following steps:
   a. Right-click the **ITSOElectronics** module and **select Test** → **Test Module**. The Integration Test Client opens.
   b. Under Detailed Properties, specify the following details as shown in Figure 10-95:
      i. Set Component to **SerialProcess** to indicate that we want to test the WS-BPEL business process. SerialProcess has only a single interface and operation, so it is automatically selected.
      ii. Under Initial request, enter the values into the parameters as shown in Figure 10-95. The important parameters are the partNo (which we set to `part1`) and qty (which we set to `1`), which determine the behavior of the business process. This test should cause WholesalerA to be selected as the preferred wholesaler.
      iii. Click the **Continue** button.

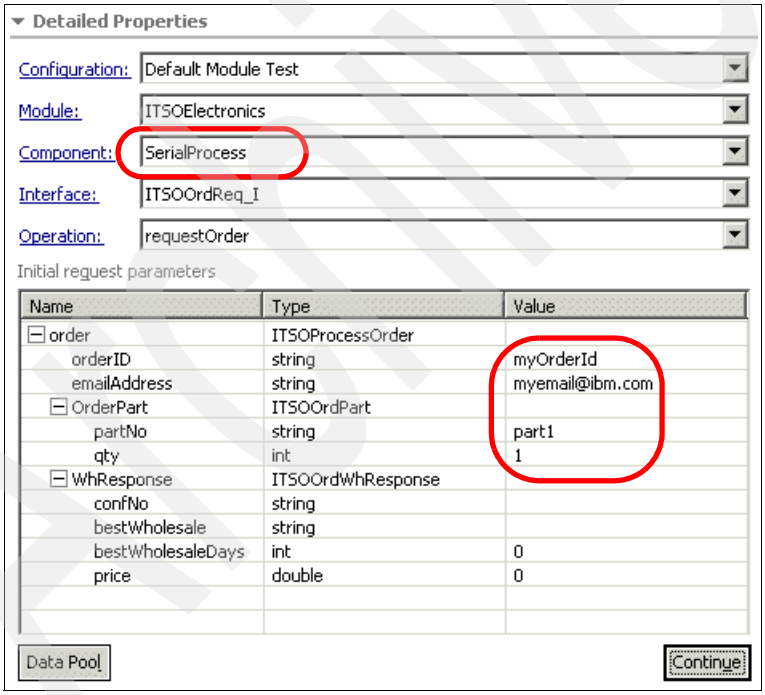


Figure 10-95 Parameters for invocation of the business process

c. The Deployment Location window opens (Figure 10-96), in which you choose the run time to execute the test against. In our case, only one server is defined, so it is selected by default. Click **Finish**.

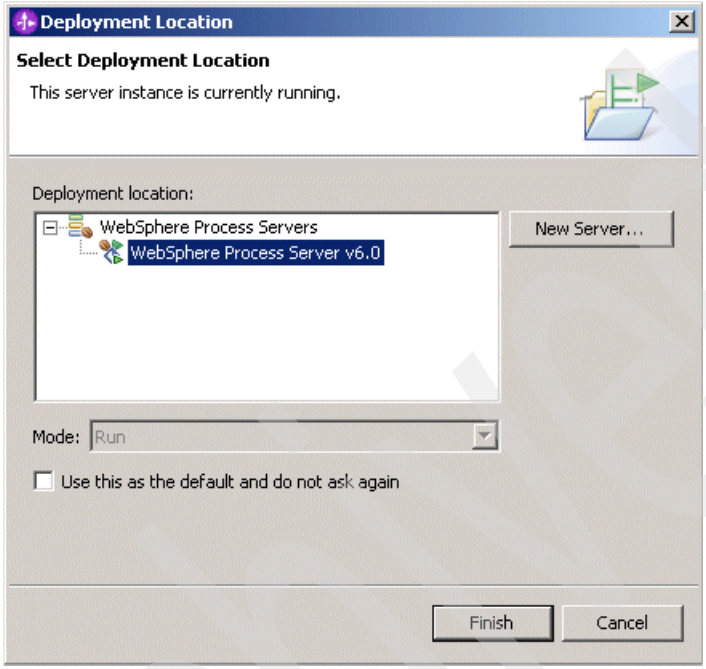


Figure 10-96 Deployment Location window

Archived

Figure 10-97 shows how the editor should look, with WholesalerA chosen, after the test is complete.

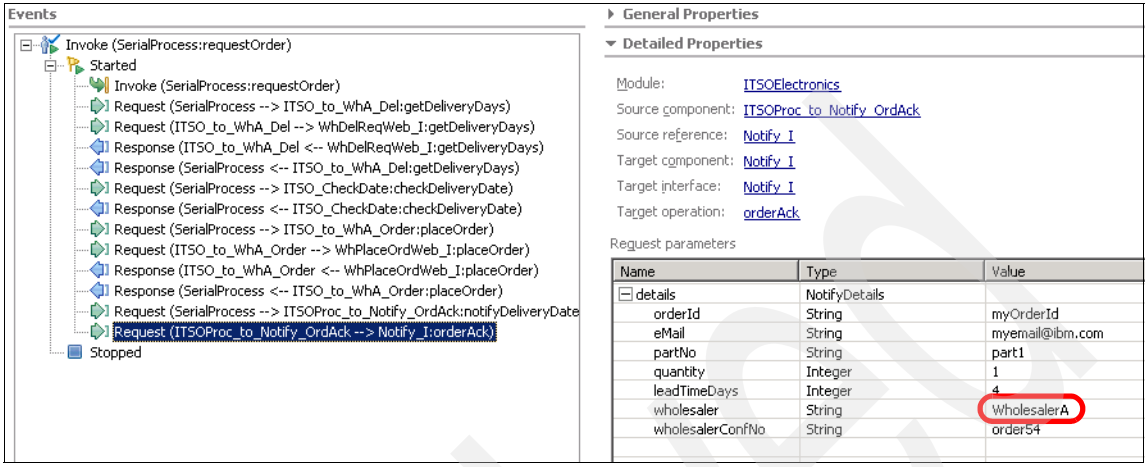


Figure 10-97 Complete test with WholesalerA as deliverer

If you click the various events in the test client, you can see the data that flowed over the wire between the components. When debugging, this allows us to trace the flow of information and see where components could be returning the wrong data. For example, click the final Request event to see the message sent to the notification service, showing that WholesalerA was selected.

It may help to refer to Figure 10-2 on page 147 to remind yourself of the business logic flow when examining the responses in the test client.

Notice the order acknowledgement that is created by the OrderAck module in the Console view. It confirms that the notification service was invoked and that an order was placed with WholesalerA (Example 10-2).

Example 10-2 Output in the Console view

```
0000008c SystemOut     O ****************************************************************
0000008c SystemOut     O Subject:order54
0000008c SystemOut     O Thank you for your order no: myOrderId
0000008c SystemOut     O Wholesaler WholesalerA will deliver 1items of part number part1 in 4
days.
0000008c SystemOut     O If there are changes, we will email you at myemail@ibm.com
0000008c SystemOut     O ****************************************************************
```

3. Test the case so that WholesalerB is chosen. In the Integration Test Client, enter the values shown in Figure 10-98 and click **Continue**. We created our business logic so that, if you specify a partNo of anything other than `part1`, WholesalerB is used.

Name	Type	Value
order	ITSOProcessOrder	
orderID	string	myOrderId
emailAddress	string	myemail@ibm.com
OrderPart	ITSOOrdPart	
partNo	string	part2
qty	int	1
WhResponse	ITSOOrdWhResponse	
confNo	string	
bestWholesale	string	
bestWholesaleDays	int	0
price	double	0

Figure 10-98 Input parameters to select WholesalerB

When the test is complete, Figure 10-99 shows how the client should look, with WholesalerB chosen.

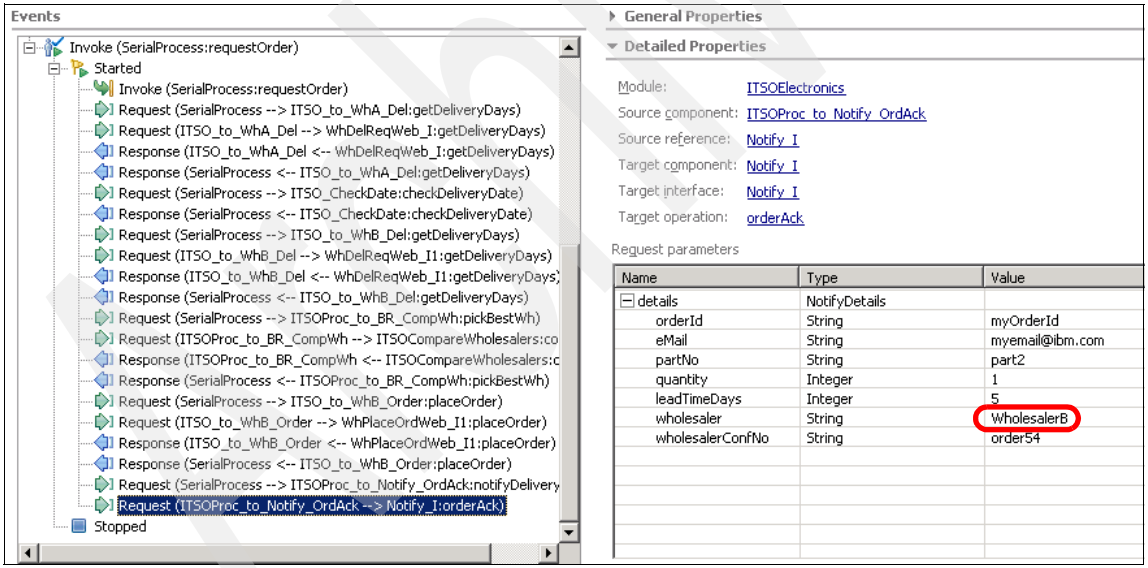


Figure 10-99 The completed test for the process that invoked WholesalerB

Notice the order acknowledgement that is created by the OrderAck module in the Console view. It confirms that the notification service was invoked and that an order was placed with WholesalerA (Example 10-3).

Example 10-3 Output in the Console view

```
0000008c SystemOut     O ************************************************************
0000008c SystemOut     O Subject:order54
0000008c SystemOut     O Thank you for your order no: myOrderId
0000008c SystemOut     O Wholesaler WholesalerB will deliver 1items of part number part2 in 5
days.
0000008c SystemOut     O If there are changes, we will email you at myemail@ibm.com
0000008c SystemOut     O ************************************************************
```

4. Close the Integration Test Client without saving it.

Archived

11

Exposed Serial Process scenario

This chapter details the construction of a process using the Exposed Serial Process pattern. It describes the following items:

- Business scenario: For the ITSO Electronics business problem
- Design guidelines: For building a process that conforms to the Exposed Serial Process pattern
- WebSphere Process Server guidelines: For implementing the process using WebSphere Process Server

This chapter expands on the features used in the Chapter 10, “Serial Process scenario” on page 145. This scenario includes the use of a business state machine and explains how to securely configure Web service calls to communicate with external enterprises.

© Copyright IBM Corp. 2006. All rights reserved.

11.1 Business scenario

Wholesaler A and Wholesaler B have completed their internal merger. After careful consideration, ITSO Electronics's branding consultants have recommended that the merged unit retain the "Wholesaler A" name, rejecting the "Wholesaler B" name. The next big challenge is to incorporate external wholesalers into the order process. By incorporating the external wholesalers, ITSO Electronics can outsource orders to them that cannot be filled internally in a timely manner.

Because ITSO Electronics made a lot of effort to automate their process, they require their partners to provide an application programming interface (API) that the process can interact within real time. They're also not going to implement different protocols for every wholesaler, and therefore expect an API that is based on standards. Effectively, this means they need Web services.

The new process includes a rule that if the internal wholesaler cannot deliver a product in a timely manner, third parties are given the opportunity to do so. More specifically, having realized that it is not practical for Wholesaler A to stock every item they want to sell, management wants to maintain a list of partners who will be given a chance to supply some items. See Figure 11-1.

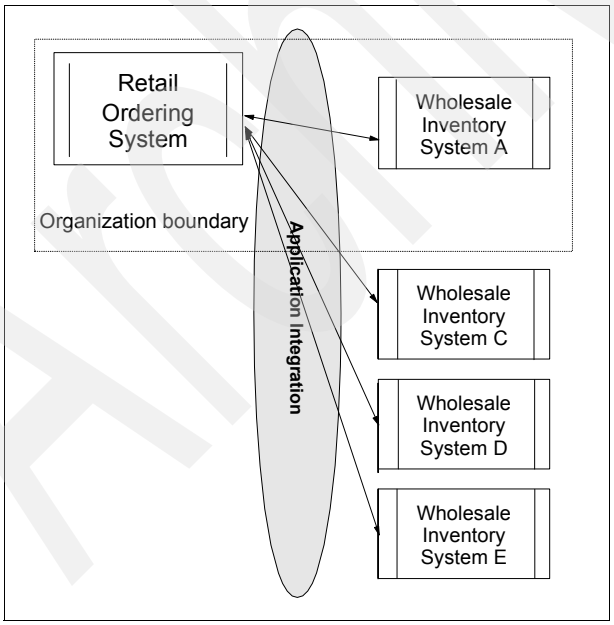


Figure 11-1 ITSO Electronics integration overview after internal merge

Management has been harmed in the past by suppliers whose delivery date estimates have proven to be incorrect. They also note that some suppliers accept an order and later cancel it. Therefore they mandate that an order be considered as filled only when it has shipped and that the rule for choosing a supplier be flexible enough to handle cancelled orders.

Finally, since business partners can and will change regularly, the process must accommodate adding and removing partners from the list of acceptable suppliers.

Figure 11-2 shows how this scenario is implemented at ITSO Electronics. The process includes a business state machine to handle the requirements involving the external suppliers. The business state machine is discussed at length in 11.2.3, “Analysis of the design options” on page 243.

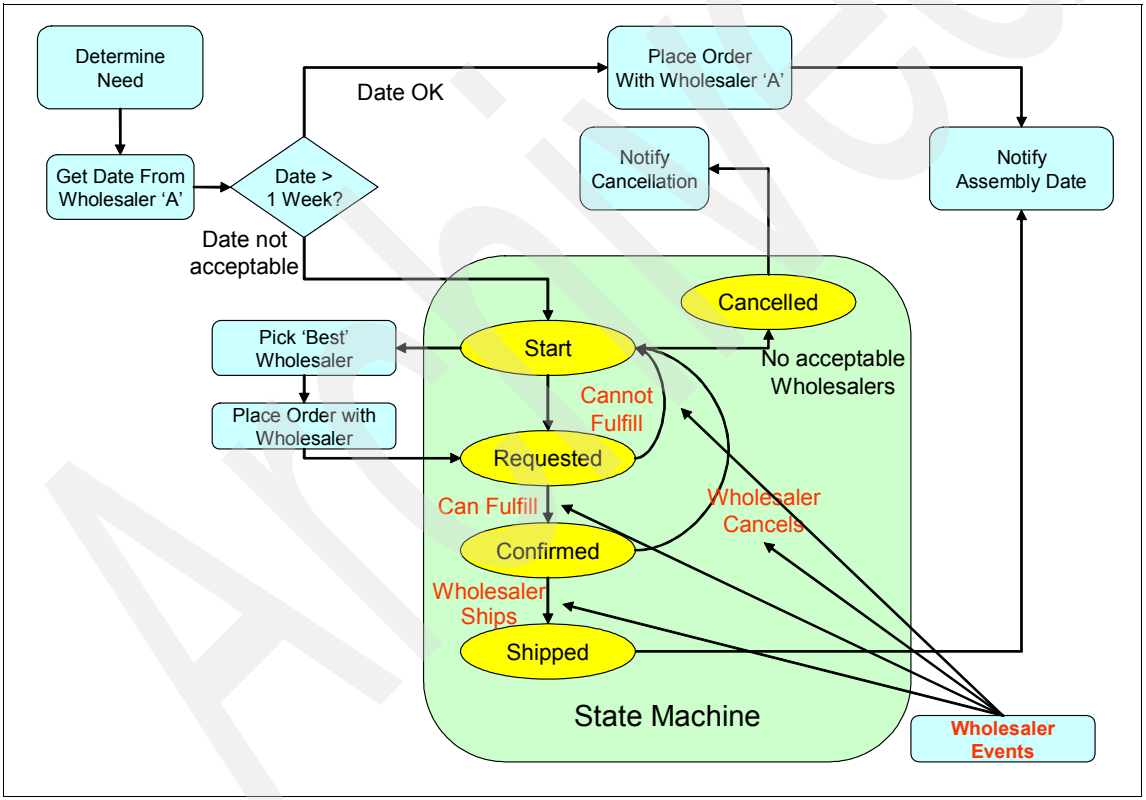


Figure 11-2 ITSO Electronics Exposed Serial Process

11.2 Design guidelines

Since this scenario is based on the one described in Chapter 10, “Serial Process scenario” on page 145, the design considerations from that scenario still apply. This particular process requires additional considerations. Because it accesses the systems of business partners outside of the company firewall, it becomes important to consider security issues. In addition, the requirement to cycle through a set of possible suppliers that lend to fulfillment by a business state machine, so we discuss state and events in this section.

We follow these guidelines:

1. Analyze the business requirements.
2. Select a pattern.
3. Analyze the design options.

11.2.1 Analysis of the business requirements

As with the scenario description, these business requirements follow on from those described for the Serial Process:

- Integrate the retail system with both internal and external wholesale systems, so that retail orders flow electronically to each system.
- Use Web services to communicate with the external systems.
- Establish a process that sends an order to Wholesaler A and determines the expected delivery date.

  The process uses a rule to decide whether that date is acceptable. If it is acceptable, the process immediately sends the order to Wholesaler A. If it is not, it uses a rule to determine which external supplier should be asked to supply the order. If that supplier cannot supply the order or cancels the order, the process is defined to find the next best choice. The process continues until the order has been shipped or until the list of suitable suppliers has been exhausted.
- Ensure that orders are secure.

  They must be sent only to approved suppliers and must not be readable by third parties.
- Provide all relevant information (wholesaler chosen, delivery date, and so forth) to the retail system.
- Ensure that the process runs until the order has been shipped or the list of suppliers has been exhausted.

Archived

- Allow wholesalers to be added and removed dynamically from the list of acceptable suppliers.
- Allow the external suppliers to notify the process of a change in order status, such as shipment or cancellation.

11.2.2 Pattern selection

In this section we use the Patterns for e-business layered asset model to determine the patterns, and ultimately the product mappings, that best fit our business scenario.

Business pattern

Since this scenario is similar to the Serial Process scenario, you expect to follow the same trail through the patterns map. However, that is not the case. The key difference is that ITSO Electronics is collaborating with partners outside its own enterprise. This leads us to the Extended Enterprise business pattern.

The Extended Enterprise business pattern, which is also known as the *Business-to-Business (B2B) pattern* addresses the interactions and collaborations between business processes in separate enterprises. If multiple enterprises are collaborating with each other, and they collaborate programmatically, then this pattern is appropriate. The programmatic nature of the collaboration is a key indicator for this pattern; it does not cover the case where someone directly uses the user interface of a business partner's application. Our scenario meets the prerequisites for using this pattern. There are direct calls between ITSO Electronics's process and those of its partners.

Archived

Application pattern

Now that we have chosen a Business pattern, the next step is to choose from among the Application patterns associated with it. Figure 11-3 shows the Application patterns for the Extended Enterprise business pattern.

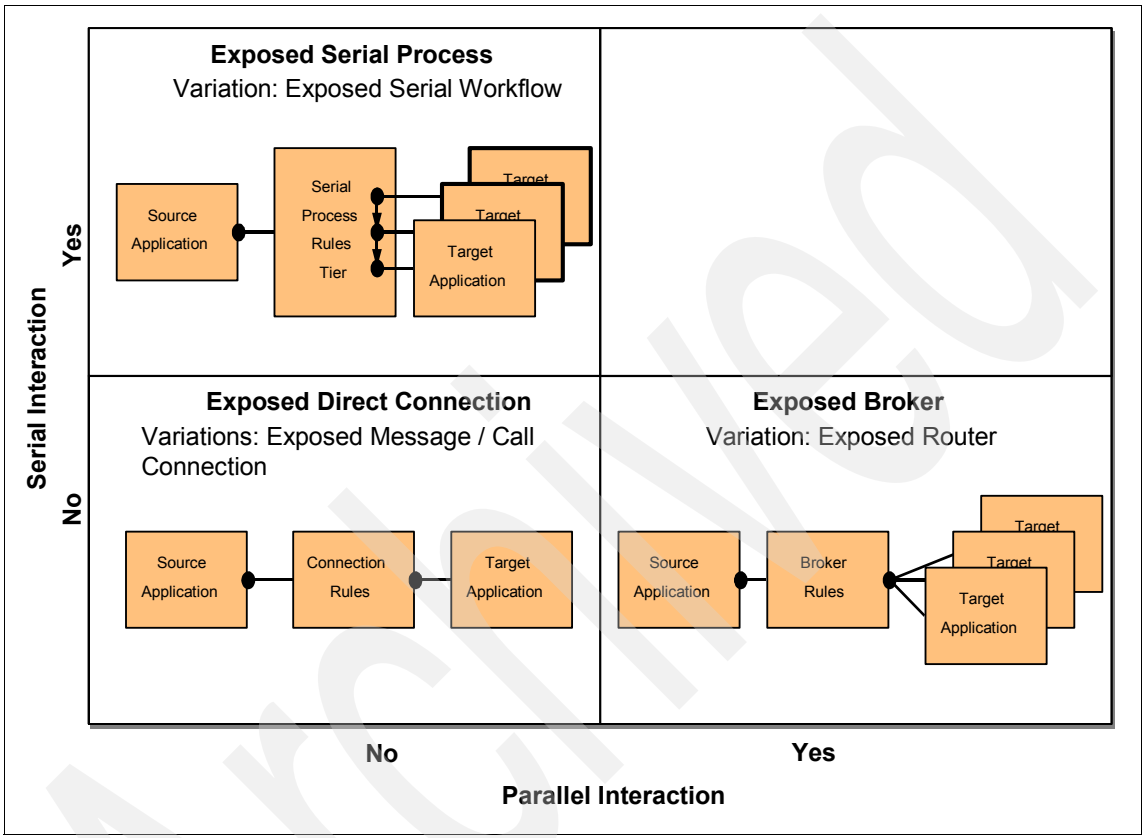


Figure 11-3 Extended Enterprise application patterns

You might note the similarity to the Application Integration patterns shown in Figure 10-4 on page 150. The major difference is that the source application lives in a different enterprise than the target applications. One other difference is that there is no exposed version of the Parallel Process. These patterns are based on the observed behavior of actual enterprises, and the Exposed Parallel Process pattern has not been seen enough to include it as an identified pattern. As process composition technologies mature, we expect to see more widespread use of the Exposed Parallel Process pattern in Extended Enterprise scenarios. This scenario is based on a sequential process and does not include any human interaction, so we choose the Exposed Serial Process pattern.

Runtime pattern

The Runtime pattern for the Exposed Serial Process application pattern has two variants: the generic profile and the service-oriented architecture (SOA) profile. Although our architecture is service-oriented, the SOA profile is intended for use with an enterprise service bus (ESB). We choose the generic profile for this scenario because we do not use an ESB. This is shown in Figure 11-4.

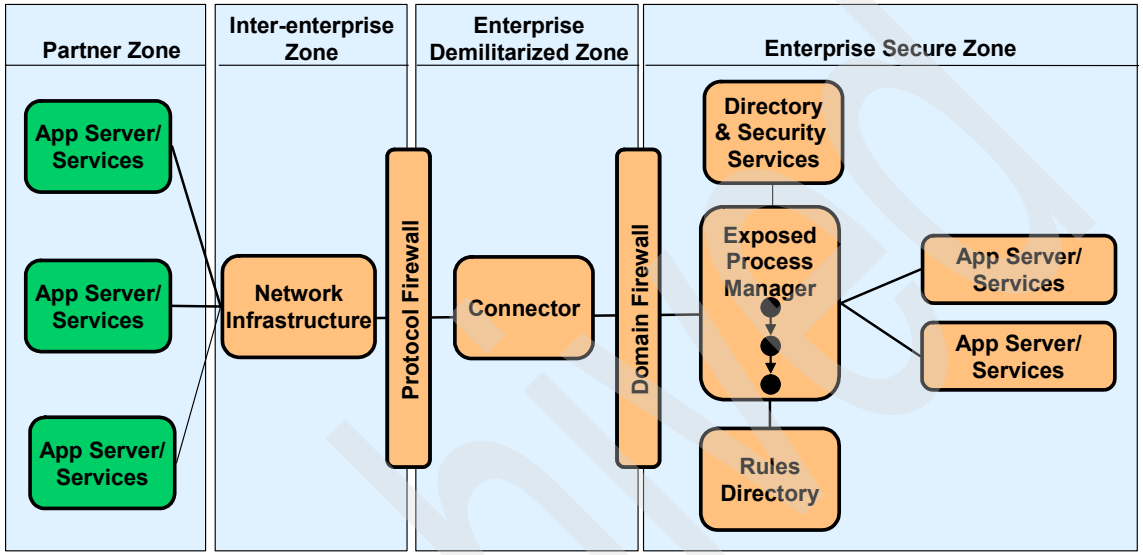


Figure 11-4 Exposed Serial Process runtime pattern: Generic profile

This is a standard topology for business-to-business interaction. Its elements are:

- Enterprise Secure Zone

  This is where ITSO Electronics's internal systems reside, including its process engine and Wholesaler A. These are vital business systems and are not exposed to the outside world.

- Enterprise Demilitarized Zone (DMZ)

  The connector in this zone is configured to expose external APIs that are intended for partner use. It is typically a Web server and might include a security application. Less secure applications and connectors may also reside here.

Archived

- Inter-enterprise Zone

  This is the Internet.

- Partner Zone

  This is where the partner's IT infrastructure lives. From ITSO Electronics's point of view, this is simply the partner's external APIs.

Product mappings

Having selected the generic profile of the Exposed Serial Process runtime pattern, we can apply product mappings to each of the nodes, as shown in Figure 11-5.

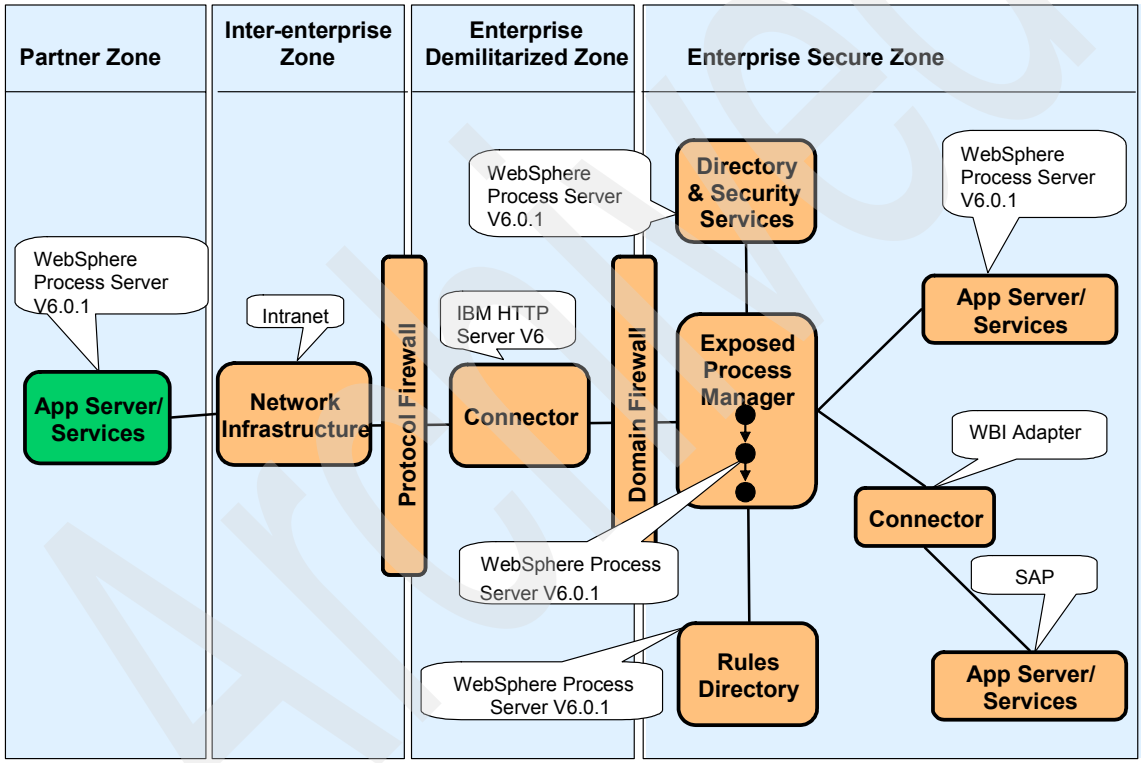


Figure 11-5 Exposed Serial Process: WebSphere Process Server product mapping

As with the Serial Process product mapping implemented in the previous chapter, WebSphere Process Server V6.0.1 is used to implement the Process Manager and the majority of the App Server/Services nodes. Each of the App Server/Services nodes running in WebSphere Process Server is Web services based. WebSphere Integration Developer V6.0.1 is the development tool used to create the artifacts for WebSphere Process Server.

In addition to this, we have a target application implemented in SAP. The Process Manager communicates to this target using a connector implemented by a WebSphere Business Integration Adapter. Note that we *emulate* an SAP system in this chapter, rather than configure and use a real one.

An application service at the partner site invokes the automated process instance implemented by the Process Manager node over the Internet using SOAP/HTTPS. The IBM HTTP Server acts as a connector by exposing SOAP/HTTPS to partner organizations while allowing the Process Manager to process standard SOAP over HTTP (SOAP/HTTP) calls.

In the Directory and Security Services node, the service integration bus within WebSphere Process Server is configured to secure all transactions to the external Partner Zone using WS-Security integrity and confidentiality.

11.2.3 Analysis of the design options

The design options that we discussed for the Serial Process are equally valid for the Exposed Serial Process. You can find that discussion in 10.2.4, “Analysis of the design options” on page 153.

Exposing the process to interaction with partners brings security concerns to the forefront. Security becomes an issue both inbound and outbound. The process must be secured, so that only specified events from trusted partners reach it. Since the process will invoke partner services, you must also consider their security needs and how your implementation will support them.

Although not specific to the Exposed Serial Process, the requirements for this scenario point to the use of a business state machine. This means the process will handle events as well. The design options discussed in this section are:

- Security
- Events and state

Securing WebSphere Process Server

Since your business processes will run on WebSphere Process Server, you must first enable security at the server level. WebSphere Process Server security is based on the WebSphere Application Server V6.0 security model. A detailed discussion of WebSphere Application Server V6.0 security is out of scope of this document. To learn more about it, see the WebSphere Application Server Information Center on the Web at:

`http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp`

Each business process that can run on WebSphere Process Server is defined by a *business process template*. You can think of the template as the cookie-cutter

that lets you create copies of business processes with the same definition. The user who wants to initiate a process must have permission to create that process from its template. If you want to control who can start and stop your process, you do that by controlling access to its template. Again, the details involved are out of scope for this document; refer to the WebSphere Process Server Information Center for details:

http://www.ibm.com/software/integration/wps

Securing processes and services

From the point of view of the Service Component Architecture (SCA), processes and services are all components. Depending on your needs, WebSphere Process Server provides several ways to secure components. Given this scenario's set of requirements, and in particular the stipulation that communication between partners use Web services, here we focus on securing Web services.

Today, securing Web services involves:

- Message level security (WS-Security)
- Transport level security (Secure Sockets Layer (SSL)/Transport Layer Security (TLS))

Message level security: WS-Security

WS-Security is a standard set of SOAP extensions. It handles three aspects of Web services security (commonly known as the *security triad of WS-Security*), namely authentication, integrity, and confidentiality.

You can use these three mechanisms independently, for example to pass a security token. Or you can use them in a tightly integrated manner, for example, signing and encrypting a message and providing a security token hierarchy associated with the keys used for signing and encryption.

Authentication

Authentication ensures that the consumer of a Web service request is really who they claim to be. WS-Security uses *security tokens* for this. WS-Security offers a general-purpose mechanism for associating security tokens with message content. It defines three approved token types:

- Username Token

  This type supports basic authentication. It contains a username and password, which can be passed as plain text or in a digest format.

- X.509 Certificate Token

  This type uses an X.509 digital certificate as user credentials.

- Security Assertion Markup Language (SAML) Token

  This type uses SAML for user credentials. SAML is an Extensible Markup Language (XML)-based security framework for exchanging authentication and authorization information. It is used to support single sign-on (SSO) over multiple applications.

Integrity

WS-Security addresses message integrity (preventing unauthorized message content modification) by using *XML Signature* in conjunction with security tokens to ensure that messages are transmitted without modifications. XML Signature is a standard that defines a schema for digitally signing XML data.

Confidentiality

WS-Security achieves message confidentiality by using XML encryption in conjunction with security tokens to keep portions of a SOAP message confidential.

WS-Security design guidelines

The following set of guidelines is for designing Web services security using WS-Security:

- Use message-level security for Web services using WS-Security when application-to-application security is required. Simple transport security can be handled with SSL.
- Use digital signatures to send important or confidential data between applications over the Internet.
- Both XML digital signature and XML encryption have a performance impact. SSL is a better choice if performance is the primary concern. Performance can also be improved by using XML Signature to sign only the sensitive parts of a SOAP message.
- As noted in the World Wide Web Consortium (W3C) specification of XML encryption, cryptographic vulnerabilities could be introduced when combining digital signatures and encryption over a common XML element. Encrypting digitally signed data, while leaving the digital signature in the clear, may allow plaintext guessing attacks. This vulnerability can be mitigated by using secure hashes in the text being processed.
- Combine digital signatures with timestamps or sequence numbers to ensure message uniqueness and prevent messages from being replayed.

WS-Security alone is not a complete security solution. If security is your paramount concern, investigate other Web service extensions and higher-level application-specific protocols.

Archived

Transport level security for Web services

HTTP, the most widely used Internet communication protocol, is currently also the most popular protocol for Web services. HTTP is inherently insecure because all information is sent in clear text between unauthenticated peers over an insecure network.

To secure HTTP, transport-level security can be applied. SSL and its successor TLS are *cryptographic protocols*, which provide secure communications on the Internet.

Typically, only the server is authenticated. Unlike message-level security, HTTPS encrypts the entire HTTP data packet. There is no option to apply security selectively on certain parts of the message. SSL and TLS provide authentication, data protection, and cryptographic token support for secure HTTP connections. They can be used when there are no intermediaries between the service provider and requester.

Applying the guidelines to ITSO Electronics

The use of security between partners generally results from negotiations between those partners. In this scenario, we secure the supplied services of the business partners in order to illustrate accessing secure services from WebSphere Process Server. One wholesaler uses WS-Security with the X.509 Certificate Token, while another uses SSL over HTTP. Again because the process is an SCA component, the steps needed to secure it are exactly the same as those used to secure the partner services. Given this observation, we chose not to secure the process or the calls to the business state machine.

Events and state

A process-oriented application moves from milestone to milestone to an ultimate goal. Our Serial Process scenario is a typical example. Over its lifetime, the order goes through a succession of steps, and is defined at all times by how far it has gone in the process. But the new elements of this scenario change that.

One of the new requirements for this scenario is that the business process wait in several places for a partner to supply some information before it can proceed. The process, or at least the part of it that deals with external suppliers, is essentially event driven. Instead of waiting for a complete command which may order it to process information, event-driven systems wait (often in a loop) for information to process and then use the event to determine what happens next.

It has been difficult to model event-driven processes in a traditional sequential business process. Web Services Business Process Execution Language (WS-BPEL) includes an event handling construct, but it is intended more for one-time events that interrupt normal process flow.

An event-driven system can be modelled by using a *finite state machine*, a construct known and loved by computer science students ever since Alan Turing used it to build his theory of computation. A finite state machine is a way of modeling the behavior of a business process by focusing on the different states the process can move through in response to incoming events known as *transitions*. The WebSphere Process Server implementation of a finite state machine is a business state machine, an implementation of a business model that moves from one state to another state based on real-time events. It is based on the state machine construct described in Unified Modeling Language (UML) V2.0. Figure 11-6 illustrates the business state machine for this scenario.

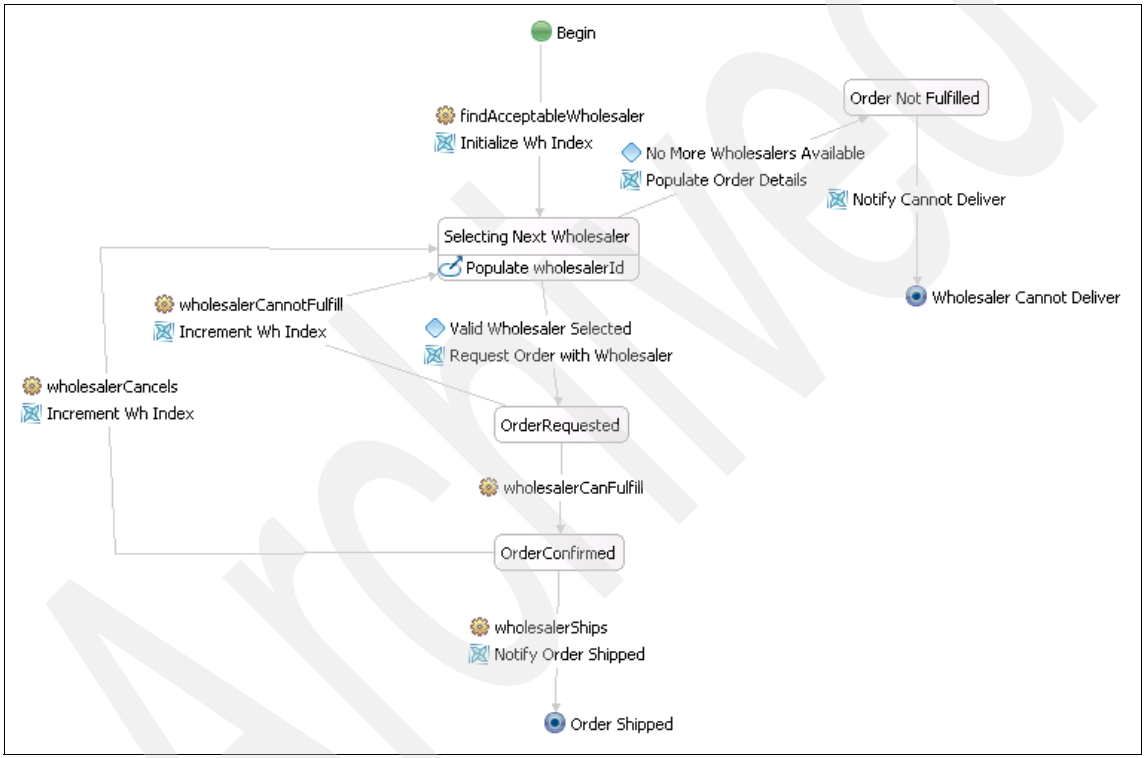


Figure 11-6 The business state machine for this scenario

This business state machine consists of the following components:

- States

  What is a state machine without states? At any given moment, the business state machine is in one particular state. The state defines the events that can be invoked and the transitions that can occur.

Archived

- Events

  An event causes a transition from one state to another. The business state machine, being an SCA component, is represented by an interface. Each event is represented by an SCA operation on that interface. In Figure 11-6, events are indicated by a gear ( ).

- Transitions

  React to an event by moving the business state machine from one state to another. They are indicated by the arrows (⟶) in Figure 11-6.

- Conditions

  Conditions are boolean functions that can block the transition. A transition only occurs if its condition evaluates to *true*. Conditions look like diamonds ( ) in Figure 11-6.

- Actions

  Actions are business logic that executes when an event is accepted and a transition is taking place. They look like propellers ( ) as shown in Figure 11-6.

A few technical details are worth noting. The business state machine is an SCA component like a process or a Web service. It can drive the entire business process, or, as in our example, it can be invoked as an activity of another process. Under the covers, the business state machine is implemented by WS-BPEL. It can be exposed via an SCA to allow assembly of the solution in WebSphere Integration Developer Assembly Editor.

When to use a business state machine

A good rule of thumb for deciding whether you want a traditional process or a business state machine is to consider whether the business process you are implementing is a noun or a verb. A typical process is a verb. For example “place order” describes the Serial Process implementation. A business state machine is a noun that represents a thing, such as a help desk or insurance policy application. Here, a verb, such as “cancel order”, is an operation upon the thing.

The business state machine used for this scenario represents the order as it is passes between ITSO Electronics and potentially multiple wholesalers. For this scenario, it makes sense to treat the subset dealing with wholesalers as a noun. Since any real process is likely to have some elements leading you in each direction, the decision is rarely completely trivial.

You must also consider how event-driven the process is. WS-BPEL processes tend to be relatively sequential with few loops. A situation with multiple re-entry points and reversion to earlier stages of the process tends to favor a business state machine.

Here are some more guidelines:

- Use a WS-BPEL process when:
  - The process is mostly sequential.
  - There is minimal looping and event handling.
  - You need business process features that are only supported in a WS-BPEL process:
    - Compensation
    - Parallel process paths
    - Human tasks
  - You operate on multiple business entities.
  - You require a standards-based implementation.
- Use a business state machine when:
  - The business process focuses on events, rather than on the business activities.
  - The reaction to events depends on the state of the process.
  - The process may revert to prior states.
  - A single business entity is used throughout the process and the object of the process is to manipulate its state.

Applying the guidelines to ITSO Electronics

We chose to implement this scenario as a process and invoke a business state machine to perform the negotiations with external suppliers. The decision to use a business state machine was based on the following factors:

- The negotiations with external suppliers are event driven.
- The process can revert to an earlier state if a supplier cancels.
- A single order object is used throughout the process.
- There is no requirement for WS-BPEL functionality not available in the business state machine.

11.3 Development guidelines

Note: The explanation in this section indicates the use of WebSphere Integration Developer V6.0.1.

This section explains how to build the Exposed Serial Process scenario using WebSphere Integration Developer. Because we use a large number of components, we guide you through the development, using an overview of the whole development and highlighting where individual development units fit within.

Archived

WebSphere Integration Developer and WebSphere Process Server provide extensive testing facilities. We recommend that you perform a unit test on the newly built components, before assembling them into the completed solution. See 11.4, “Runtime guidelines” on page 312, for testing and runtime deployment guidelines. Figure 11-7 shows the scenario overview that we build in this chapter.

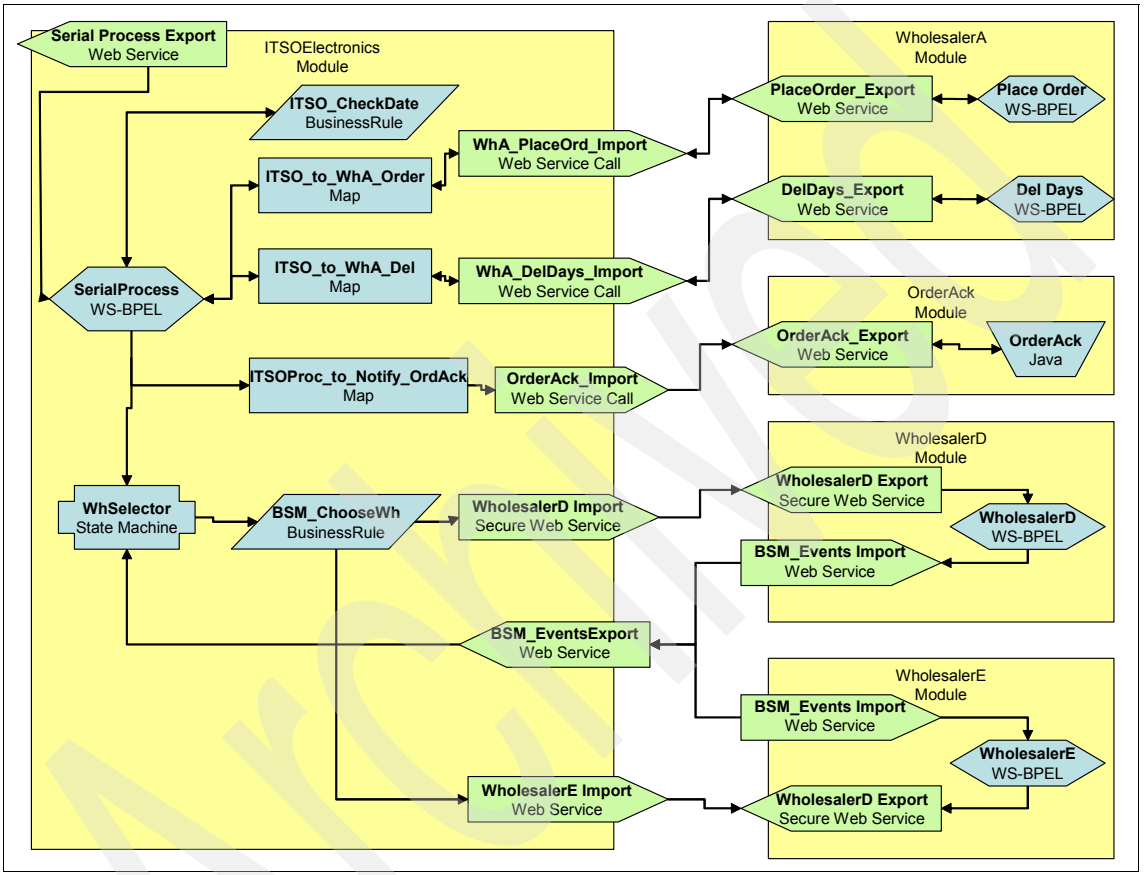


Figure 11-7 Overview of the scenario

Archived

11.3.1 Development plan

In this chapter, we build the components using the following tasks. This is not a hard and fast order of construction, but it allows each new component to reuse components that were previously built.

1. Set up the WebSphere Integration Developer workspace and copy the artifacts to reuse from the Serial Process.
2. Build the business state machine. See 11.3.3, “Creating the business state machine” on page 253.
3. Build the business rule called by the business state machine. See 11.3.4, “Creating the business rule BSMChooseWh” on page 280.
4. Build Wholesalers D and E. See 11.3.6, “Creating the Wholesaler service providers” on page 285.
5. Create an assembly diagram for ITSOElectronics, which links the business rule and OrderAck module using SCA. See 11.3.5, “Creating the ITSOElectronics assembly diagram” on page 284.
6. Use assembly diagrams to link the business state machine and Wholesalers D and E. See 11.3.7, “Linking WholesalerD and WholesalerE to the state machine” on page 298.
7. Build the WS-BPEL process which calls Wholesaler A and the state machine. 11.3.8, “Creating the Exposed Serial business process” on page 302.
8. Update the assembly diagram to link the WS-BPEL process to the state machine and Wholesaler A. See 11.3.9, “Linking the business process in the assembly diagram” on page 308.
9. Update the assembly diagram to link the ITSO_CheckDate business rule into the business process. See 11.3.10, “Adding the ITSO_CheckDate business rule” on page 310.
10. Expose the business process as a Web service. See 11.3.11, “Exposing the business process as a Web service” on page 311.

Archived

11.3.2 Setting up the workspace

We create this scenario in a new workspace and reuse some components of the Serial Process scenario. We reuse the WholesalerA and OrderAck modules from the Serial Process scenario completely, as well as some components from other modules within the Serial Process scenario, as highlighted in Figure 11-8.

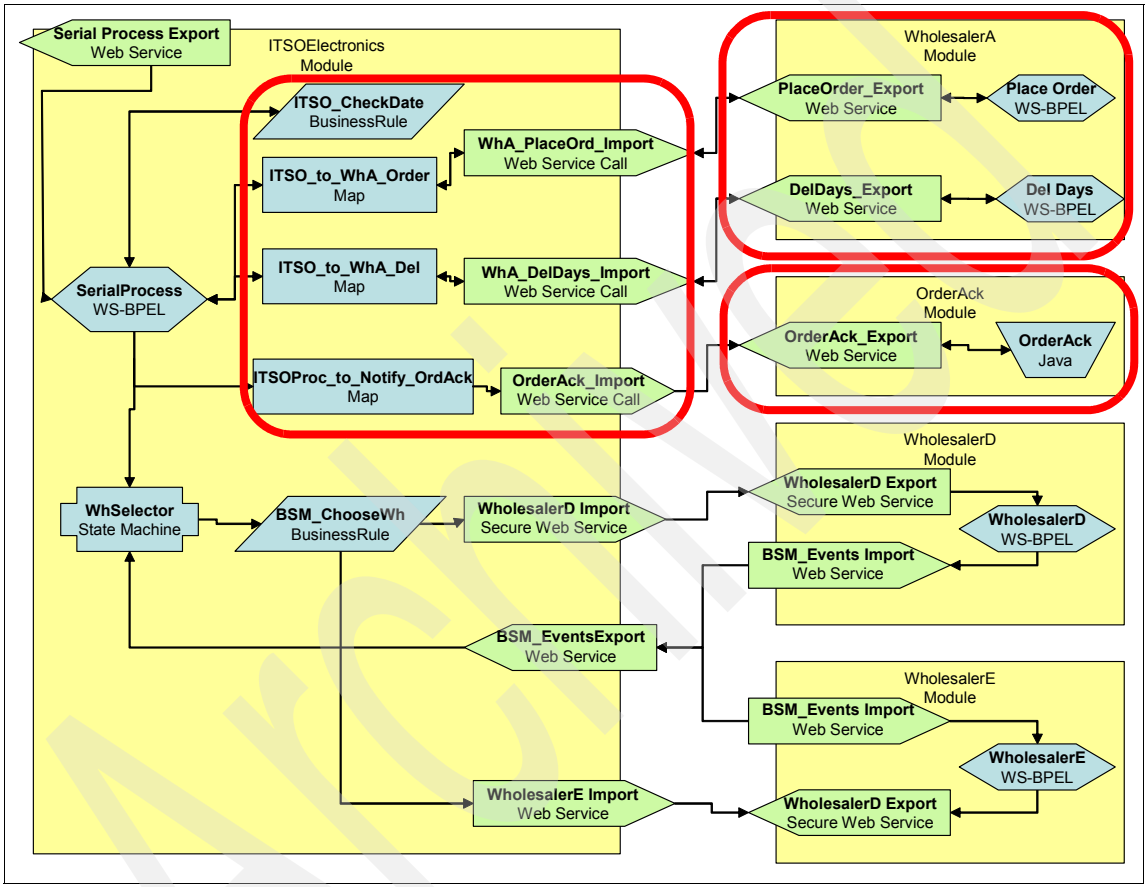


Figure 11-8 Reuse of WholesalerA and OrderAck components

Perform the following steps:

1. Start a new workspace in WebSphere Integration Developer V6.0.1.
2. Import the WholesalerA module into the workspace. You can find this module in the additional material supplied with this IBM Redbook in the file \Wholesalers\WholesalerA\WholesalerA.zip. Select **File** → **Import** → **Project Interchange** to import it.

3. Import the following modules into the workspace using the same method. You can find these modules in the additional material supplied with this book in the \ExposedSerialProcess directory (see Appendix E, “Additional material” on page 689):
   - ITSOLibrary
   - OrderAck
   - ITSOElectronics

   Note: When the ITSOElectronics module is imported, it imports with seven warning messages, which you can ignore.

4. Create following new modules in the workspace, by selecting **File** → **New** → **Project** → **Module**. These are currently empty modules with no dependencies and are populated during this chapter.
   - WholesalerD
   - WholesalerE

   Figure 11-9 shows the Business Integration view in the workspace.

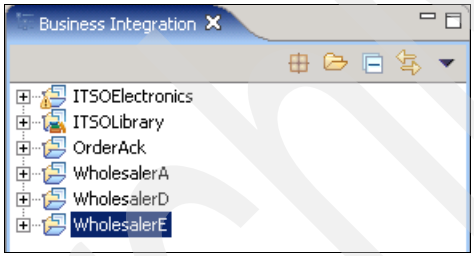


Figure 11-9 Prepared workspace

11.3.3 Creating the business state machine

In this section, we build the business state machine which is used to iterate through several suppliers and hold the state of the order while the suppliers either confirm, cancel, or ship the order.

The flow of the logic through the state machine is summarized as follows:

1. Receive an order request. The state is now *Selecting Next Wholesaler*.
2. Select the first supplier from an index and place an order with that supplier. The state is now *Order Requested*.
3. If the supplier cannot fulfill the order, the state is reset to *Selecting Next Wholesaler*. The state machine then selects the next supplier in the index. If

Archived

no more suppliers are available, the state is set to *Wholesalers Cannot Deliver* and the state machine ends.

4. If the supplier can fulfill the order, the state is set to *Order Confirmed*.
5. If the supplier ships the order, the state is set to *Order Shipped* and the state machine ends.
6. If the supplier cancels the order, the state is reset to *Selecting Next Wholesaler* and the next wholesaler is selected as before.

The state machine in this scenario simply uses an integer (either 1 or 2) to indicate to which supplier the order request is to be sent. We use a business rule to translate the integer into a request to either Wholesaler D or Wholesaler E.

This business state machine fits into the overall scenario as shown in Figure 11-10.

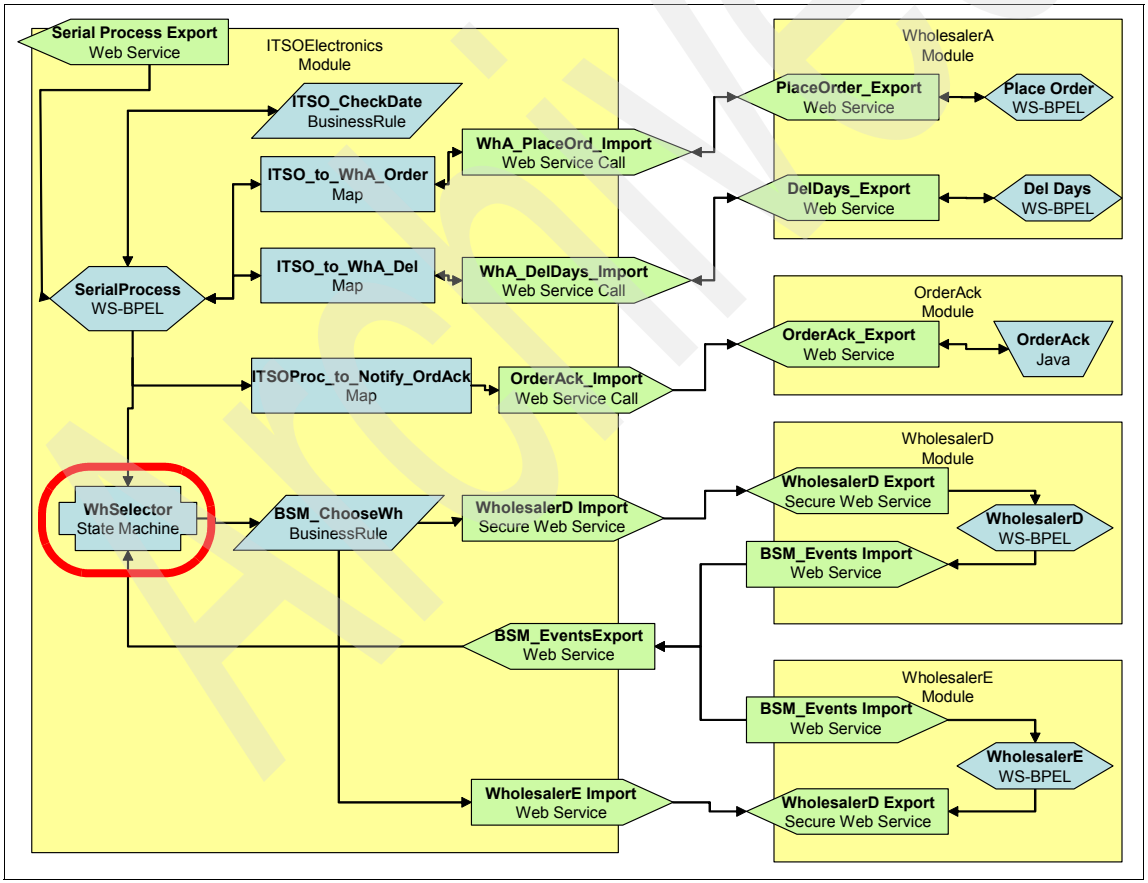


Figure 11-10 Overview of the scenario showing the interlinked components and modules

Creating the business state machine

To create the state machine:

1. Create the interfaces, one inbound and one outbound, that the state machine uses.
2. Create the state machine artefact in the ITSOElectronics module.
3. Add the interfaces and references to the state machine. This is how the state machine calls other components and responds to incoming events.
4. Create the states. These are the states that the machine can be in at any one time.
5. Create the transitions between the states. Transitions indicate how the machine moves from one state to another and governs which states can follow other states.
6. Add actions, operations, and conditions to the transitions. This is what dictates which transitions the machine uses at what time.
7. Add a correlation for the state machine so that the correct instance can be referenced by the incoming events.

Creating the interfaces for the state machine

In this section, we build two interfaces:

- The outbound interface BSMOrderReq_I

  This interface allows the state machine to request an order of a wholesaler.

- The inbound interface BSMEvents_I

  This interface allows the state machine to receive events and take action accordingly. The events are:

  – *findAcceptableWholesaler*: The initial request is sent to the state machine.
  – *wholesalerCanFulfill*: The wholesaler accepts the order.
  – *wholesalerCannotFulfill*: The wholesaler declines the order request.
  – *wholesalerCancels*: The wholesaler cancels the order after accepting it.
  – *wholesalerShips*: The wholesaler ships the order after accepting it.

  Each event corresponds to an operation of the inbound interface.

We also reuse the interface ITSOOrdNotify, which already exists in the ITSOElectronics module. This interface allows the state machine to notify the requester of the final status of the order.

Archived

To build the two new interfaces:

1. In the ITSOElectronics module, create the interface BSMEvents_I by right-clicking **ITSOElectronics** and selecting **New** → **Interface**. The interface consists of five one-way operations, each with one input. Figure 11-11 shows how the operations should look.

> **Important:** Create the interface operations as one-way operations because they only have inputs and no outputs.

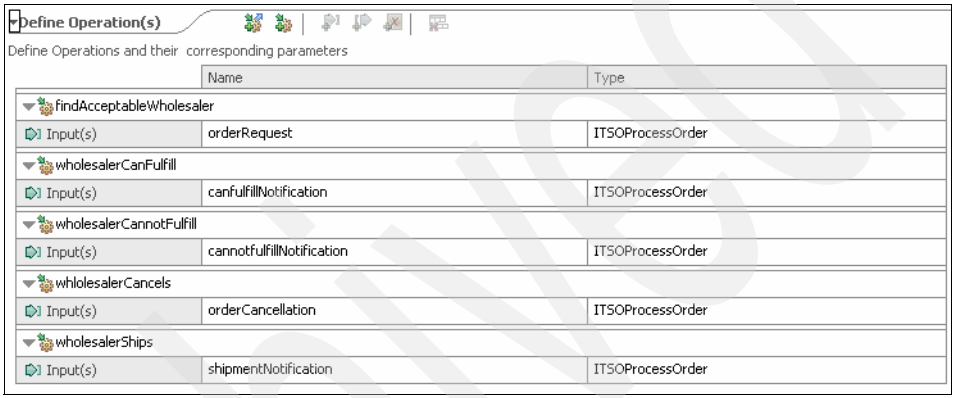


Figure 11-11 The interface BSMEvents_I

The operations, inputs, and business objects used in the interface are shown in Table 11-1. The business object ITSOProcessOrder should already exist in the ITSOElectronics module.

Table 11-1 Details of the BSMEvents_I interface operations

Operation	Input	Type
findAcceptableWholesaler	orderRequest	ITSOProcessOrder
wholesalerCanFulfill	canfulfillNotification	ITSOProcessOrder
wholesalerCannotFulfill	cannotfulfillNotification	ITSOProcessOrder
wholesalerCancels	orderCancellation	ITSOProcessOrder
wholesalerShips	shipmentNotification	ITSOProcessOrder

Archived

2. In the ITSOElectronics module, create the interface BSMOrderReq_I. Figure 11-12 shows how the interface should look.

> **Important:** Again, the operation should be one-way because it only has an input and no output.

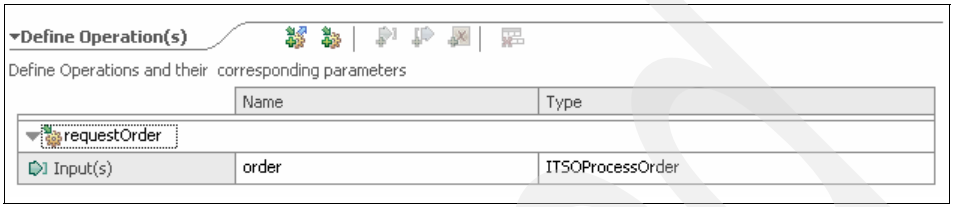


Figure 11-12 The interface BSMOrderReq_I

Table 11-2 outlines the details of the operation.

Table 11-2 Details of the BSMOrderReq_I interface operation

Operation	Input	Type
requestOrder	order	ITSOProcessOrder

Archived

Creating the state machine artefact

Now we create state machine. Figure 11-13 shows a preview of the finished result.

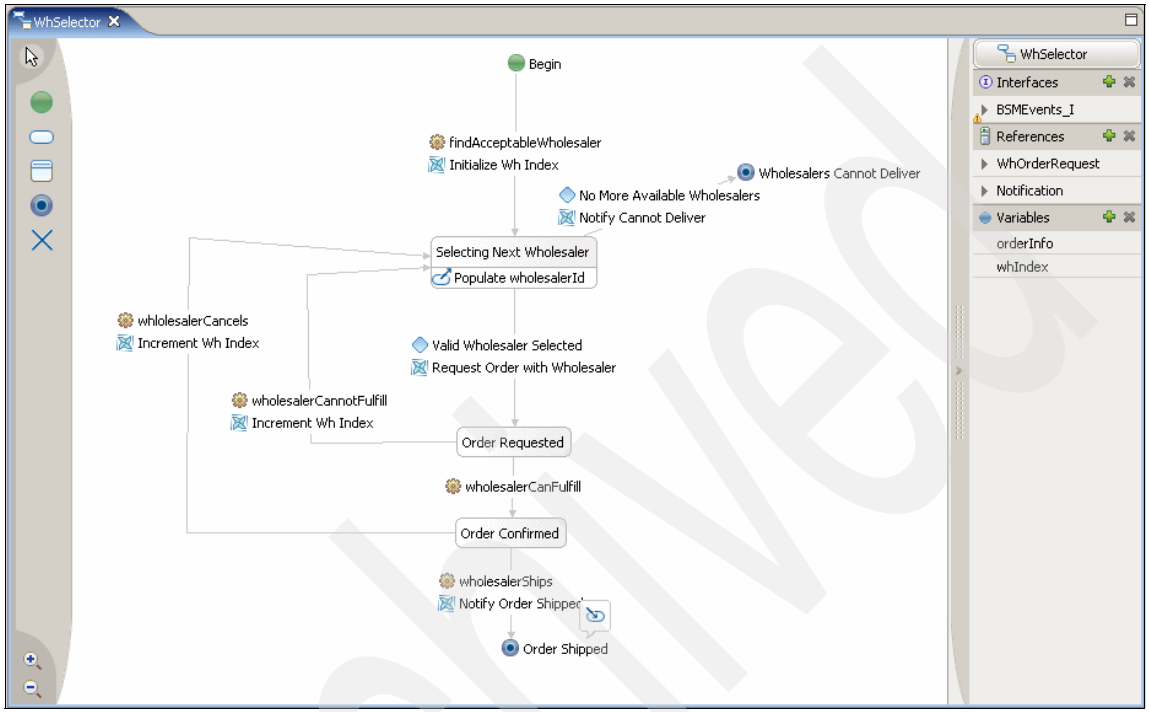


Figure 11-13 The WhSelector business state machine

> **Note:** The exact layout of the machine is not important as long as all of the states are connected correctly. Do not spend large amounts of time trying to make the machine aesthetically pleasing.

Archived

To build the business state machine, perform the following actions:

1. In the ITSOElectronics module, in the Business Logic folder, right-click **State Machines** → **New** → **Business State Machine**.
2. In the New Business State Machine window (Figure 11-14), in the Name field, type `WhSelector`. Click **Finish**.

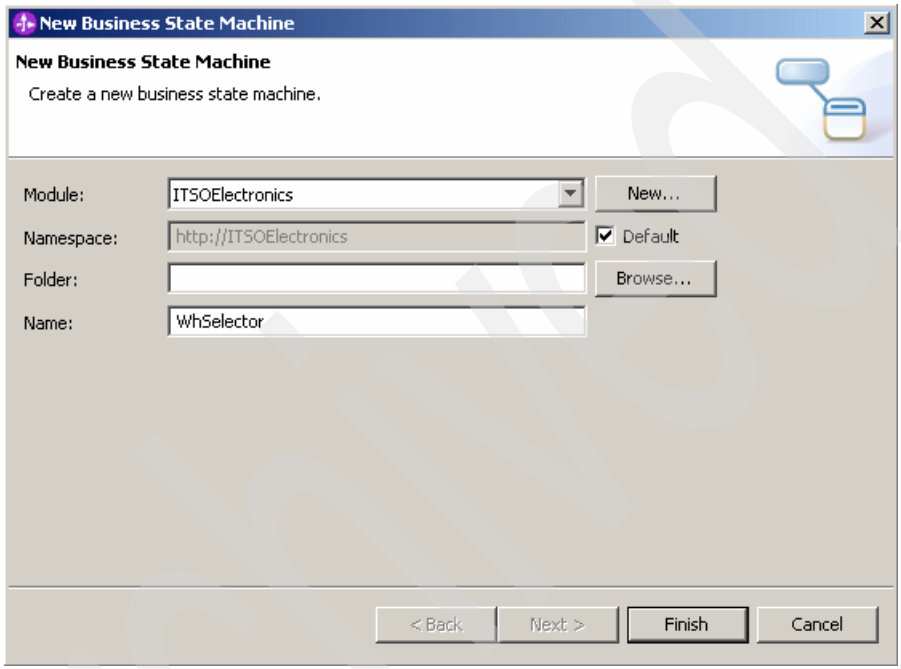


Figure 11-14 New Business State Machine window

Archived

The Business State Machine Editor opens as shown in Figure 11-15, indicating the changes. This shows stage one of building the solution (Figure 11-13 on page 258) in the editor.

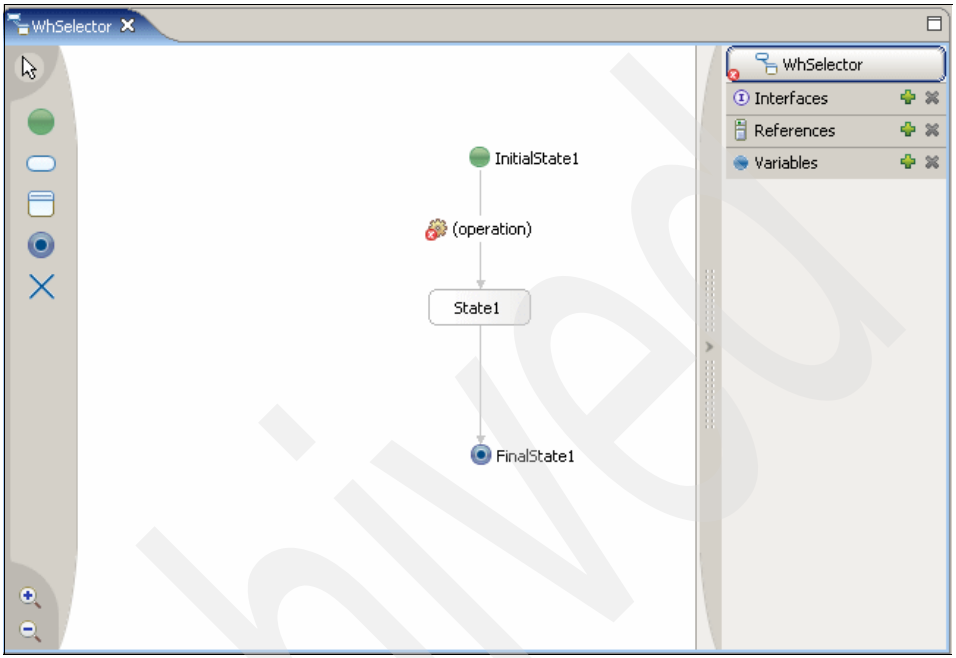


Figure 11-15 Business State Machine Editor

Note: There may be an error indicated by a red X (operation). This is because the default operation is not yet configured and will be fixed later.

Adding interfaces and references to the state machine

We add the interfaces and references shown in Table 11-3 to the state machine to allow it to communicate with other components.

Table 11-3 Interfaces and references used by the state machine

Name	Interface or reference	Type
BSMEvents_I	Interface	BSMEvents_I
WhOrderRequest	Reference	BSMOrderReq_I
Notification	Reference	ITSOOrdNotify

Perform the following actions:

1. Click the plus (**+**) sign next to **Interfaces** to add an interface.
2. In the Interface Selection panel, select **BSMEvents_I** and click **OK**.
3. Click the **+** sign next to **References** to add a reference.
4. In the Reference Selection panel, select **BSMOrderReq_I** and click **OK**.
5. Click the **Properties** tab and the **Description** subtab. Rename Reference1 to `WhOrderRequest`. Add another reference of type `ITSOOrdNotify_I` and rename it to `Notification`.

Figure 11-16 shows how the Business State Machine Editor looks now after making the changes.

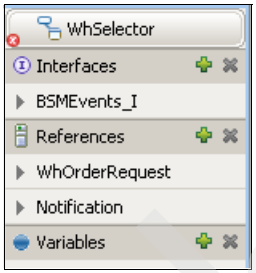


Figure 11-16 Business State Machine Editor showing interfaces and references

Creating the states for the state machine

We create the following six states for the state machine:

- Begin (Initial State)
- Selecting Next Wholesaler
- Order Requested
- Order Confirmed
- Order Shipped (Final State)
- Wholesaler Cannot Deliver (Final State)

To create the states:

1. Click **State1** and rename it to `Selecting Next Wholesaler`.

   Note: Spaces are allowed in the display names of the states.

2. Click the **New State** icon ( ) and click the **Business State Machine Editor canvas** to add a new state to the editor.
3. Click the newly-created **State2** and rename it to `Order Requested`.

Archived

4. Add another state and name it `Order Confirmed`.

   Figure 11-17 shows how the Business State Machine Editor canvas looks now.

Figure 11-17 Business State Machine Editor canvas with three states

5. Click **InitialState1** and rename it to `Begin`.
6. Click **FinalState1** and rename it to `Wholesaler Cannot Deliver`.
7. Click the **Final State** icon ( ) to add another Final State to the canvas.
8. Rename the newly created FinalState2 to `Order Shipped`.

> **Tip:** To avoid overlaps, keep rearranging the components on the state machine canvas. Make sure that all the components are connected correctly. The canvas does not need to look exactly like the figures shown in this section.

Figure 11-18 shows how the Business State Machine Editor canvas looks after completing these steps.

Archived

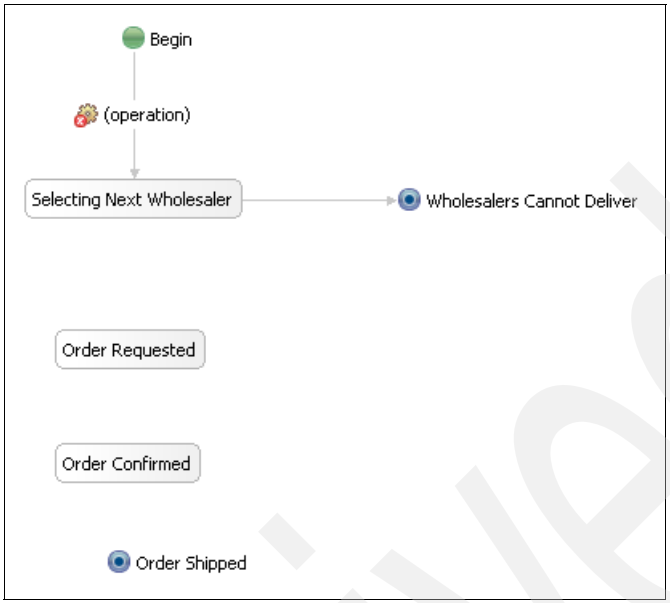


Figure 11-18 Business State Machine Editor canvas with two final states

Adding transitions to the state machine

Now that we have all the states that we need, we can add transitions between them. Transitions define which states can follow others, effectively defining the order of execution of the state machine logic.

The actions shown in Table 11-4 are used in the WhSelector business state machine. Not all of these transitions are added in this section; some are added in later sections.

Table 11-4 State transitions used in the business state machine

From state	To state
Begin	Selecting Next Wholesaler
Selecting Next Wholesaler	Wholesalers Cannot Deliver
Selecting Next Wholesaler	Order Requested
Order Requested	Selecting Next Wholesaler
Order Requested	Order Confirmed
Order Confirmed	Selecting Next Wholesaler
Order Confirmed	Order Shipped

Archived

Perform the following steps:

1. To add the first transition, move the mouse over the state **Selecting Next Wholesaler** until the border is highlighted and drag it to the state **Order Requested**. A transition arrow is displayed between the two states.
2. Add a transition from Order Requested to Order Confirmed.
3. Add a transition from Order Confirmed to Order Shipped.

Figure 11-19 shows how the Business State Machine Editor canvas looks after completing these steps.

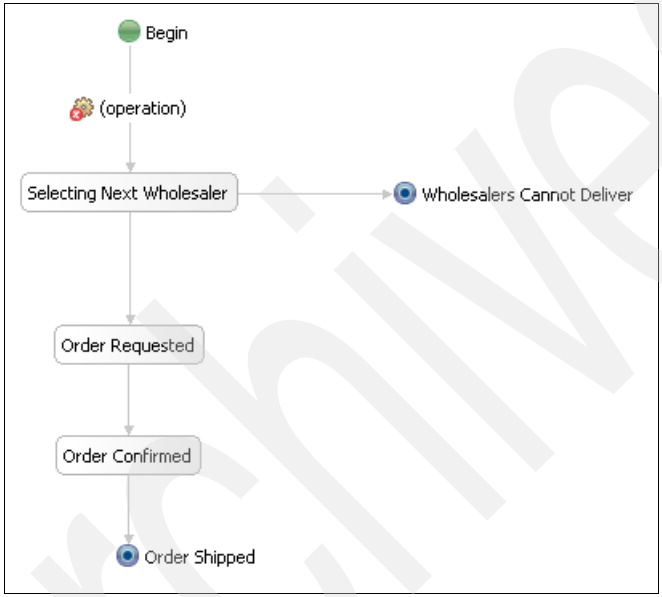


Figure 11-19 Business State Machine Editor canvas with transitions

Archived

Adding actions to the transitions

Now we need to add actions to the transitions that we have created. An *action* is something that the state machine does when moving from state to state along a transition. The actions shown in Table 11-5 are used in the WhSelector business state machine.

Table 11-5 Actions used on state transitions

Action	From state	To state
Initialize Wh Index	Begin	Selecting Next Wholesaler
Notify Cannot Deliver	Selecting Next Wholesaler	Wholesalers Cannot Deliver
Notify Order Shipped	Order Confirmed	Order Shipped
Increment Wh Index	Order Requested	Selecting Next Wholesaler
Increment Wh Index	Order Confirmed	Selecting Next Wholesaler
Request Order with Wholesaler	Selecting Next Wholesaler	Order Requested

Note: Not all of the actions in Table 11-5 are created in this section, because we have not yet defined all the transitions that are needed. They are created in later sections.

Perform the following steps:

1. To add the first action, move the mouse over the transition between Selecting Next Wholesaler and Wholesalers Cannot Deliver until a pop-up window with four icons is displayed (see Figure 11-20).

Figure 11-20 Pop-up window showing the four icons

2. Click the **Add an Action** icon ( ).
3. Rename the newly created Action1 to `Notify Cannot Deliver`.

4. At the transition between Selecting Next Wholesaler and Order Requested, add an action and name it `Request Order with Wholesaler`.
5. At the transition between Order Confirm and Order Shipped, add an action and name it `Notify Order Shipped`.

Figure 11-21 shows how the Business State Machine Editor canvas looks after making these changes.

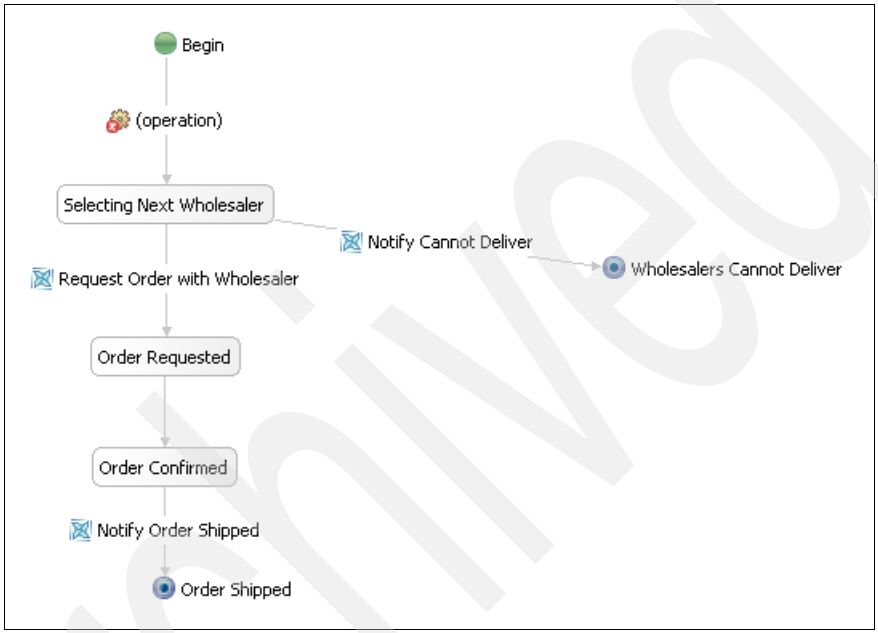


Figure 11-21 Business State Machine Editor canvas with actions added

Adding more transitions and actions

One of the advantages of using a business state machine over a WS-BPEL process is that it is easy to move backwards in a process, meaning to return to a previous point or state. Now add some more transitions that allow the state machine to return to the Selecting Next Wholesaler state if, at any point, the currently selected wholesaler cannot fulfill the requested order. For the full list of transitions used in the business state machine, refer to Table 11-4 on page 263.

Archived

Perform the following steps:

1. Add a transition between Order Requested and Selecting Next Wholesaler.

> **Tip:** After the new transition is created, drag the **transition arrow anchor points** to arrange it on the canvas. You need to move the new arrow because it is initially created on top of an existing one.

2. Add the action `Increment Wh Index` to the transition between Order Requested and Selecting Next Wholesaler.
3. Add a transition between Order Confirmed and Selecting Next Wholesaler. You may need to reshape the transition to make it visible on the canvas.
4. Add the action `Increment Wh Index` to the transition between Order Confirmed and Selecting Next Wholesaler.
5. Add the action `Initialize Wh Index` to the transition between Begin and Selecting Next Wholesaler.

Figure 11-22 shows how the Business State Machine Editor canvas looks now after completing these steps.

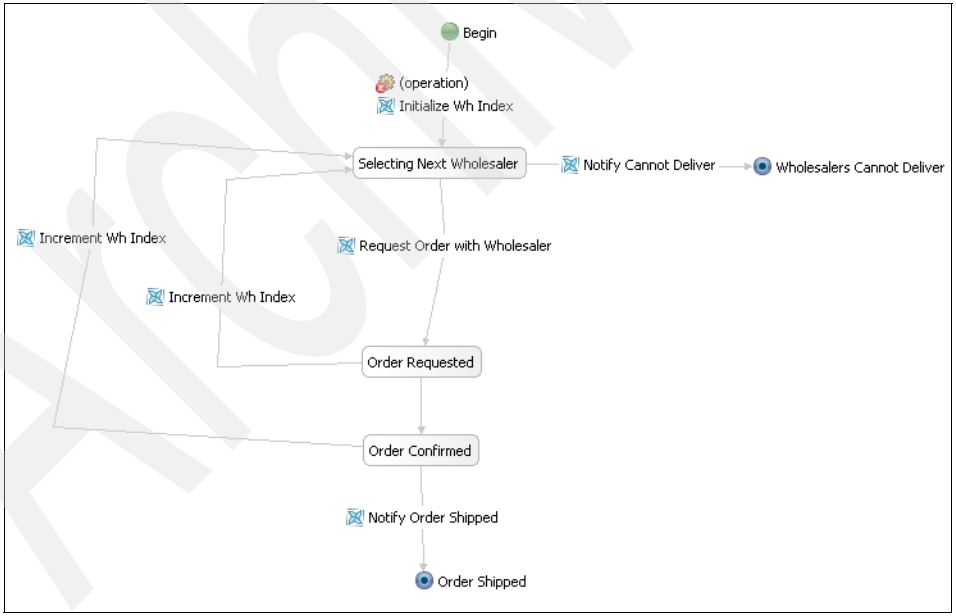


Figure 11-22 Business State Machine Editor canvas with all actions and transitions

Archived

Adding operations

Operations allow the business state machine to respond to external events. Each operation in the state machine corresponds to an operation of an interface that we previously added to the state machine.

> **Note:** Operations are inbound only. Although operations correspond to the operations defined in interfaces, they cannot be used to invoke operations on other components. Actions are used for outbound calls.

The operations shown in Table 11-6 are used in the business state machine.

Table 11-6 Operations used on state transitions

Operation	From state	To state
findAcceptableWholesaler	Begin	Selecting Next Wholesaler
wholesalerCanFulfill	Order Requested	Order Confirmed
wholesalerShips	Order Confirmed	Order Shipped
wholesalerCannotFulfill	Order Requested	Selecting Next Wholesaler
wholesalerCancels	Order Confirmed	Selecting Next Wholesaler

Perform the following steps to add an operation:

1. Rename and define the default operation, which is used to create the state machine. Click the ***operation*** that you want to add.
2. Click the **Properties** tab and the **Description** subtab.
   a. In the Interface field, select **BSMEvents_I**.
   b. In the Operation field, select **findAcceptableWholesaler**.
3. Click the **Add an operation** icon ( ), in the pop-up menu of the transition between Order Requested and Order Confirmed, to add another operation.
4. Click the **Properties** tab and the **Description** subtab.
   a. In the Interface field, select **BSMEvents_I**.
   b. In the Operation field, select **wholesalerCanFulfill**.
5. Add the operation **wholesalerShips** to the transition between Order Confirmed and Order Shipped.
6. Add the operation **wholesalerCannotFulfill** to the transition between Order Requested and Selecting Next Wholesaler.
7. Add the operation **wholesalerCancels** to the transition between Order Confirmed and Selecting Next Wholesaler.

Archived

Figure 11-23 shows the changes in the Business State Machine Editor canvas.

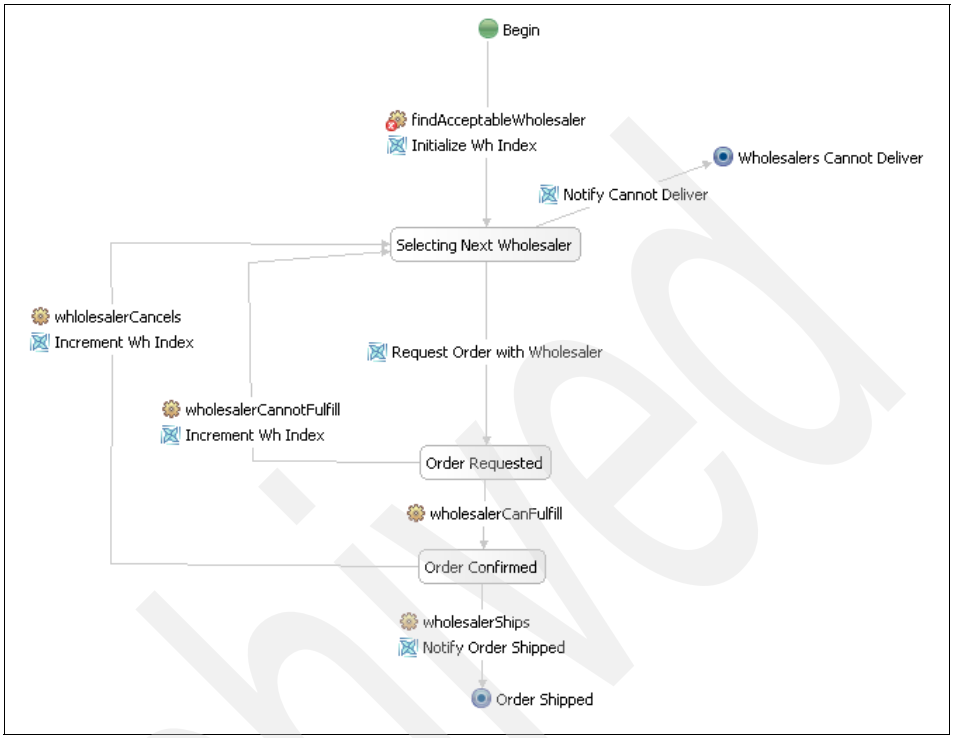


Figure 11-23 The business state machine with the operations added

Adding variables

Like business processes, state machines use variables to hold the values of the entities that it uses. Here we use variables to hold the details of the order and the index of the wholesaler that we are currently using. Table 11-7 shows the details of the variables that are used.

Table 11-7 Variables used in the business state machine

Variable	Type
orderInfo	ITSOProcessOrder
whIndex	int

Click the **+** sign next to **Variables** to add following variables:

- `orderInfo` of type `ITSOProcessOrder`
- `whIndex` of type `int`

Figure 11-24 shows the Variables section with the added variables.

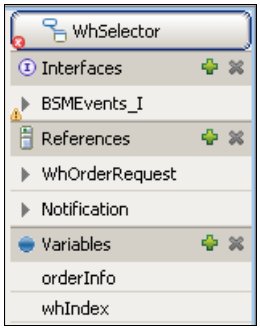


Figure 11-24 Variables added to the business state machine

Defining the properties of the actions which use the variables

When we originally added the actions, they were empty. Now that we have added the variables, we can define the properties of the actions which use them. Actions can be of the type *invoke*, which invokes another component, or of the type *visual snippet*, which is used to manipulate internal variables or to perform more complex action logic.

Initializing the warehouse index

When the business state machine is first created, we need to initialize the whIndex variable. The first wholesaler is referenced by the index 1. Complete the following steps:

1. Click the **Initialize Wh Index** action.
2. Select the **Properties** tab and the **Details** subtab.
3. Create the visual snippet shown in Figure 11-25.

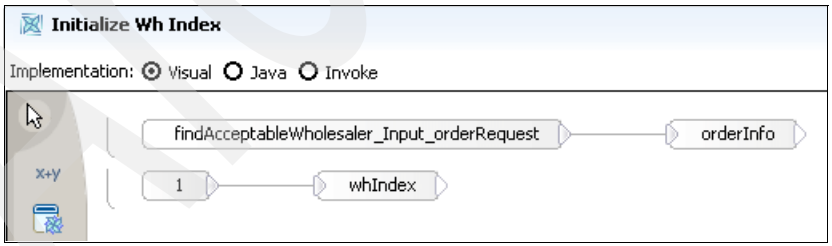


Figure 11-25 Visual snippet for the Initialize Wh Index action

Incrementing the warehouse index

When a wholesaler cannot fulfil an order, or cancels a confirmed order, we need to increment the wholesaler index so that the next wholesaler can be selected. Because we use a simple integer for the index, we need to add one to it.

Add following visual snippet to both occurrences of the **Increment Wh Index** action as shown in Figure 11-26. It is necessary to add the snippet to both occurrences of the action, even though they have the same name.

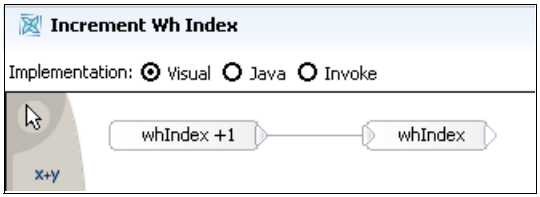


Figure 11-26 Visual snippet for the Increment Wh Index action

Placing the order request with the wholesaler

This action invokes the service which places the order with the currently selected wholesaler. The wholesaler to be used is indicated by the current value of the whIndex variable. A business rule is created in the next section, to translate the index into the correct wholesaler name.

1. Click the action **Request Order with Wholesaler**.
2. Select the **Properties** tab and the **Details** subtab.
3. Select **Invoke** as the implementation.
4. It the warning message shown in Figure 11-27 appears, select **Yes**.

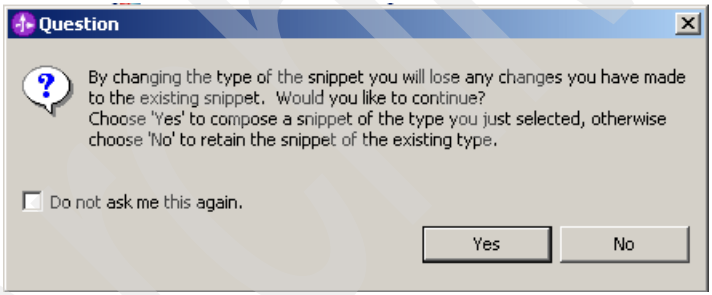


Figure 11-27 Warning message window

5. In the Reference field, select **whOrderRequest**.
6. In the Operation field, select **requestOrder**.
7. In the Variables box, click **orderInfo** to highlight it.
8. In the Operation input box, click **order** to highlight it.
9. Click **Set**.

10. Verify that the Operation input box now contains order < orderInfo as shown in Figure 11-28.

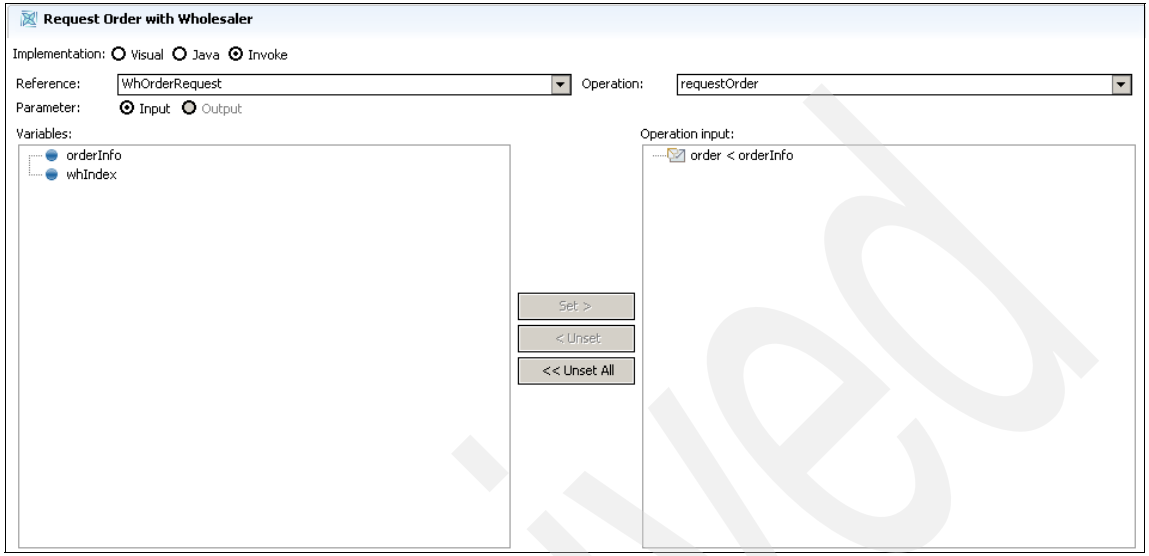


Figure 11-28 Invoking Request Order with Wholesaler

Notifying that the order is shipped

When the order is shipped, we need to notify the original requestor that the shipment has occurred. We do this by calling the OrdNotify service:

1. Click **Notify Order Shipped**, and for Implementation, select **Invoke**.
2. For Reference, select **Notification**, and for Operation, select **notifyDeliveryDate**.
3. Expand **WholesalerShips input** and select the **shipment notification** variable to assign it the Operation order. Highlight both the variable and the operation and click **Set**.
4. Click **Notify Cannot Deliver**, and for Implementation, select **Invoke**.
5. For Reference, select **Notification**, and for Operation, select **notifyDeliveryDate**.
6. Assign the **orderInfo** variable to the Operation order. Highlight both the variable and the operation and click **Set**.

Adding conditions

When the business state machine moves from state to state along the transitions, it needs the ability to determine which transition to follow if there is more than one outgoing transition from a particular state. An operation can be

Archived

used to detect an incoming event and choose a transition. However, if no operation is appropriate, conditions can be used to choose the correct transition.

Because there are two transitions from Selecting Next Wholesaler without operations, we need to add two conditions, one for each transition, to tell the state machine which transition to use, depending on the value of the whIndex variable. If the value of the variable is greater than 2, we have run out of wholesalers and need to notify a failure. If the value is less than or equal to 2, we can request that the indexed wholesaler fulfill the order. Table 11-8 specifies the conditions that are used.

Table 11-8 Conditions used by the business state machine

Condition	From state	To state
Valid Wholesaler Selected	Selecting Next Wholesaler	Order Requested
No More Available Wholesalers	Selecting Next Wholesale	Wholesalers Cannot Deliver

When creating the conditions, we need to return a boolean value for each one. All of the outgoing conditions for a state are evaluated, and the transition used is the one that evaluates to true. Perform the following actions:

1. Click the **transition** between Selecting Next Wholesaler and Order Requested.
2. Click the **Add a Condition** icon ( ) in the pop-up menu to add a condition.
3. Rename Condition1 to `Valid Wholesaler Selected`.
4. Add the visual snippet shown in Figure 11-29 to the condition Valid Wholesaler Selected.

> **Tip:** When in the Visual Snippet Editor, to add the return snippet to the canvas, right-click the canvas and select **Add** → **Return**.

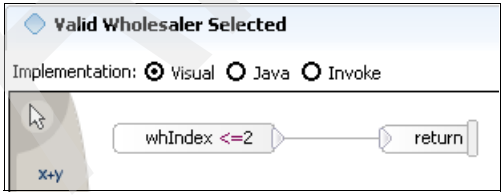


Figure 11-29 Visual snippet of the Valid Wholesaler Selected condition

5. Add another condition between Selecting Next Wholesaler and Wholesaler Cannot Deliver. Name it `No More Available Wholesalers`.

6. Add the visual snippet shown in Figure 11-30 to the condition No More Available Wholesalers.

Figure 11-30 Visual snippet of the No More Wholesalers Available condition

Figure 11-31 illustrates the results as they are displayed in the Business State Machine Editor canvas now.

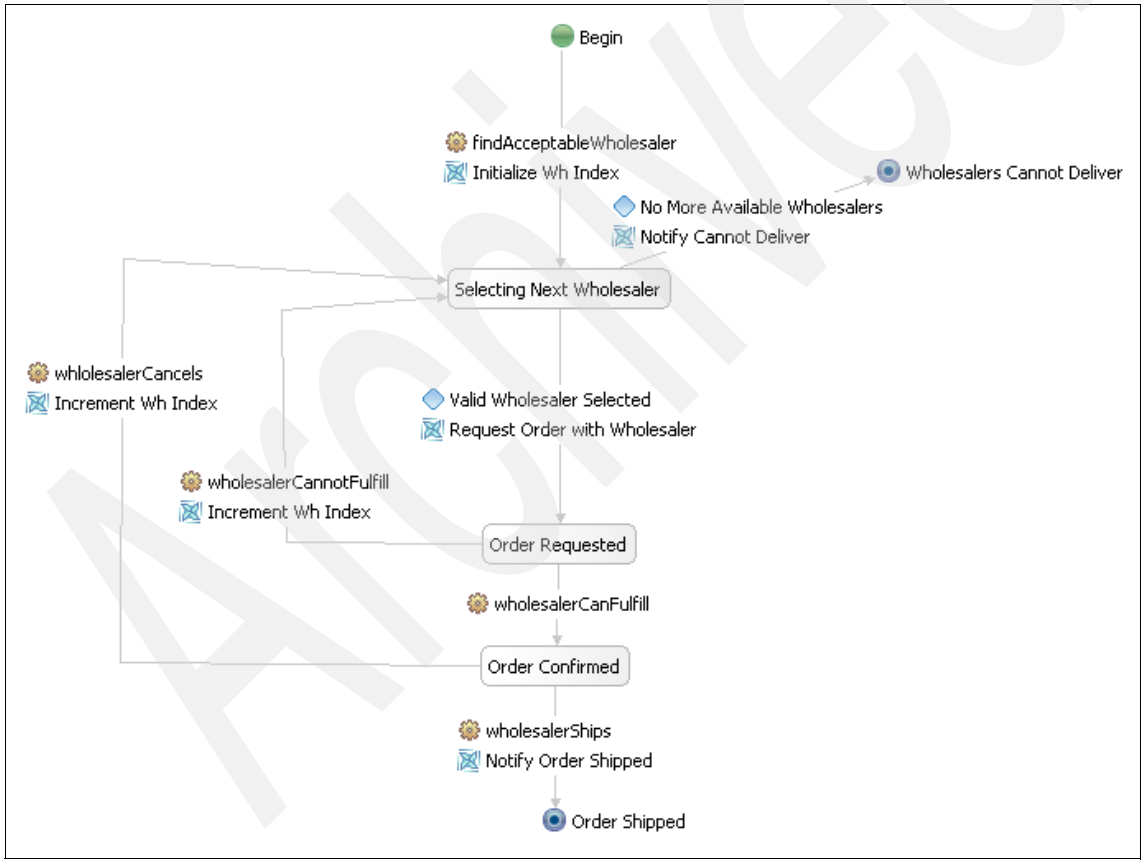


Figure 11-31 Business State Machine Editor canvas with all conditions

Archived

Adding another state to allow more actions

The Notify Correct Order action invokes a service call to notify that the order cannot be fulfilled by any of the available wholesalers. At present, the orderInfo object contains no information to indicate that the order was not fulfilled, when the invoke is made. As it stands, the invoke will send a notify for wholesaler 3, even though there is no wholesaler 3.

The solution is to add another action before the invoke, to populate the orderInfo fields attributes with non-fulfillment information. WebSphere Integration Developer does not allow multiple actions in a single state transition. Therefore, we must consider alternatives:

- Add the code to the state exit of state Selecting Next Wholesaler.

  This exit runs regardless of whether there are more wholesalers. Therefore, it is not suitable because we need to recode the “Do we have any more wholesalers?” condition.
- Change the action type of Notify Cannot Deliver to be visual or Java, and make the invoke call after populating the attributes.
- Add an additional state of Order Not Fulfilled between states Selecting Next Wholesaler and Wholesalers Cannot Fulfill with no conditions or operations. This means that we gain an extra transition, so that we can use the first transition to populate the attributes and the second to perform the invoke.

We choose the option to add an additional state and transition and follow these steps:

1. Add an new state called *Order Not Fulfilled*.
2. Move the transition between Selecting Next Wholesaler and Wholesaler Cannot Deliver to be between Selecting Next Wholesaler and the new state Order Not Fulfilled.
3. Add a new transition between Order Not Fulfilled and Wholesalers Cannot Deliver.
4. This now gives us a new transition so that we can add another action. The new action (to populate the orderInfo attributes) must come before the invoke. Therefore we must move the Notify Cannot Deliver action.

   Drag the action **Notify Cannot Deliver** to the newly created transition between Order Not Fulfilled and Wholesalers Cannot Deliver.
5. Add a new action `Populate Order Details` on the transition between Selecting Next Wholesaler and Order Not Fulfilled.

Archived

6. Add a visual snippet as shown in Figure 11-32 to populate the order.

Figure 11-32 The visual snippet to populate the orderInfo attributes

Table 11-9 shows the attributes and values.

Table 11-9 Populating the values in the orderInfo object

Value	Attribute
“IMPORTANT: YOUR ORDER WILL NOT BE SHIPPED”	orderInfo.WhResponse.confNo
“NO WHOLESALER AVAILABLE - ORDER WILL NOT BE SHIPPED”	orderInfo.WhResponse.bestWholesale
0	orderInfo.WhResponse.bestWholesaleDays
0	orderInfo.WhResponse.price

Adding a state exit

We are going to use a business rule to determine which wholesaler is represented by which index number. Therefore, we must copy the whIndex variable into the orderInfo business object before we request that order be fulfilled by the wholesaler. Because this copy must happen before the Request Order with Wholesaler action happens, we do it when we exit the Selecting Next Wholesaler state. WebSphere Process Server allows us to add both entries and exits to a state. We use a state exit to perform the copy.

To add the exit and the logic:

1. Right-click the state **Selecting Next Wholesaler** and select **Add** → **Exit**.
2. Rename Exit1 to `Populate wholesalerId`.
3. Select the **Properties** tab and the **Details** subtab.
4. Add the visual snippet as shown in Figure 11-33.

Archived

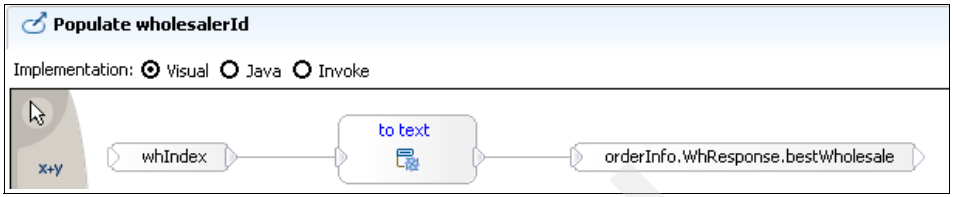


Figure 11-33 Visual snippet of the Exit

> **Note:** Because the whIndex variable is of type int and the orderInfo.WhResponse.bestWholesale is of type String, we need to use the snippet to text to convert between the types. orderInfo.WhResponse.bestWholesale may be considered an inappropriate reuse of an existing field, but it serves the purpose for the scenario we are using.

Figure 11-34 shows how the Business State Machine Editor canvas looks now.

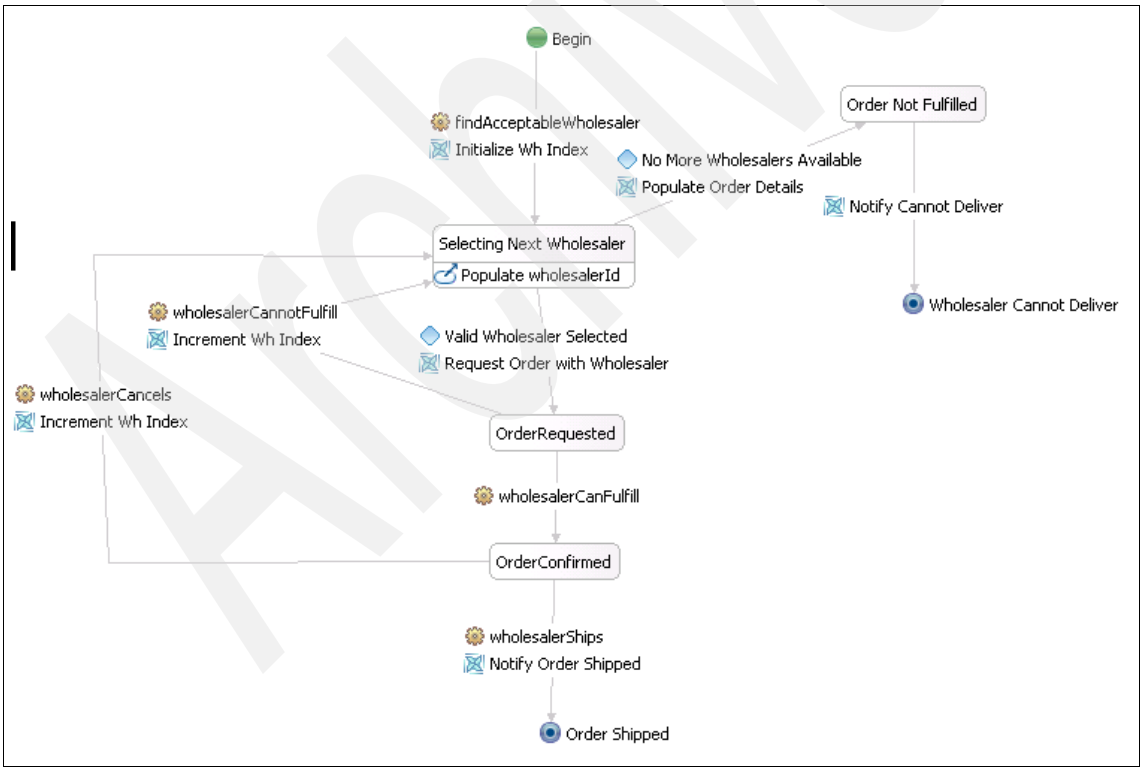


Figure 11-34 Completed business state machine logic

Adding a correlation

The state machine logic is now finished. Finally, we need to add a correlation so that WebSphere Process Server can find the correct instance when an event is received. A *correlation* is a unique identifier for a state machine instance as many instances of the same state machine may be active in WebSphere Process Server at any one time.

In this scenario, we use the orderId of the original order request as the correlation. Each incoming event must refer to that orderId so that the correct instance of the state machine can be found. In a full implementation, the orderId may not be sufficient, so multiple fields can be used as correlations.

For each incoming event, the field on the incoming datatype which corresponds to the orderId must be defined, so that WebSphere Process Server can tell which field to use. This is known as an *alias*. In our case, this is the same for each operation because they all use the ITSOProcessOrder type. We must to create one alias for each operation, even though each one uses the same field in the same business object type.

To add the correlation and aliases, perform the following steps:

1. Click the state machine canvas (not any of the artifacts within it) and select the **Properties** tab and the **Correlation** subtab.
2. Click **New**.
3. In the Create Correlation Property window, complete these tasks:
   a. In the Name field, enter `orderNumber`.
   b. Click **Browse**. In the window that opens, select **string** and click **OK**.
   c. In the Alias section, click **New**.
   d. In the Properties Alias Selection window, complete these steps:
      i. For Matching interfaces, select **BSMEvents_I**.
      ii. In the Select a data type section, expand **findAcceptableWholesaler input** → **orderRequest** and highlight **orderID**.
      iii. Click **Add**. The Property Alias Selection window should confirm that the property alias was added.

      > **Note:** You do not have to close the Properties Alias Selection window before you add more messages.

      iv. In the Select a data type section, expand **wholesalerCancels input** → **orderCancellation** and highlight **orderID**.
      v. Click **Add**.

vi. Add orderId correlation aliases for the following input types:

- wholesalerCanFulfill input
- wholesalerCannotFulfill input
- wholesalerShips input

vii. Click **OK**.

The Create Correlation Property window (Figure 11-35) shows the results of these actions.

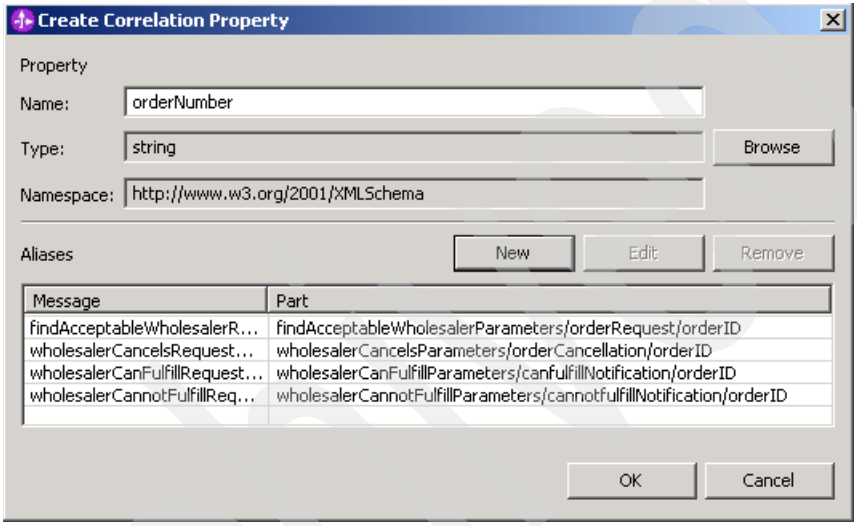


Figure 11-35 Create Correlation Property window

e. Click **OK** to create the correlation.

4. Press Ctrl+S to save the business state machine, and then close the Business State Machine Editor. The business state machine is now complete.

> **Test point:** At this point, we recommend that you perform a unit test of the state machine to ensure that it functions correctly in isolation. You can find guidelines for this in 11.4, “Runtime guidelines” on page 312.
>
> **Timeouts**: The business state machine that we constructed does not contain timeouts. In a full implementation, it is good practice to add timeouts to some transitions to allow processing to continue, even when a wholesaler has not responded in a reasonable amount of time. For example, when the machine is in the Order Requested state, a timeout allows it to transition back to the Selecting Next Wholesaler state if the response to the order request does not arrive in time.

11.3.4 Creating the business rule BSMChooseWh

When the WhSelector business state machine sends order requests to wholesalers, it identifies the wholesaler by means of an integer index. An index value of 1 represents WholesalerD. An index value of 2 represents WholesalerE. To allow the integer index to invoke the desired wholesaler, we use a business rule.

The use of a business rule was chosen for two reasons. First, it allows a quick way to build an *if this condition, invoke this service* structure. Second, through the use of templates, it allows the user to change the business rule without redeploying the component. In this example, a business rule can be used to change the order of the wholesalers called from the business state machine, for example to try E before trying D. For simplicity, we do not use business rule templates here.

We create the rule group with one interface, to allow the business state machine to call it, and two references which allow the rule to call the appropriate wholesaler's service. Wholesalers D and E, together with their services are defined in 11.3.6, "Creating the Wholesaler service providers" on page 285. Because we already have defined the interface that they will use, we can reference them here at this time.

Archived

Figure 11-36 shows the location of the business rule in relation to the overall components.

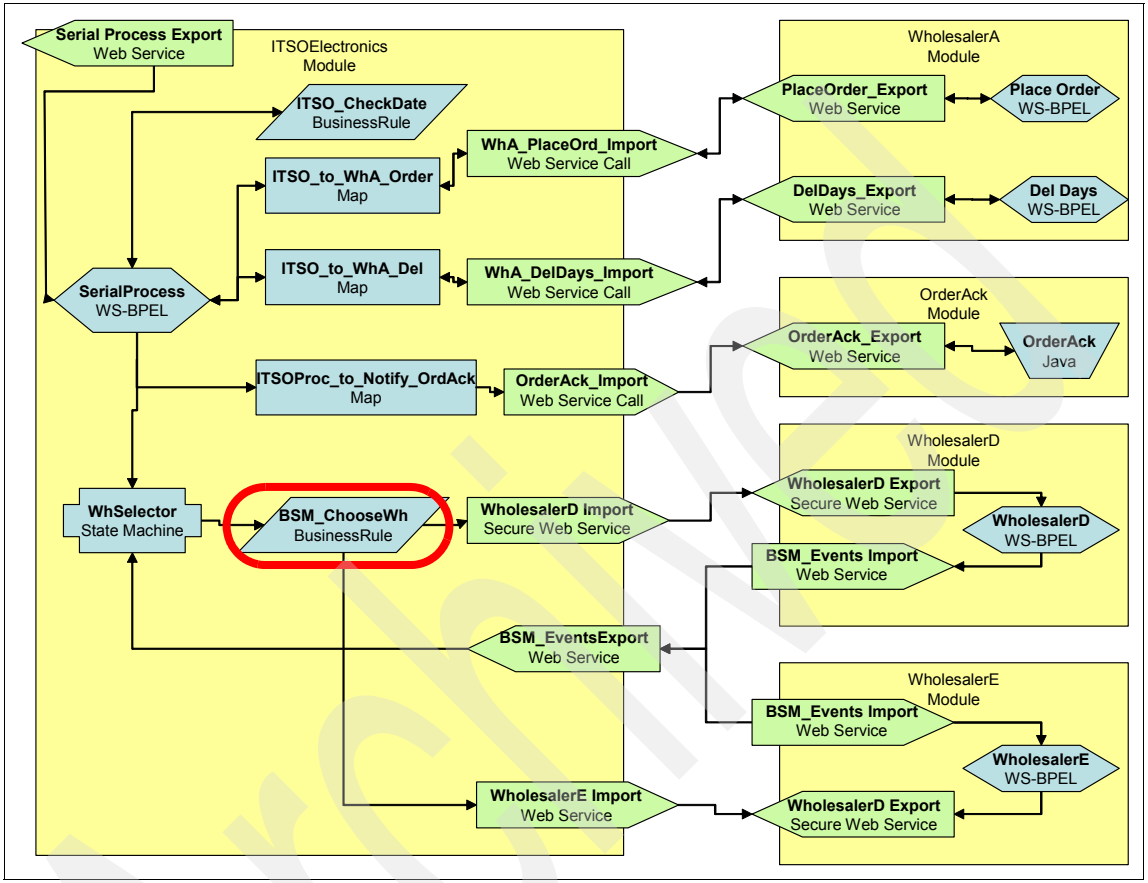


Figure 11-36 Overview of the scenario showing the interlinked components and modules

Archived

To create the rule group, perform the following steps:

1. In the ITSOElectronics module, create a new rule group named `BSMChooseWh` with an interface of `BSMOrderReq_I`.
2. Add two references to the rule group, both using the interface BSMOrderReq_I (Figure 11-37):
   - `WholesalerD`
   - `WholesalerE`

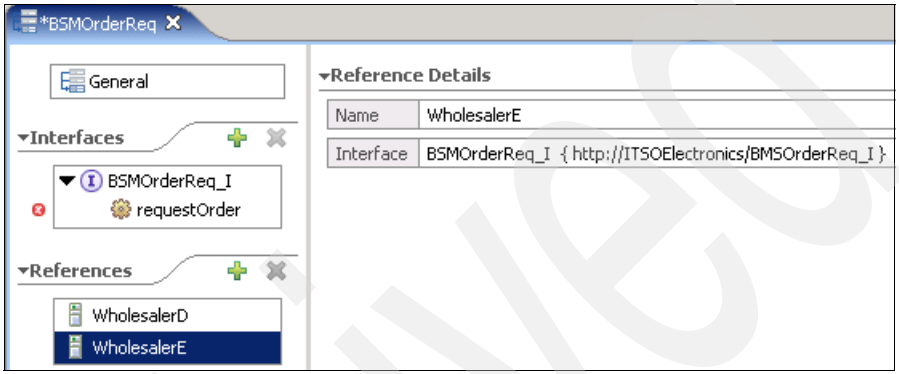


Figure 11-37 BSMChooseWh rule group

3. Press Ctrl+S to save the rule group.
4. Create the ruleset, which the rule group uses, from within the Rule Group Editor.
   a. In the Interfaces section, click the **requestOrder** operation of the BMSOrderReq_I interface.
   b. In the Default Destination field, click **Enter Destination** and select **New Ruleset**.
   c. Name the new Ruleset `chooseWh_RS`.

Archived

d. In the Rule Set, add two If-Then rules, where the Then part of the rule is an Invoke, as shown in Figure 11-38.

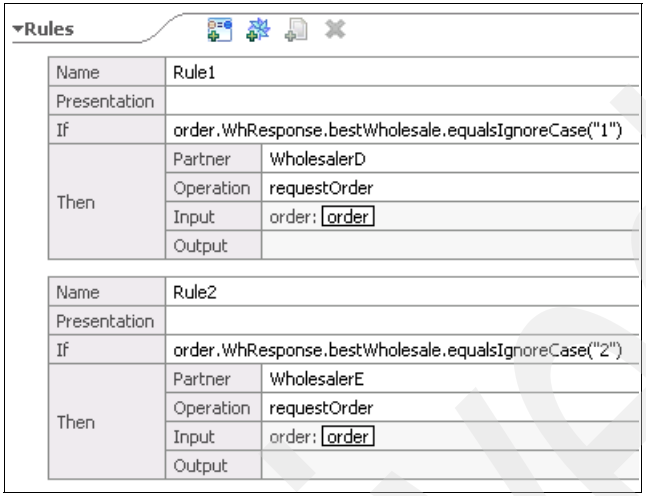


Figure 11-38 The business rule chooseWh_RS

e. Press Ctrl+S to save the rule set and close the Ruleset Editor. Figure 11-39 shows how the rule group BSMChooseWh looks now.

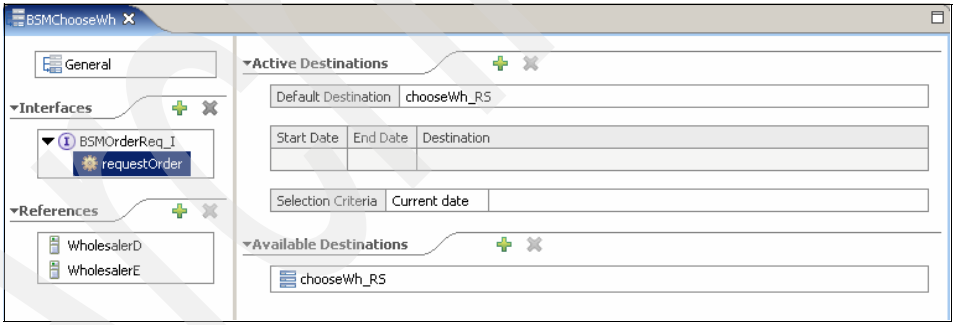


Figure 11-39 Rule group BSMChooseWh

5. Press Ctrl+S to save the rule group, and close the Rule Group Editor.

> **Test point:** At this point, we recommend that you perform a unit test of the business rule group to ensure that it functions correctly in isolation. You can find the guidelines for this in 11.4, “Runtime guidelines” on page 312.

Archived

11.3.5 Creating the ITSOElectronics assembly diagram

We create an ITSOElectronics assembly diagram which contains the business state machine and the BSMChooseWh business rule. It also calls the OrderAck module.

1. Open the ITSOElectronics assembly diagram in the ITSOElectronics module.
2. Drag the state machine **WhSelector** to the assembly diagram.
3. Drag the rule group **BSMChooseWh** to the assembly diagram.
4. Right-click **WhSelector** and select **Wire to Existing**. Figure 11-40 shows how the assembly diagram looks.

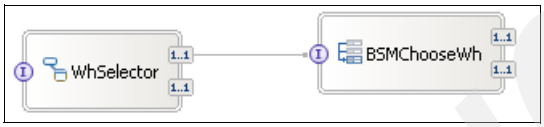


Figure 11-40 Assembly diagram with wired WhSelector and BSMChooseWh

5. Drag **OrderAckExport** from the OrderAck assembly diagram to the ITSOElectronics assembly diagram.
6. In the Component Creation wizard, select **Import with SCA Binding**.
7. In the ITSOElectronics assembly diagram, rename this component to `OrderAckImport`.
8. Because WhSelector uses Interface ITSOOrdNotify_I and OrderAck uses Interface Notify_I, we need to use an interface map between them.
   a. Drag the interface map **ITSOProc_to_Notify_OrdAck** from the ITSOElectronics module to the canvas.
   b. Right-click **ITSOProc_to_Notify_OrdAck** and select **Wire to Existing**. Figure 11-41 shows the assembly diagram after completing these steps.

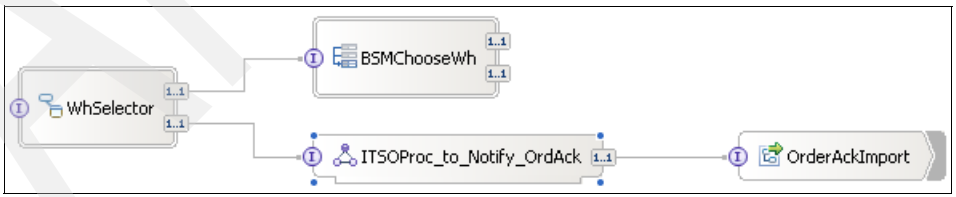


Figure 11-41 Assembly diagram with the wired interface map

9. Save the assembly diagram.

11.3.6 Creating the Wholesaler service providers

Now that we have created the business state machine and the rule group, we need to create the two wholesalers which will interact with them. For the purposes of this scenario, Wholesalers D and E are identical except for their behavior based on the part number that they are asked to supply.

Each wholesaler receives a request to supply a particular part. Depending on the part number, one of three actions happens:

- The wholesaler declines the order, sending a wholesalerCannotFulfill response.
- The wholesaler accepts the order, sending a wholesalerCanFulfill response. It waits 20 seconds and then ships the order, sending a wholesalerShips response to the business state machine.
- The wholesaler accepts the order, sending a wholesalerCanFulfill response. It waits 20 seconds and then cancels the order, sending a wholesalerCancels response to the business state machine.

Table 11-10 summarizes this and shows which part triggers which responses from which wholesalers. An entry of “none” indicates that the wholesaler does not expect to have to respond to a request for this part, because it is already supplied by another wholesaler (for part1, that wholesaler is wholesaler A).

Table 11-10 Wholesaler actions based on part numbers

Part number	WholesalerD	WholesalerE	Outcome
part1	None	None	Shipped by Wholesaler A
part2	Accept and Ship	None	Shipped by WholesalerD
part3	Accept and Cancel	Accept and Ship	Shipped by WholesalerE
part4	Decline	Accept and Cancel	Not shipped
part5	Decline	Decline	Not shipped

The request to supply the part is a one-way operation as are the responses from the wholesalers. This means that each call must contain the context from the originator, so that the state machine can correlate the response into the correct instance. All calls to and from Wholesalers D and E therefore contain the orderID, which is the correlation used by the state machine.

Both wholesalers expose their services as Web services. They also call the business state machine using a Web service.

Archived

Because Wholesalers D and E represent separate business entities, we build them in their own individual modules. This means that many components are duplicated, which is more representative of a real world situation, where the implementation is hidden behind the Web service.

Figure 11-42 shows the two wholesalers in context of the overall solution.

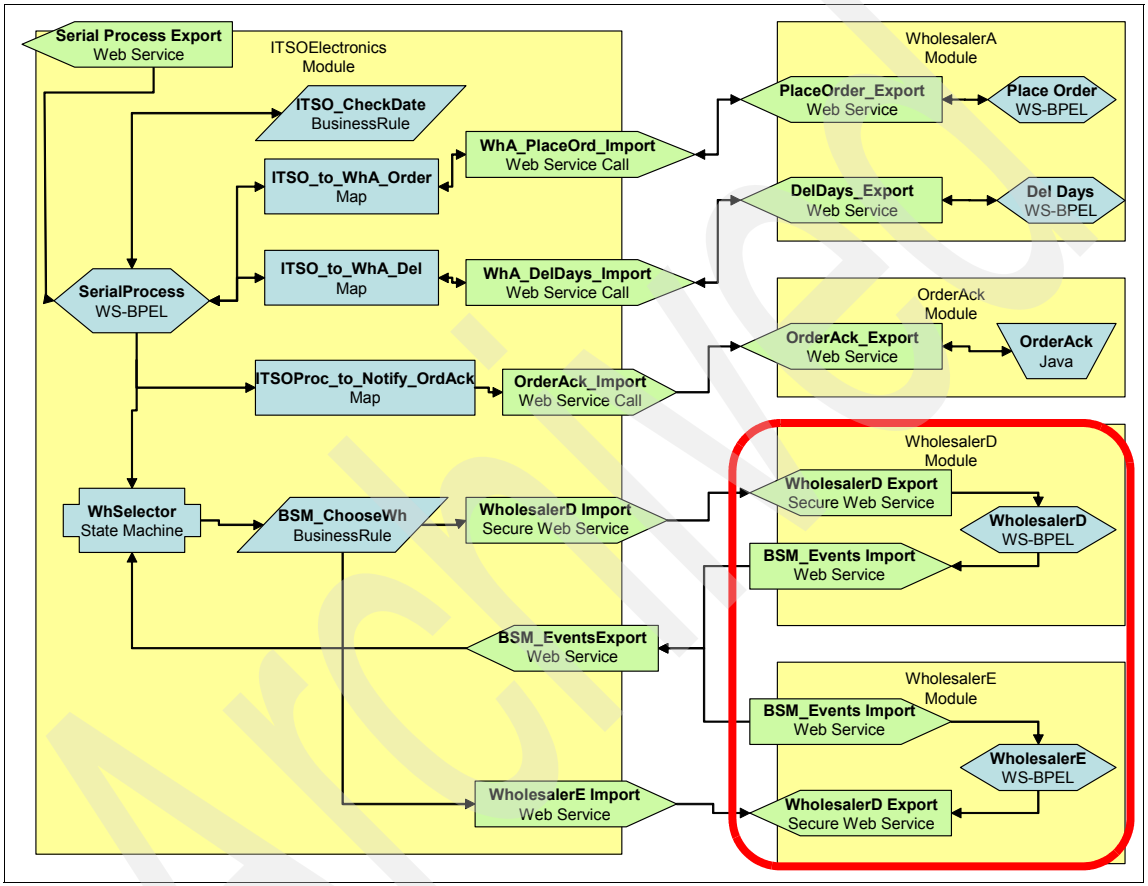


Figure 11-42 Overview of the scenario showing the interlinked components and modules

Archived

Creating Wholesalers D and E

Because the two wholesalers are nearly identical, we create Wholesaler D first in a step-by-step fashion and then create Wholesaler E reusing what we can from Wholesaler D.

1. Create a new WholesalerD module and copy any artifacts that we can reuse from other modules into the WholesalerD module. This creates the interfaces and data types.
2. Create the business logic for WholesalerD.
3. Reuse the WholesalerD artifacts for WholesalerE.
4. Change the business logic for WholesalerE to reflect the different part numbers.

Creating the WholesalerD module

To simplify the implementation for this scenario, we decided to have the business state machine and the wholesalers use the same interfaces, eliminating the need for mapping. This means that we can reuse the interfaces and business objects that we created for the business state machine.

Create the artifacts listed in Table 11-11 by copying them from the ITSOElectronics module to the existing WholesalerD module.

Table 11-11 ITSOElectronics artifacts reused in the WholesalerD module

Name	Namespace	Type
BSMEvents_I	http://ITSOElectronics/BSMEvents_I	Interface
BSMOrderReq_I	http://ITSOElectronics/BSMOrderReq_I	Interface
ITSOOrdPart	http://ITSOElectronics	Data Types
ITSOOrdWhResponse	http://ITSOElectronics	Data Types
ITSOProcessOrder	http://ITSOElectronics	Data Types

Note: These artifacts, although not modified in the WholesalerD module, are copied rather than referenced in order to enforce the isolation of the implementation of the WholesalerD module.

Archived

Creating the business logic for WholesalerD

We create the business logic as a WS-BPEL process. We need to ensure that the WholesalerD process is long-running because we use the *wait* activity to pause the process to simulate waiting for responses from the wholesaler. The wait activity requires that the process be long running.

Follow these steps:

1. Create a new business process in the WholesalerD module and name it `WholesalerD`. Select the interface **BSMOrderReq_I** and the operation **requestOrder** when prompted.
2. Add a Reference Partner named `ITSO_BSM`. For Reference Interface (for the partner), select **BSMEvents_I**. This reference interface is used to send responses back to the business state machine (Figure 11-43).

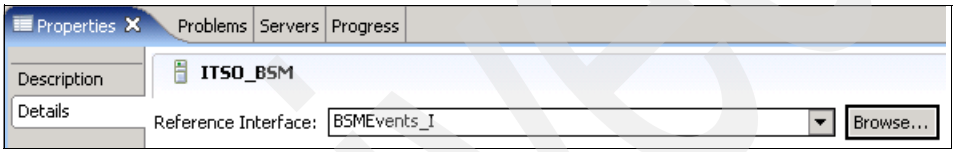


Figure 11-43 Reference partner ITSO_BSM

3. Verify that the business process already defines a variable named `order` of type `ITSOProcessOrder`.
4. Click the canvas of the WholesalerD business process.
   a. Click the **Properties** tab and the **Details** subtab.
   b. Select the **Process is long-running** check box as shown in Figure 11-44.

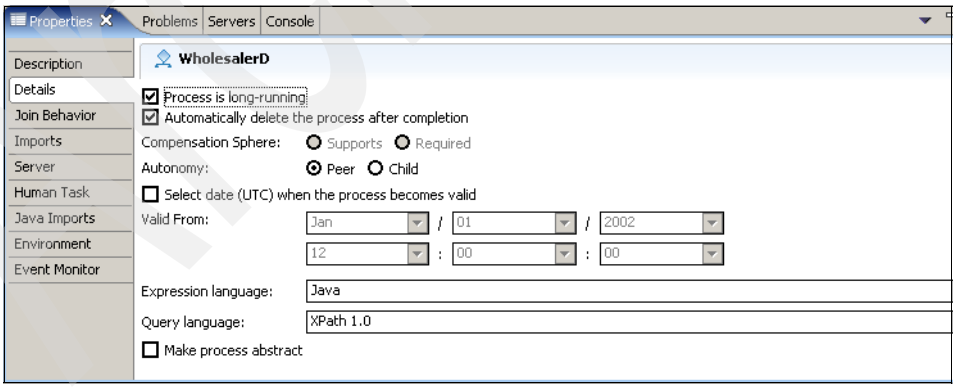


Figure 11-44 Ensuring the process is long-running

5. Create the WS-BPEL diagram as shown in Figure 11-45. Use the information in Table 11-12 and Table 11-14, as well as the visual snippet instructions to help complete the diagram.

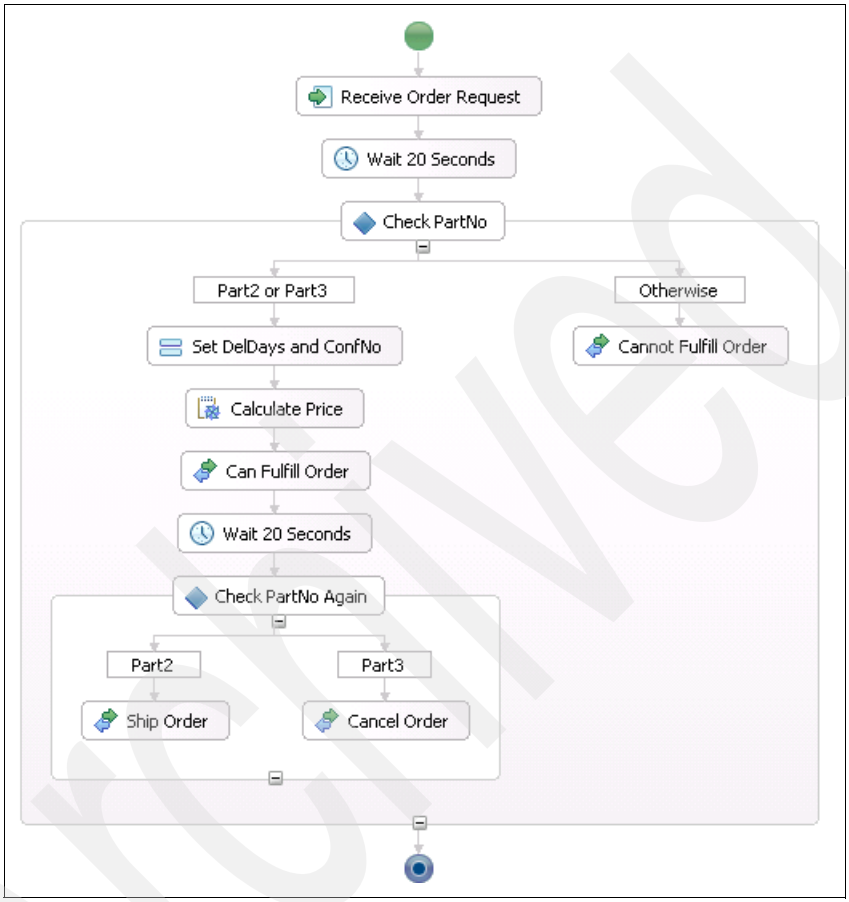


Figure 11-45 WS-BPEL process of WholesalerD

The WS-BPEL diagram contains the activities shown in Table 11-12. This shows the type of each activity from Figure 11-45.

Table 11-12 Activity names and types of the WS-BPEL process diagram

Name	Type
Receive Order Request	Receive
Wait 20 Seconds	Wait
Check PartNo	Choice

Archived

Name	Type
Part2 or Part3	Choice case
otherwise	Choice otherwise
Set DelDays and ConfNo	Assign
Calculate Price	Snippet
Can Fulfill Order	Invoke
Cannot Fulfill Ordert	Invoke
Check PartNo Again	Choice
Part2	Choice case
Part3	Choice case
Ship Order	Invoke
Cancel Order	Invoke

6. Set the Set DelDays and ConfNo assign node as shown in Table 11-13.

Table 11-13 Details of the assign activity Set DelDays and ConfNo

Name	Order	From type	From	To type	To
Set DelDays and ConfNo	1	Fixed value	3	Variable	Order.WhResponse. bestWholesaleDays
	2	Fixed value	order1976	Variable	Order.WhResponse. confNo

7. Implement the Calculate Price snippet as shown in Figure 11-46.

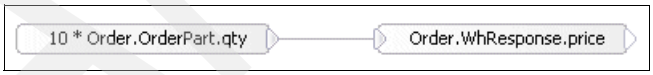


Figure 11-46 Calculate Price Snippet

Archived

8. Table 11-14 shows the details of the invoke activities. Use this information to populate the invoke activity property details.

Table 11-14 The invoke activity property details

Name	Partner	Interface	Operation	Input	Variable
Can Fulfill Order	ITSO_BSM	BSMEvents_I	wholesalerCan Fulfill	canfulfillNotification	Order
Cannot Fulfill Order	ITSO_BSM	BSMEvents_I	wholesalerCannot Fulfill	cannotfulfillNotification	Order
Ship Order	ITSO_BSM	BSMEvents_I	wholesaler Ships	shipmentNotification	Order
Cancel Order	ITSO_BSM	BSMEvents_I	wholesaler Cancels	orderCancellation	Order

9. Configure the two Wait 20 Seconds wait activities as follows (follow the instructions for both wait activities):
   a. Select the **Wait** activity.
   b. Click the **Properties** tab and the **Details** subtab.
   c. Click **Create a New Condition**.
   d. Click the **Duration** radio button.
   e. Create a visual snippet as shown in Figure 11-47 to return a value of 20 to the condition. This ensures that the wait activity waits for 20 seconds. Both wait activities in the diagram are identical.

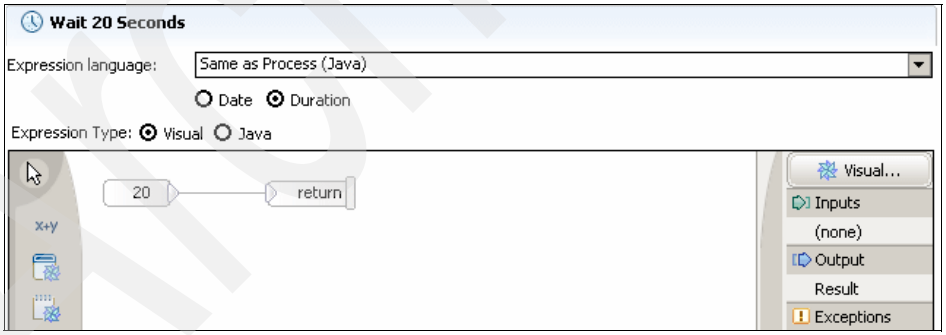


Figure 11-47 Wait snippet

10. Refer to Figure 11-48 to create a visual snippet for the choice case activity Part2 or Part3. This snippet returns *true* if the part number stored in Order.OrderPart.partNo is either part2 or part3.

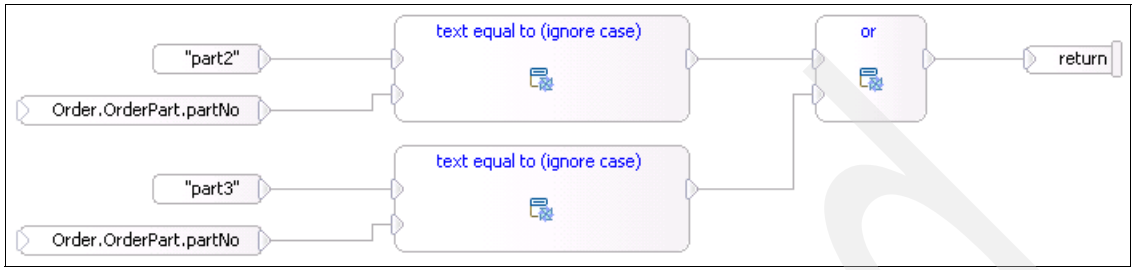


Figure 11-48 Choice case Part2 or Part3

11. Refer to Figure 11-49 to create visual snippet for the choice case activity `Part2`. This snippet returns *true* if the part number stored in Order.OrderPart.partNo is part2.

Figure 11-49 Choice case Part2

12. Refer to Figure 11-50 to create visual snippet for the choice case activity Part3. This snippet returns *true* if the part number stored in Order.OrderPart.partNo is part3.

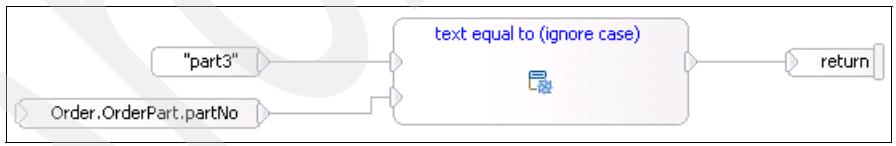


Figure 11-50 Choice case Part3

13. Press Ctrl+S to save the business process.

Archived

You should now see the artifacts as in Figure 11-51 in module WholesalerD in the Business Integration view.

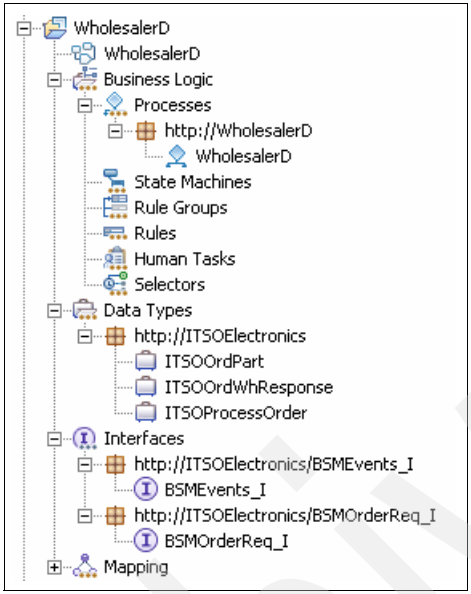


Figure 11-51 Artifacts in the module WholesalerD

Test point: At this point, we recommend that you perform a unit test of the WholesalerD module to ensure that it functions correctly, especially since it is used as the basis for WholesalerE. For information about the guidelines, see 11.4, “Runtime guidelines” on page 312.

Creating the WholesalerE module

Build WholesalerE in the same way as WholesalerD, except for the Business Process, which is called *WholesalerE* and has different part numbers in the choices.

Note: It is not currently possible to copy and paste the process diagram directly from WholesalerD to WholesalerE, or to copy individual activities from one to another. However, it is possible to copy visual snippets from an activity in one module to an activity in another module, using the clipboard.

Archived

1. Copy all of the interfaces and data types from the WholesalerD module to the WholesalerE module.
2. In the module WholesalerE, define a new business process called `WholesalerE`. This business process should specify an interface of **BSMOrder_Req_I** and an operation of **requestOrder**. Because the business process will include a wait activity, set the process to be long-running.
3. In the WholesalerE business process, define a new Reference Partner named `ITSO_BSM`, specifying the reference interface **BSMEvents_I**.
4. Create the WS-BPEL for the process WholesalerE as it is displayed in Figure 11-52. The interfaces, references, variables, and most of the activities are identical to those for WholesalerD. The only difference is the part numbers in the choice cases.

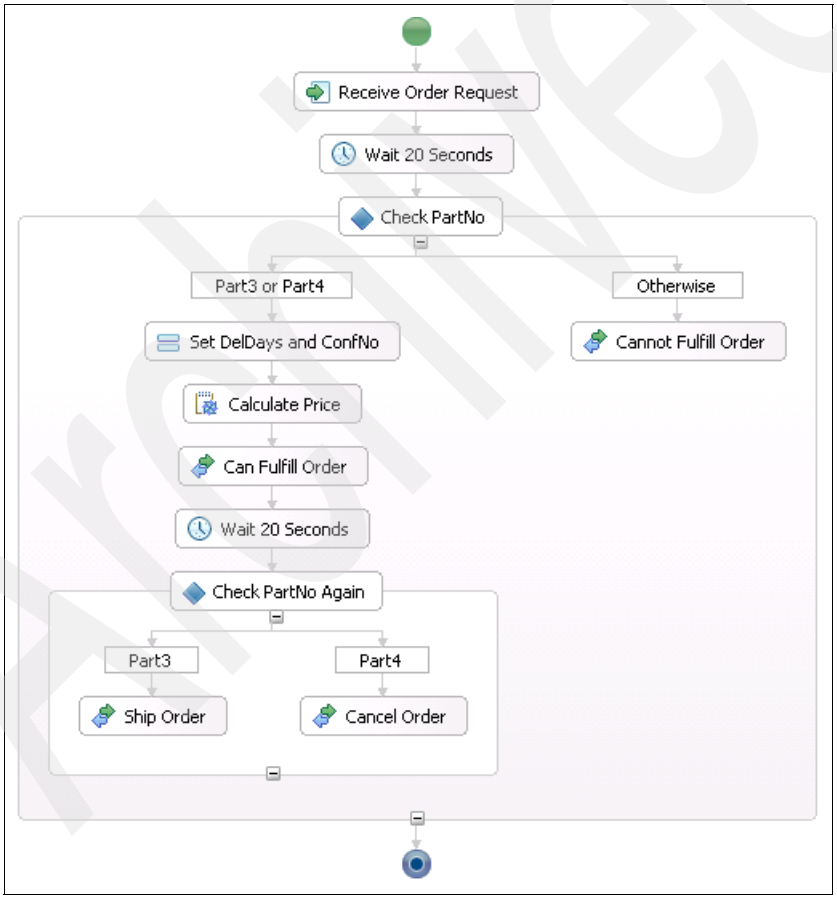


Figure 11-52 WS-BPEL Process of WholesalerE

Table 11-15 also shows the WholesalerE WS-BPEL activities. Note that only the choice cases have changed from WholesalerD.

Table 11-15 Activity names and types

Name	Type
Receive Order Request	Receive
Wait 20 Seconds	Wait
Check PartNo	Choice
Part3 or Part4	Choice case
otherwise	Choice otherwise
SetDelDays and ConfNo	Assign
Calculate Price	Snippet
Can Fulfill Order	Invoke
Cannot Fulfill Ordert	Invoke
Check PartNo Again	Choice
Part3	Choice case
Part4	Choice case
Ship Order	Invoke
Cancel Order	Invoke

5. Create a new condition for each wait activity to wait for a duration of 20 seconds (Figure 11-53).

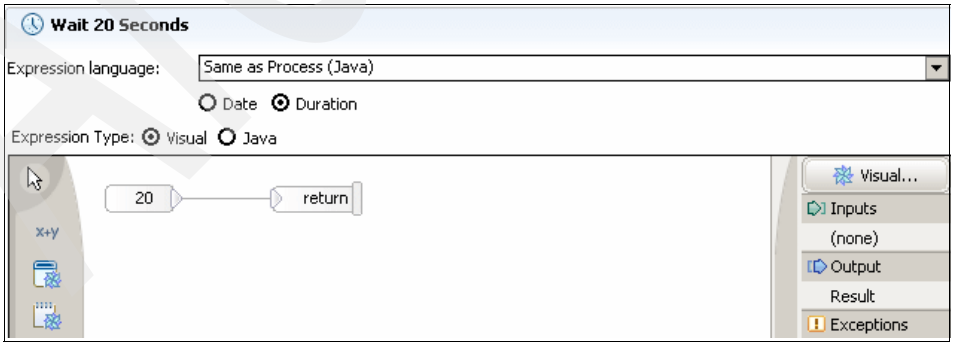


Figure 11-53 Wait snippet

Archived

6. Set the Set DelDays and ConfNo assign node as shown in Table 11-16.

Table 11-16 Details of the assign activity Set DelDays and ConfNo

Name	Order	From type	From	To type	To
Set DelDays and ConfNo	1	Fixed value	5	Variable	Order.WhResponse. bestWholesaleDays
	2	Fixed value	order7	Variable	Order.WhResponse. confNo

7. Implement the Calculate Price snippet as shown in Figure 11-54.

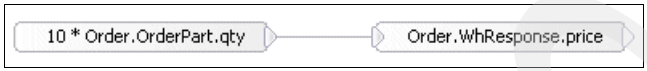


Figure 11-54 Calculate Price snippet

8. Table 11-14 on page 291 shows the details of the invoke activities, although this table was for WholesalerD it equally applies to WholesalerE. Use the information in this table to populate the invoke activity property details.
9. Refer to Figure 11-55 to create a visual snippet for the choice case activity Part3 or Part4. This snippet returns *true* if the part number stored in Order.OrderPart.partNo is either part3 or part4.

> **Tip:** Although it is not possible to copy complete activities from one process to another, it is possible to copy visual snippets. Press Ctrl+A to select all the snippets in the relevant wholesalerD activity, and then press Ctrl+C to copy them. In the snippet editor for the corresponding wholesalerE activity, press CTRL+V to paste the completed snippets. If you use this tip, remember to change the part numbers in the snippets when copying from wholesalerD to wholesalerE.

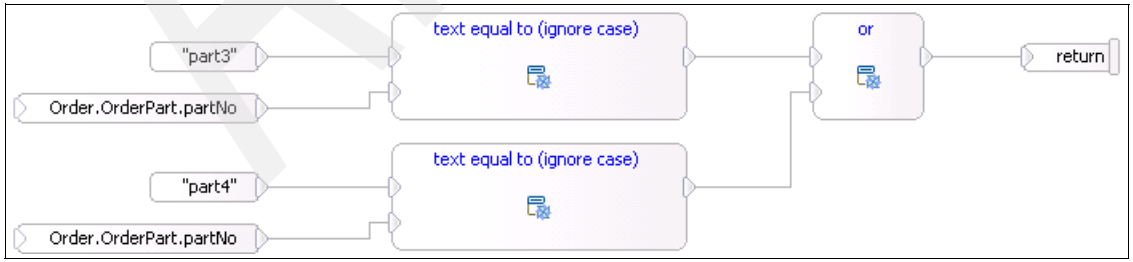


Figure 11-55 Choice case Part3 or Part4

10. Refer to Figure 11-56 to create a visual snippet for the choice case activity Part3. This snippet returns *true* if the part number stored in Order.OrderPart.partNo is part3.

Figure 11-56 Choice case Part3

11. Refer to Figure 11-57 to create a visual snippet for the choice case activity Part4. This snippet returns *true* if the part number stored in Order.OrderPart.partNo is part4.

Figure 11-57 Choice case Part4

12. Press Ctrl+S to save the completed business process.

The artifacts in Figure 11-58 should now be in the Business Integration view of the WholesalerE module.

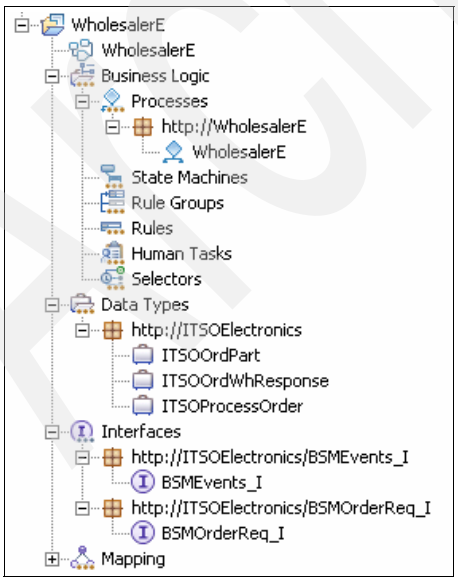


Figure 11-58 Artifacts in WholesalerE

Archived

The implementations for both WholesalerD and WholesalerE logic are now complete.

11.3.7 Linking WholesalerD and WholesalerE to the state machine

Currently, the WholesalerD and WholesalerE modules only contain interfaces, business objects, and process logic. To use them with WebSphere Process Server, we must create assembly diagrams for each of them.

The business state machine and a wholesaler have a two-way interdependent relationship. Each relies on the other for a service. The business state machine calls the wholesaler to place an order request. The wholesaler calls the business state machine to update it with the status of the order request. This means that we must link the business state machine and a wholesaler in a looped fashion.

Plan for linking wholesalers and the business state machine

To link the wholesalers and the business state machined:

1. Expose the business state machine as a Web service.
2. Copy the Web service reference from the ITSOElectronics module to the wholesaler modules.
3. Import the business state machine Web service into the wholesaler assembly diagrams.
4. Export the wholesaler business logic as Web services.
5. Copy the Web service reference from the wholesaler modules to the ITSOElectronics.
6. Import the wholesaler Web services into the ITSOElectronics assembly diagram and wire them to the business state machine.

Each of these steps is explained further in the sections that follow.

Exposing the business state machine as a Web service

Complete these steps:

1. In the ITSO Electronics assembly diagram, right-click **WhSelector** and select **Export** → **Web Service Binding**.
2. In the Binding File Generation alert box, click **Yes** to generate the Web Services Description Language (WSDL) file.

3. In the Select Transport window, select **soap/http** and click **OK**.

   The WhSelectorExport component is created. Figure 11-59 shows how the assembly diagram looks.

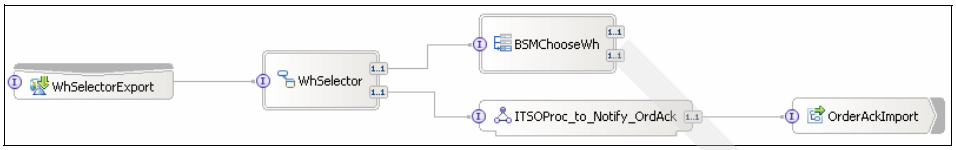


Figure 11-59 Assembly diagram with wired export component

4. Press Ctrl+S to save the ITSOElectronics assembly diagram.

> **Tip:** You may want to experiment with the auto-layout settings provided with WebSphere Integration Developer for the assembly diagrams. To access the settings, right-click the assembly diagram canvas and select **Arrange Contents Automatically**, **Layout Contents**, or both from the menu.

Copying the state machine Web service reference to the wholesalers

The WSDL file generated in the previous step is in the ITSOElectronics module Web Service Ports folder. WholesalerD and WholesalerE need this file, which must be copied into their modules. Follow these steps:

1. Copy the artifact **WhSelectorExport_BSMEvents_IHttpPort** from the Web Service Port folder of the ITSOElectronics module to the WholesalerD module.
2. Repeat this process to copy **WhSelectorExport_BSMEvents_IHttpPort** to the WholesalerE module.

> **Important:** The WSDL file in the Web Service Ports folder must be copied into the modules that want to call that Web service. It is not sufficient to point an import to a Web services port in another module, for example using the Wire to New Import right-click menu. WebSphere Process Server will fail to find the reference at run time.

Now WholesalerD and WholesalerE are able to reference the Web service provided by the business state machine WhSelector. We can now create assembly diagrams for both wholesalers, which can import the business state machine Web service.

Archived

Adding the state machine Web service into the wholesalers

Perform the following steps to add the business state machine Web service as a Web service import into both wholesalers. In this section, we work with WholesalerD.

1. Open the WholesalerD assembly diagram.
2. Drag the business process **WholesalerD** onto the assembly diagram canvas.
3. Drag the Web service port **WhSelectorExport_BSMEvents_IHttpPort** onto the canvas.

> **Important:** Make sure that you drag the Web service port from the WholesalerD module to the canvas, not from the ITSOElectronics module.

4. In the Component Creation window, select **Import with Web Service Binding** and click **OK**.
5. Rename Import1 to `BSMEventsImport`.
6. Right-click the **WholesalerD** process component and select **Wire to Existing**.

Figure 11-60 shows the WholesalerD assembly diagram with the wired import component.

Figure 11-60 Assembly diagram with the wired import component

Exporting the wholesaler process as a Web service

Now that WholesalerD can call the business state machine service, we must export the WholesalerD process as a Web service so that the business state machine can call it.

1. Right-click the **WholesalerD** process component and select **Export** → **Web Service Binding**.
2. In the Binding File Generation alert box, click **Yes** to generate the WSDL file.
3. In the Select Transport window, select **soap/http** and click **OK**.

Archived

The WholesalerDExport component was created. Figure 11-61 shows the assembly diagram with the wired export component.

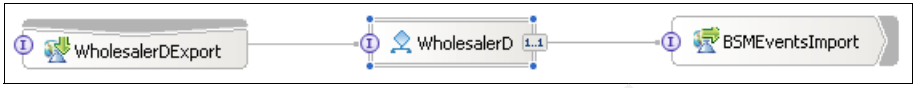


Figure 11-61 Assembly diagram with wired export component

4. Press Ctrl+S to save the assembly diagram.
5. Create the WholesalerE assembly diagram as we have done for WholesalerD. The assembly diagram of WholesalerE must resemble the example in Figure 11-62.

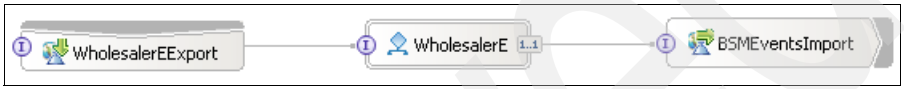


Figure 11-62 Assembly diagram of WholesalerE

We have now generated a Web service port for each WholesalerD and WholesalerE. The port is needed by the business state machine so that it can consume the services provided.

Copying the Wholesaler Web services references to ITSOElectronics

Copy the service port files for both Wholesalers D and E:

1. Copy the **WholesalerDExport_BSMOrderReq_IHttpPort** generated WSDL file in the Web Service Port folder of the WholesalerD module to the ITSOElectronics module. We need the WSDL file in the ITSOElectronics module so the business state machine can use it.
2. Copy the **WholesalerEExport_BSMOrderReq_IHttpPort** generated WSDL file in the Web Service Port folder of the WholesalerD module to the ITSOElectronics module.

Importing the Wholesaler Web services and wiring to the state machine

Now we can import the Wholesalers D and E Web services into the ITSOElectronics assembly diagram. We can then wire the services to the business state machine.

1. Open the ITSOElectronics assembly diagram.
2. Drag the Web service port **WholesalerDExport_BSMOrderReq_IHttpPort** onto the diagram canvas.
3. Select **Import with Web Service Binding** from the window.

4. Rename the import to `WholesalerDImport`.
5. Drag the Web service port **WholesalerEExport_BSMOrderReq_IHttpPort** onto the diagram canvas.
6. Select **Import with Web Service Binding** from the window.
7. Rename the import to `WholesalerEImport`.
8. In the assembly diagram, right-click the **BSMChooseWh** rule group and select **Wire to Existing**.
9. The wiring cannot take place automatically because Wholesalers D and E have the same interface.
   a. In the Advanced Wiring window, in the Wire source box, highlight **WholesalerD**. Then in the Wire target box, select the **WholesalerDImport** check box.
   b. In the Wire source box, highlight **WholesalerE**, and in the Wire target box, select the **WholesalerEImport** check box.
   c. Click **OK**.

Figure 11-63 shows how the assembly diagram looks now.

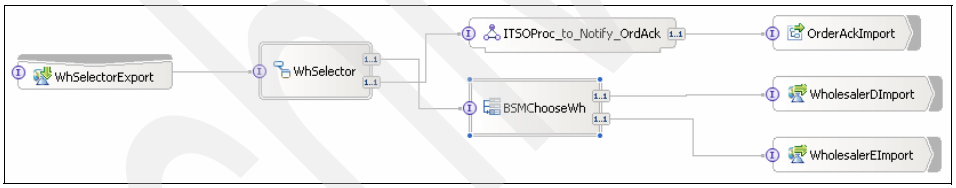


Figure 11-63 Assembly diagram with imports

Test point: At this point, we recommend that you test the assembly, especially to ensure that all of the Web service imports and exports are correctly configured. Refer to 11.4, “Runtime guidelines” on page 312, for guidelines.

11.3.8 Creating the Exposed Serial business process

In the ITSOElectronics module, we create a business process which will first try to fulfill an order from Wholesaler A. If this is not possible, it will call the business state machine. The business state machine will try to fulfill the orders from WholesalerD and WholesalerE.

This business process fits into our overall scenario as shown in Figure 11-64.

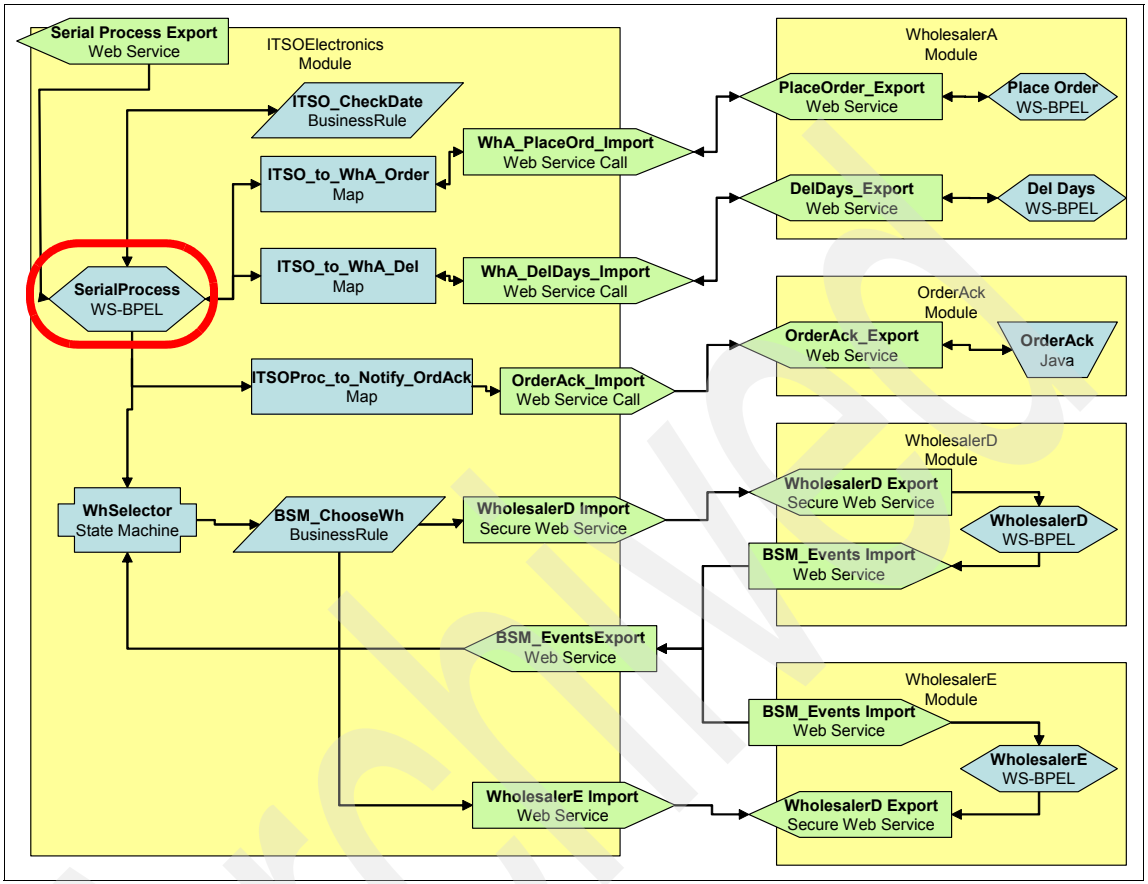


Figure 11-64 Overview of the scenario showing the interlinked components and modules

Archived

The logic for *Can wholesalerA fulfill the order* is reused from the Serial Process scenario.

1. In the ITSOElectronics module create a new business process called `SerialProcess` with the interface `ITSOOrdReq_I`.
2. Add five reference partners as shown in Table 11-17.

Table 11-17 Reference partners

Name	Interface
WhAOrderPartner	ITSOPlaceOrd_I
WhADeliveryPartner	ITSODelDays_I
DateCheckPartner	ITSODateCheck_I
OrderNotifyPartner	ITSOOrdNotify_I
StateMachinePartner	BSMEvents_I

3. Add three variables as shown in Table 11-18.

Table 11-18 Variables

Name	Data type
currentOrder	ITSOProcessOrder
wholesalerResponse	ITSOProcessOrder
isDateOK	boolean

Archived

4. Create the WS-BPEL business process as shown in Figure 11-65.
   Table 11-19, Table 11-20, and Table 11-21 describe the required activities.

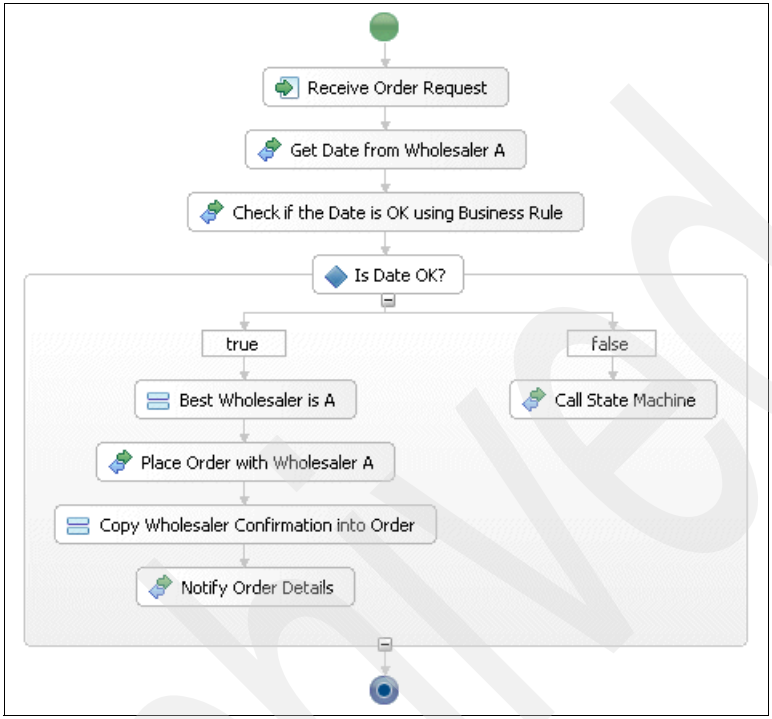


Figure 11-65 Exposed Serial Process

Table 11-19 lists the activity names and types.

Table 11-19 Activity names and types

Name	Type
Receive Order Request	Receive
Get Date From WholesalerA	Invoke
Check if the Date is OK using Business Rule	Invoke
Is Date OK?	Choice
True	Choice case
False	Choice case
Best Wholesaler is A	Assign
Call State Machine	Invoke

Archived

Name	Type
Place Order with WholesalerA	Invoke
Copy Wholesaler Confirmation into Order	Assign
Notify Order Details	Invoke

Table 11-20 lists the details of the invoke activities.

Table 11-20 Details of the invoke activities

Name	Partner	Interface	Operation	Input variable	Output variable
Get Date From WholesalerA	WhADelivery Partner	ITSODelDays_I	getDelivery Days	currentOrder	wholesaler Response
Check if the Date is OK using Business Rule	DayCheck Partner	ITSODate Check_I	checkDelivery Date	wholesaler Response	isDateOK
Place Order with WholesalerA	WhAOrder Partner	ITSO_Place Ord_I	placeOrder	currentOrder	wholesaler Response
Notify Order Details	OrderNotify Partner	ITSOOrd Notify_I	notifyDelivery Date	currentOrder	N/A
Call State Machine	StateMachine Partner	BSMEvents_I	findAcceptable Wholesaler	currentOrder	N/A

Table 11-21 lists the details of the assign activities.

Table 11-21 Details of the assign activities

Name	Order	From type	From	To type	To
Best Wholesaler is A	1	Fixed value	WholesalerA	Variable	currentOrder.Wh Response.best Wholesale
	2	Variable	wholesalerResponse.Wh Response.bestWholesale Days	Variable	currentOrder.Wh Response.best WholesaleDays
Copy Wholesaler Confirmation into Order	1	Variable	wholesalerResponse.Wh Response.confNo	Variable	currentOrder.Wh Response.confNo

Archived

5. Set Receive Order Request to use the variable `currentOrder` as its input (see Figure 11-66).

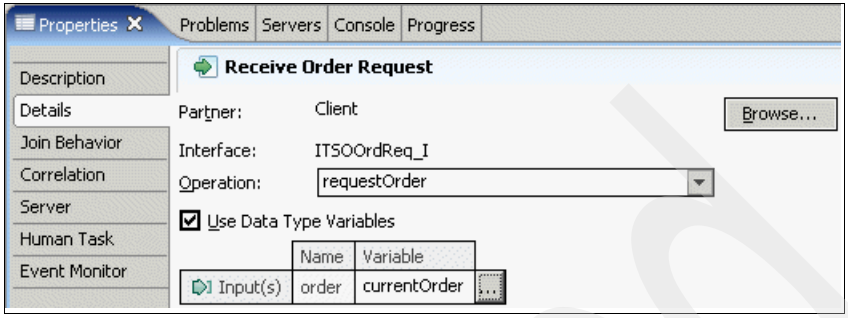


Figure 11-66 Receive Order Request input

6. Create the Choice case Activity `IsDateOK true`, as shown in Figure 11-67.

Figure 11-67 Choice case true

7. Create the Choice case Activity `IsDateOK false`, as shown in Figure 11-68.

Figure 11-68 Choice case false

8. Press Ctrl+S to save the business process.
9. Close the Business Process Editor.

> **Test point:** At this point, we recommend that you perform a unit test on the business process to ensure that it functions correctly in isolation.

Archived

11.3.9 Linking the business process in the assembly diagram

Now that the business process is built, we link it in the assembly diagram to the state machine as explained in the following steps.

Figure 11-69 shows how the ITSOElectronics assembly diagram currently looks.

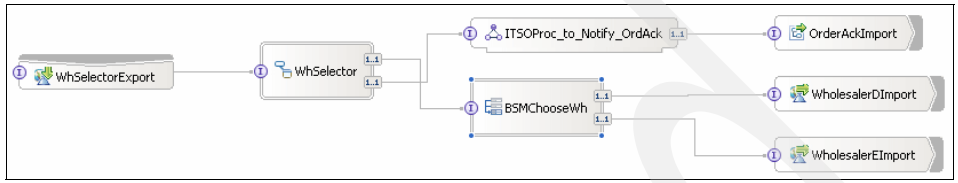


Figure 11-69 Assembly diagram without the business process

1. We now need to add the business process and wire it.
   a. Open the ITSOElectronics assembly diagram.
   b. Drag the business process **SerialProcess** to the assembly diagram.
   c. In the assembly diagram, right-click the **SerialProcess** and select **Wire to Existing**.

   Figure 11-70 shows how the assembly diagram looks after completing these steps.

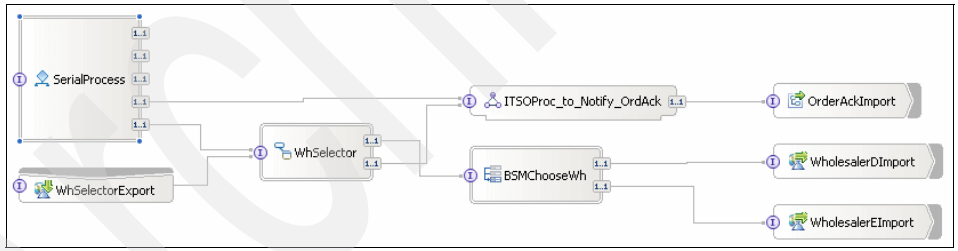


Figure 11-70 Assembly diagram with the wired business process

2. Now that we have the business process in place, we need to import and wire the WholesalerA Web service into the ITSOElectronics module. The references are already in the library ready to go. We need to import both the getDeliveryDate and placeOrder services.
   a. Drag the **WholesalerA_GetDeliveryDateExport_WhDelReq Web_IHttpPort** Web service port from ITSOLibrary to the ITSOElectronics assembly diagram and select **Import with Web Service Binding**.
   b. Rename the new Import component Import1 to `WhA_DelDayImport`.

Archived

c. Drag the **WholesalerA_PlaceOrderExport_WhPlaceOrdWeb_IHttpPort** Web service port from the ITSOLibrary to the ITSOElectronics assembly diagram and select **Import with Web Service Binding**.

d. Rename the new Import component Import1 to `WhA_PlaceOrderImport`.

3. Because the SerialProcess uses different interfaces to Wholesaler A, we need to use interface maps to transform the data between the two interface formats.

   a. In ITSOElectronics, drag the interface maps **ITSO_to_WhADel** and **ITSO_to_WhA_ Order** to the ITSOElectronic assembly diagram.

   b. In the assembly diagram, right-click **ITSO_to_WhADel** and select **Wire to Existing**.

   c. In the assembly diagram, right-click **ITSO_to_WhA_Order** and select **Wire to Existing**.

   Figure 11-71 shows the assembly diagram after completing these steps.

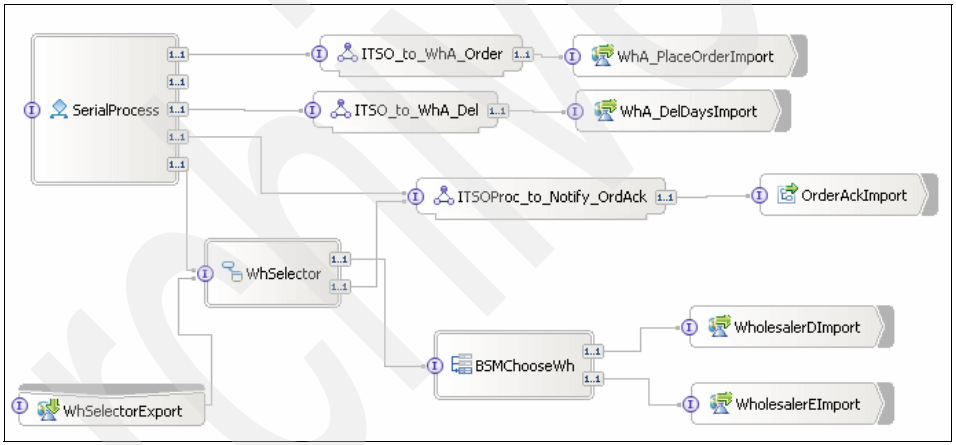


Figure 11-71 Assembly diagram with the wired wholesalers

4. Press Ctrl+S to save the assembly diagram.

Archived

11.3.10 Adding the ITSO_CheckDate business rule

The Exposed Serial Process also needs the ITSO_CheckDate business rule. The rule helps to decide whether Wholesaler A (the preferred supplier) can deliver within the agreed lead time or whether the business state machine should find an alternative. The ITSO_CheckDate business rule is re-used from a previous scenario. Figure 11-72 shows its position in the overall scenario.

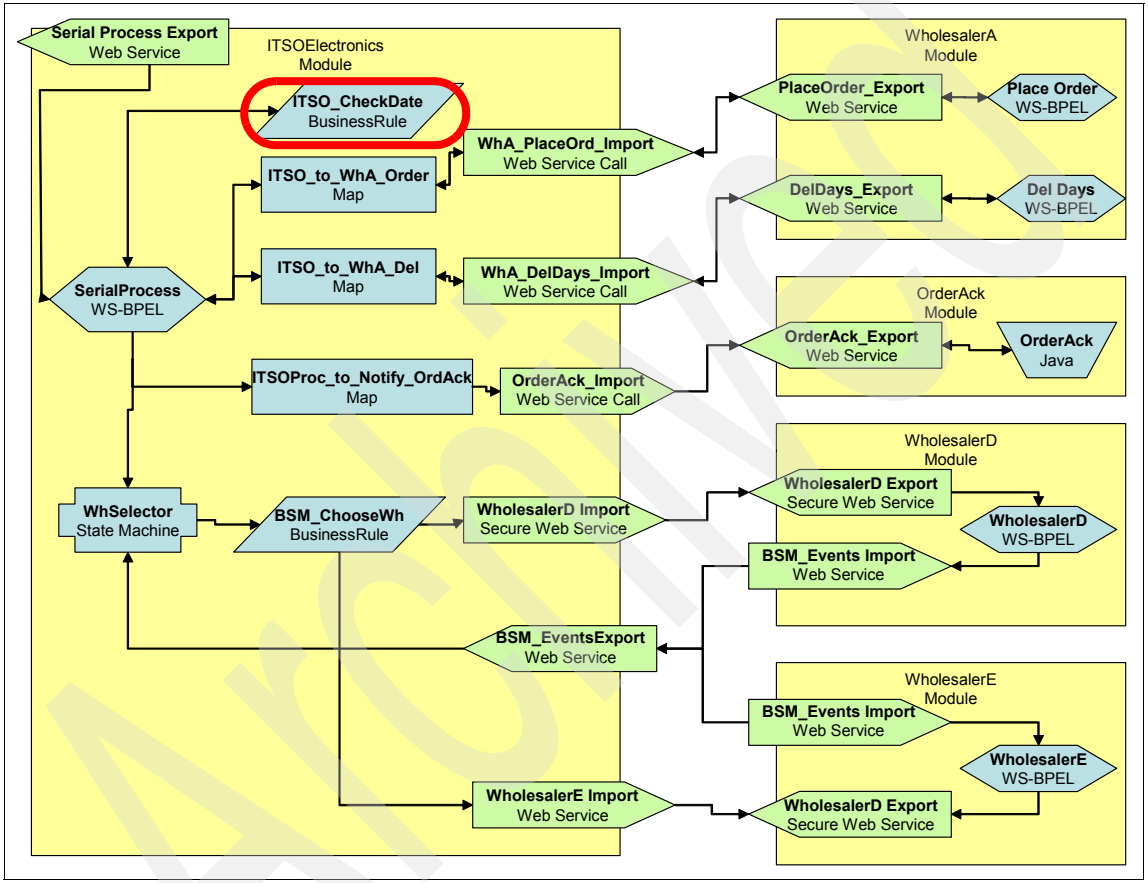


Figure 11-72 Overview of the scenario showing the interlinked components and modules

Complete the following steps:

1. Open the ITSOElectronic assembly diagram.
2. Drag the rule group **ITSO_CheckDate** from the ITSOElectronics module to the ITSOElectronic assembly diagram.

3. Right-click the **ITSO_CheckDate** rule group and select **Wire to Existing**.

   Figure 11-73 shows the changes to the assembly diagram.

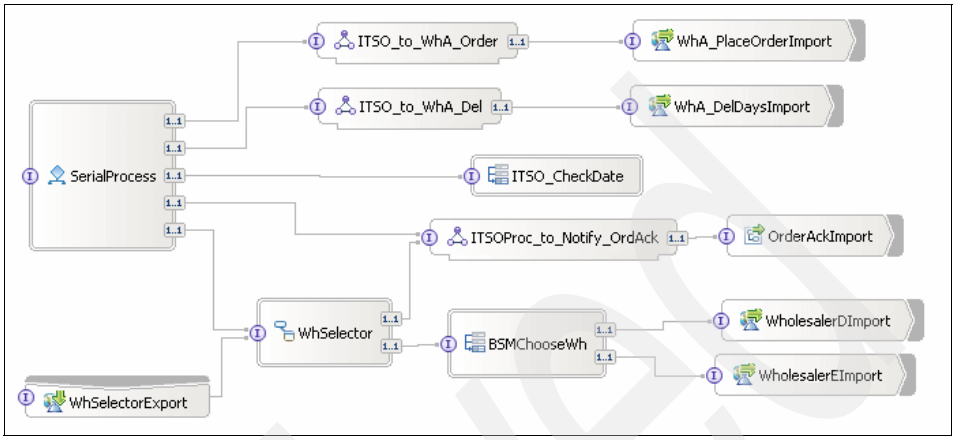


Figure 11-73 Assembly diagram with the wired business rule

4. Press Ctrl+S to save the assembly diagram.

> **Tip:** You can move the references within a component up and down for a better layout of the assembly diagram so that the wires do not cross. Drag the **Reference 1.1** symbol to the desired position.
>
> Compare Figure 11-73 to Figure 11-71 on page 309, where the references of the SerialProcess component have been rearranged.

11.3.11 Exposing the business process as a Web service

Currently, the SerialProcess business process is available for testing only by the test client. We now need to expose it as a Web service so that it can be called by any Web service enabled system.

1. Open the ITSOElectronics assembly diagram.
2. Right-click **SerialProcess** and select **Export** → **Web Service Binding**.
3. In the Binding File generation window, select **Yes** to create a WSDL file.
4. Select **soap/http** as the transport.

   Figure 11-73 shows how the assembly diagram looks after completing these steps.

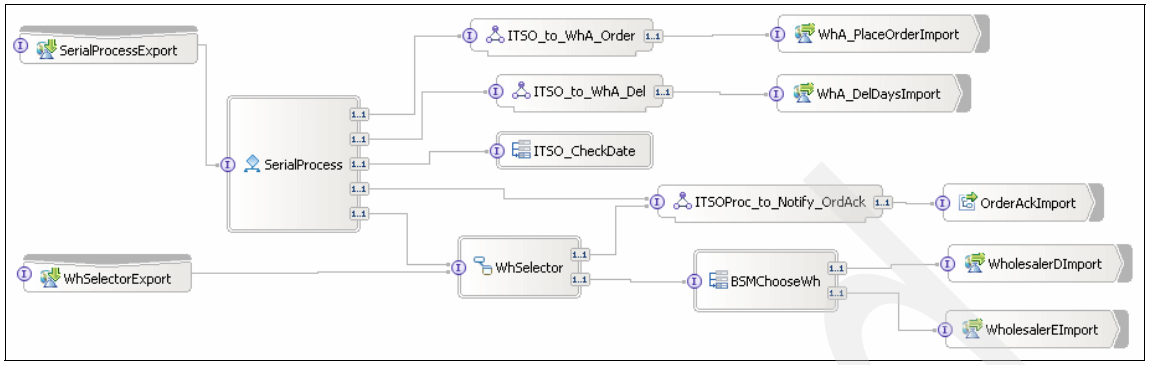


Figure 11-74 Completed assembly diagram

5. Press Ctrl+S to save the assembly diagram.

Notice that the WSDL file has been generated in the Web Service Ports folder of the ITSOElectronics module. Other applications can use this file to call the service that we just exposed, in addition to being callable by other WebSphere Process Server modules.

> **Note:** To call the Web service externally, you must provide all of the components required for the calling application to successfully make the Web service call. In addition to the port file, you must also provide any interface files and business object .xsd files that are referenced. Providing only the port file is often not enough.

11.4 Runtime guidelines

You can add an HTTP server to provide support for SOAP/HTTP. This is necessary when communicating with Web services across an untrusted network such as the Internet. This section presents an overview of how to implement WS-Security and explains how to test the application that we built in this chapter.

11.4.1 Configuring an HTTP server for SSL pass-through

> **Note:** This section requires IBM HTTP Server V6 to be installed.

In the generic profile of the Exposed Direct Connection runtime pattern, a Connector is required in the DMZ. In the product mapping that we selected, this

Connector is implemented with IBM HTTP Server V6, as indicated in Figure 11-75.

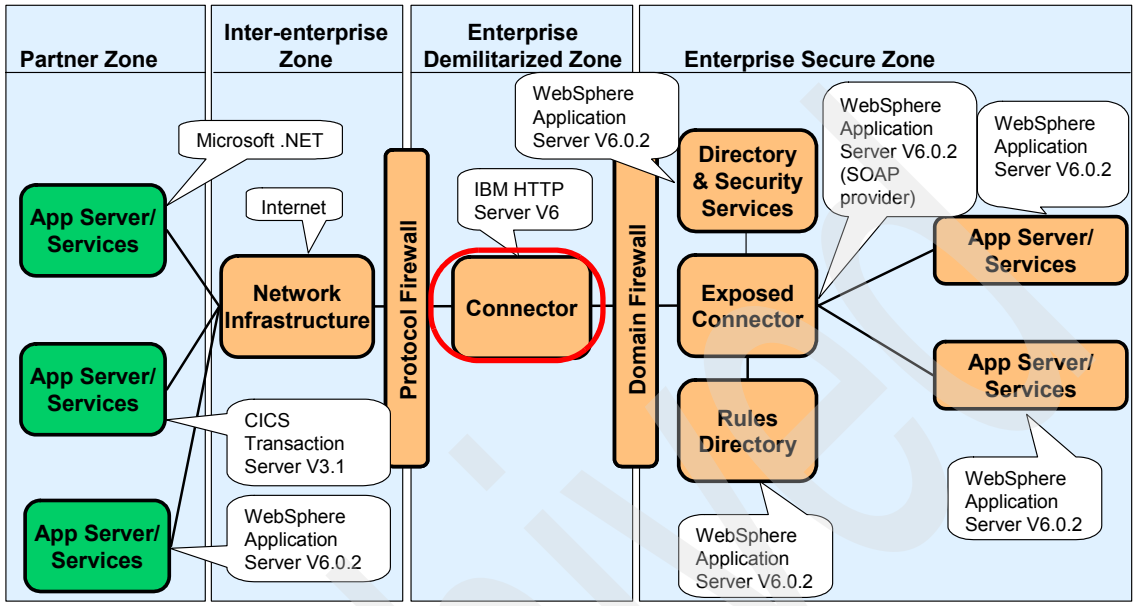


Figure 11-75 Use of IBM HTTP Server as a Connector node

The purpose of the HTTP server in this implementation is to serve as a connector between the applications in the Enterprise Secure Zone and those in the Inter-enterprise Zone.

The HTTP server contains SSL pass-through rules for receiving requests from Web service clients over SOAP/HTTP and forwarding them to the relevant WholesalerD service endpoint URL using SOAP/HTTPS. Figure 11-76 illustrates this.

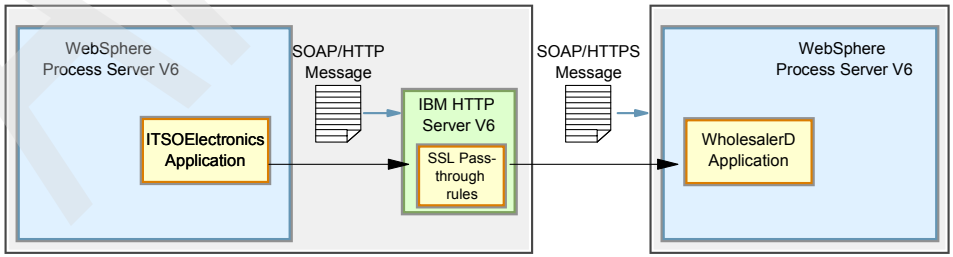


Figure 11-76 Using an IBM HTTP Server for SSL pass-through

Creating a keystore for use by the HTTP server

The secured connection between the HTTP server and the application server requires a certificate key store to store its own private and public key files as well as the public certificate from the Web container key file. We reuse a precreated key store, named *ihs.kdb*. You can find this file along with other essential files in the additional material supplied with this book in \ExposedSerialProcess\keystore\ihsssl.zip. For more information, see Appendix E, “Additional material” on page 689.

Once you obtain the ihsssl.zip file, complete the following steps:

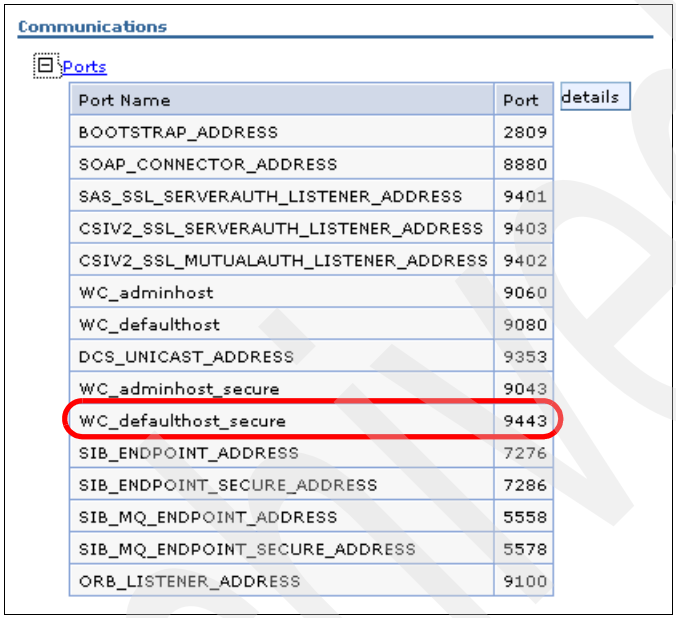
1. On the system where the IBM HTTP Server is installed, create a directory named `C:\keys` (assuming you are using Windows).
2. Unzip all the contents of the ihsssl.zip file into the newly created directory.

> **Note:** If you want to create your own self-signed certificate and use it in the key store instead of using the one supplied with this book, follow the steps listed in the WebSphere Application Server Information Center at:
>
> http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.ihs.doc/info/aes/ae/tihs_certselfsigned.html
>
> If created your own self-sign certificate, ensure that the certificate is imported into the server-based key file. The key file is used by the application server to store private keys for client-certificate authentication purposes. The default location and file name for WebSphere Application Server V6 is WAS_HOME/profiles/*server_name*/etc/DummyServerKeyFile.jks.

3. Log in to the administrative console for the ITSOElectronics server profile.

Archived

4. Determine the TCP/IP port assigned to SSL incoming connections.
   a. Select **Servers** → **Application servers**.
   b. Click **server1**.
   c. Under Communications, expand **Ports**.
   d. Note the value of WC_defaulthost_secure (see Figure 11-77). In our case it is 9443.



Communications

Ports

details

Port Name	Port
BOOTSTRAP_ADDRESS	2809
SOAP_CONNECTOR_ADDRESS	8880
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	9401
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	9403
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	9402
WC_adminhost	9060
WC_defaulthost	9080
DCS_UNICAST_ADDRESS	9353
WC_adminhost_secure	9043
WC_defaulthost_secure	9443
SIB_ENDPOINT_ADDRESS	7276
SIB_ENDPOINT_SECURE_ADDRESS	7286
SIB_MQ_ENDPOINT_ADDRESS	5558
SIB_MQ_ENDPOINT_SECURE_ADDRESS	5578
ORB_LISTENER_ADDRESS	9100

Figure 11-77 Port used to receive SSL requests

Archived

Configuring SSL pass-through in the HTTP server

We need to modify the HTTP server configuration file to enable SSL, specify the keystore to use, and add a ProxyPass statement for the WholesalerD request that will flow through the HTTP server.

1. Open the **httpd.conf** HTTP server configuration file in a text editor. You can find it in the *HTTP_SERVER_HOME*\conf\ directory.
2. Add the statements shown in Example 11-1 to the end of the configuration file. These entries indicate to the HTTP server that incoming HTTP requests are to be encrypted using the keys in the indicated key store file before being routed to the various Manufacturer Web services based on the distinct URIs.

Example 11-1 Additions to httpd.conf

```
SSLProxyEngine on
Keyfile "C:\keys\ihs.kdb"
ProxyPass /WholesalerDWeb/ https://localhost:9443/WholesalerDWeb/
```

Notes:

- Example 11-1 uses the value 9443 for the SSL port on WebSphere Process Server for WholesalerD. Use the correct port number in your environment based on the value of the WC_defaulthost_secure port name.
- In our scenario, because all applications are running on the same server, we use *localhost* as the host name. In a real life scenario, you use the host name of the WholesalerD server.

3. Append the entries shown in Example 11-2 to httpd.conf to load the various code modules necessary for SSL-based routing to work.

Example 11-2 Loading SSL modules to the httpd.conf file

```
LoadModule ibm_ssl_module     modules/mod_ibm_ssl.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_connect_module modules/mod_proxy_connect.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule proxy_ftp_module modules/mod_proxy_ftp.so
```

4. Save and close the httpd.conf file.
5. Restart the HTTP server so the new changes take effect.

Changing the Web service client bindings configuration

We have configured an HTTP server to add SSL to SOAP/HTTP messages sent between ITSOElectronics and WholesalerD. Now we need to make sure that we are using this HTTP server. We do this by modifying the Web service client bindings for WholesalerD.

Perform the following steps in the administrative console of the server where ITSOElectronics is deployed:

1. From the administrative console where the ITSOElectronics enterprise application is installed, select **Applications**, and click **Enterprise Applications**.
2. From the list of installed applications, click **ITSOElectronicsApp**.
3. In the application configuration panel, click the link **EJB Modules**.
4. In the server-module installation binding for an EJB Module panel, in the list of EJB modules, click **ITSOElectronicsEJB.jar**.
5. In the selected EJB module configuration panel, click the link **Web services client bindings**.
6. In the next panel (see Figure 11-78), in the Port Information column in the sca/import/WholesalerDImport row, click **Edit**.

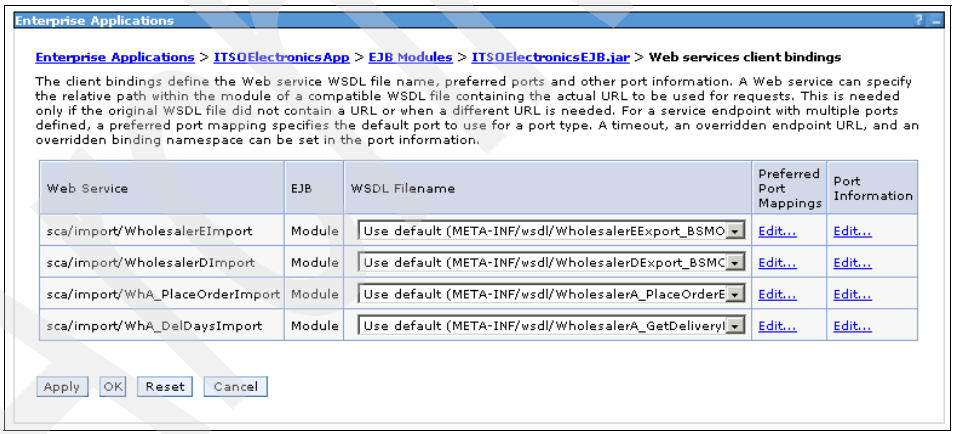


Figure 11-78 Web services client bindings

7. In the Port Information panel, we can specify to override the endpoint URL. Set the Overridden Endpoint URL to `http://localhost/WholesalerDWeb/sca/WholesalerDExport`.

   The Web service client ITSOElectronics will use this URL to contact the Web service. This URL points to the HTTP server, where the request is redirected to the external WholesalerD Web service. See Figure 11-79.

   In our scenario, because ITSOElectronics and the HTTP server are running on the same server, we use localhost as the host name.

   Click **OK**.

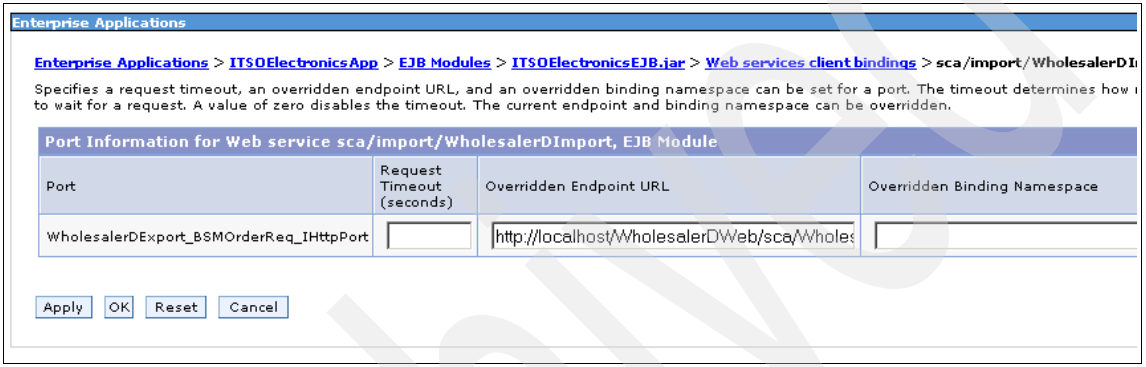


Figure 11-79 Overriding the endpoint URL for WholesalerD

8. Save your changes to the configuration.

> **Note:** These changes will be lost if you republish the ITSOElectronics module.

11.4.2 Using WS-Security

WS-Security provides an additional way to secure Web service calls. In the previous section, we encrypted SOAP/HTTP requests using SSL. Using WS-Security, we can extend this support using for example:

- Integrity, which adds a digital certificate to a SOAP message
- Confidentiality, which encrypts specific parts of a SOAP message (such as the SOAP body) instead of encrypting the entire HTTP message

To configure integrity and confidentiality using WS-Security, you must define the resources shown in Figure 11-80.

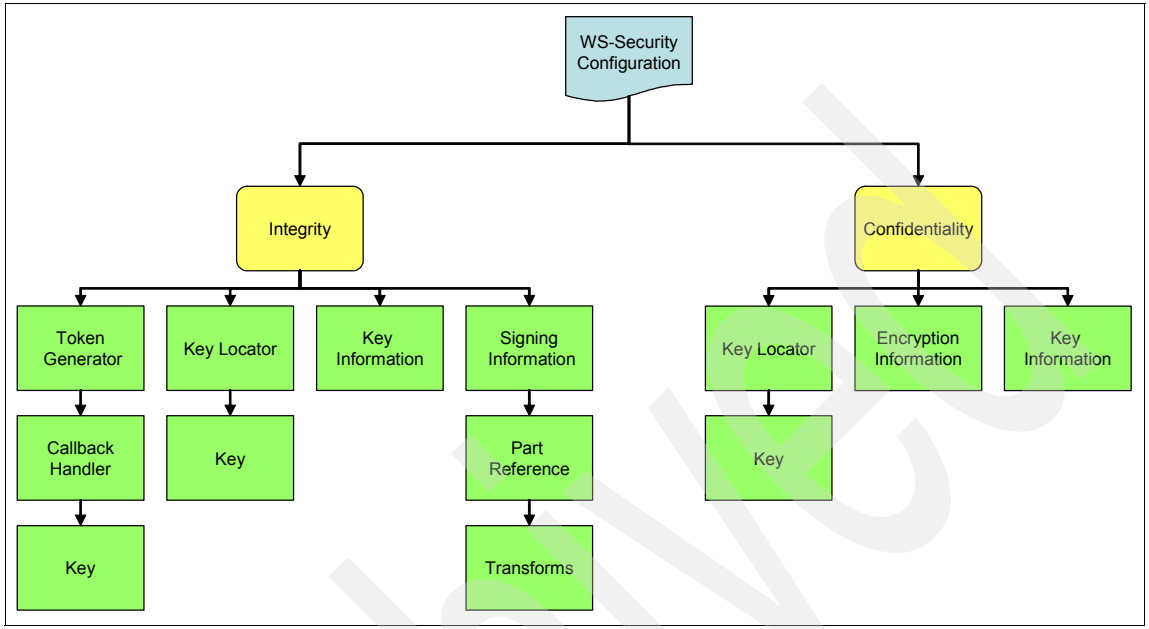


Figure 11-80 WS-Security binding elements

To enable WS-Security, you perform the following steps:

1. Define the WS-Security resources, such as those shown in Figure 11-80. Use either the appropriate editor in WebSphere Integration Developer, such as the EJB Deployment Descriptor Editor, or use the administrative console of WebSphere Process Server.
2. Enable security on WebSphere Process Server.
3. Add a Web services server security binding for each Web service that will use WS-Security.

For more information about WS_Security, see *Patterns: Extended Enterprise SOA and Web Services*, SG24-7135.

11.4.3 Testing the application

The application can be tested using the Integration Test Client. For full details about how to use this test client, see “Starting run time and deploying modules to the test environment using the Integration Test Client” on page 572.

In this section, we make four test cases as shown in Figure 11-81:

- Placing an order at Wholesaler A for part1
- Placing an order with the business state machine for part2 fulfilled by Wholesaler D
- Placing an order with the business state machine for part3 fulfilled by Wholesaler E
- Placing an order for part4 where the business state machine is unable to find an appropriate wholesaler

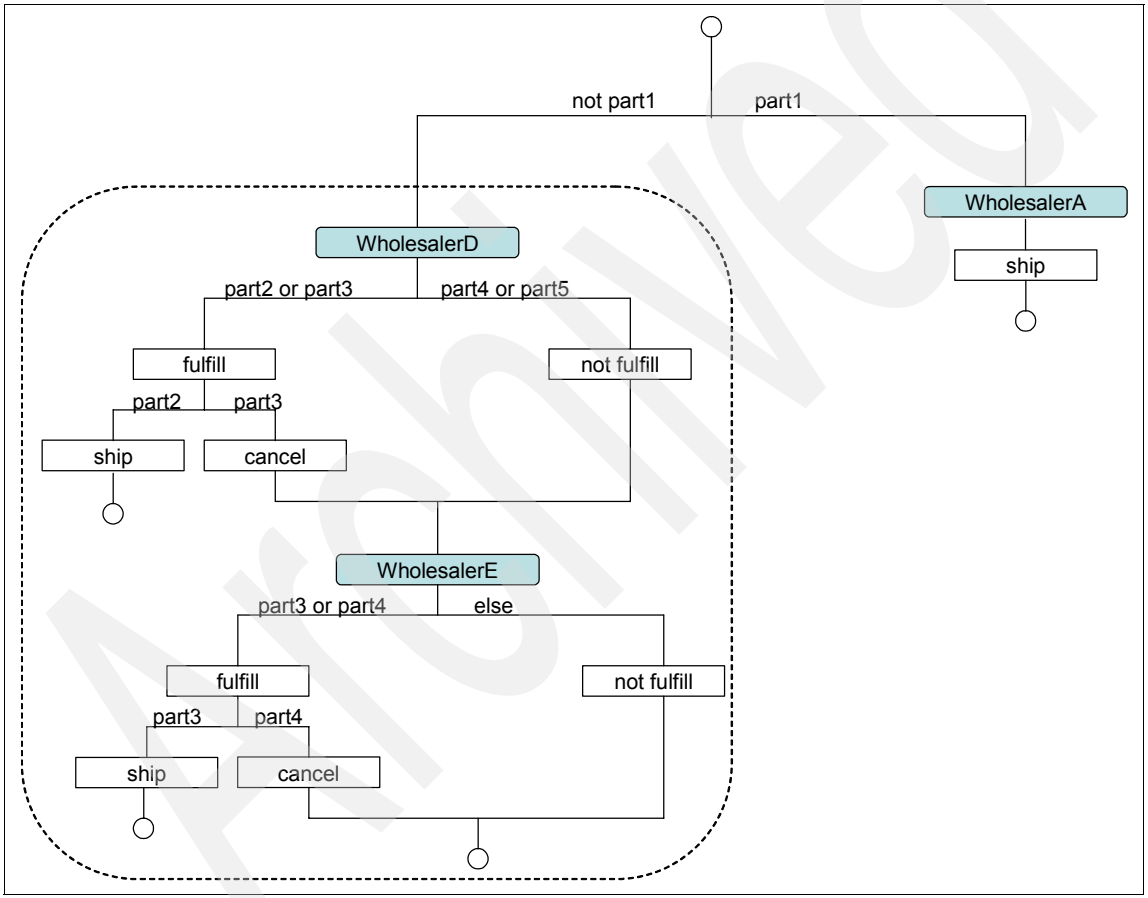


Figure 11-81 Test scenario

Testing Wholesaler A

Before starting the test, ensure that the server is started and the following projects are deployed to it:

- WholesalerAApp
- WholesalerDApp
- WholesalerEApp
- ITSOElectronicsApp
- OrderAckApp

To launch the Integration Test Client:

1. Right-click the **ITSOElectronics** module and select **Test** → **Test Module**.
2. In the Integration Test Client, select **SerialProcess** as the component. Because this component only has a single interface and operation, they are selected automatically.
3. We first use a test case where an order is placed with Wholesaler A. This test case does not use the business state machine. Enter the following values under Initial request parameters (Figure 11-82):
   - orderId: `myOrderId`
   - emailAddress: `myemail@ibm.com`
   - partNo: `part1`
   - qty: `1`

   Click the **Continue** button.

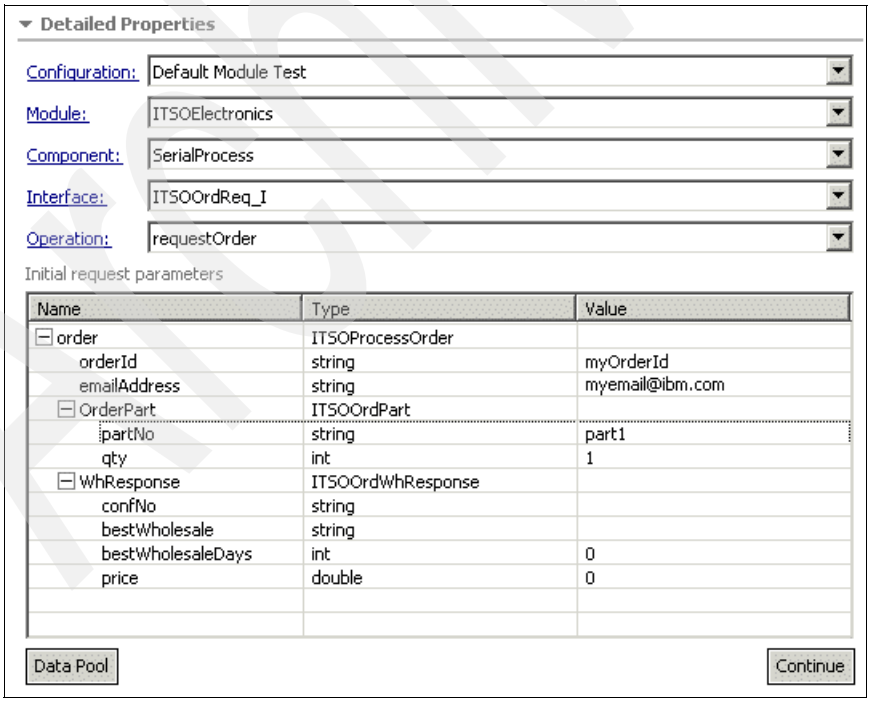


Figure 11-82 Parameters for invocation of the SerialProcess component

4. In the Deployment Location window (Figure 11-83), you can choose the run time to execute the test against. In our case, only one server is defined, which is selected by default. Click the **Finish** button.

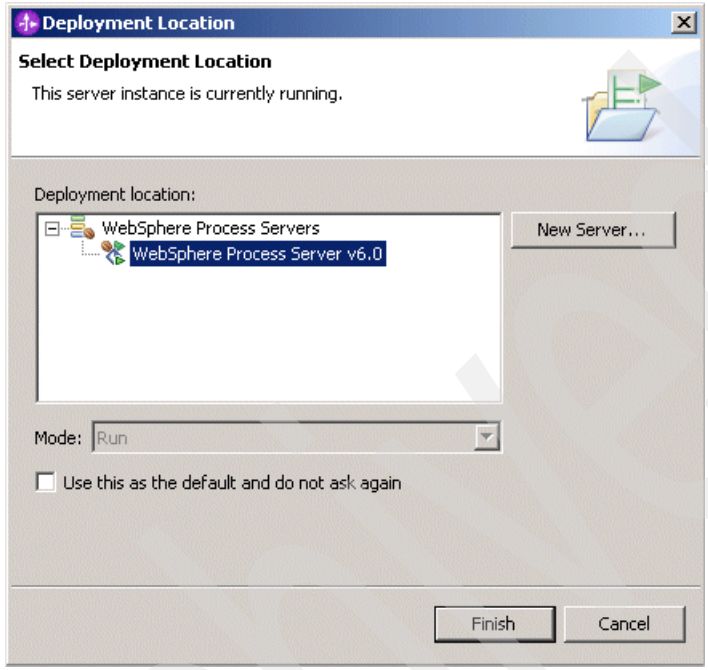


Figure 11-83 Deployment Location window

Archived

The test now runs. Figure 11-84 shows how the editor looks after the test is complete.

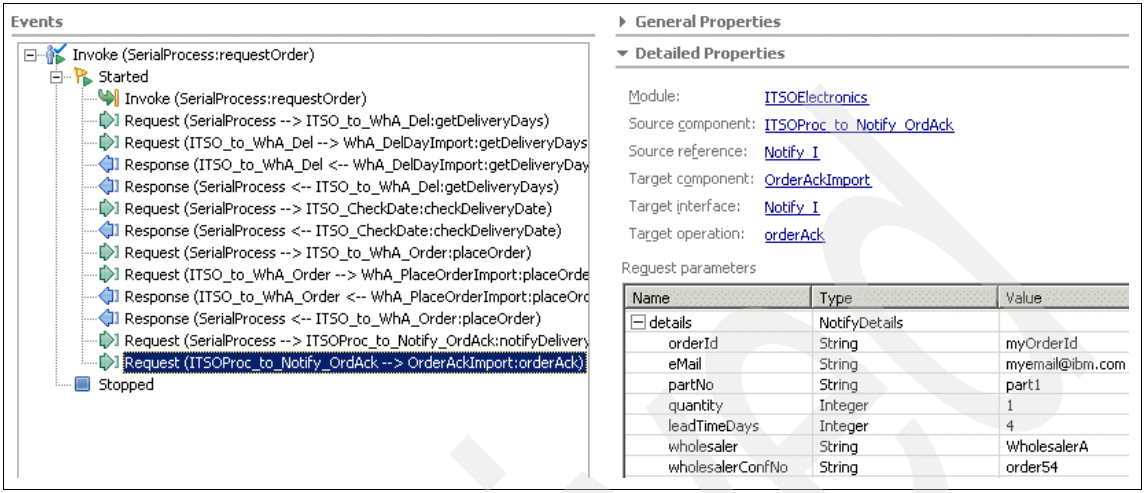


Figure 11-84 Complete test with WholesalerA as the deliverer

If we click the various events in the test client, we can see the data that flowed over the wire between the components. When debugging, this allows us to trace the flow of information and see where components could return the wrong data. In this case, we see that Wholesaler A was used to place the order.

Notice the order acknowledgement that is created by the OrderAck module in the Console view (Example 11-3).

Example 11-3 Successful invocation, placing an order with Wholesaler A

```
SystemOut     O ****************************************************************
SystemOut     O Subject:order54
SystemOut     O Thank you for your order no: myOrderId
SystemOut     O Wholesaler WholesalerA will deliver 1items of part number part1 in 4 days.
SystemOut     O If there are changes, we will email you at myemail@ibm.com
SystemOut     O ****************************************************************
```

Testing the business state machine and WholesalerD

Now we test the application so that the business state machine is invoked, which results in an order being placed with WholesalerD. Figure 11-85 illustrates the flow of logic through the application.

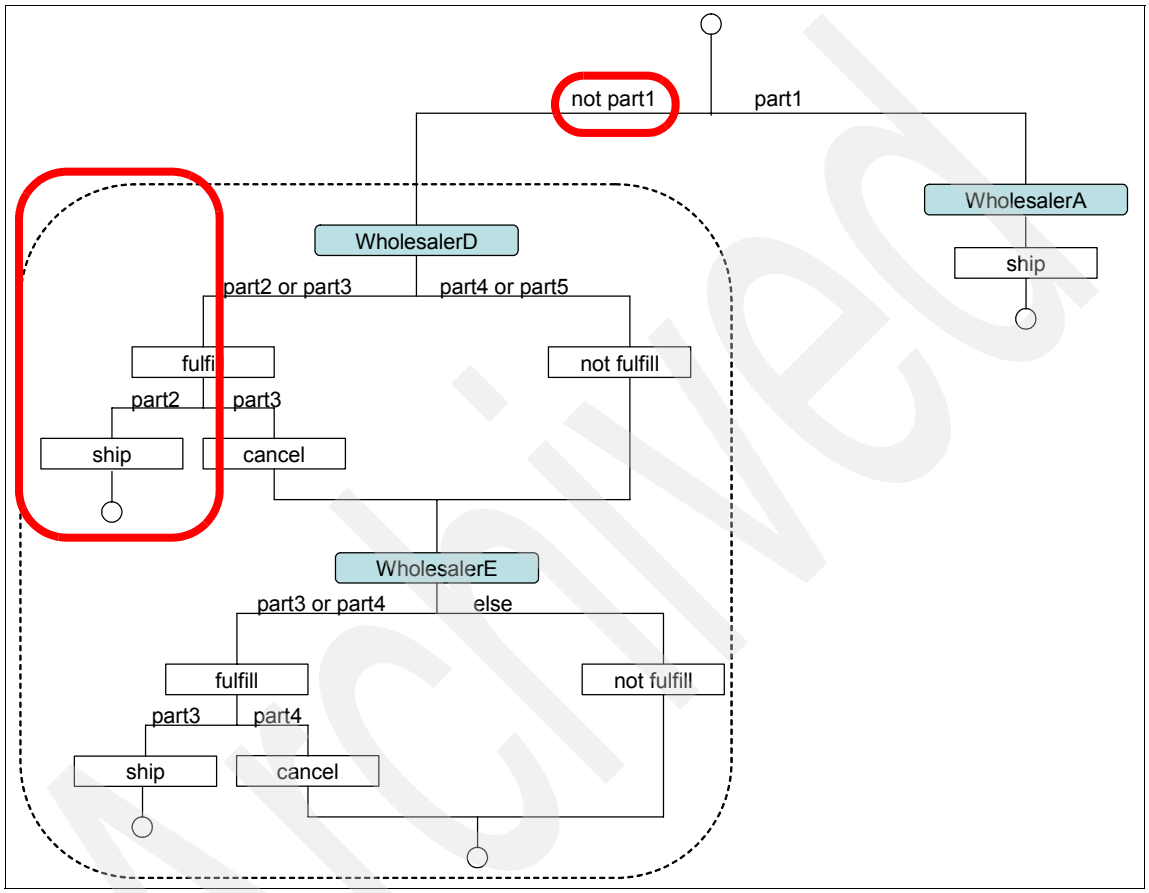


Figure 11-85 Logical flow resulting in an order shipped from WholesalerD

Archived

Use the Integration Test Client to test the SerialProcess component. Enter the following values under Initial request parameters:

- orderId: `myOrderId2`
- emailAddress: `myemail@ibm.com`
- partNo: `part2`
- qty: `1`

Click **Continue**.

Figure 11-86 shows how the client should look after the test is complete. Be aware that WholesalerD contains a wait activity of 20 seconds.

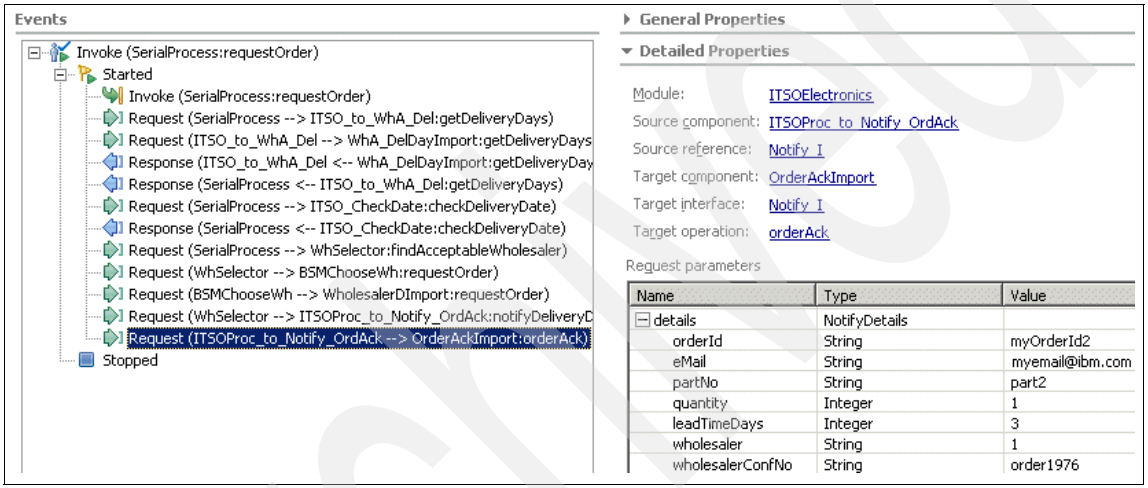


Figure 11-86 The completed test for the process that invoked and chose WholesalerD

Notice the invocation of the business state machine and the order acknowledgement that is created by the OrderAck module in the Console view (Example 11-4).

Example 11-4 Successful invocation, placing an order with WholesalerD

```
SystemOut     O ************************************************************
SystemOut     O Subject:order1976
SystemOut     O Thank you for your order no: myOrderId2
SystemOut     O Wholesaler 1 will deliver 1items of part number part2 in 3 days.
SystemOut     O If there are changes, we will email you at myemail@ibm.com
SystemOut     O ************************************************************
```

Testing the business state machine and WholesalerE

Next, we test the application so that the business state machine is invoked. An order is placed with WholesalerD, which is subsequently cancelled, and then an order is placed and fulfilled by WholesalerE. Figure 11-87 illustrates the flow of logic through the application.

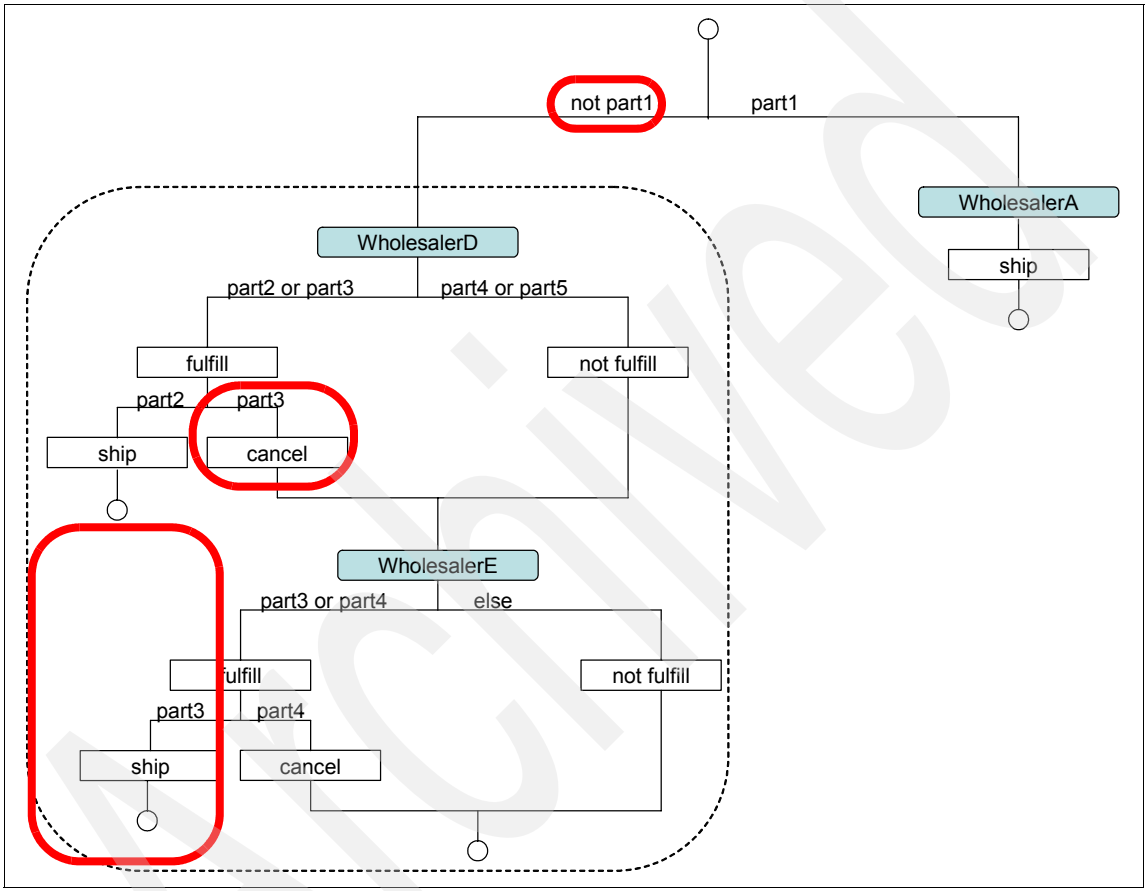


Figure 11-87 Logical flow resulting in an order shipped from WholesalerE

Archived

Use the Integration Test Client to test the SerialProcess component. Enter the following values under Initial request parameters:

- orderId: `myOrderId3`
- emailAddress: `myemail@ibm.com`
- partNo: `part3`
- qty: `1`

Click **Continue**.

Figure 11-88 shows how the client should look after the test is complete. Be aware that Wholesalers D and E contain wait activities of 20 seconds.

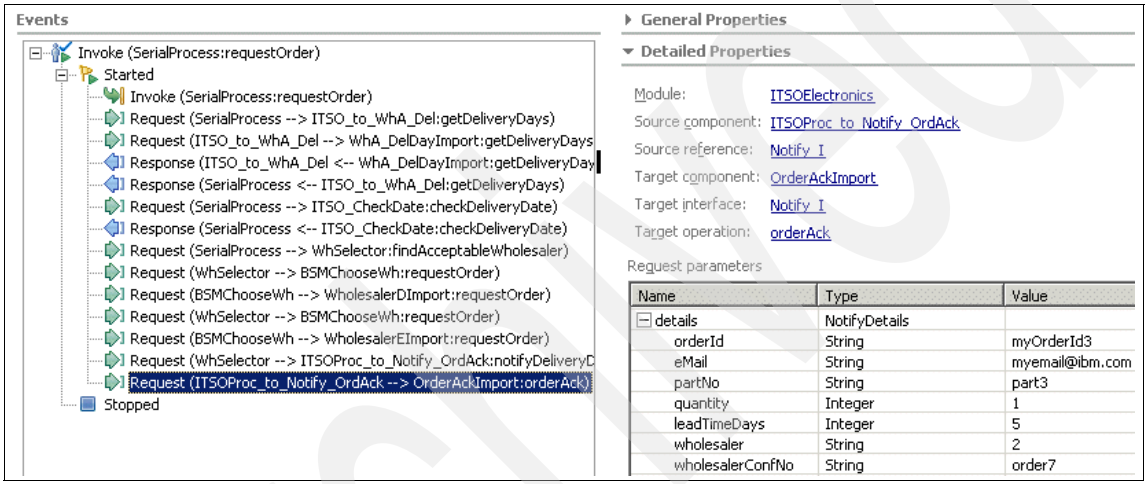


Figure 11-88 The completed test for the process that invoked and chose WholesalerE

Notice the invocation of the business state machine and the order acknowledgement that is created by the OrderAck module in the Console view (Example 11-5).

Example 11-5 Successful invocation, placing an order with WholesalerE

```
SystemOut     O ***************************************************************
SystemOut     O Subject:order7
SystemOut     O Thank you for your order no: myOrderId3
SystemOut     O Wholesaler 2 will deliver 1items of part number part3 in 5 days.
SystemOut     O If there are changes, we will email you at myemail@ibm.com
SystemOut     O ***************************************************************
```

Testing the business state machine with no wholesaler

Finally we test the application so that the business state machine is invoked, but no Wholesalers can fulfill the order. Both Wholesalers D and E are invoked but neither can ship the order. Figure 11-89 illustrates the flow of logic through the application.

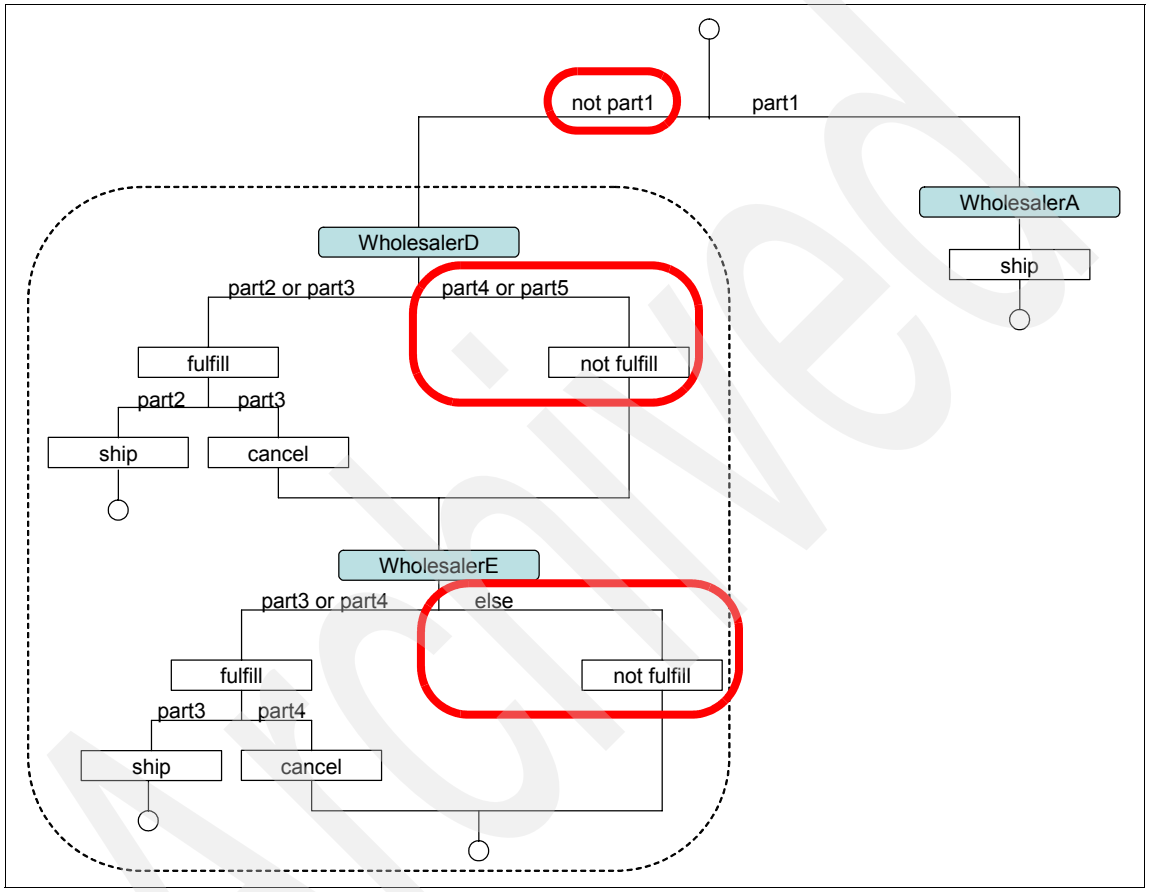


Figure 11-89 Logical flow where no wholesaler can fulfill the order

Archived

Use the Integration Test Client to test the SerialProcess component. Enter the following values under Initial request parameters:

- orderId: `myOrderId5`
- emailAddress: `myemail@ibm.com`
- partNo: `part5`
- qty: `1`

Click **Continue**.

Figure 11-90 shows how the client should look after the test is complete. Be aware that Wholesalers D and E contain wait activities of 20 seconds.

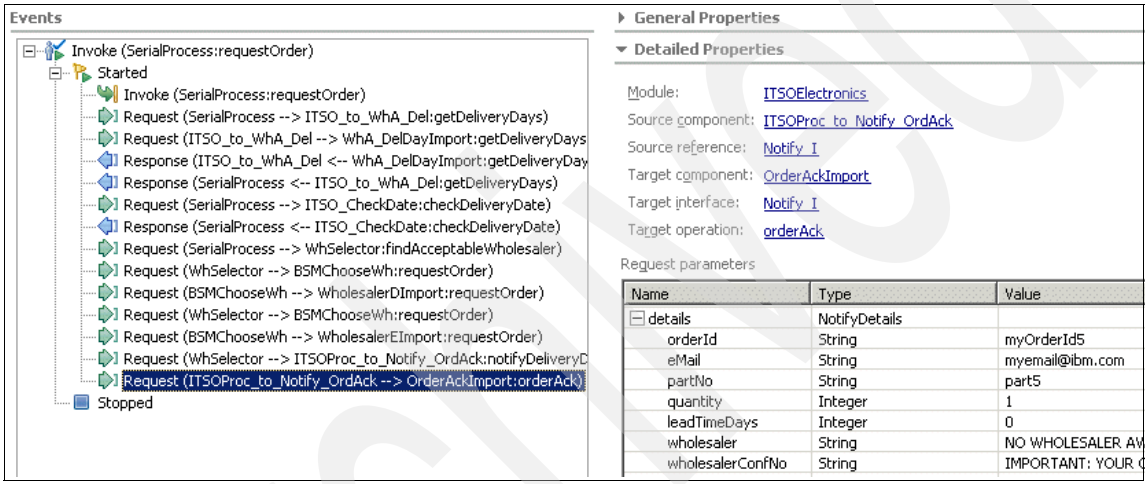


Figure 11-90 The completed test for the process that could not fulfill an order

Notice the invocation of the business state machine, and Wholesalers D and E. Also notice the order acknowledgement that is created by the OrderAck module in the Console view (Example 11-6).

Example 11-6 Invocation where an order could not be fulfilled

```
SystemOut     O ****************************************************************
SystemOut     O Subject:IMPORTANT: YOUR ORDER WILL NOT BE SHIPPED
SystemOut     O Thank you for your order no: myOrderId5
SystemOut     O Wholesaler NO WHOLESALER AVAILABLE - ORDER WILL NOT BE SHIPPED will deliver
1items of part number part5 in 0 days.
SystemOut     O If there are changes, we will email you at myemail@ibm.com
SystemOut     O ****************************************************************
```

Archived

12

Parallel Process scenario

This chapter details the construction of a process using the Parallel Process pattern. It describes the following items:

- Business scenario: For the ITSO Electronics business problem
- Design guidelines: For building a process conforming to the Parallel Process pattern
- WebSphere Process Server guidelines: For implementing the process using WebSphere Process Server

This chapter illustrates the use of the following WebSphere Process Server features:

- Parallel paths of execution in a long-running process
- Use of the WebSphere Adapter for Java Database Connectivity (JDBC) to update DB2 tables
- Transactional considerations
- Common Event Infrastructure (CEI)
- Selectors

© Copyright IBM Corp. 2006. All rights reserved.

12.1 Business scenario

Through the analysis of its Serial Process, ITSO Electronics has noticed that more and more orders are being referred to Wholesaler B, as a result of the inability of Wholesaler A to fulfill their order requests. Because ITSO Electronics wants to become a fully On Demand Business, minimizing the cycle time of the end-to-end business processes is critical to enable it to compete in a market where Quality of Service can be a bigger differentiator than price.

As a result of using WebSphere Process Server to automate their process, ITSO Electronics has established that Wholesalers A and B respond in approximately the same elapsed time of 30 seconds when asked to quote for the order. Because ITSO Electronics takes a considerable number of orders over the telephone directly from customers, they need to provide a quick response to their customers when asked to supply.

In the Serial Process, it takes a minimum of 30 seconds for a quote to be received from Wholesaler A. This expands to one minute if Wholesaler B is also asked to quote.

ITSO Electronics decided to reduce the typical elapsed time for all order quotes to 30 seconds by implementing parallel processing. This meant that Wholesaler B had to provide a quote for every order.

This led to business complications, because Wholesaler B was now competing on equal footing with Wholesaler A, even though wholesaler A was providing preferred supplier benefits. After negotiations with both wholesalers, the decision was made to treat both suppliers equally, removing Wholesaler A's preferred supplier status.

In addition to the cycle time reduction, as part of the on demand commitment to integrating its processes end-to-end, ITSO Electronics decided to automate the update of their financial systems at the end of the process. The company replaced the manual update that used to happen on receipt of the notification.

Archived

Figure 12-1 shows the outline process flow. The order clerk from ITSO Electronics places an order for a part that will be fulfilled by one of ITSO Electronics's wholesalers. Since there is no preferred wholesaler in this scenario, the wholesaler with the earliest delivery date is selected.

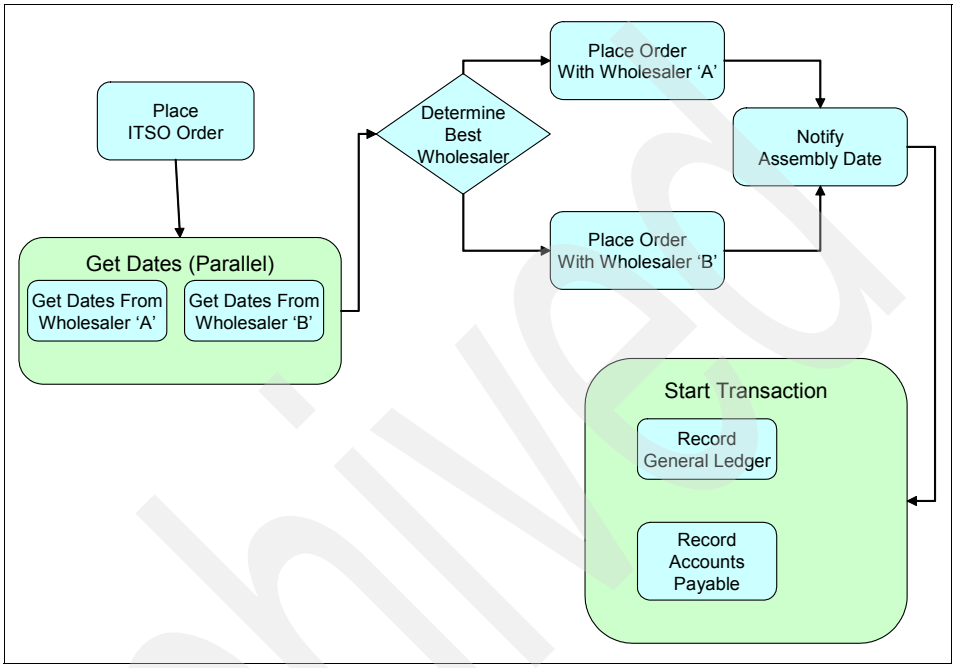


Figure 12-1 ITSO Electronics Parallel Process

12.2 Design guidelines

In this chapter, we discuss the concepts of transactions and audit trailing which are demonstrated in the implementation. We also model parallel paths of execution.

By obtaining the delivery dates from the two wholesalers in parallel, we potentially shorten the time it takes to complete the process. More importantly, we reduce the maximum average time to quote by 50%.

When the defined business process allows it, splitting the process into two or more paths of parallel execution is often desirable. For long-running processes, cycle time reduction is a constant focus for business executives. Shorter cycle times in processes typically equate to an organization being more responsive to its customers.

Archived

We follow these guidelines:

1. Analyze the business requirements.
2. Select a pattern.
3. Analyze the design options.

12.2.1 Analysis of the business requirements

The business requirements are summarized as follows:

- Reduce the cycle time for a wholesaler quote to 30 seconds.
- Choose the wholesaler with the shortest lead time. Do not apply any preferred supplier rules because we rescinded Wholesaler A's preferred supplier status.
- Update the financial system automatically and transactionally. Ensure that either both the general ledger and accounts payable system are updated or none of them are updated. Updates made to one system, but not the other, will result in a failed financial audit.
- Provide an audit trail of events.

12.2.2 Pattern selection

In this section, we use the Patterns for e-business layered asset model to determine the patterns, and ultimately the product mappings, that best fit our business scenario.

Business pattern

This Application Integration pattern brings together multiple applications and information sources. We recommend that you use it when multiple applications must be combined in order to automate a new business function. Therefore, this pattern is an obvious choice.

Archived

Application pattern

Now that we have chosen a Business pattern, next we choose from among the Application patterns associated with it. Since we are implementing a new process communicating with internal wholesalers, we focus on the Process Integration patterns, shown in Figure 12-2.

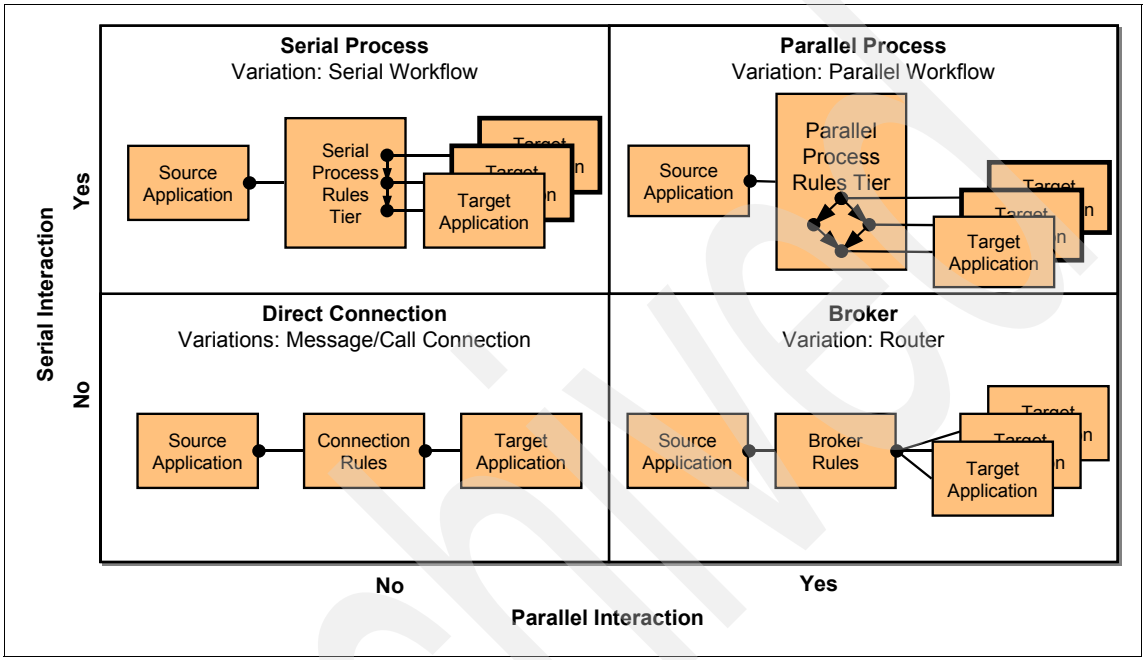


Figure 12-2 Process Integration patterns

Our scenario includes six distinct interactions with four targets. Therefore we look in the top row. Since we query both Wholesaler A and Wholesaler B in parallel, we choose the Parallel Process from the top right quadrant of the table. This pattern does have a variation if human intervention is required. In this case, we do not need human interaction. Therefore, the final result is the Parallel Process application pattern.

Archived

Runtime pattern

Figure 12-3 illustrates the Runtime pattern for the Parallel Process application pattern. This Runtime pattern has two variants: the generic profile and the service-oriented architecture (SOA) profile. Although our architecture is service-oriented, the SOA profile is intended for use with an enterprise service bus (ESB). We choose the generic profile for this scenario because we do not use an ESB.

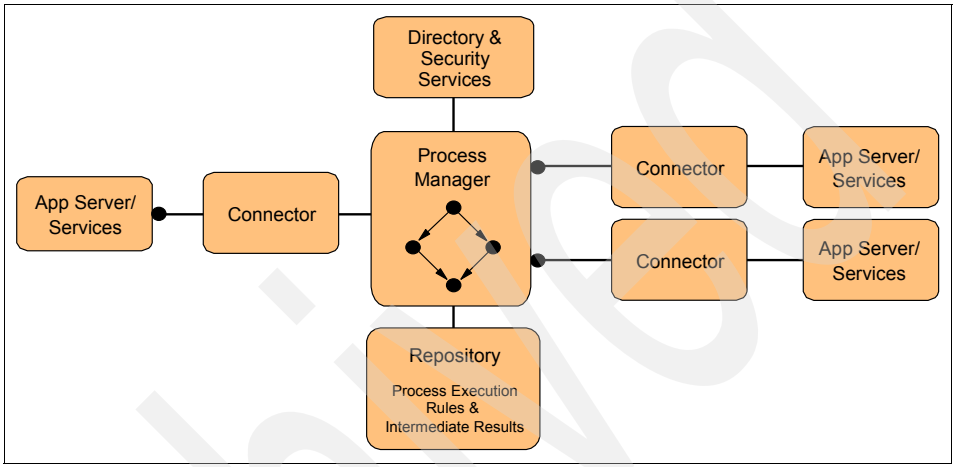


Figure 12-3 Parallel Process runtime pattern: Generic profile

Product mappings

Figure 12-4 shows the Parallel Process product mapping. In this product mapping WebSphere Process Server provides the capability to start parallel threads of execution in the Process Manager node. This product mapping is similar to the Serial Process product mapping. See 7.1.1, “Serial Process product mapping” on page 102, for a detailed description of each node.

Product wise, the only other difference is the use of DB2 Universal Database as the Repository node. WebSphere Process Server does not truly start multiple threads of execution in parallel if an IBM Cloudscape database is used.

Finally, one of the App Server/Services nodes is now implemented by DB2 Universal Database. Connectivity is through a JDBC adapter, which in this case is also part of WebSphere Process Server.

Archived

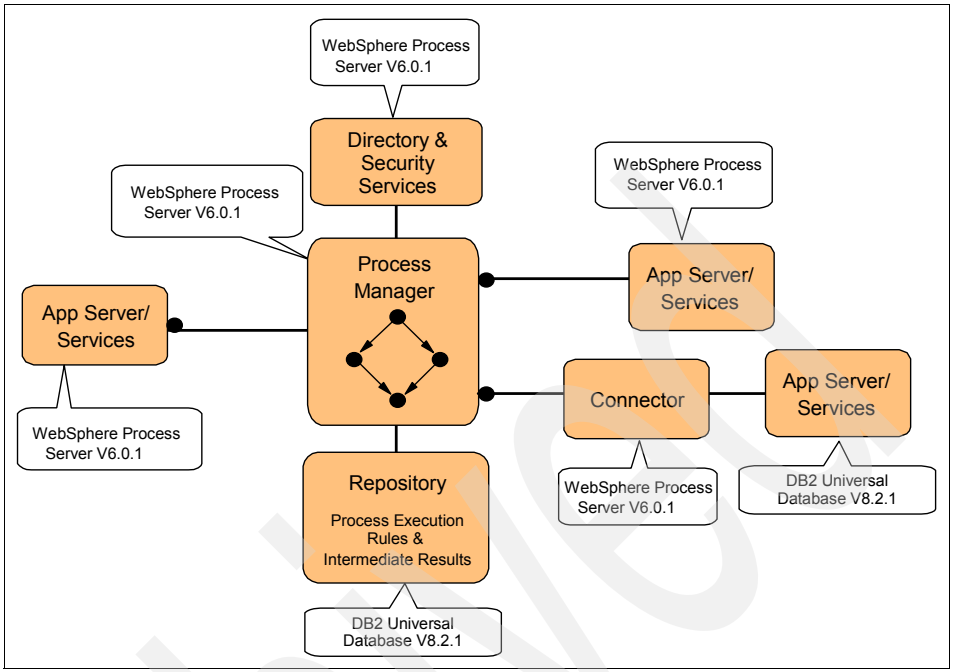


Figure 12-4 Parallel Process: WebSphere Process Server product mapping

12.2.3 Analysis of the design options

Many of the design options for parallel processing are similar or identical to those considered for the Serial Process. These options are discussed in 10.2.4, “Analysis of the design options” on page 153.

We consider the following options for this scenario:

- Transactions
- Monitoring and auditing events
- Dynamicity and future changes
- Adapters

In addition, the following design considerations apply specifically to parallel processing, although some of them may equally apply to running multiple instances of a Serial Process concurrently. See “Does my process genuinely need the parallel pattern?” on page 342.

- Parallel pattern
- Degrees of parallelism
- Compensation
- Performance

- Deadlocking and dependencies
- Resource loading
- Complexity
- Process modeling and simulation

We also discuss the implications of each of these options with regard to the ITSO Electronics scenario.

Transactions

When multiple flows execute at the same time, the problem of error handling arises. As long as all of the parallel paths complete successfully, there is no problem. The difficulties arise when one (or more) flows fail. What happens?

The use of transactions (for those services that support it) can help solve this problem. If all of the flows that execute in parallel participate in the same transaction, then the failure of one flow causes the rollback of the noncommitted transactions from all of the flows, and the transactional integrity is maintained across all the services.

Each service that is called transactionally is asked to participate in a transaction by the transaction coordinator. Each service makes a first-phase commit and notifies the coordinator that it is ready to accept a second-phase commit request. When all of the participating services have notified the coordinator that they are ready to commit, the coordinator initiates the second-phase commit, and all of the transactions are committed together.

If one of the participants fails, then the coordinator tells the remaining participants in the transaction to roll back to their previous state.

Note: When considering transactions with adapters, the WebSphere Business Integration Adapters do not support transactions, while the WebSphere Adapters do. When considering transactions, consider this point both in the design and product-selection phases.

Applying the guidelines to ITSO Electronics

When the accounting update is made, either both updates have to be successful or none of them are successful. Because the accounting systems are represented in our scenario by XA supporting databases (DB2 Universal Database), which are accessed using the WebSphere JDBC Adapter, the two service calls can participate in the same transaction.

The option to include the order placement and notification in the same transaction as the accounting systems was considered. However, because

neither of these services is transactional, and the notification depends serially on the output of the order placement call, this was not possible.

If the JDBC transaction fails, the implementation does not explicitly handle the error. See 13.2, “Design guidelines” on page 395, which discusses error handling. It is important to note that, even though the error is not explicitly handled, the transactional processing still ensures that either both or none of the transactions are completed. The error is thrown at transactional level, not at the individual service level.

Monitoring and auditing events

Business events undergo a potentially complex series of interactions typically involving multiple components. It is becoming increasingly critical to track these business events for both technical and business reasons.

IT departments have historically tracked and monitored system-level events to measure performance and to verify compliance with service-level agreements. Many enterprises have also established audit trails for critical systems. In addition, organizations attempt to understand their business processes better and capitalize on the knowledge that is inherent in the execution of the business. In doing so, they expect to be able to log and retrieve *business-level events*. A business level event captures activity that can be observed and monitored by anyone who wants to analyze the effectiveness of the business process that emits it. This enables such functionality as:

- Offer a means to investigate a particular sequence of events.
- Understand why an order took three weeks longer than expected.
- Allow historical analysis of past performance to predict future behavior.
- Help maintain a historical record of business commitments.
- Provide a central point where data can be assembled and passed to a tamper-proof log for an audit trail.
- Trigger an action when an unexpected event occurs.

The CEI is a consistent, unified format for the creation, transmission, persistence, and distribution of a wide range of business, system, and network events. Events are emitted to an infrastructure that can either persist the event or distribute it to active monitors. This provides a basis for both monitoring and auditing events at all levels.

WebSphere Integration Developer allows you to activate events at development time. Events can be emitted on the entry or exit of any component, and depending on the component, additional events may be available. Events can

also be activated through the administrative console of WebSphere Process Server while the process is running.

The WebSphere Process Server administrative console includes the Common Base Event (CBE) browser to view emitted events. The event server allows clients to query events through a Session Enterprise JavaBean (EJB) application programming interface (API), and it can publish events to Java Message Service (JMS) consumers.

When to use the Common Event Infrastructure

In general, you use the CEI when you need to monitor your business processes or when you need a historical record of what happened. Any need you have to visualize systems events, either dynamically as they happen or in an audit mode, can be supported by the CEI.

Applying the guidelines to ITSO Electronics

This scenario's requirements specifically ask for an audit trail. We use the CEI to capture the business events to provide that trail.

Dynamicity and future changes

When building a process, we must not only think of what the process will do, but also how it will need to expand and change in the future. While our business process may remain static, the services that it calls may change. In our example, the suppliers have already changed status with Wholesaler A losing its preferred supplier label. In the future, the wholesalers might change the way that they interact with the process, or might change completely.

Applying the guidelines to ITSO Electronics

We can use selectors to isolate the calls to the wholesalers from the process logic. This enables us to change the wholesaler implementation with less impact on the process logic and WebSphere Process Server deployed components. In our implementation, we use a selector to isolate Wholesaler A from the business process

Appendix D, "WebSphere Process Server dynamicity" on page 661, shows how Wholesaler A can easily be replaced by another wholesaler, wholesaler C using the selector functionality of WebSphere Process Server.

Adapters

Not every application or business service that we want to use exposes its services as Web services. WebSphere Process Server integrates easily with both the WebSphere Adapters and the WebSphere Business Integration Adapters. These adapters allow bi-directional communication between

WebSphere Process Server and a variety of technologies and applications where a Web service is not available or suitable.

There are a number of differences between the WebSphere adapters and the WebSphere Business Integration Adapters, for example:

- WebSphere Adapters are based on Java 2 Platform, Enterprise Edition (J2EE) Connector Architecture V1.5. WebSphere Business Integration Adapters run under the WebSphere Business Integration adapter framework based on Java 2 Platform, Standard Edition (J2SE™) and communicate with WebSphere Process Server over JMS.
- WebSphere Adapters are more closely integrated with WebSphere Process Server. WebSphere Business Integration Adapters can integrate with WebSphere Interchange Server, WebSphere Message Broker, and WebSphere Server Foundation using a common framework which is not optimized explicitly for WebSphere Process Server.
- WebSphere Business Integration Adapters run in a separate Java virtual machine (JVM). WebSphere Adapters run inside the WebSphere Process Server.

We recommend that you use the WebSphere Adapters where available. However, there are WebSphere Business Integration Adapters which do not have WebSphere Adapter equivalents.

When an adapter is used with WebSphere Process Server, it is called like any other service using interfaces and business objects. The designer of a process or implementer does not need to know that there is an adapter behind the service call as long as they are aware of the interface, data structures, and quality of service provided by the adapter service.

Both WebSphere Adapters and WebSphere Business Integration Adapters provide introspection tools which allow the application's data structures (such as database tables) to be built easily as business objects within WebSphere Process Server and WebSphere Integration Developer.

Applying the guidelines to ITSO Electronics

The accounting system used by ITSO Electronics does not expose a Web service interface and therefore requires the use of an adapter. We choose the WebSphere JDBC adapter to provide connectivity to the accounting systems, which are implemented in our example as database tables.

Because we have chosen the WebSphere JDBC adapter and not the WebSphere Business Integration JDBC adapter, we can make the calls transactionally.

Archived

Does my process genuinely need the parallel pattern?

In our scenario, we have two ways to do more than one thing at one time:

- In a single order, we can query two wholesalers in parallel.
- We can process multiple part request orders simultaneously.

The first task requires the parallel pattern, which is what we have chosen here. The second task requires only the serial pattern, but uses the functionality of WebSphere Process Server to enable multiple instances of that pattern to run at the same time. It is important to keep the difference in mind, although some of the design considerations for the parallel pattern are equally applicable when running multiple instances of the serial pattern.

We can also run multiple instances of the parallel pattern, which is what will happen most likely in ITSO Electronics. The reason for implementing the parallel pattern is to reduce the cycle time for the process. The reason for running multiple concurrent instances is to allow more orders to be taken at the same time and increase order throughput. See “Performance” on page 344.

Applying the guidelines to ITSO Electronics

We implement the parallel pattern because it is the only way to reduce the cycle time per the business requirement. Even if we run 30 process instances of the serial pattern concurrently, we can process 30 orders in one minute, but each order still takes one minute, not two seconds. This difference is crucial and is sometimes referred to as the “pregnant woman restriction”. That is we cannot make a baby in one month using nine mothers.

Short-running or long-running process

Within WebSphere Process Server, a parallel process only truly executes in parallel if the process is defined as long running. WebSphere Integration Developer allows the construction of parallel flows within a process defined as short running. However, the actual execution of the process is performed serially, but it is not defined in the order in which the process paths are executed.

Although a short-running process executes faster than a long-running one, the overhead of the process running in WebSphere Process Server is minimal compared to the response times of the services called, in our example the two wholesalers. Often it is the case that the services being consumed by a WebSphere Process Server process take more time to execute than the process execution logic.

Note: A WebSphere Process Server non-interruptible process only executes in a single thread. A decision to have a parallel process in WebSphere Process Server means that the process needs to be defined as interruptible. A change from a non-interruptible process to an interruptible process does impact performance.

Applying the guidelines to ITSO Electronics

We choose a long-running process to ensure that both wholesalers' requests can be sent at the same time.

Degrees of parallelism

After the decision is made to select a parallel pattern, we must determine the number of degrees of parallelism or the number of concurrent flows to implement. This is one of the most difficult issues in process-based integration. Opinions range from "Use serial unless you have a reason not to," such as for one flow, to 'Use as many threads as you have resources for."

In our example, we call two suppliers in parallel, but revert to a serial pattern to place the order with the appropriate supplier and send a notification of the complete order. We then move to parallel again to update the accounting systems. Although we can alternatively sent the notification of the order at the same time as updating our accounting systems (therefore processing all three flows in parallel), we choose not to do so. We decided this because the call to send the notification is not transactional and cannot participate in the transaction, which updates the accounting systems.

There are two ways to execute processes in parallel:

- Design a process so that there are multiple parallel paths through the process. This implements the Parallel Process pattern.
- Run multiple instances of a process in parallel with each other. This implements the Serial Process pattern but runs multiple parallel instances.

Many factors influence this decision. Each service provider may limit the number of simultaneous service requests, for example, maximum connections to a database. There may be limited processor capacity, or each process instance may consume too many resources, such as memory.

In a process where there are many complex interactions with service providers, it may be worth your time to construct a Serial Process first to test all of the endpoints, which will be easier to verify and debug. After the serial version is proven, you can implement the parallel version.

Applying the guidelines to ITSO Electronics

Because we have two suppliers, the decision to design the process with two paths was easy. We know and trust our two suppliers and can rely on the fact that they take approximately the same amount of time to process their requests. We also have two accounting systems that we want to update and decided to have two parallel flows here too.

The decision to “revert” to serial to send the notification was based on the fact that the service to send the notification was not transactional and could not participate in the same transaction as the accounting systems.

Compensation

In general, compensation is what you do when your resources do not support two-phase transactionality. If all of your service providers support two-phase commits and the time duration of the process is small, then use transactions.

In the real world, not all systems support two-phase transactions. Such resources as non-transactional, traditional systems, WebSphere Business Integration Adapters, and Web services, which do not support transaction management provided by WS-Transaction or similar, cannot participate in a transaction. For these resources, compensation is necessary to ensure the integrity of our processes.

Even if all the resources support transactions, for a long-running process, it is not desirable to lock the resources necessary for transactional processing for long periods of time. Therefore consider compensation here too.

A combination of transactionality and compensation is often a good approach. It allows a group of short-running services to be bound transactionally, while allowing the rest of a long-running process to have its integrity protected by compensation. The two techniques are not exclusive within the same process.

Applying the guidelines to ITSO Electronics

Since this process does not take a long time to execute, and our parallel updates are transactional, we do not use compensation in this scenario. We discuss compensation in 13.2, “Design guidelines” on page 395.

Performance

The usual driving factor for parallel execution is performance. Providing the processing capacity is available, doing two things at one time is quicker than doing them one after the other.

Be aware that there are multiple measures of performance. The two important measures are cycle time (or response time) and throughput:

- **Cycle time**: The length of time taken for a complete end-to-end execution of *one instance* of the process; typically expressed as absolute elapsed time, for example a cycle time of 45 seconds
- **Throughput**: The amount of instances of a process that can be completed in a given amount of time; typically expressed in processes per time unit, for example 50 orders per minute

For example, ITSO Electronics is becoming a real-time On Demand Business. Cycle time is the driving factor for customer-facing orders because the orders might be taken over the Web or via telephone. Our metric may be, for example, "Reduce the time to quote for an order from 1 minute to 30 seconds."

Consider the need for processing bulk inter-company orders. Each order does not need to be processed within a specific cycle time, but all of the orders need to be processed within a batch window.

Make sure you know exactly what the business requires before tying your process performance down to a metric which may not be achievable or applicable.

Applying the guidelines to ITSO Electronics

In our scenario, the goal is to reduce the cycle time. Since we have established that both wholesalers take approximately the same time to quote, we can reduce the cycle time by querying both wholesalers in parallel.

Deadlocking and dependencies

Deadlocking is the hidden danger of parallel processing. In its simplest form, this is where flow 1 needs a resource that is in use by flow 2, while at the same time flow 2 is waiting for a resource that flow 1 has locked. The resources can be physical, such as warehouse locations, humans, or IT resources such as database rows. The result is the same.

There are a number of strategies to resolve deadlocking, one of which is a timeout while waiting for a resource. One process times out and releases the resource so that the deadlock is resolved. Unless designed explicitly to retry, the process releasing the lock typically fails and must be recovered.

Applying the guidelines to ITSO Electronics

In our scenario, there are two parallel flow sets, retrieving the order lead time quotes and updating the financial systems. Since each of the four resources that are used is independent, there is no need to consider a deadlocking avoidance strategy.

Archived

Resource loading

We must consider two areas with respect to resource loading:

- The service consumer, in our case the process running on WebSphere Process Server
- The service providers, which are represented here by Wholesalers A and B

In our examples, the resource loading on the consumer can be controlled. All of the relevant factors, such as the number of processors, software license, and amount of memory, are typically under the direct governance of the process owner, who can vary them to the needs of the company.

External service providers are less controllable and might introduce more constraints. This can range from a slowdown of response from a Web service, to a hard limit such as a maximum number of database connections available to a process. No amount of parallelism can cure a resource bottleneck without investigating and correcting the underlying cause.

Applying the guidelines to ITSO Electronics

In our scenario, each service is independent. However in reducing our cycle time for the wholesaler queries by a half, we also double the rate at which these queries are executed for any given part. This increases the load on both our system and the Wholesaler B system for any given part order.

Complexity

With each activity that is processed in parallel, complexity tends to increase considerably if the processing does not complete successfully. In using parallel execution, consider the following questions in error situations:

- If one activity fails or times out, what is the desired behavior of the process? Should the process end and the error be returned to the component that initiated this process?
- What about the other parallel activities that have executed and completed successfully?
- Is compensation required? If so, does the development and processing cost justify ending the process if only one activity fails?
- How does a parallel path know if an activity failed in one of the other parallel paths? If the path activities are required to communicate, the synchronization of the communication method may negate the benefits of running in parallel.
- Should the failure of one of the activities prevent the continuation of processing along the other parallel paths?

- Should the process be put into a state that uses human interaction to investigate the problem? If so, how is the state of all of the parallel paths made available for manual investigation?

Applying the guidelines to ITSO Electronics

We have kept our process simple to illustrate the principles of parallel processing. We have used a transaction to solve the problem of parallel updates to the accounting system that needs to be processed in an “all or nothing” style.

Our parallel calls to the wholesalers do not need to be transactional or compensated because they are, effectively, retrieve operations which do not create, delete, or update any data.

Process modeling and simulation

A great aid to process optimization, especially with multiple parallel flows and flows which have multiple decision points, is to model the process and simulations. Doing so enables you to perform analysis of the processes and identify bottlenecks, critical paths, slack time, and other improvement opportunities. The use of process modelling allows a large number of “what if” simulations to be carried out, without the expense of re-engineering the deployed process on a “try it and see” basis.

With software, such as WebSphere Business Modeler, multiple processes can be simulated per second and results graphed in minutes. In addition, actual process metrics can be measured and added to the simulation to improve the accuracy.

Three of the major benefits of using process modeling and simulation are:

- Identifying and alleviating bottlenecks through bottleneck analysis
- Simulating and measuring the benefits of changing the degree of parallelism
- Identifying the critical path within a multipath process

Applying the guidelines to ITSOElectronics

ITSO Electronics used the data from their system to create timing metrics from which they could calculate the savings due to parallel process implementation. Knowing that Wholesaler B was called 50% of the time, as well as knowing the average response times, enabled ITSO Electronics to simulate the process. They found that the overhead of the business process execution in WebSphere Process Server was minimal compared to the length of time for the service calls to their two wholesalers, so they decided to adopt the parallel pattern.

12.3 Development guidelines

Note: This section refers to the use of the following products:

- IBM WebSphere Integration Developer V6.0.1
- IBM DB2 Universal Database V8.2.1
- IBM WebSphere Adapter for JDBC V6.0.0

This section contains step-by-step instructions for building a Parallel Process for the ITSO Electronics scenario. Unlike the other implementation examples, this chapter uses a top-down approach. In the top-down design, we define the interfaces, references, component types, and wires in the Assembly Editor before we create any of the actual implementations. This allows us to visualize the entire solution at once to ensure that the data flows as expected.

12.3.1 Setting up the workspace

To prepare the WebSphere Integration Developer workspace for this scenario:

1. Start a new workspace in WebSphere Integration Developer V6.0.1.
2. Select **File** → **Import** → **Project Interchange** to import the WholesalerA module into the workspace. You can find this module in the additional material supplied with this IBM Redbook in the file \Wholesalers\WholesalerA\WholesalerA.zip.
3. Repeat this process to import the WholesalerB module. This is also supplied in the additional material supplied with this book in the file \Wholesalers\WholesalerB\WholesalerB.zip.
4. Import the **ITSOLibrary** module. This contains Wholesaler data types and interfaces. Import it from the additional material, using the file \ParallelProcess\ITSOLibrary.zip.

To access the additional material supplied with this book, see Appendix E, “Additional material” on page 689.

12.3.2 Creating the Parallel Process module

In this section, we create the parallel process module using the top-down design pattern:

1. Select **File** → **New** → **Project** and select **Module** to create a new module. Name the module *ITSOPModule*.

2. In order for the module to have visibility to the artifacts contained in the library project, add the library as a project dependency. Double-click **ITSOPModule** to open the dependency editor.

   a. Click the **Add** button.
   b. Select the **ITSOLibrary** project, and click **OK**.
   c. Close the editor and save the changes.

> **Note:** In a top-down design, it is beneficial to create the interfaces and business objects before wiring components in the Assembly Editor. The following steps explain how to create the required artifacts in the ITSOPModule module for the overall process.

3. Create a business object called *Order* using the attributes shown in Table 12-1. The user is required to enter these parameters to start our overall OrderEntry process. Create this, and all artifacts in this section, in the ITSOPModule using the default assigned namespace.

Table 12-1 Order business object

Attribute	Type
orderId	string
partId	string
quantity	int
email	string
department	string

4. Create an interface called *OrderEntry* with a one-way operation named *createOrder* as shown in Table 12-2.

Table 12-2 OrderEntry interface: createOrder operation

Input/output	Name	Type
Input	initialOrder	Order

> **Tip:** If you accidently create a Request/Response operation instead of a one-way operation, you receive an error in the problems view stating that the response object in the interface could not be resolved. Ensure that you are using a one-way operation.

Archived

5. In our scenario, we have a requirement to determine which of our two wholesalers is the better one to choose. The business logic behind this decision may change, so we have a component in our solution that compares two wholesaler orders and returns the best one. To support this, we need to create an additional business object and interface.

   a. Create a business object named *WholesalerOrder* with the properties shown in Table 12-3.

   Table 12-3 WholesalerOrder business object

Attribute	Type
wholesalerName	string
daysToDeliver	int
order	Order
wholesalerId	string
price	double

   b. Create an interface called *BestWholesaler* with a request-response operation named *getBetterWholesaler* that contains the attributes shown in Table 12-4.

   Table 12-4 BestWholesaler interface: getBetterWholesaler operation

Input/output	Name	Type
Input	company1	WholesalerOrder
Input	company2	WholesalerOrder
Output	selectedWholesaler	WholesalerOrder

12.3.3 Constructing a solution in the Assembly Editor

We have now defined enough information to begin constructing our solution in the Assembly Editor. The key idea to remember is that we are trying to define all the individual pieces required in the solution before we create any implementations.

Given our design requirements, we know that we will use a business process to drive our entire scenario. Let's create a business process component in our diagram.

Archived

1. Open the Assembly Editor for the ITSOPModule project.
2. Click the **Component (with no implementation type)**( ) icon from the palette, and then click the canvas to add it. Configure it as explained in the following steps:
   a. Right-click the **component**.
   b. Click **Show In Properties**.
   c. Click the **Description** tab.
   d. Change the Display Name to `PlaceOrderProcess`.
   e. Change the type from an unimplemented component to a process. Right-click **PlaceOrderProcess** and select **Change Type** → **Process**.
   f. Add an interface to the component. Right-click **PlaceOrderProcess** and select **Add** → **Interface**. Double-click **OrderEntry**.
3. Our process requires references to the partners that will return the shipping date and references to place the order for the given wholesaler. These are defined by the WhPlaceOrderWeb_I and the WhDelReqWeb_I interfaces that were copied from the Wholesalers. Add these references:
   a. Right-click **PlaceOrderProcess** and select **Add** → **Reference**.
   b. In the Add Reference window, double-click **WhDelReqWeb_I**.
   c. Right-click the ***newly created reference*** and click **Show In Properties**.
   d. Change the name of the reference to `WholesalerADeliveryDaysPartner`.
   e. Create four more references on the component using the information from Table 12-5.

Table 12-5 Additional references for the PlaceOrderProcess Component

Interface name	Reference name
WhDelReqWeb_I	WholesalerBDeliveryDaysPartner
WhPlaceOrdWeb_I	WholesalerAOrderPartner
WhPlaceOrdWeb_I	WholesalerBOrderPartner
BestWholesaler	BestWholesalerPartner

Note: One more reference is required that will wire to the JDBCOutbound interface. The interface is created dynamically by the enterprise service discovery wizard and is added later.

Archived

Figure 12-5 shows how the assembly diagram should look.

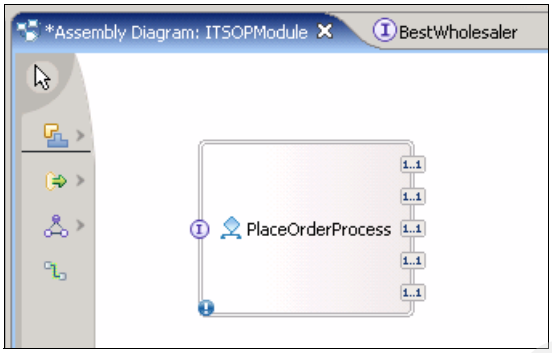


Figure 12-5 The PlaceOrderProcess component

We now need to create the following components, which will be invoked from the partner references of the process:

- The invocation of Wholesaler A
- The invocation of Wholesaler B
- The invocation of the business logic to determine the best wholesaler

Creating the imports for Wholesaler B

So far, we have defined references on the component for the wholesaler. We now need to create the imports that will be executed when that reference is called. We are using an import because Wholesaler B has exposed their service via a Web service and we want to use it.

1. Click the **Import** icon ( ) from the palette and click the canvas.
2. Change the name of the component to `WholesalerBGetDate`.
3. Right-click **WholesalerBGetDate** and click **Add Interface**.
4. In the Add Interface window, double-click **WhDelReqWeb_I**.
5. Hover over the WholesalerBDeliveryDaysPartner reference on the PlaceOrderProcess component. Drag the **handle** to the interface of the **WholesalerBGetDate** component to create a connection between the two (Figure 12-6).

Archived

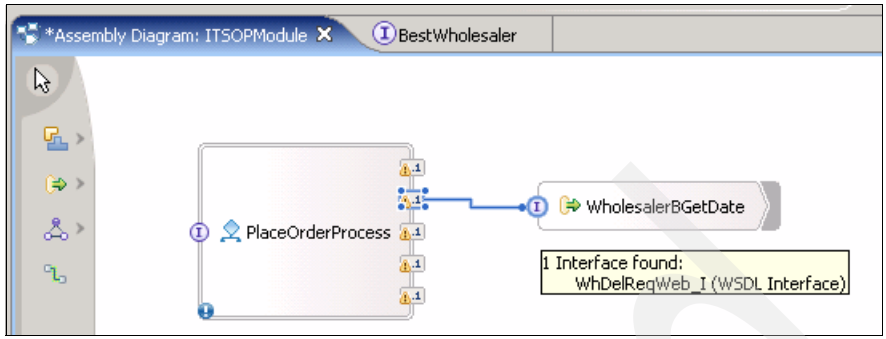


Figure 12-6 Connecting PlaceOrderProcess with WholesalerBGetDate

6. Repeat the previous steps to create a new import called *WholesalerBPlaceOrder*, which has an interface of WhPlaceOrderWeb_I. Wire this component to the WholesalerBOrderPartner reference.

Figure 12-7 shows how the Assembly Editor should look.

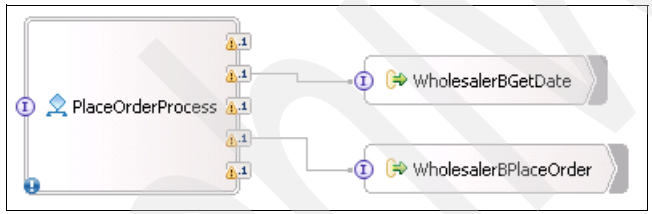


Figure 12-7 The Assembly Editor after wiring Wholesaler B

Creating the components for Wholesaler A

In the previous section, we wired the PlaceOrderProcess to Wholesaler B. In this section, we wire Wholesaler A. We create a different set of components and imports to implement Wholesaler A. We expect that there will be the potential for a lot of change once the PlaceOrder service is implemented. To make our solution as flexible as possible, we do not wire Wholesaler A directly to the import. Instead, we use the selector/rule group pattern.

A selector dynamically invokes a component based on a condition at run time. The limitation of a selector is that it can only invoke components that implement the same interface as the selector. This limitation prevents the scenario where you need to invoke a different interface. To provide this capability, rule groups are used. Rule groups (and business rules) can invoke any references it defines. To solve the problem of allowing dynamic invocation of a changing interface, you put these two technologies together. The selector simply chooses a rule group to

invoke, and then the rule group invokes the new interface, likely using an interface map.

1. In the assembly diagram, click the **selector palette entry** (Figure 12-8) and click the canvas to add it.

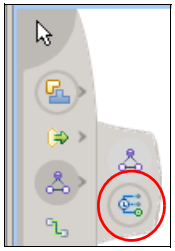


Figure 12-8 The selector palette entry

2. Rename the selector to `WholesalerGetDateSelector`.
3. Create another selector and set its name to `WholesalerPlaceOrderSelector`.
4. Wire the **WholesalerADeliveryDaysPartner** reference on the PlaceOrderProcess component to the WholesalerGetDateSelector component.
5. In the Add Wire window (Figure 12-9), click **OK** to continue.

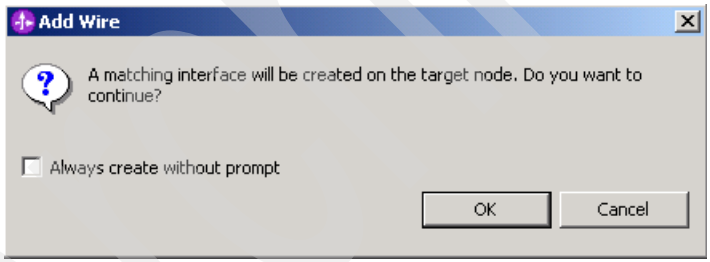


Figure 12-9 The Add Wire window

6. Repeat these steps to wire the **WholesalerAOrderPartner** reference to the WholesalerPlaceOrderSelector component.

Archived

7. Now that we have created the selectors, we need to complete the pattern by creating the rule group components. These components will be called from the selector.
   a. Click the **Rule Group** icon ( ) from the Assembly Editor palette and add it to the canvas.
   b. Change the name of the component to `WholesalerAGetDateRG`.
   c. Create another rule group component and change its name to `WholesalerAPlaceOrderRG`.
   d. For each rule group component:
      i. Right-click the **component** and Click **Add** → **Interface**.
      ii. For the WholesalerAGetDateRG component select **WhDelReqWeb_I**. For the WholesalerAPlaceOrderRG component select **WhPlaceOrdWeb_I**.
8. Create the two imports that will call the Web service exposed for Wholesaler A.
   a. Name the first one `WholesalerAGetDate`.
   b. Name the second one `WholesalerAPlaceOrder`.
9. Click the reference from the **WholesalerAGetDateRG** component and wire it to the WholesalerAGetDate import.
   a. If the Add Wire window (Figure 12-10) opens, click **OK**.
   b. Double-click the **WhDelReqWeb_I** interface.

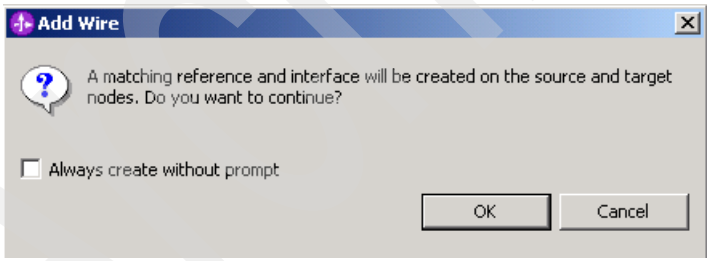


Figure 12-10 Creating a source and reference at the same time

10. Repeat step 9 to connect the reference from the **WholesalerAPlaceOrderRG** component and wire it to the WholesalerAPlaceOrder import using the WhPlaceOrdWeb_I interface.

Archived

Creating the component for the BestWholesaler

The final component that we need to wire in our Assembly Editor is the logic that takes two WholesalerOrder business objects, compares them, and returns the wholesaler to order from. In this section, we create the empty component that represents this logic.

1. Click the **Component (with no implementation type)** icon ( ) from the palette and add it to the palette.
2. Change the name of the new component to `BestWholesalerRG`.
3. Add the BestWholesaler interface to the component.
4. Wire the **BestWholesalerPartner** reference from the PlaceOrderProcess component to the BestWholesaler component.
5. Right-click **BestWholesalerRG** and select **Change Type** → **Rule Group**.
6. The wiring in the Assembly Editor is now complete. Save the assembly diagram. Figure 12-11 shows how the diagram should look.

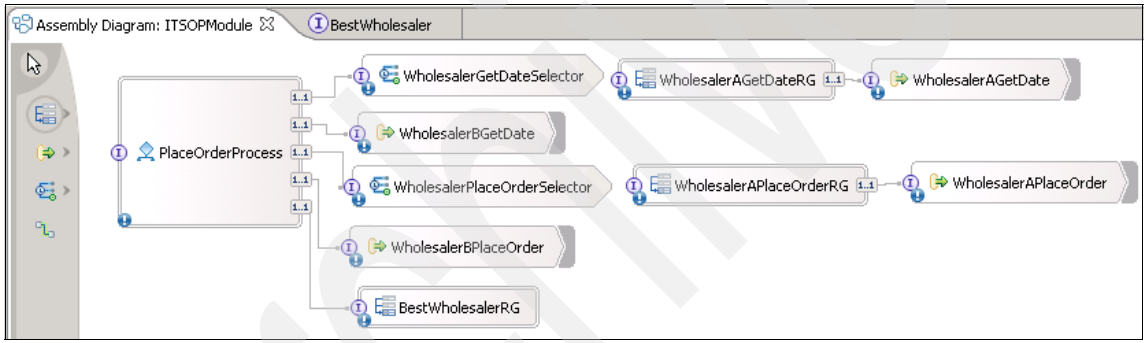


Figure 12-11 The completed assembly diagram

> **Attention:** Selectors do not have references. Therefore, you will not see any wires created for the components that they use.

12.3.4 Creating the component implementations

In the previous section, we created all the components but provided no implementation details. In this section, we generate implementations that provide the behavior required to complete the scenario.

Implementing the PlaceOrderProcess component

The PlaceOrderProcess is the primary component for invocation. When the clerk wants to place an order, this is the component that will begin the required processing. Since this component will control flow, we implement it as a Web Services Business Process Execution Language (WS-BPEL) process.

1. In the Assembly Editor, right-click the **PlaceOrderProcess** component and click **Generate Implementation**.
2. Accept the defaults in the window that opens and click **OK**.

Figure 12-12 shows the business process editor when it opens.

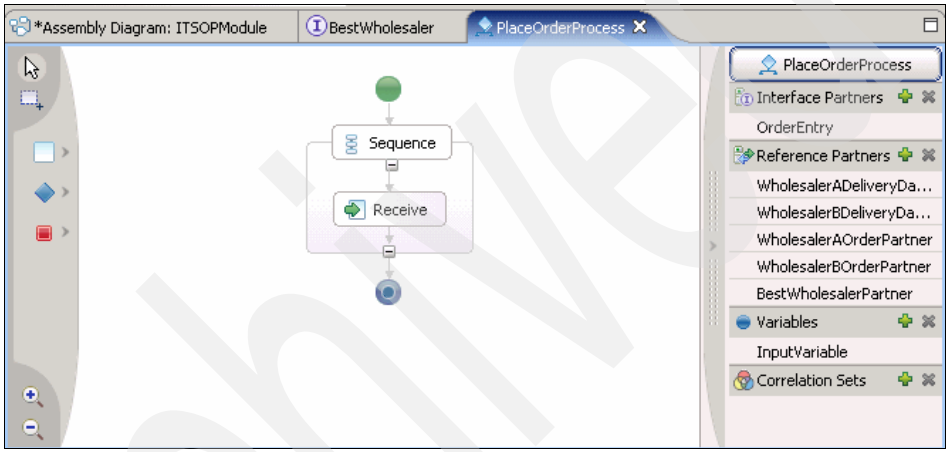


Figure 12-12 The generated business process for the PlaceOrderProcess component

The first stage of the process

In this stage of the process, we configure the initial receive, create the parallel wholesaler invocations, and make our process long running.

1. In the Business Process Editor, create the following items, which are shown in Figure 12-13, in the order presented:
   a. A Parallel Activities node
   b. Three assign nodes: Create Wholesaler A Request, Create WholesalerOrder, and Create Wholesaler B Request
   c. Two invoke activities: Invoke Wholesaler A GetDays and Invoke Wholesaler B GetDays
   d. One snippet: Create WholesalerOrder

Archived

e. Links between each activity. To link components, right-click the **first component**, select **Add Link**, and then click the **second component** to complete the link.

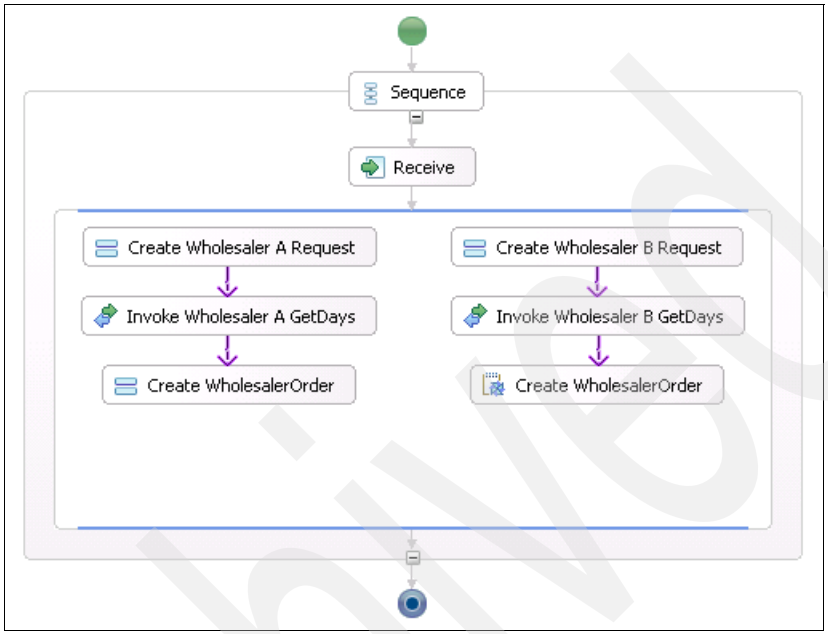


Figure 12-13 The first stage of the business process

2. Create a variable named `InitialOrder` with the data type `Order`.
3. Click the **Receive** node.
   a. Click the **Details** page
   b. Enable the **Use Data Type Variables** check box.
   c. Set the initialOrder variable to **InitialOrder**.
4. Create a variable named `WholesalerARequest` with the data type `WholesalerDeliveryDaysRequest`.
5. Create a variable named `WholesalerAResponse` with data type `WholesalerDeliveryDaysResponse`.

Archived

6. Click the **Create Wholesaler A Request** assign node and make the following assignments from Table 12-6.

Table 12-6 The Create Wholesaler A Request assign activity

Order	From		To	
Create Wholesaler A Request				
1	Variable	InitialOrder → partId	Variable	WholesalerARequest → partNo
2	Variable	InitialOrder → quantity	Variable	WholesalerARequest → quantity

7. Click the **Invoke Wholesaler A Get Days** invoke activity.
   a. Set Partner to **WholesalerADeliveryDaysPartner**.
   b. Set the input data type variable to **WholesalerARequest**.
   c. Set the output data type variable to **WholesalerAResponse**.
   d. Click the **Server** tab.
   e. Set the Transactional Behavior to **Requires Own**.

Restriction: To achieve true parallelism in a business process, each path must be included in a separate transaction. If the paths are included in the same transaction, serial behavior occurs.

8. Create a new variable called `WholesalerOrderA` of data type `WholesalerOrder`.
9. Click the **Create WholesalerOrder** assign activity and make the assignments from Table 12-7.

Table 12-7 The Create WholesalerOrder assign activity

Order	From		To	
Create WholesalerOrder				
1	Variable	WholesalerAResponse → daysToDeliver	Variable	WholesalerOrderA → daysToDeliver
2	Variable	InitialOrder	Variable	WholesalerOrderA → order
3	Fixed Value	Wholesaler A	Variable	WholesalerOrderA → wholesalerName
4	Fixed Value	A	Variable	WholesalerOrderA → wholesalerId
5	Variable	WholesalerAResponse → price	Variable	WholesalerOrderA → price

Archived

10. Create a new variable named `WholesalerBRequest` with the data type `WholesalerDeliveryDaysRequest`.
11. Create a new variable named `WholesalerBResponse` with the data type `WholesalerDeliveryDaysResponse`.
12. Click the **Create Wholesaler B Request** assign node and make the assignments from Table 12-8.

Table 12-8 The Create Wholesaler B Request assign activity

Order	From		To	
Create Wholesaler B Request				
1	Variable	InitialOrder → partId	Variable	WholesalerBRequest → partNo
2	Variable	InitialOrder → quantity	Variable	WholesalerBRequest → quantity

13. Click the **Invoke Wholesaler B GetDays** activity.
    a. Set Partner to **WholesalerBDeliveryDaysPartner**.
    b. Set the input to **WholesalerBRequest**.
    c. Set the output to **WholesalerBResponse**.
    d. Click the **Server** tab.
    e. Set the Transactional Behavior to **Requires Own**.
14. Create a variable named `WholesalerOrderB` of the data type `WholesalerOrder`.

Archived

15. Click the other **Create WholesalerOrder** snippet and create a visual snippet as shown in Figure 12-14.

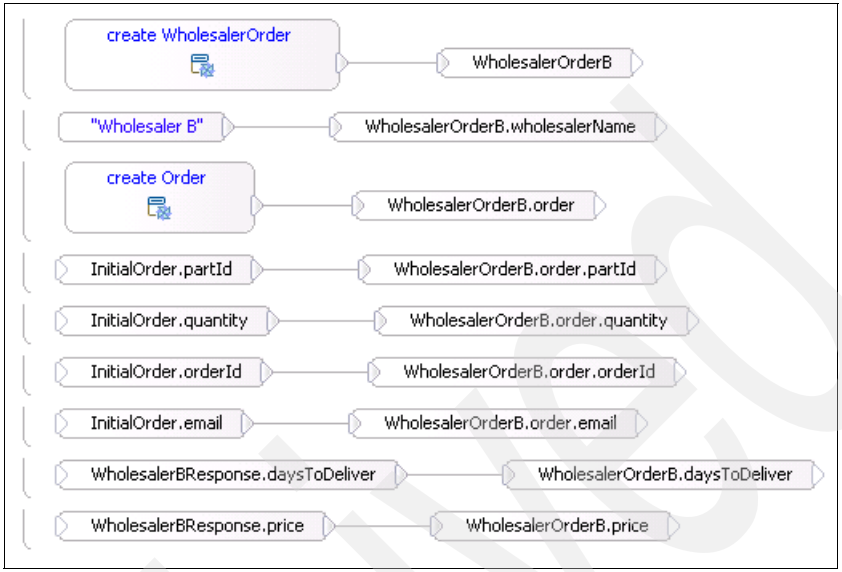


Figure 12-14 The visual snippet for the Create WholesalerOrder activity

Archived

The second stage of the process

In this section, we continue developing our solution by creating the Best Wholesaler logic. We also place the actual order with the selected wholesaler and obtain the wholesaler's confirmation code. In addition, we enable a CEI event for when a wholesaler order is placed.

1. In the Business Process Editor for PlaceOrderProcess, add the activities as shown in Figure 12-15. Add these activities to the end of the process, after the Parallel Activities container.

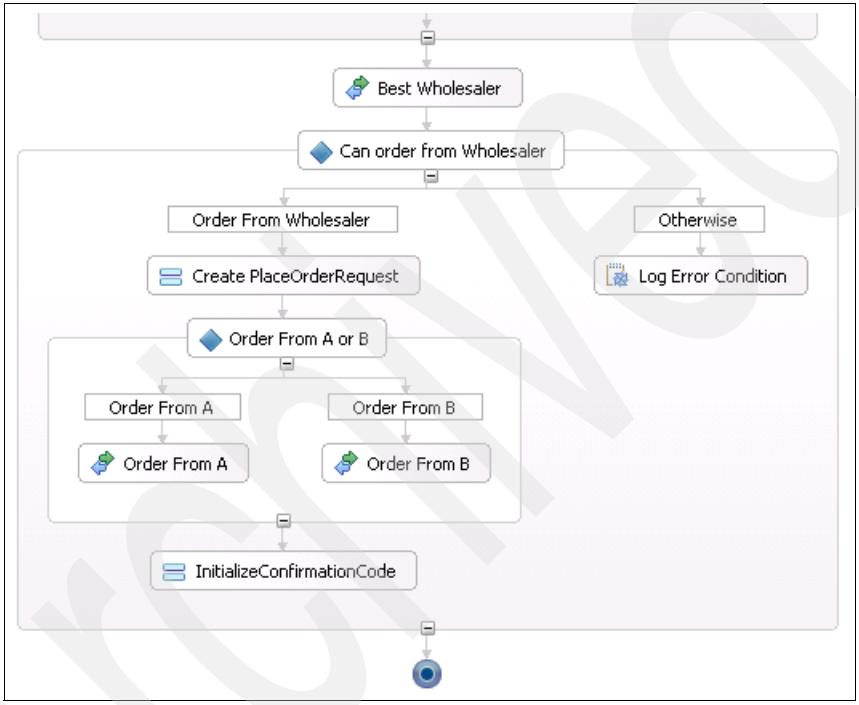


Figure 12-15 The Business Process for the second stage.

2. Create a new variable named `SelectedWholesaler` of type `WholesalerOrder`.
3. Click the **Best Wholesaler** invoke activity.
   a. Click the **Details** tab.
   b. Set Partner to **BestWholesalerPartner**.
   c. Set the data type input for company1 to **WholesalerOrderA**.
   d. Set the data type input for company2 to **WholesalerOrderB**.
   e. Set the data type output for selectedWholesaler to **SelectedWholesaler**.

Archived

4. Click the **Order From Wholesaler** condition.
   a. Click the **Details** tab.
   b. Change the expression language to **Java**.
   c. Complete the visual snippet as shown in Figure 12-16.

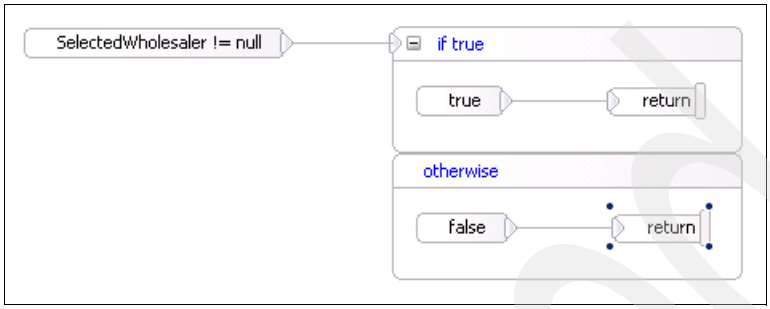


Figure 12-16 The Order From Wholesaler conditional visual snippet

> **Note:** To create the `if true / otherwise` construct, use the Choice component in the Visual Snippet Editor.

5. Create a new variable named `PlaceOrderRequest` of the data type `WholesalerPlaceOrderRequest`.
6. Create a new variable named `PlaceOrderResponse` of the data type `WholesalerPlaceOrderResponse`.
7. Create a new variable named `Price` of the data type `double`.
8. Click the **Create PlaceOrderRequest** assign activity and make the assignments from Table 12-9.

Table 12-9 The Create PlaceOrderRequest assign activity

Order	From		To	
Create PlaceOrderRequest				
1	Variable	SelectedWholesaler → order → partId	Variable	PlaceOrderRequest → partNo
2	Variable	SelectedWholesaler → order → quantity	Variable	PlaceOrderRequest → qty
3	Variable	SelectedWholesaler → price	Variable	Price

Archived

9. Click the **Order From A** condition.
   a. In the Properties view, click the **Details** tab.
   b. Change the expression language to **Java**.
   c. Make the visual snippet look like the one in Figure 12-17.

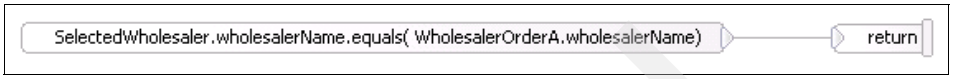


Figure 12-17 The Order From A visual snippet

10. Click the **Order From B** condition.
   a. In the Properties view, click the **Details** tab.
   b. Change the expression language to **Java**.
   c. Make the visual snippet look like the one shown in Figure 12-18.

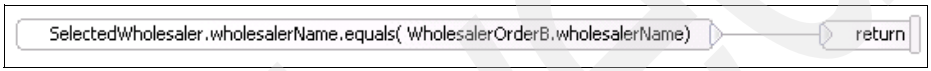


Figure 12-18 The Order From B visual snippet

11. Click the **Order From A** invoke activity.
   a. Set Partner to **WholesalerAOrderPartner**.
   b. Set Input variable to **PlaceOrderRequest**.
   c. Set Output variable to **PlaceOrderResponse**.
   d. Click the **Event Monitor** tab. We can turn on CEI event monitoring here (Figure 12-19).
   e. Click the **Selected** radio box.
   f. Select the **Exit** check box to enable the option.
   g. Set Event Content to **Full**.
   h. Select the **On** check box to enable the option.

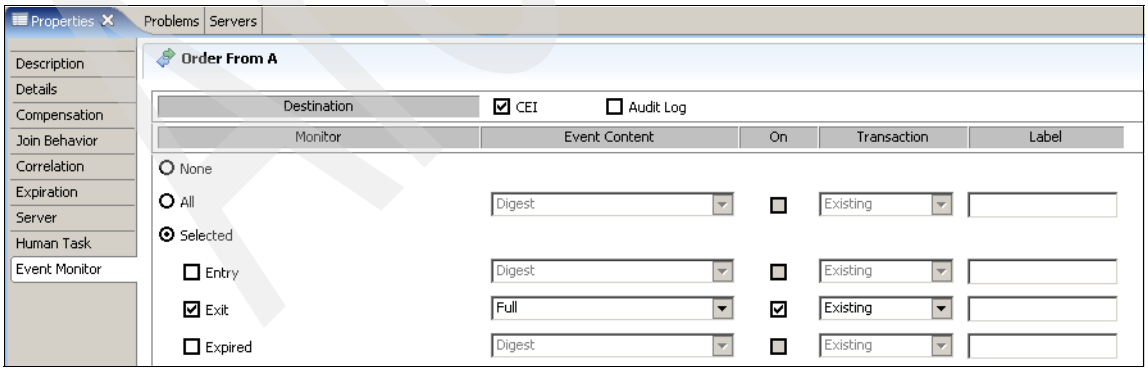


Figure 12-19 Turning on event monitoring

> **Tip:** The Label text box is used only at development time. Any content placed in the field does not appear at run time.

12. Click the **Order From B** invoke activity.
    a. Set Partner to **WholesalerBOrderPartner**.
    b. Set Input variable to **PlaceOrderRequest**.
    c. Set Output variable to **PlaceOrderResponse**.
    d. Click the **Event Monitor** tab.
    e. Click the **Selected** radio box.
    f. Select the **Exit** check box to enable it.
    g. Set the Event Content to **Full**.
    h. Select the **On** check box to enable it.
13. Create a new variable named `ConfirmationCode` of type `string`.
14. Click the **InitializeConfirmationCode** assign activity and configure it as shown in Table 12-10.

Table 12-10 The InitializeConfirmationCode assign activity

Order	**From**		**To**	
InitializeConfirmationCode				
1	Variable	PlaceOrderResponse → confirmCode	Variable	ConfirmationCode

15. Click the **Log Error Condition** snippet and implement the visual snippet shown in Figure 12-20.

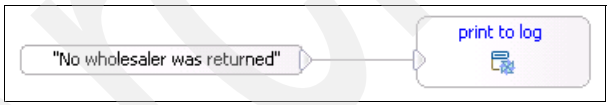


Figure 12-20 The Log Error Condition snippet

16. Click the **Business Process Editor** canvas.
    a. Click the **Details** tab.
    b. Select the **Process is long-running** check box.
17. Save the business process.

This completes the implementation of the PlaceOrderProcess component. We return to this component later to implement the JDBC adapter queries.

Archived

Tip: With the assembly diagram complete, we can use the Integration Test Client to test our process even though we have not defined any implementations for our partner references. The test client automatically emulates the unimplemented components and allows you to enter return values manually.

Setting reference qualifiers

In the Problems view, you see a number of error messages. One of the error messages is:

```
The reference x in the process component file y needs the reference qualifier
'Asynchronous Invocation' with a value of 'commit'
```

To correct this error, complete the following steps:

1. Open the Assembly Editor for ITSOPModule and define an asynchronous invocation reference qualifier as follows:
   a. Right-click a **reference** with the error marker on it. Ensure that the error is the `asynchronous invocation` error as stated previously).
   b. Click **Show In Properties**.
   c. Click the **Details** tab.
   d. Click the **Qualifiers** Tab.
   e. Click the **Add** button.

Archived

f. Select **Asynchronous Invocation** and click **OK**.

g. Click the **Asynchronous Invocation** entry and change the Value field to **Commit** (Figure 12-21).

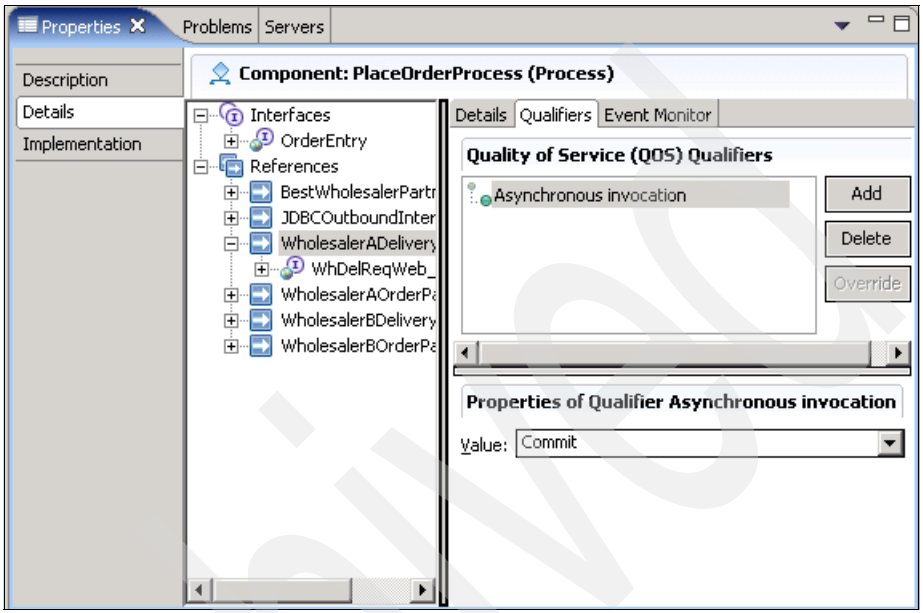


Figure 12-21 Setting a reference qualifier

2. Repeat this process for each of the `asynchronous invocation` errors.

You see another error message in the Problems view stating:

```
The interface x in the process component file y needs the
preferredInteractionStyle attribute with a value of async
```

Archived

To correct this error, perform the following steps in the Assembly Editor of ITSOPModule:

1. Right-click the **error marker** and click **Show In Properties**.
2. Click the **Details** tab
3. Expand **Interfaces**, and select **OrderEntry**.
4. Click the **Details** subtab. Change the value of Preferred interaction style to `Asynchronous` (Figure 12-22).

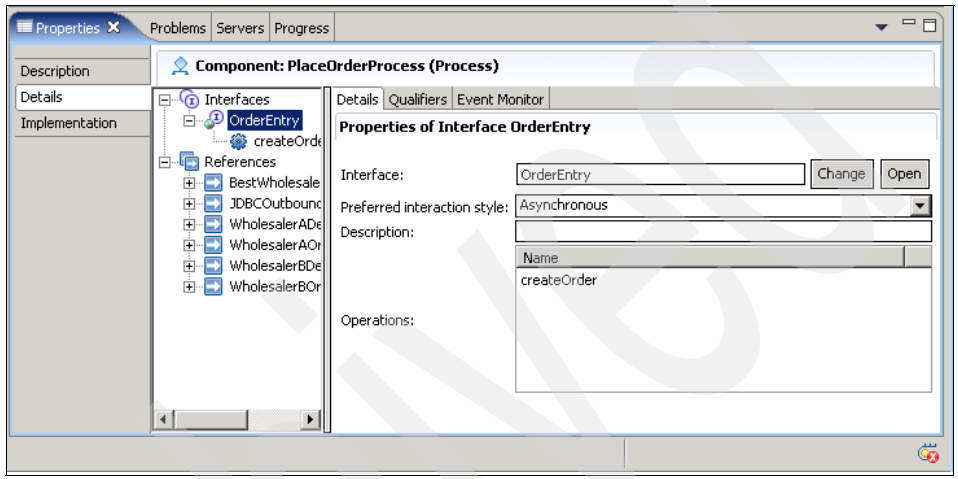


Figure 12-22 The Preferred interaction style interface setting

And finally, a third error message that needs to addressed is:

```
The operation x of the interface y in the process component file z does not
contain the JoinTransaction interface qualifier with a value of 'false'
```

To correct this error, perform the following steps in the Assembly Editor of ITSOPModule:

1. Right-click the **error marker** and click **Show In Properties**.
2. Click the **Details** tab and click the **Qualifiers** subtab.
3. Click **Join transaction** entry, and change the value combo box to `False` (Figure 12-23).

Archived

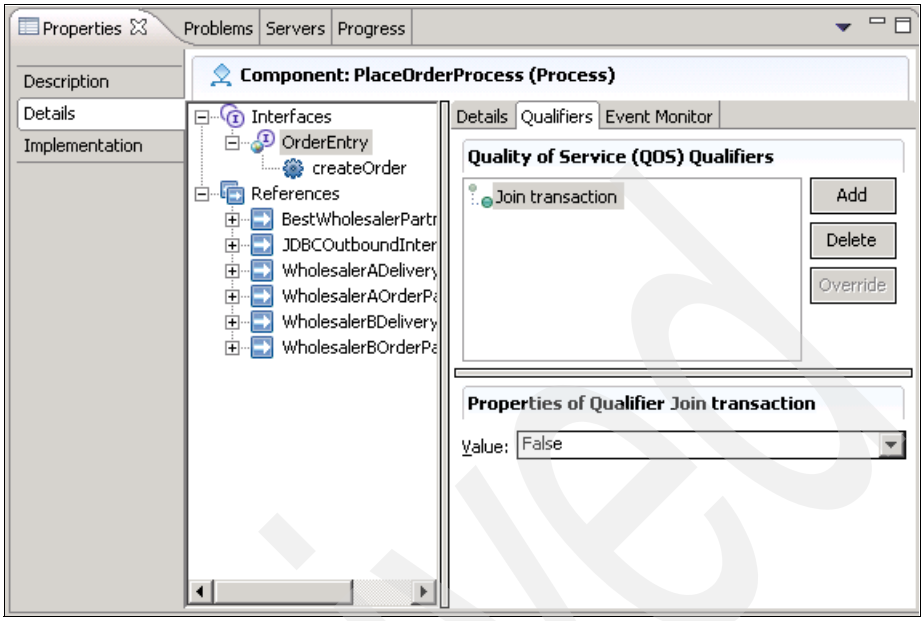


Figure 12-23 The Join transaction interface qualifier

4. Save the assembly diagram. You should see no error messages (only warnings) in the Problems view.

Implementing the components for Wholesaler B

In this section, we create all the implementations required to call Wholesaler B's exposed Web service. This involves copying the Web service port definitions from the WholesalerB module and creating Web service bindings for the two imports (WholesalerBGetDate and WholesalerBPlaceOrder) in the ITSOPModule.

Attention: It is easy to believe that we do not need to copy the Web service ports from the Wholesaler B module into the ITSOPModule. We have to remember that it is strictly coincidence that Wholesaler B exists in our workspace. In a typical scenario, Wholesaler B only provides us with a Web Services Description Language (WSDL) definition and the port binding information, not their entire module. The key to remember for our scenario is that ITSOPModule does not define a dependency on Wholesaler B at all, so it has zero visibility to its artifacts. Thus, we need to copy the information.

Archived

1. In the Business Integration view, expand the **WholesalerB** module and then expand **Web Services Ports**.
2. Copy all three Web service ports to the ITSOPModule module.
3. In the ITSOPModule Assembly Editor, right-click the **WholesalerBGetDate** import, and select **Generate Binding** → **Web Service Binding**.
4. Right-click **WholesalerBGetDate** and click **Show In Properties**.
5. In the Properties view, click the **Binding** tab.
6. Click the **Browse** button.
7. In the Select a WSDL file with binding/service window, expand **ITSOPModule**. Highlight the **WhBExpDelv_WhDelReqWeb_IHttp_Service.wsdl** file and click **OK**.

   Attention: If the binding WSDL file that is chosen is not the one from the ITSOPModule, runtime errors may occur because the module will not be able to resolve the interface.

8. In the Assembly Editor, right-click the **WholesalerBPlaceOrder** import, and select **Generate Binding** → **Web Service Binding**.
9. Right-click **WholesalerBPlaceOrder** and click **Show In Properties**.
10. In the Properties view, click the **Binding** tab.
11. Click the **Browse** button.
12. In the Select a WSDL file with binding/service window, expand **ITSOPModule**. Highlight the **WhBExpPlaceOrd_WhPlaceOrdWeb_IHttp_Service.wsdl** file and click **OK**.
13. Save the changes to the Assembly Editor.

Implementing the components for Wholesaler A

To implement Wholesaler A, we need to implement two selections, the two rule groups (with corresponding business rules) and the two imports that make the call to the Wholesaler A Web service.

1. In the ITSOPModule assembly diagram, right-click the **WholesalerGetDateSelector** component and select **Generate Implementation**.
2. Click **OK** to accept the default location. The selector is generated.

Archived

3. When the Selector Editor (Figure 12-24) opens, under the interfaces heading, click the **getDeliveryDays** operation.

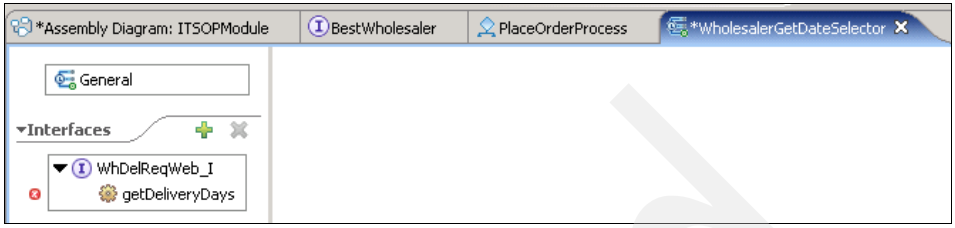


Figure 12-24 The Selector Editor

4. Click **Enter SCA Destination** and click the **WholesalerAGetDateRG** component.
5. Save and close the editor.
6. Back in the ITSOPModule assembly diagram, right-click the **WholesalerPlaceOrderSelector** component and select **Generate Implementation**.
7. Click **OK** to accept the default location.
8. In the Selector Editor, under the interfaces heading, click the **placeOrder** operation.
9. Click **Enter SCA Destination** and click the **WholesalerAPlaceOrderRG** component.
10. Save and close the editor.

Archived

Now that we have implemented the selectors, we need to implement the components that the selectors call. In this case, these components are the WholesalerAGetDateRG and the WholesalerAPlaceOrderRG rule groups.

1. In the ITSOPModule Assembly Editor, right-click the **WholesalerAGetDateRG** component and click **Generate Implementation**.
2. Click **OK** to accept the default location.
3. When the Rule Group Editor (Figure 12-25) opens, click the **getDeliveryDays** operation.

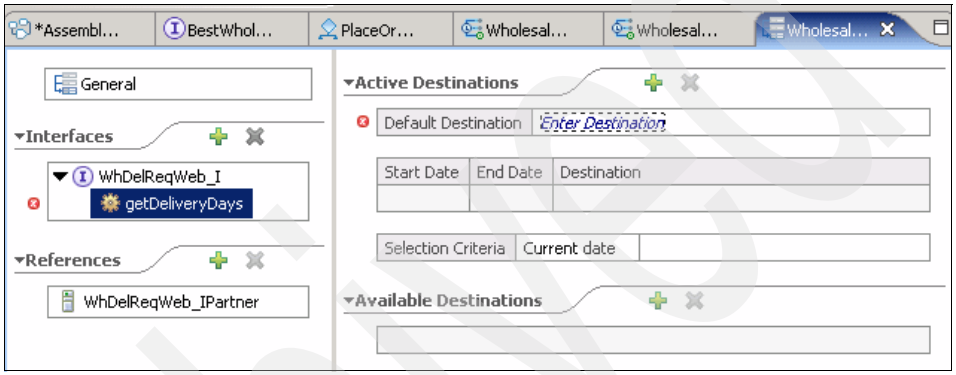


Figure 12-25 The Rule Group Editor

4. Click **Enter Destination**, and select **New Ruleset**.
5. In the New Rule Set window, accept the defaults and click **Finish**. A ruleset has been created.
6. The Rule Set Editor opens. We now need to create an action rule that invokes the import to call the Wholesaler A Web service.
   a. Under the Rules heading, click the **Add Action Rule** icon to create an action.
   b. Click the **Action** hyperlink.
   c. Click **Invoke**.
   d. Enter the following values:
      - Partner: `WhDelReqWeb_IPartner`
      - Operation: `getDeliveryDays`
      - Input: `getDeliveryDaysWholesalerRequest`
      - Output: `getDeliveryDaysWholesalerResponse`
7. Save and close the Rule Set Editor.
8. Save and close the Rule Group Editor.

Archived

We now need to create the corresponding rule group and rule set for the Place Order components of Wholesaler A.

1. In the ITSOPModule Assembly Editor, right-click the **WholesalerAPlaceOrderRG** component and click **Generate Implementation**.
2. Click **OK** to accept the default location.
3. Click the **placeOrder** operation.
4. Click **Enter Destination**, and select **New Ruleset**.
5. In the New Rule Set window, accept the defaults and click **Finish**. A ruleset has been created.
6. The Rule Set Editor opens in which we now need to create an action rule that invokes the import to call the Wholesaler A Web service.
   a. Under the Rules heading, click the **Add Action Rule** icon to create an action.
   b. Click the **Action** hyperlink.
   c. Click **Invoke**.
   d. Enter the following values:
      - Partner: `WhPlaceOrdWeb_IPartner`
      - Operation: `placeOrder`
      - Input: `placeOrderRequest`
      - Output: `placeOrderResponse`
7. Save and close the Rule Set Editor.
8. Save and close the Rule Group Editor.

> **Note:** In our example, we can simply pass the input parameters into the invocation. If the interface we wanted to call is different, then we invoke an interface mapping which then calls the new interface.

We have completed the Selector/Rule Group pattern. To complete the setup of logic to invoke Wholesaler A, we need to configure the imports.

1. In the Business Integration view, expand the **WholesalerA** module, and then expand **Web Services Ports**.
2. Copy all three Web service ports to the ITSOPModule module.
3. In the ITSOPModule Assembly Editor, right-click the **WholesalerAGetDate** import, and select **Generate Binding** → **Web Service Binding**.
4. Right-click **WholesalerAGetDate** and click **Show In Properties**.

Archived

5. In the Properties view, click the **Binding** tab.
6. Click the **Browse** button.
7. In the Select a WSDL file with binding/service window, expand **ITSOPModule**. Highlight the **WholesalerA_GetDeliveryDateExport_WhDelReqWeb_IHttp_Service.wsdl** file and click **OK**.
8. In the Assembly Editor, right-click the **WholesalerAPlaceOrder** import, and select **Generate Binding** → **Web Service Binding**.
9. Right-click **WholesalerAPlaceOrder** and click **Show In Properties**.
10. In the Properties view, click the **Binding** tab.
11. Click the **Browse** button.
12. In the Select a WSDL file with binding/service window, expand **ITSOPModule**. Highlight the **WholesalerA_PlaceOrderExport_WhPlaceOrdWeb_IHttp_Service.wsdl** file and click **OK**.
13. Save the changes to the Assembly Editor. Figure 12-26 shows how the assembly diagram should look.

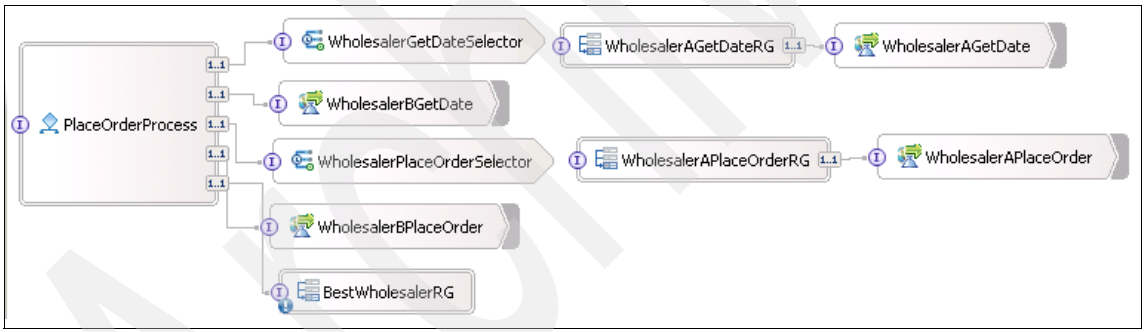


Figure 12-26 The current state of the assembly diagram

Wholesaler A is now fully implemented in our solution. The final part of the scenario is to implement the BestWholesaler rule group.

Implementing the BestWholesaler rule group

In this section, we provide an implementation for the business logic that chooses the better wholesaler given two options.

1. In the ITSOPModule Assembly Editor, right-click the **BestWholesalerRG** component, and select **Generate Implementation**.
2. Click **OK** to accept the default location.

3. In the Rule Group Editor that opens, complete these tasks:
   a. Click the **getBetterWholesaler** operation.
   b. Click the **Enter Destination** hyperlink.
   c. Click **New RuleSet**.
   d. Click **Finish** to accept the default name.
4. In the Ruleset Editor, complete the following steps:
   a. Under the Rules heading, click the **Add If-Then Rule** icon ( ) to create a rule.
      i. Set the name of the rule to `Company2FasterThanCompany1`.
      ii. Click the **Condition** hyperlink and set the condition to `company1.daysToDeliver > company2.daysToDeliver`.
      iii. Click the **Action** hyperlink and set the Action to `selectedWholesaler = company2`.
   b. Under the Rules heading, click the **Add If-Then Rule** icon ( ) to create another rule.
      i. Set the name of the rule to `Company1FasterThanCompany2`.
      ii. Click the **Condition** hyperlink and set the condition to `company1.daysToDeliver <= company2.daysToDeliver`.
      iii. Click the **Action** hyperlink and set the Action to `selectedWholesaler = company1`.
5. Save the getBetterWholesaler Ruleset.
6. Save the BestWholerSalerRG rule group
7. Save the assembly diagram.

All the components are now authored, and our solution is ready for deployment to WebSphere Process Server and invocation in the Integration Test Client.

12.3.5 Configuring the JDBC adapter with the business process

In this section, we guide you through the steps required in WebSphere Integration Developer to use the WebSphere Adapter for JDBC that connects to a DB2 database. In these steps, the JDBC adapter and the DB2 Universal Database are already installed on the same machine as WebSphere Integration Developer.

We use the following software levels:

- IBM DB2 Universal Database V8.2.1
- IBM WebSphere Adapter for JDBC V6.0.0

Archived

Preparing a DB2 database

Using the DB2 Control Center, perform the following administrative tasks:

1. Create a new database called *ITSO_E*.
   a. In the DB2 Control Center, right-click **All Databases** and select **Create Database** → **Standard**.
   b. In the Create Database wizard, specify a database name of `ITSO_E`.
2. Create a table called *Order* in the ITSO_E database. The Order table is used to store orders.
   a. In the DB2 Control Center, click **All Databases** → **ITSO_E** → **Tables** → **Create**.
   b. In the Create Table Wizard, select a schema of **DB2ADMIN** and specify a table name of `Order`.
3. Add the columns to this table using the details from Table 12-11.

Table 12-11 Columns for the DB2 table Order

Field name	Type	Length
ORDERID	VARCHAR	100
PARTID	VARCHAR	100
QUANTITY	BIGINT	
EMAIL	VARCHAR	100
CONFIRMATIONCODE	VARCHAR	100
WHOLESALERID	VARCHAR	100

4. Create a second table in the ITSO_E database called *AccountsPayable* (again with a schema of DB2ADMIN). The AccountsPayable table is used to bill the appropriate department that placed the order. Add the columns using the details shown in Table 12-12.

Table 12-12 Columns for the DB2 table AccountsPayable

Field name	Type	Length
ORDERID	VARCHAR	100
PRICE	DOUBLE	
DEPARTMENT	VARCHAR	100

Archived

Importing the resource adapter

Now we import the JDBC resource adapter into WebSphere Integration Developer and generating the business objects using the Enterprise Service Discovery (ESD) wizard.

1. In WebSphere Integration Developer, click **File** → **New** → **Enterprise Service Discovery**.
2. In the Enterprise Service Discovery window (Figure 12-27), click the **Import Resource Adapter** button.

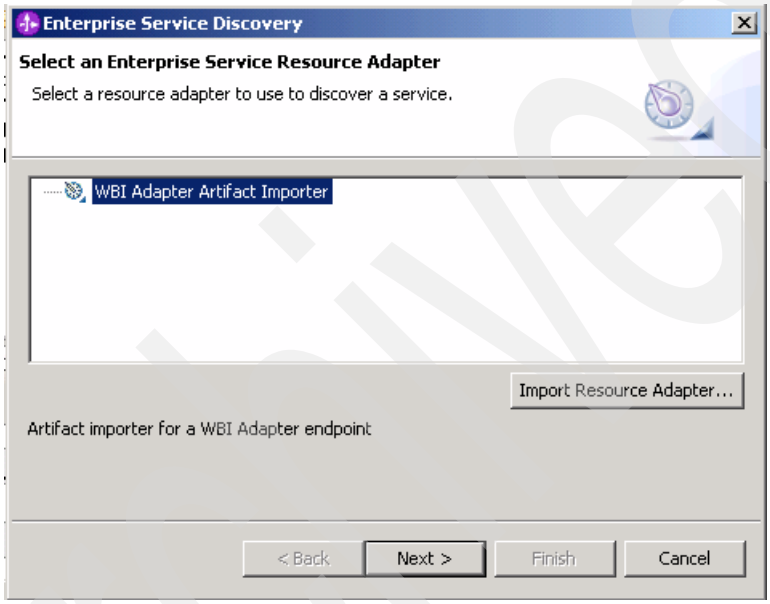


Figure 12-27 The Enterprise Service Discovery wizard

Archived

3. In the Import window (Figure 12-28), click the **Browse** button and select the **CYBC_JDBC.rar** file from the resource adapters installation directory. Then click **Finish**.

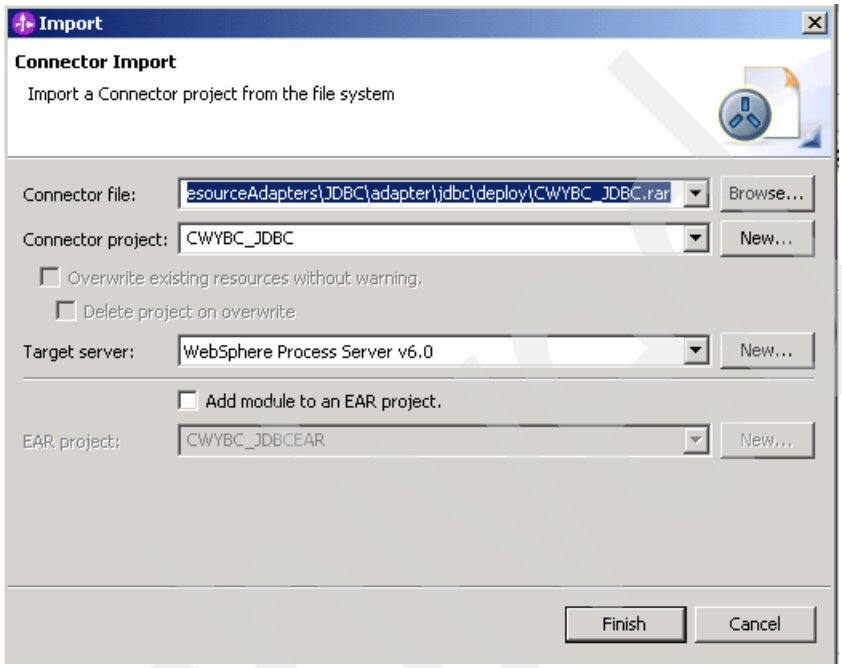


Figure 12-28 The New Connector Import wizard

The WebSphere Adapter for JDBC is imported into the workspace into a connector project called *CWYBC_JDBC*. We need to copy the db2java.zip file into the connector project and add it to the classpath.

> **Important:** If we attempt to use the Enterprise Service Discovery Wizard before adding the db2java.zip to the connector project, the adapter is unable to connect to the database with the following error message:
>
> `The resource adapter named 'JDBC EMD Adapter' returned the following error: 'Failure in connection to EIS COM.ibm.db2.jdbc.app.DB2Driver'`
>
> This translates into an error stating that the adapter could not connect to the DB2 database because the db2java.zip was missing on the classpath.

4. Do not switch to the J2EE perspective if asked. Click **Cancel** to close the Enterprise Service Discovery wizard.

5. Right-click any projects in the Business Integration view and click **Show Files**. The Physical Resources view opens.
6. Expand the **CWYBC_JDBC** connector project.
7. Copy the **db2java.zip** file from the DB2 installation into the connectorModule folder (Figure 12-29). You can find the db2java.zip file in the *DB2_SQL_LIB*/java/ directory. Drag this file from the file system into your WebSphere Integration Developer workspace.

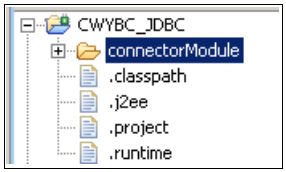


Figure 12-29 The connectorModule folder

8. Right-click the **CWYBC_JDBC** project and click **Properties**.
9. Click the **Java Build Path** property page.
10. Click the **Libraries** tab. Click the **Add Jar** button and click the **db2java.zip** file. Click **OK**.
11. Click **OK** again to close the Properties window.

> **Note:** You may now need to restart WebSphere Integration Developer before proceeding to the next step.

The connector is now sufficiently configured to allow us to progress in the Enterprise Service Discovery Wizard.

12. Click **File** → **New** → **Enterprise Service Discovery**.
13. Click the **JDBC EMD Adapter** entry and click **Next**.

Archived

14. Complete the details of the Enterprise Service Directory window (Figure 12-30) as follows.
    a. For Username and Password, enter valid credentials.
    b. The DatabaseURL field expects a value in the format of `jdbc:db2:`*`databasename`*. In this scenario, set this value to `jdbc:db2:ITSO_E`.
    c. The JDBCDriverClass is `COM.ibm.db2.jdbc.app.DB2Driver`. We use this driver because DB2 is installed on the same machine. If it is installed on a remote machine, consult the product documentation for the proper URL and driver class name.
    d. Click **Next**.

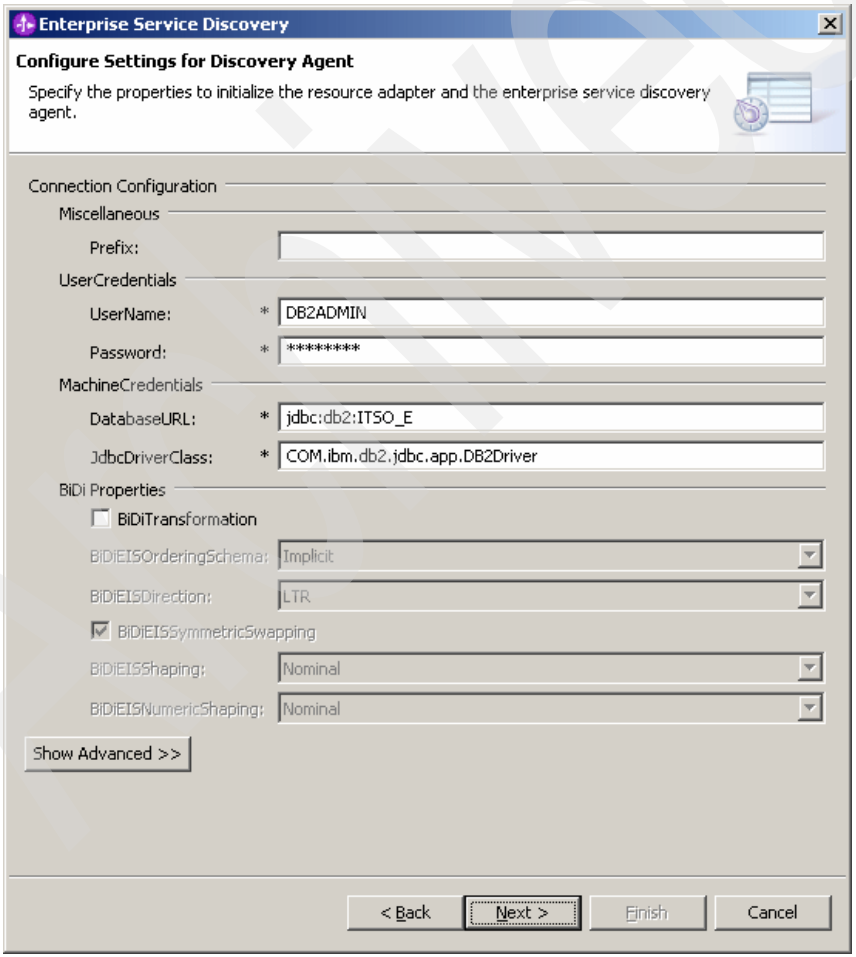


Figure 12-30 Enterprise Service Discovery wizard

15. In the next page of the wizard, we query the DB2 database to find the tables we want to create business objects for.
    a. Click the **Run Query** button.
    b. Import the two tables that we created earlier (Figure 12-31).
        i. Expand **DB2ADMIN** → **Tables**. Highlight **ACCOUNTSPAYABLE** and click **Add**.
        ii. Highlight **ORDER** and click **Add**. Now both ACCOUNTSPAYABLE and ORDER added to the Objects to the imported list.
    c. Click **Next**.

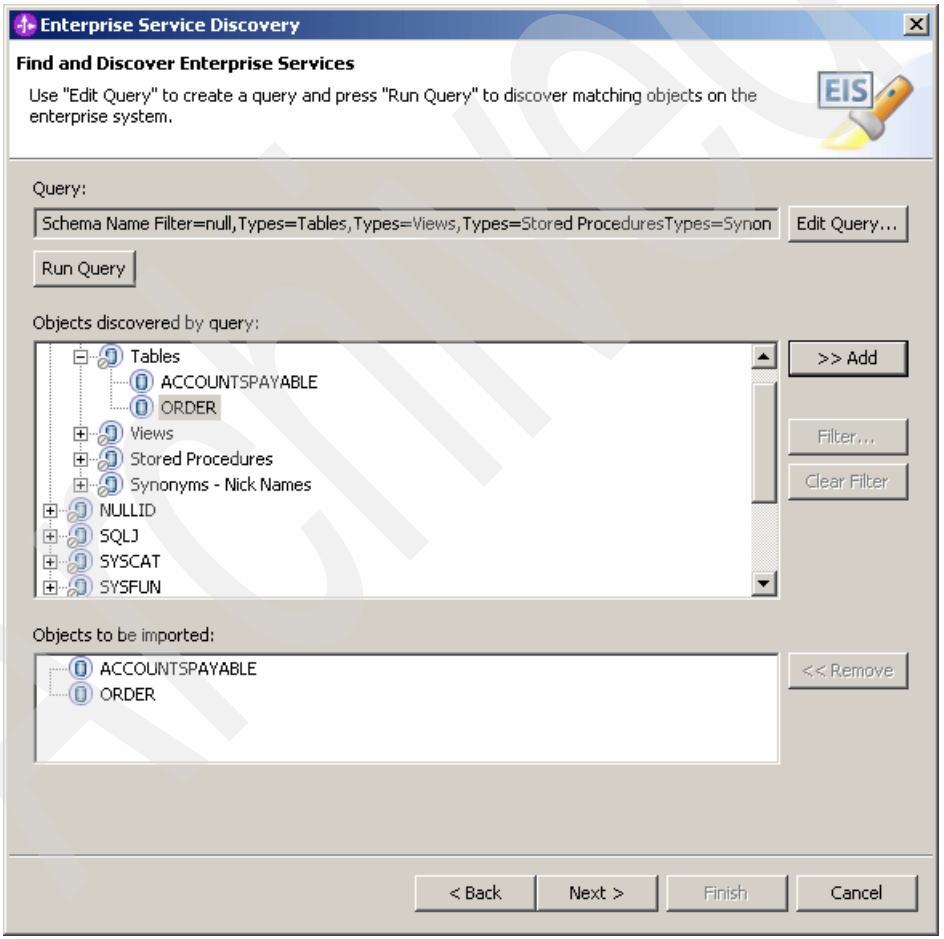


Figure 12-31 Importing tables from the ITSO_E database

Archived

16. In the next page of the wizard, configure the direction for the information to flow to the DB2 database. *Inbound* means that our implementation is started by an event occurring in the database. This can be the addition of a row or any other supported event. When selected, an Export is created to represent the adapter in our assembly diagram.

    Outbound means that our process queries our database and push information out into the database. When selected, an Import is created to represent the adapter in our assembly diagram. We then create invokes in our components to call the import and obtain the data from the database.

    The operations heading tells the adapter which operations on the interface to generate. If an operation is never required, we recommend that you remove it from the interface. An example is removing the delete operation because business guidelines prevent rows from ever being explicitly deleted.

    In our example, we want to query data and be able to insert, update and retrieve data. We create an outbound interface with those three operations allowed.

    a. For Service Type, select **Outbound**.

    b. Select the **Delete**, **RetrieveAll**, and **ApplyChanges** operations and click **Remove** (Figure 12-32).

    c. Click **Next**.

Archived

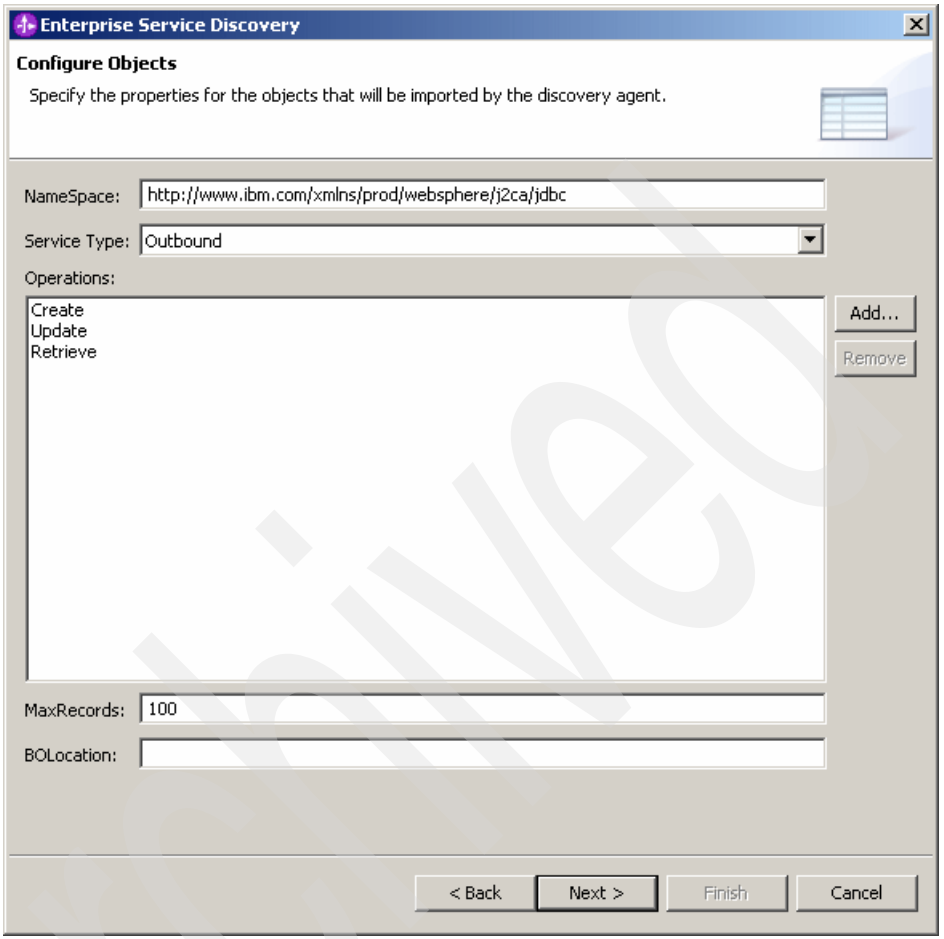


Figure 12-32 Configuring the adapter for outbound services

17. In the Generate Artifacts window, click the **Use discovered connection properties** radio button. The section expands. Enter the values shown in Table 12-13.

Table 12-13 The fields to complete in the wizard page

Field name	Value
Username and Password	The credentials to log into the database
J2C Authentication Data Entry	widNode/db2admin
DatabaseVendor	DB2

Archived

> **Attention:** The runtime guidelines contain the steps to properly configure the J2C Authentication Alias field. If you do not follow these steps, the server is unable to create a connection to DB2, and a `NullPointerException` is thrown at run time.
>
> Also, if the Authentication Alias cannot be found, this exception occurs:
>
> `javax.resource.ResourceException: LoginException getting Subject`
>
> **Note:** If you need to update these values, you can find them in the properties of the import binding in the Assembly Editor.

Click **Finish**.

18. The wizard creates two new *business graphs* of the form *SchemaName*BG and *TableName*BG. In our scenario, it created Db2adminAccountspayableBG and AdminOrderBG. You can find these in the Data Types section of ITSOPModule in the Business Integration view.

    The business graphs are wrappers that provide additional metadata functionality to purely data oriented business objects.

19. Open the Assembly Editor for ITSOPModule. It contains a new import named *JDBCOutboundInterface*.
20. Right-click **JDBCOutboundInterface** and select **Show in Properties**.
21. Click the **Details** tab and expand **Interfaces** → **JDBCOutboundInterface**. Notice there are six operations, three for each table (Create, Retrieve, and Update).
22. Save the assembly diagram.

Updating the business process

Now that we have created the import in our Assembly Editor, we need to update our business process to use it.

1. Open the ITSOPModule Assembly Editor.
2. Create a reference from the PlaceOrderProcess component to the JDBCOutboundInterface import.
   a. Right-click **PlaceOrderProcess** and select **Add** → **Reference**.
   b. In the Add Reference window, select **JDBCOutboundInterface** and click **OK**.
3. Wire a connection between the newly created reference and the JDBCOutboundInterface component.
4. Save the assembly diagram.

The reference has been created. We need to update the business process to use it.

1. In the Business Process Editor, open the **PlaceOrderProcess** business process.
2. Add a new business logic at the end of the existing business process logic. Add a scope that contains an assign node and two invoke nodes to the end of the process, as shown in Figure 12-33.

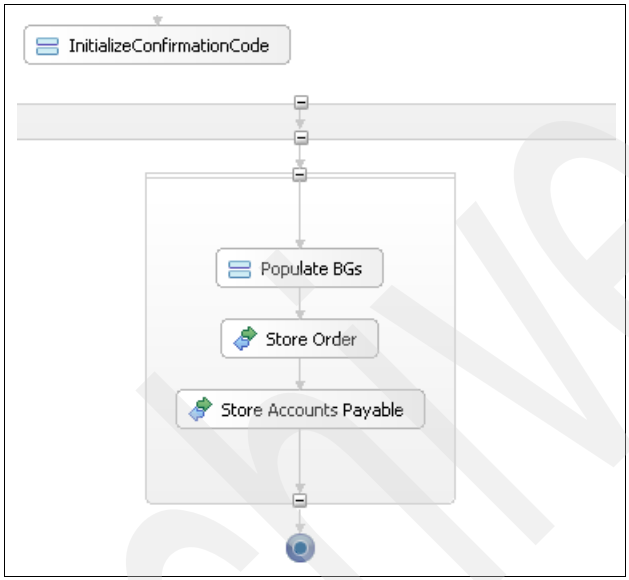


Figure 12-33 Additional activities to add to the PlaceOrderProcess business process

3. Create a new variable named `OrderBG` of the data type `Db2adminOrderBG`.
4. Create a new variable named `AccountspayableBG` of the data type `Db2adminAccountspayableBG`.

Archived

5. Configure the Populate BGs assign activity with the details from Table 12-14.

Table 12-14 The Populate BGs assign activity

Order	From		To	
PopulateBGs assign activity				
1	Variable	SelectedWholesaler → wholesalerName	Variable	OrderBG → Db2adminOrder → wholesalerid
2	Variable	SelectedWholesaler → order -> partId	Variable	OrderBG → Db2adminOrder → partid
3	Variable	SelectedWholesaler → order → quantity	Variable	OrderBG → Db2adminOrder → quantity
4	Variable	ConfirmationCode	Variable	OrderBG → Db2adminOrder → confirmationcode
5	Variable	SelectedWholesaler → order → email	Variable	OrderBG → Db2adminOrder → email
6	Variable	SelectedWholesaler → order → orderId	Variable	OrderBG → Db2adminOrder → orderid
7	Fixed Value	Create	Variable	OrderBG → verb
8	Fixed Value	Create	Variable	AccountspayableBG → verb
9	Variable	SelectedWholesaler → order → orderId	Variable	AccountspayableBG → Db2adminAccountspayable → orderid
10	Variable	Price	Variable	AccountspayableBG → Db2adminAccountspayable → price

6. Create a new reference partner named `JDBCOutboundInterfacePartner`.
   a. In the Properties view, click the **Details** tab.
   b. Click the **Browse** button and choose the **JDBCOutboundInterface** interface.

Archived

7. Right-click the **Store Order** invoke activity and click **Show In Properties**.
   a. Click the **Details** tab.
   b. Set the partner to **JDBCOutboundInterfacePartner**.
   c. Set the operation to **createDb2adminOrder**.
   d. Set the operation input variable to **OrderBG**.
   e. Set the operation output variable to **OrderBG**.
8. Right-click the **Store Accounts Payable** invoke activity, and click **Show In Properties**.
   a. Click the **Details** tab.
   b. Set the partner to **JDBCOutboundInterfacePartner**.
   c. Set the operation to **createDb2adminAccountspayable**.
   d. Set the operation input variable to **AccountspayableBG**.
   e. Set the operation output variable to **AccountspayableBG**.
9. Save the business process.

Completing the process

The final task is to fix the errors in the Problems view.

1. Open the ITSOPModule assembly diagram.
2. You see another `asynchronous invocation` error message in reference to the JDBCOutboundInterface component. To remove the error:
   a. Right-click the **error marker** on the reference and click **Show In Properties**.
   b. Click the **Details** tab and then the **Qualifiers** tab. Click the **Add** button.
   c. Select **Asynchronous Invocation** and click **OK**.
   d. Click the **Asynchronous Invocation** entry and change the Value field to `Commit`.
3. You may see two more errors, relating to the PlaceOrderProcessArtifacts.wsdl file. To resolve these errors:
   a. Switch back to the **Physical Resources** view.
   b. Right-click **PlaceOrderProcessArtifacts.wsdl** and select **Open With** → **XML Source Page Editor**.
   c. In the WSDL file, note the `name=""` declaration:

      ```
      xmlns:xsd="http://www.w3.org/2001/XMLSchema" name=""
      ```

      Remove `name=""` and save the WSDL file.

There should be no further errors in the Problems view. This completes the development of the Parallel Process application. The next section explains how to prepare the runtime environment and test the process.

Attention: Before you can execute the business process, you need to configure a JDBC adapter as explained in “WebSphere Adapter for JDBC” on page 618.

12.4 Runtime guidelines

In the following sections, we explain how to configure resources to run and test the Parallel Process application.

12.4.1 Configuring DB2 as Business Process Choreography database

The WebSphere Process Server test server provided with WebSphere Integration Developer uses IBM Cloudscape as the database for Business Process Choreography. The advantages of using Cloudscape are:

- It requires no configuration or administration.
- It uses a small footprint.
- It is automatically installed as part of the WebSphere Integration Developer installation.

While Cloudscape is an excellent database for use in a test environment, it should never be used in production. Furthermore, where a process contains concurrent paths of execution (a Parallel Process), each path only truly executes in parallel if a supported database, other than Cloudscape, is used for the Business Process Choreography database.

For testing purposes, Cloudscape can still be used with parallel processes, but users should be aware that the process does not execute in parallel at run time. For production solutions, we recommend that you use DB2 Universal Database for the Business Process Choreography database, which supports parallel execution.

For instructions to help you configure DB2 Universal Database for this purpose, see the WebSphere Process Server Information Center at:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp

Specifically, you need to create a database, schema, and tablespace using a set of Data Definition Language (DDL) scripts provided with the WebSphere Process Server run time, and then assign that database to the Business Process Container. You can learn how to do this by referring to the WebSphere Process Server information Center at:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wsps.ins.doc/doc/bpc/t2codbdb.html

12.4.2 Configuring the WebSphere Adapter for JDBC

Before you can test the complete business process, you must first configure the WebSphere Adapter for JDBC in the test environment of WebSphere Integration Developer.

1. Start the server.
2. Right-click the server and select **Add and remove projects**. Add the **ITSOPModuleApp** project to the server. This project uses the WebSphere Adapter for JDBC, which we need to configure at run time.
3. Right-click the **server** again and select **Run administrative console**.
4. Log in to the administrative console, and then follow the instructions in "WebSphere Adapter for JDBC" on page 618.

12.4.3 Testing the application

You can test the application using the Integration Test Client. For full details about how to use this test client, see "Starting run time and deploying modules to the test environment using the Integration Test Client" on page 572.

In this section, we make two test cases:

- Test Wholesaler B as the selected wholesaler
- Test Wholesaler A as the selected wholesaler

Before starting the test, ensure the server is started, and the following projects are deployed to it using the Add/Remove Projects wizard:

- WholesalerAApp
- WholesalerBApp
- ITSOPModuleApp (This should already be added from the previous section.)

Testing Wholesaler B as the selected wholesaler

To launch the Integration Test Client:

1. Right-click the **ITSOPModule** module and **select Test** → **Test Module**.
2. In the Integration Test Client under Detailed Properties, select **PlaceOrderProcess** as the component. This component only has a single interface and operation, so they are automatically selected.
3. We use a test case where an order is placed with Wholesaler B. Enter the following parameters under Initial request parameters (Figure 12-34):
   - orderId: `order#1`
   - partId: `part4`
   - quantity: `10`
   - email: `orders@ibm.com`
   - department: `Accounting`

   Click **Continue**.

Initial request parameters

Name	Type	Value
⊟ initialOrder	Order	
orderId	string	order#1
partId	string	part4
quantity	int	10
email	string	orders@ibm.com
department	string	Accounting

Figure 12-34 Parameters for invocation of the parallel process

4. The deployment location window opens. In this window, you can choose the run time to execute the test against. In our case, only one server is defined, so it is selected by default. Click the **Finish** button.

The test now runs. After the test is complete, the editor should look like the example in Figure 12-35. You should see the following information:

- Calls in parallel to both Wholesaler A and Wholesaler B
- A delivery day estimate of 10 days from Wholesaler A and 5 days from Wholesaler B
- The selection of Wholesaler B as the better wholesaler
- Order placement with Wholesaler B
- A request to create entries in the Order table and AccountsPayable table

Archived

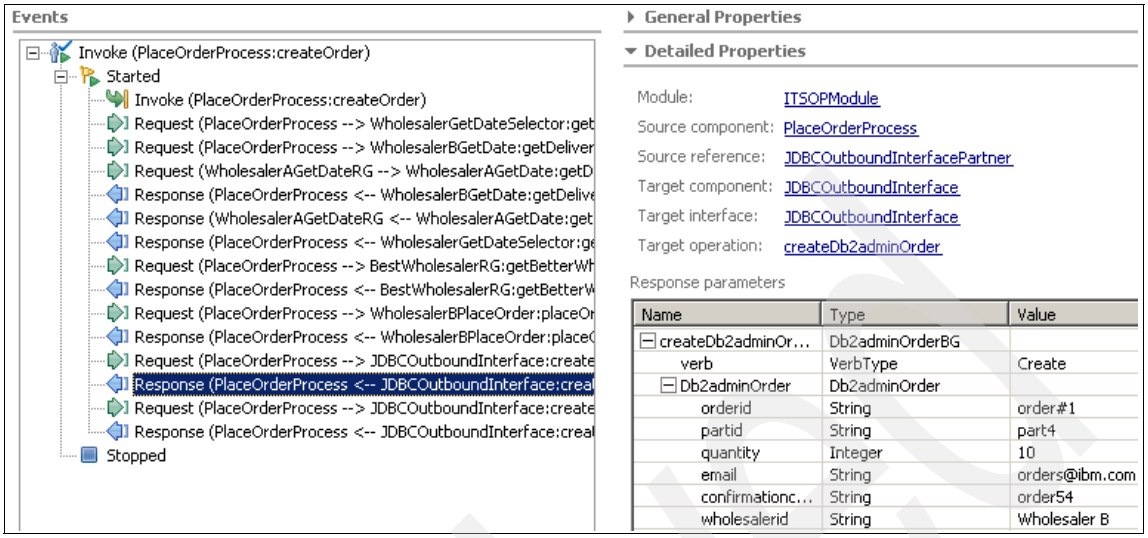


Figure 12-35 Successful test invoking Wholesaler B

Checking the entries in the DB2 tables

Check that the two entries were made in DB2:

1. Open a DB2 Command window.
2. Enter the following commands:

```
db2 connect to ITSO_E
db2 select * from DB2ADMIN.ORDER
```

   You should see the table is populated with the values shown in Figure 12-35.

3. Additionally, check that the AccountsPayable table has also been updated:

```
db2 select * from DB2ADMIN.ACCOUNTSPAYABLE
```

Testing Wholesaler A as the selected wholesaler

We now test the case so that Wholesaler A is chosen. Test the PlaceOrderProcess again this time specifying the following settings:

- orderId: `order#2`
- partId: `part1`
- quantity: `15`
- email: `orders@ibm.com`
- department: `Accounting`

Click **Continue** and **Finish** in the window to start the test.

Archived

The test now runs. After the test is complete, the editor should look like the example in Figure 12-36. You should see the following information:

- Calls in parallel to both Wholesaler A and Wholesaler B
- A delivery day estimate of 4 days from Wholesaler A and 10 days from Wholesaler B
- The selection of Wholesaler A as the better wholesaler
- Order placement with Wholesaler A
- A request to create entries in the Order table and AccountsPayable table

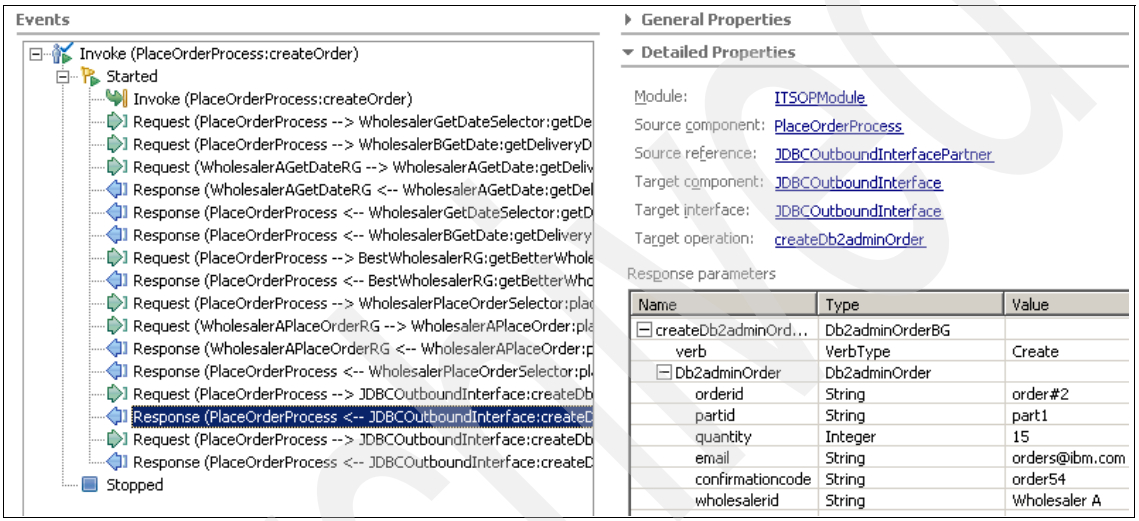


Figure 12-36 Successful test invoking Wholesaler A

Check the DB2 tables to ensure the Order and AccountsPayable tables were updated appropriately.

Archived

13

Parallel Workflow scenario

This chapter details the construction of a process using the Parallel Workflow pattern. It describes the following items:

- Business scenario: For the ITSO Electronics business problem
- Design guidelines: For building a process conforming to the Parallel Workflow pattern
- WebSphere Process Server guidelines: For implementing the process using WebSphere Process Server

This chapter illustrates the use of the following WebSphere Process Server features:

- Compensation
- Event handling
- Fault handling
- Common Event Infrastructure (CEI) monitoring
- Human tasks

© Copyright IBM Corp. 2006. All rights reserved.

13.1 Business scenario

In this scenario, we build upon the Parallel Process scenario described in Chapter 12, “Parallel Process scenario” on page 331, but make some changes.

Immediately when a request is placed to ITSO Electronics, an acknowledgement of the request is generated to indicate acceptance of the order. After the order is placed with the wholesaler, ITSO Electronics now requires that the assembly date is manually approved before it is returned to the consumer. This manual approval is handled by an approver. If the original approver is not available, the task is escalated. If the date is not approved, the order with the wholesaler, and the acknowledgement, both need to be cancelled.

In addition it was decided that the process needs to accept a cancellation of an ITSO Electronics order at any time. If this happens, the process needs to cope with the cancellation event and to compensate any requests. Figure 13-1 outlines the process flow.

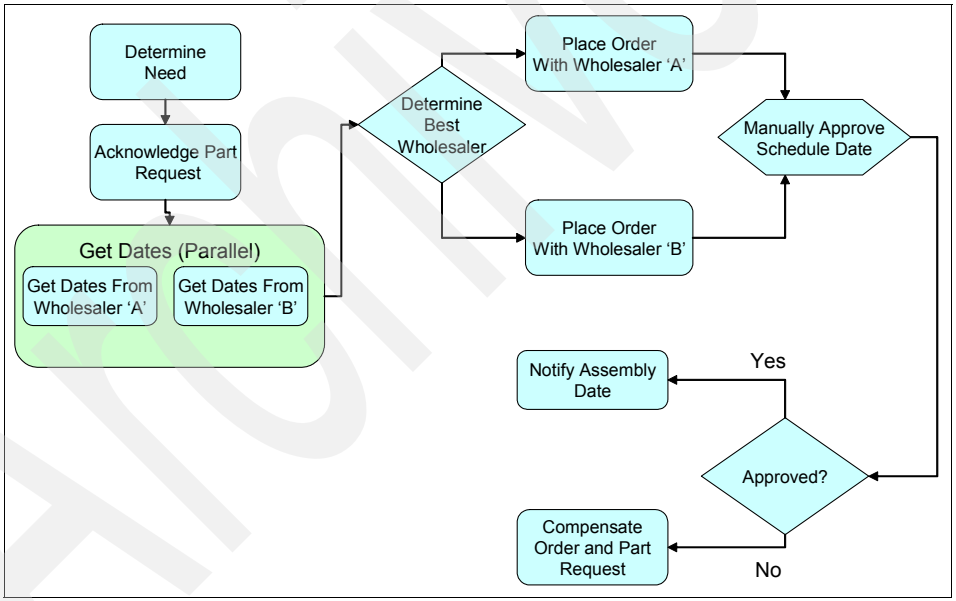


Figure 13-1 The ITSO Electronics parallel workflow process

Note: For simplicity in this book, and because it is covered in Chapter 12, “Parallel Process scenario” on page 331, the transactional update of the accounting system is removed from this flow. The logic could be added after the “notify assembly date” activity shown in Figure 13-1.

13.2 Design guidelines

Many of the guidelines from the previous scenario chapters apply here. In this chapter, we concentrate on the new concepts introduced with this scenario, including human interaction, compensation of previously completed events, and handling events which arrive in midprocess.

We reuse the services that we built elsewhere in this book, especially Wholesalers A and B, where we can now use the order cancellation operation provided in the interface.

We follow these steps:

1. Analyze the business requirements.
2. Select a pattern.
3. Analyze the design options.

13.2.1 Analysis of the business requirements

The business requirements specific to this implementation are:

- Acknowledge the order immediately after it is received.
- Add human approval of the wholesaler order after it is placed.
- If the wholesaler order is not approved, cancel the wholesaler order and the acknowledgement.
- Ensure that the process responds correctly to an order cancellation received at any time during the process, compensating if necessary.

13.2.2 Pattern selection

In this section, we use the Patterns for e-business layered asset model to determine the patterns, and ultimately the product mappings, that best fit our business scenario.

Business pattern

We choose the Application Integration pattern because this pattern is described as bringing together multiple applications and information sources. We recommend that you use this pattern when multiple applications must be combined to automate a new business function.

Archived

Application pattern

After choosing a Business pattern, we choose an associated Application pattern. Because this scenario is a business process, we focus on the Process Integration patterns shown in Figure 13-2.

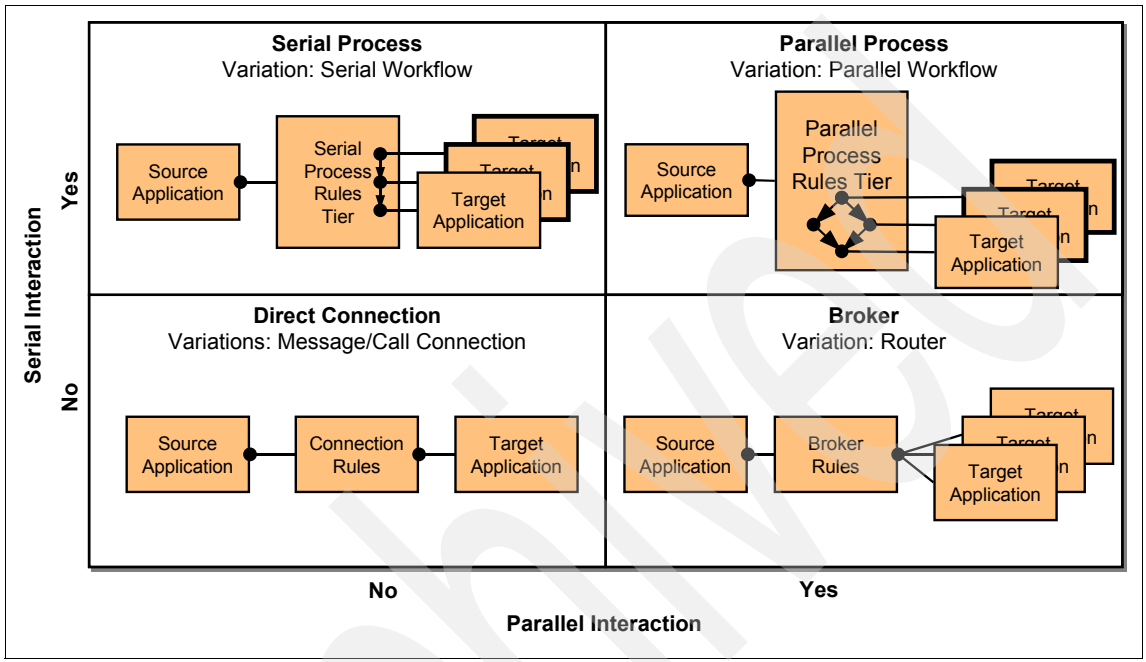


Figure 13-2 Process Integration patterns

Business processes are described by the Serial Process and Parallel Process application patterns. Since we query both Wholesaler A and Wholesaler B in parallel, we choose the Parallel Process from the top right quadrant of the table.

The pattern has a variation if human intervention is required. In this case, because human interaction is used to approve the order, the final result is the Parallel Workflow variation.

Archived

Runtime pattern

Figure 13-3 illustrates the Runtime pattern for the Parallel Workflow runtime pattern. This Runtime pattern has two variants: the generic profile and the service-oriented architecture (SOA) profile. Although our architecture is service-oriented, the SOA profile is intended for use with an enterprise service bus (ESB). We choose the generic profile for this scenario because we do not use an ESB.

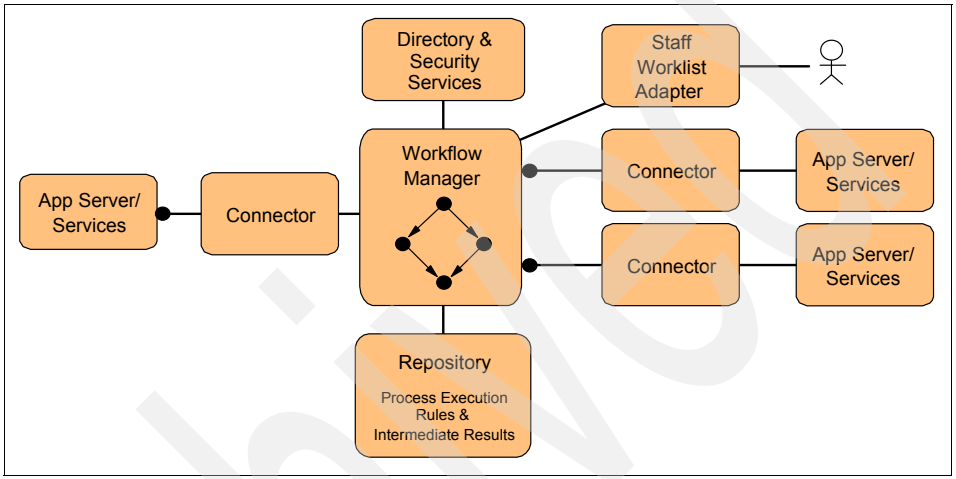


Figure 13-3 Parallel Workflow runtime pattern: Generic profile

Archived

Product mappings

Having selected the Parallel Workflow variation, we can add product mappings to each of the nodes as shown in Figure 13-4. This product mapping adds a few nodes to the mapping for the Parallel Process to represent human interaction. The Process Manager node is replaced with the Workflow Manager node, although this node is still implemented by WebSphere Process Server.

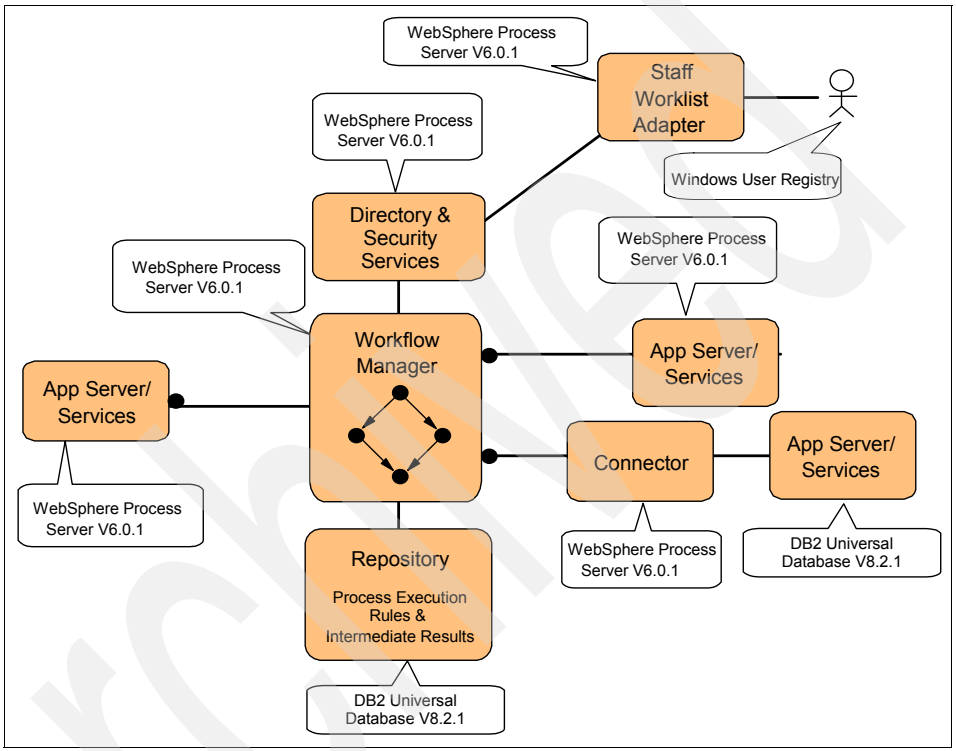


Figure 13-4 Parallel Workflow variation: WebSphere Process Server product mapping

The Workflow Manager invokes processes that require human interaction. The human interactions within the process flow are resolved and handled by the Staff Worklist Adapter, which in this case is the Business Process Explorer included in WebSphere Process Server. Another good option for the Staff Worklist Adapter is the Process Portal component of WebSphere Portal. WebSphere Process Server includes plug-ins for different user directories to store the users and roles necessary for human interaction. We used the Windows User Registry. Another popular choice, and one more likely to be used in production, is a Lightweight Directory Access Protocol (LDAP) directory such as Tivoli Directory Server.

13.2.3 Analysis of the design options

We consider the following options for this scenario:

- Human interaction
- Fault handling and compensation
- Event handling
- Dynamicity

Also, the design options for the Parallel Process pattern that we discussed in Chapter 12, “Parallel Process scenario” on page 331, are relevant to the parallel workflow pattern. For the sake of brevity, we do not discuss them again here.

Human interaction

In many cases, some activities within a process are not automated, but rather they are completed by a person or a team. These tasks can be modeled in WebSphere Process Server using the human interaction capabilities.

Performance

The major problems with using human interaction are speed and availability factors. Performance figures measured in requests per second are not a viable measurement when dealing with human tasks.

This may seem obvious, but it is important when considering other factors that are affected by our process. If we want to use transactions, then we should not hold open transactionally-based resources while waiting for a human to respond. Also when implementing parallel flows, remember that a flow containing a human task is likely to end up on the critical path.

Reducing the cost

We also need to consider the cost of involving humans in our processes. Often humans are used for approval, but we must take into account the cost of the approval. If it costs USD$35 (35 U.S. dollars) to reject a USD$25 purchase order request, you must decide whether it is worth the USD$10 loss to preserve the integrity of the company’s purchasing policy.

It is important that you provide your human task owners with enough information to accomplish the task you want them to perform. People are expensive when it comes to process activities; use them wisely and efficiently. Help them along where you can.

Escalation

Escalation is the practice of moving a human task up the chain, when either no action has been taken against the task for a set period of time or a higher level of authority is needed to advance the task. WebSphere Process Server provides built-in escalation support which can be used in a variety of situations.

Do not confuse escalation with the need to have a “four-eye” check on an activity. In this case, the requirement may simply be that two different people have approved a document, not necessarily that one has a higher approval level than another.

Using Service Component Architecture

Because WebSphere Process Server can provide Service Component Architecture (SCA) bindings to human tasks, it is easy to start a process with a task being performed by people. Then you can replace it later in the evolution of the process by an automated task, perhaps composed of further processes and business rules.

Applying the guidelines to ITSO Electronics

ITSO Electronics wants to approve orders manually using human interaction. They have decided that, in the future, they will implement some business rules to remove some of the simpler decisions, but at the moment, each order needs to be approved by a staff member.

Escalation is implemented as a matter of course, because not all ITSO Electronics’s staff are available to approve or reject orders all of the time.

Fault handling and compensation

Although fault handing is a vital part of business processes, we leave the discussion of it until this chapter so that we can combine it with compensation.

Fault handling

At the simplest level, for any fault that we encounter, we must provide a fault handler to catch the fault and take appropriate action. Appropriate actions include:

- Reporting the error and waiting for system operator recovery
- Retrying a condition, if the fault was generated as a result of a timeout or non-availability of a resource
- Selecting an alternative service which provides similar capabilities to a failed service

- Ignoring the error and carrying on

  This is a valid step in a parallel process or parallel workflow pattern because multiple parallel flows might be trying to find the best alternative, as in our wholesaler A and B example in this scenario.
- Referring the error to a human task for manual intervention
- Compensate any previous steps and cancel the process

Compensations compared to transactions

We can compare compensation to the use of transactions when ensuring process and data integrity. The two concepts are different but help us accomplish the same thing. They ensure that, in a process where multiple updates are made through multiple service calls, either all of the updates are complete or none of them are complete at the end of the process.

> **Important:** Compensations apply only to service calls made through invoke activities. The compensation facilities in WebSphere Process Server do not automatically undo nonservice related logical changes that you have made in your process, for example updating variable values.

Using transactions ensures that none of the updates happens before all are ready to be committed. This requires that all of the services that participate in a transaction can support two-phase commits. It also requires that those resources hold the transactions open for the duration of the transactions.

Some of the reasons why we use compensation as opposed to transactions are:

- The services or resources that we need do not support transactions.
- The time over which the transaction is to take place is too long and we do not want to hold the transactional resources open for that duration.
- Serial dependencies occur here the process requires that we fully complete one update before another.

Archived

Types of compensating actions

The most obvious type of compensating action is to directly undo a call that we previously made. Table 13-1 lists examples of such actions.

Table 13-1 Actions and some compensating activities

Action	Compensation
Create customer	Delete customer
Delete customer	Recreate customer
Update customer record	Replace previous version
Move inventory	Move it back

Often a compensating activity needs to be more involved than the primary service call. It might need knowledge of the previous state of a data item before an update call, for example.

Important: When attempting to compensate an update of a data item, remember that using a compensation approach typically means that you do not have any kind of lock over the update that you originally performed. It may not be safe to presume that the state of a data item is the same as that to which you set it in the activity that you are trying to compensate.

Table 13-2 provides examples of more complex compensating activities.

Table 13-2 More actions and compensations

Action	Compensation
Accounting transaction	Reversal transaction
Invoice sent	Raise credit note
Component used in manufacturing build for order	Order replacement component. Add newly manufactured assembly to stock.

Adverse effects of using compensation

It is important to ensure that you always take into account the full consequences of compensating previously complete actions. For example, you mistakenly deduct USD$1000 from a customer's account and the account balance is only USD$500. If you then compensate by crediting the account with the original USD$1000 that you deducted, the customer is not in the same position as before, because the customer now has to cope with an unauthorized overdraft.

In this situation, if you hold the debit transaction open, and then roll back the transaction before committing, the customer's account stays at USD$500 throughout the transaction, without causing any problems.

Compensating multiple paths and conditional processes

When a process has multiple (not necessarily parallel) paths, it is important to compensate only those steps which have been fully completed.

How then, do we know which activities require compensation? The simple answer is that WebSphere Process Server does it for us. It tracks which activities are successfully completed for each process and compensates only those activities which are complete. If there is a request to compensate an uncompleted activity, WebSphere Process Server recognizes this and does not perform the compensation.

Applying the guidelines to ITSO Electronics

We use compensation in two places in this process:

- If the human interaction to validate the order results in a rejection, we compensate the order with the wholesaler by calling the cancelOrder operation. We also compensate the acknowledgement by sending an "order cancelled" acknowledgement.
- If the external cancellation event is received, we compensate the actions that are successfully completed.

Event handling

Event handling is where a process receives an external event at any point, without an explicit check for that event being designed into every step of the process.

When an event arrives, a process must respond to it. Typically we can respond in four ways:

- Ignore the event, because it is not appropriate here.
- Reject the event, because it is invalid.
- Adjust the process variables and move on.

  The event may add extra information or change something which does not affect the running of the process.
- Stop the process and apply compensation if necessary.

A single process can use all or any of these methods when responding to an event. If you find that your process needs to respond to multiple events, consider whether your process has become event driven. In this case, consider using a business state machine as discussed in Chapter 11, "Exposed Serial Process

scenario" on page 235. A business state machine has the major advantage when responding to events that it is easy for the machine to return to a previous state.

Applying the guidelines to ITSO Electronics

For this scenario, ITSO Electronics wants the process to accommodate the original order being cancelled. Therefore, our process accepts cancellation events at any point along the process and reacts accordingly, by compensating if necessary.

We choose not to use a business state machine here because the process does not return to a previous state when the event is received. The process is cancelled completely.

13.3 Development guidelines

This section describes the development guidelines for building the ITSO Electronics Parallel Workflow process. Figure 13-5 illustrates the process that we build in this chapter.

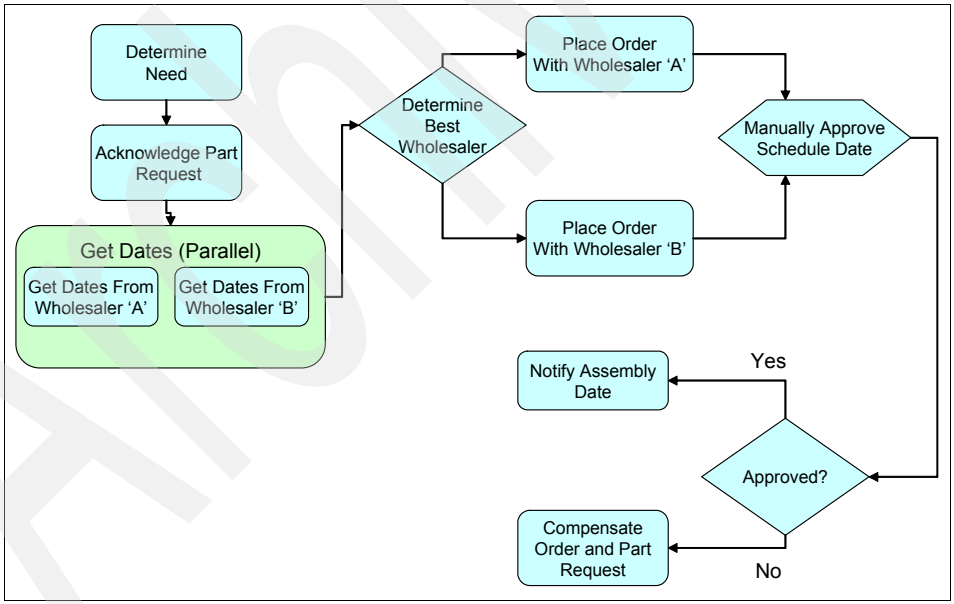


Figure 13-5 Parallel Workflow process overview

We develop the Parallel Workflow process using a bottom-up approach, which means that we develop the business process first. The process contains many of the Web Services Business Process Execution Language (WS-BPEL) elements

such as invoke, choice, parallel activities, visual snippets, assign, throw, rethrow, and compensate. We then wire the process partners with the Wholesaler A and B Web service imports and the BestWholesaler business rule. Then we generate the implementations for the remaining Selectors, business rules, and the human task, and wire the process reference partners to these implementations.

During the development of the Parallel Workflow process, we show you how to implement a number of important features such as:

- Compensation
- Event handling
- Fault handling
- CEI monitoring
- Human tasks

Figure 13-6 shows the components that we build or reuse in this section.

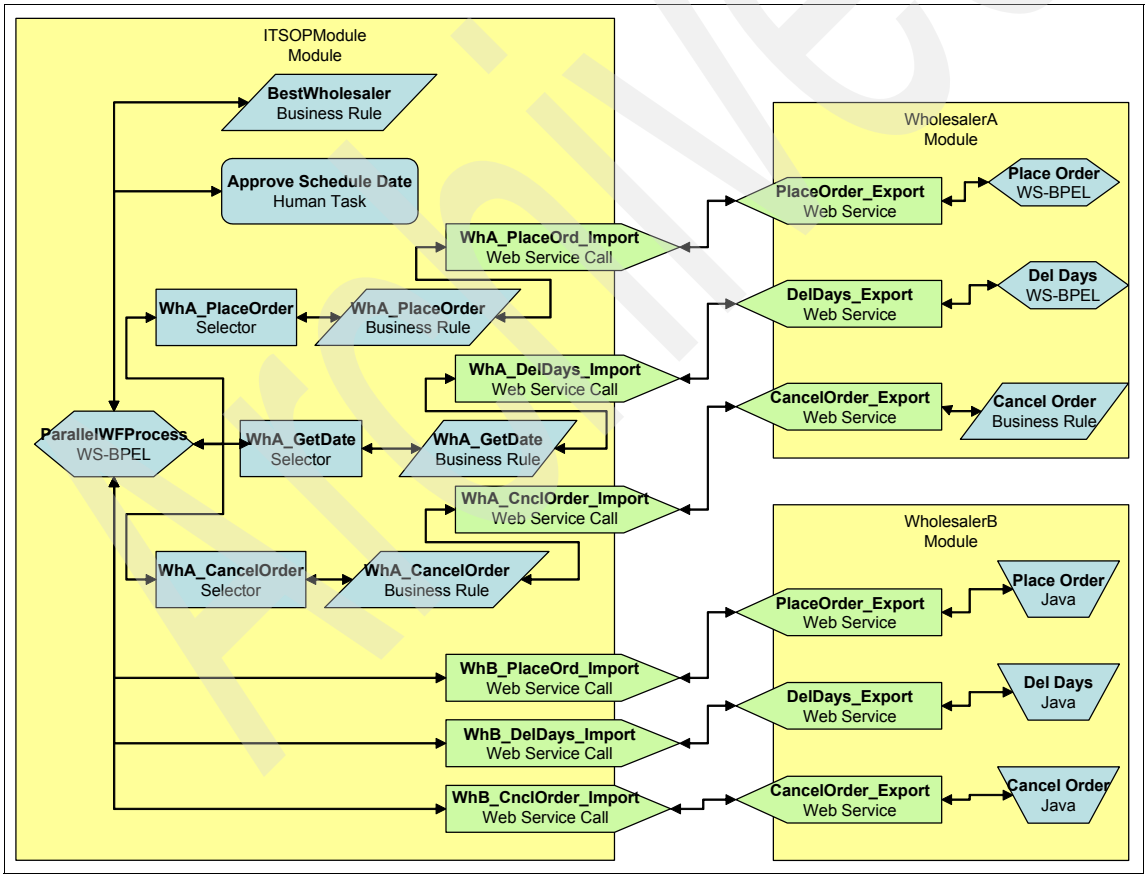


Figure 13-6 Overview of the scenario showing the interlinked components and modules

13.3.1 Setting up the workspace

The following sections explain how to set up the WebSphere Integration Developer workspace. Before you begin these steps, start WebSphere Integration Developer as explained here:

1. Start WebSphere Integration Developer.
2. Select a Workspace directory, for example: **c:\workspaces\ParallelWorkflowProcess**. Click **OK**.
3. Click **Window** → **Open Perspective** → **Business Integration** to open the Business Integration perspective (if not open by default).

Importing ITSOLibrary, WholesalerA, and WholesalerB

This scenario implementation reuses the WholesalerA and WholesalerB modules. For step-by-step instructions to build these modules, see Appendix A, “Building Wholesaler A and B service implementations” on page 501.

You also need to import some business objects and interfaces common to scenarios used throughout this IBM Redbook, stored in the project ITSOLibrary. The ITSOLibrary, WholesalerA, and WholesalerB resources are packaged in the additional material supplied with this book. To import these resources:

1. In the Business Integration perspective, right-click the **Business Integration** view and click **Import**.
2. From the Import wizard that opens, complete these steps:
   a. Select **Project Interchange**. Click **Next**.
   b. Click **Browse** and navigate to the file ITSOLibrary.zip. This file is supplied with the additional material of this book in the \ParallelWorkflow directory.
   c. Select the **ITSOLibrary** project and click **Finish**.
3. Repeat this process to import the WholesalerA.zip and WholesalerB.zip files. These files are packaged in the additional material supplied with this book in the \Wholesaler\WholesalerA and \Wholesaler\WholesalerB directories.

You should now have three projects in your workspace: ITSOLibrary, WholesalerA, and WholesalerB.

Creating the Parallel Workflow process module

Create the Parallel Workflow process module, ITSOPModule:

1. In the Business Integration perspective, right-click the **Business Integration** view and click **New** → **Module**.

2. From the New Module wizard, complete these steps:
   a. In the Module name field, enter `ITSOPModule`.
   b. For the Module Location Directory, select **Use default**.
   c. Click **Finish**.
3. Configure the ITSOPModule required library ITSOLibrary:
   a. Right-click **ITSOPModule** and click **Open Dependency Editor**.
   b. In the ITSOPModule Dependencies window, click **Add**.
   c. From the Library Selection window that opens, select the **ITSOLibrary** library and click **OK**.
4. Press Ctrl+S to save the module.

The workspace set up is now complete and artifacts can be created.

13.3.2 Creating the business objects

You build the required business objects as explained in the sections that follow. The ITSOPModule requires four business objects:

- WholesalerOrder
- NotApprovedFault
- NotAvailableFault
- OrderCancelledFault

WholesalerOrder business object

The WholesalerOrder business object is used in the PlaceOrderProcess as input and output from the BestWholesaler business rule. This business rule determines the wholesaler with whom to place the order. The WholesalerOrder business object uses the default namespace `http://ITSOPModule`.

The WholesalerOrder business object has the attributes shown in Figure 13-7.

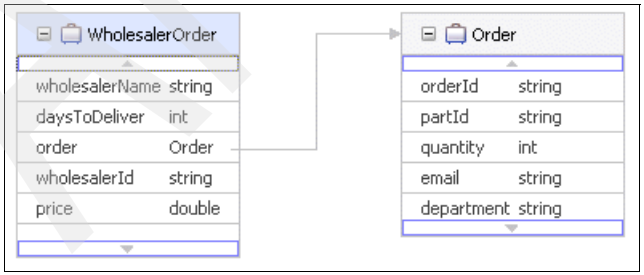


Figure 13-7 Attributes of the WholesalerOrder business object

The order attribute type is Order from the ITSOLibrary.

NotApprovedFault business object

The NotApprovedFault is thrown when the schedule date is not approved by the clerk. The NotApprovedFault business object uses the default namespace `http://ITSOPModule`.

The NotApprovedFault business object has the attributes shown in Figure 13-8.

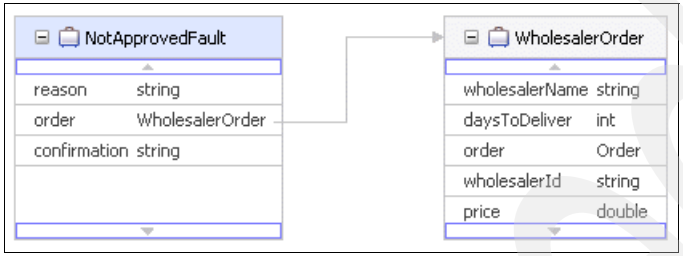


Figure 13-8 Attributes of the NotApprovedFault business object

NotAvailableFault business object

The NotAvailableFault is thrown when the order cannot be fulfilled. The NotAvailableFault business object uses the default namespace `http://ITSOPModule`.

The NotAvailableFault business object has the attributes shown in Figure 13-9.

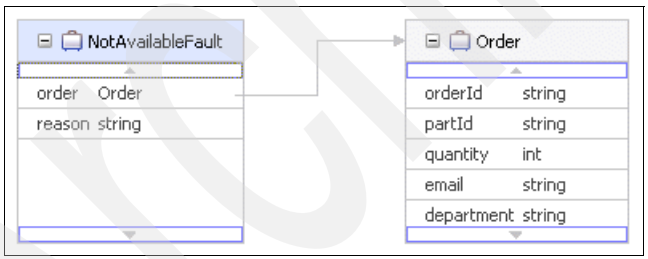


Figure 13-9 Attributes of the NotAvailableFault business object

Archived

OrderCancelledFault business object

The OrderCancelledFault is thrown when the order is cancelled. The OrderCancelledFault business object uses the default namespace `http://ITSOPModule`.

The OrderCancelledFault business object has the attributes shown in Figure 13-10.

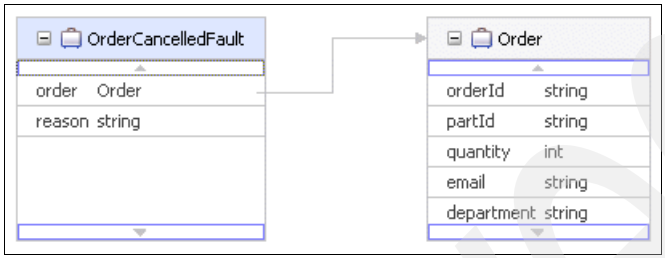


Figure 13-10 Attributes of the OrderCancelledFault business object

13.3.3 Creating the interfaces

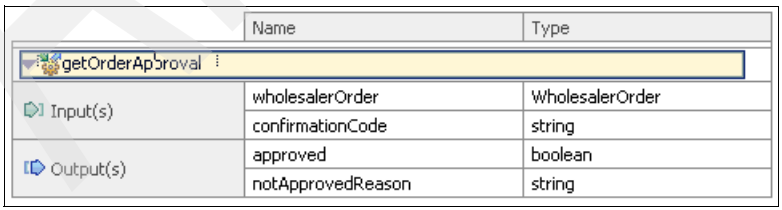
Now you create the required interfaces as explained in the sections that follow. The ITSOPModule requires three interfaces:

- ApproveSchedule
- BestWholesaler
- CancelOrder

ApproveSchedule interface

The ApproveSchedule interface is used as the interface to the Approve Schedule Date human task. The ApproveSchedule interface uses the default namespace `http://ITSOPModule/ApproveSchedule`.

The ApproveSchedule interface has the operations shown in Figure 13-11.



	Name	Type
getOrderApproval		
Input(s)	wholesalerOrder	WholesalerOrder
	confirmationCode	string
Output(s)	approved	boolean
	notApprovedReason	string

Figure 13-11 Operations of the ApproveSchedule interface

Archived

BestWholesaler interface

The BestWholesaler interface is used as the interface to the BestWholesaler business rule. This business rule determines the wholesaler with whom to place the order. The BestWholesaler interface uses the default namespace `http://ITSOPModule/BestWholesaler`.

The BestWholesaler interface has the operations shown in Figure 13-12.

	Name	Type
getBetterWholesaler		
Input(s)	company1	WholesalerOrder
	company2	WholesalerOrder
Output(s)	selectedWholesaler	WholesalerOrder

Figure 13-12 Operations of the BestWholesaler interface

CancelOrder interface

The CancelOrder interface is used as the interface to the Cancel Order event. The CancelOrder interface uses the default namespace `http://ITSOPModule/CancelOrder`.

The CancelOrder interface has the operations shown in Figure 13-13.

	Name	Type
cancelOrder		
Input(s)	orderToCancel	Order

Figure 13-13 Operations of the CancelOrder interface

Note: Unlike the other interfaces, this interface uses a one-way operation.

13.3.4 Building the PlaceOrderProcess Parallel Workflow process

Next you create and build the PlaceOrderProcess Parallel Workflow process.

Creating the Parallel Workflow business process component

To create the Parallel Workflow process component PlaceOrderProcess:

1. Right-click **ITSOPModule** and click **New** → **Business Process**.
2. In the New Business Process wizard, in the Process name field, enter `PlaceOrderProcess`. Click **Next**.

3. In the Select an Interface window, complete these actions:
   a. Click the **Select an existing Interface** radio button and then click **Browse**.
   b. In the Interface Selection window, select the **OrderEntry** interface, and click **OK**.
   c. Back in the Select an Interface window, select the **createOrder** operation, and click **Next**.
4. On the Advanced window, deselect the **Disable WebSphere Process Server BPEL Extensions**, and click **Finish**.

   The new business process PlaceOrderProcess opens in the Business Process editor and looks like the example in Figure 13-14.

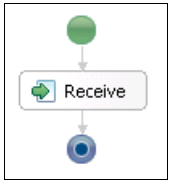


Figure 13-14 The new business process: PlaceOrderProcess

5. To change the properties of PlaceOrderProcess, complete these steps:
   a. Click the **Properties** page.
   b. Click the **Details** tab.
   c. Select the **Process is long-running** check box.
   d. Ensure that the **Automatically delete the process after completion** check box is selected.
6. Press Ctrl+S to save the process.

Adding reference partners to the PlaceOrderProcess

You must add the reference partners to the PlaceOrderProcess Parallel Workflow process. The reference partners are added to the process to enable the process to invoke:

- The BestWholesaler business rule

  This business rule determines the wholesaler with who to place the order, based on the number of days to delivery. For step-by-step instructions on how to implement the BestWholesaler business rule, see “Creating the BestWholesaler business rule” on page 435.

- The ApproveScheduleDate human task

  This human task enables the clerk to approve or decline the schedule date. You can find step-by-step instructions on how to implement the

Archived

ApproveScheduleDate human task in “Generating implementation for the Approve Schedule Date human task” on page 478.

- The Web services exposed by Wholesaler A

  The Wholesaler A Web service is invoked using the selector-business rule pattern. To implement the selector-business rule pattern, 13.3.5, “Building the ITSOPModule assembly diagram” on page 466.

- The Web services exposed by Wholesaler B

  The Wholesaler B Web service is invoked using Web service imports, as explained in “Adding the Wholesaler A and B Web service imports” on page 466.

The reference partners are wired to the specific components in “Wiring the PlaceOrderProcess reference partners” on page 481.

To add the reference partners:

1. Drag the following interfaces from the ITSOLibrary module to the Business Process Editor canvas to add reference partners. Drag each interface twice because each is used for both Wholesaler A and Wholesaler B:
   - WhCnclOrdWeb_I
   - WhDelReqWeb_I
   - WhPlaceOrdWeb_I
2. Drag the following interfaces from the ITSOPModule module to the Business Process Editor canvas once to add reference partners:
   - ApproveSchedule
   - BestWholesaler
3. Rename the reference partners as indicated in Table 13-3.

Table 13-3 PlaceOrderProcess reference partner renaming

Rename from	Rename to
WhCnclOrdWeb_I	CompanyACancelOrderPartner
WhCnclOrdWeb_I	CompanyBCancelOrderPartner
WhDelReqWeb_I	CompanyADeliveryDaysPartner
WhDelReqWeb_I	CompanyBDeliveryDaysPartner
WhPlaceOrdWeb_I	CompanyAOrderPartner
WhPlaceOrdWeb_I	CompanyBOrderPartner
ApproveSchedule	ApproveSchedulePartner
BestWholesaler	BestWholesalerPartner

After you rename the reference partners, the Business Process Editor tray should look like the example in Figure 13-15.

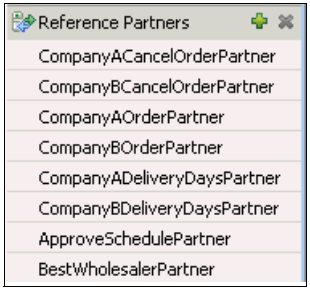


Figure 13-15 The PlaceOrderProcess reference partners tray

4. Press Ctrl+S to save the process.

Adding variables to the PlaceOrderProcess

Add the variables shown in Table 13-4 to the PlaceOrderProcess Parallel Workflow process.

Table 13-4 PlaceOrderProcess variables

Variable name	Data type	Additional information
NotAvailableFault	NotAvailableFault	Thrown when the order cannot be fulfilled.
NotApprovedFault	NotApprovedFault	Thrown when the schedule date is not approved.
CancelledOrderFault	OrderCancelledFault	Thrown when the order is cancelled.

After you add the variables, the Business Process Editor variables tray should look like the example in Figure 13-16.

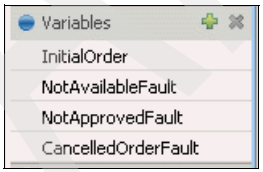


Figure 13-16 The PlaceOrderProcess variables tray

Press Ctrl+S to save the process.

Archived

Adding the CancelOrderClient interface partner to the PlaceOrderProcess

The CancelOrderClient interface partner is used to enable the process to receive the incoming cancel order event at any time while the process is running. Add the CancelOrderClient interface partner to the PlaceOrderProcess:

1. In the Business Process Editor tray, click the **plus sign** ( ) next to the Interface Partners heading.
2. Click the **Properties** page.
3. Click the **Description** tab. In the Name field, type `CancelOrderClient`.
4. Click the **Details** tab. Click **Browse**.
5. In the Interface Selection window, select the **CancelOrder** interface, and click **OK**.

   After you add the interface partner, the Business Process Editor interface partner tray should look like the example in Figure 13-17.

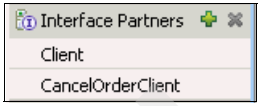


Figure 13-17 The PlaceOrderProcess interface partner tray

6. Press Ctrl+S to save the process.

Adding the OrderIdCorrelationSet correlation set to the PlaceOrderProcess

The OrderIdCorrelationSet is used to identify a specific process instance based on the orderId when an cancel event is sent to the process. Add the OrderIdCorrelationSet correlation set to the PlaceOrderProcess:

1. In the Business Process Editor tray, click the **plus sign** ( ) next to the Correlation Sets heading.
2. Click the **Properties** page.
3. Click the **Description** tab. In the Name field, enter `OrderIdCorrelationSet`.
4. Click the **Details** tab. Click **Add**.
5. In the Select a Property window, click **New**.
6. In the Create Property window, complete the following steps:
   a. In the Name field, type `orderId`.
   b. Click **Browse**. Select **string** data type and click **OK**.
   c. Click **New**.

Archived

i. In the Interface field, select the **OrderEntry** interface.
ii. In the Operation field, select the **createOrder** operation.
iii. Click the **Direction Input** radio button.
iv. In the Part field, select **createOrderParameters** → **initialOrder** → **orderId**.
v. Click **OK**.

d. Click **New**.

i. In the Interface field, select the **CancelOrder** interface.
ii. In the Operation field, select the **cancelOrder** operation.
iii. Click the **Direction Input** radio button.
iv. In the Part field, select **cancelOrderParameters** → **orderToCancel** → **orderId**.
v. Click **OK**.

e. Click **OK**.

Figure 13-18 shows how the Business Process Editor correlation sets tray looks after adding the correlation process.

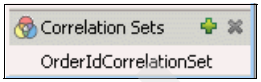


Figure 13-18 The PlaceOrderProcess correlation sets tray

Next, add the OrderIdCorrelationSet correlation set to the Receive activity:

1. Click the **Receive** activity.
2. Click the **Properties** page.
3. Click the **Correlation** tab.
4. Click **Add**.
5. For Initiation, select **Yes**.
6. For Correlation Set, select **OrderIdCorrelationSet**.

Finally, press Ctrl+S to save the process.

Adding activities to the PlaceOrderProcess

This section explains how to add activities to the PlaceOrderProcess Parallel Workflow process. Figure 13-19 shows how the completed PlaceOrderProcess Parallel Workflow process looks.

Archived

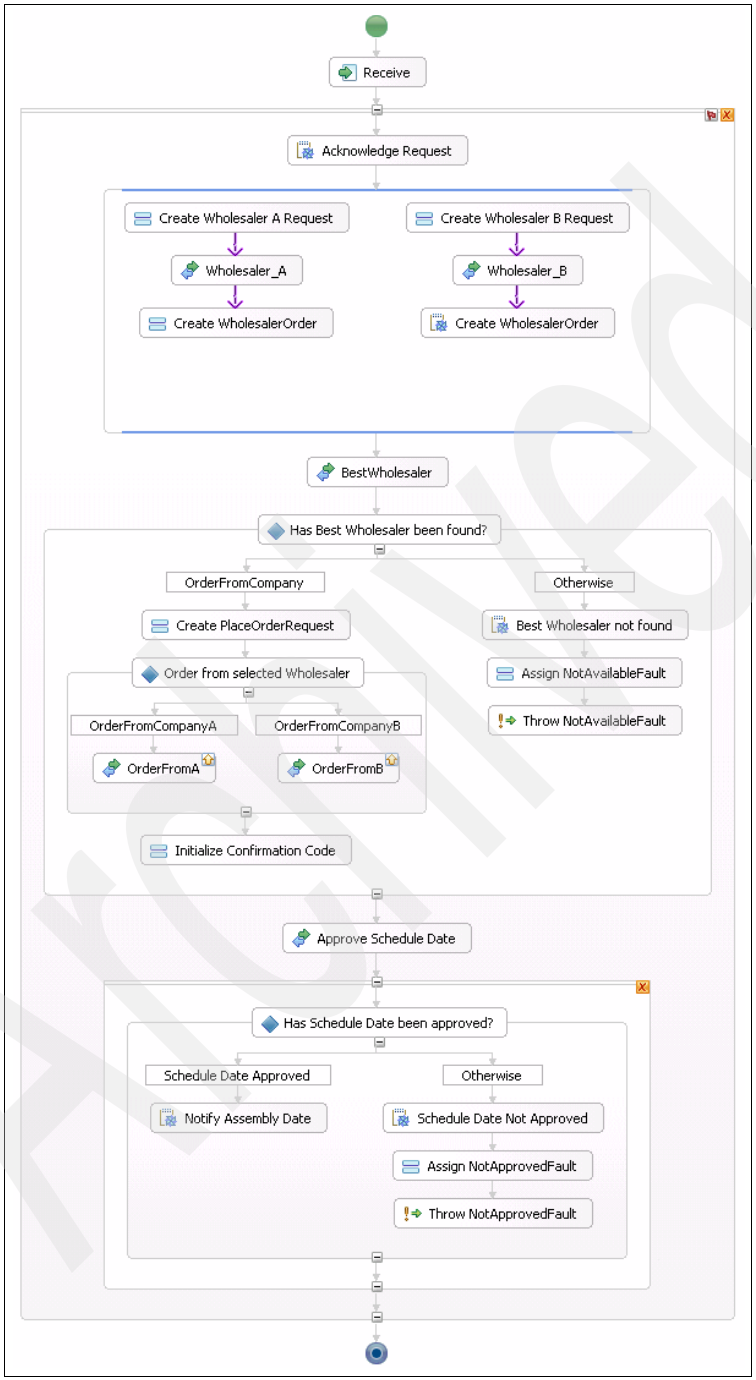


Figure 13-19 PlaceOrderProcess Parallel Workflow process overview

Figure 13-20 shows the PlaceOrderProcess Parallel Workflow process with the Schedule Date Approval scope collapsed.

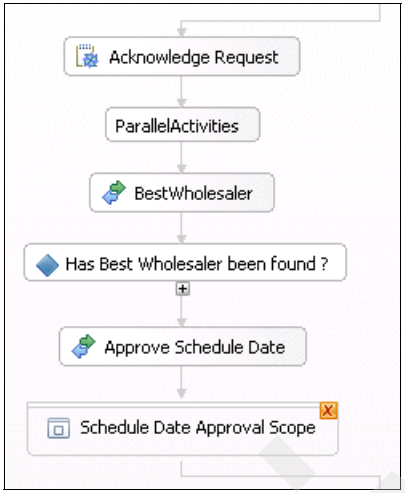


Figure 13-20 PlaceOrderProcess with the Schedule Date Approval Scope collapsed

Figure 13-21 shows the PlaceOrderProcess Parallel Workflow process with the Place Order Process Scope collapsed.

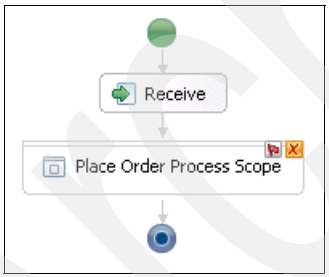


Figure 13-21 PlaceOrderProcess with the Place Order Process Scope collapsed

Archived

The PlaceOrderProcess Parallel Workflow main process contains the activities that are listed in Table 13-5.

Table 13-5 PlaceOrderProcess Parallel Workflow process activities

Activity name	Activity type	Additional information
Receive	Receive	This is the only receive activity in the process. It is the starting activity, and it receives the process input.
Place Order Process Scope	Scope	This is the scope that surrounds the entire process except the receive activity.
Acknowledge Request	Visual snippet	This activity prints an acknowledgement to the log. In the real world, we might send an e-mail to acknowledge the request.
ParallelActivities	Parallel Activities	This is the Parallel Activities activity which ensures that the Wholesaler A and Wholesaler B getDeliveryDays Web services are invoked concurrently.
Create Wholesaler A Request	Assign	This activity assigns the input to the Wholesaler B getDeliveryDays Web service.
Wholesaler_A	Invoke	This activity invokes the Wholesaler A getDeliveryDaysWeb service. This invoke activity is wired to the Wholesaler A getDeliveryDays Web service, in the ITSOPModule assembly diagram, using the selector-business rule pattern which enabled us to replace Wholesaler A at run time without changing the process.
Create WholesalerOrder	Assign	This activity assigns the Wholesaler A input to the BestWholesaler business rule.
Create Wholesaler B Request	Assign	This activity assigns the input to the Wholesaler B getDeliveryDays Web service.
Wholesaler_B	Invoke	This activity invokes the Wholesaler B getDeliveryDays Web service.
Create WholesalerOrder	Visual snippet	This activity assigns the Wholesaler B input to the BestWholesaler business rule.
BestWholesaler	Invoke	This activity invokes the BestWholesaler business rule. The BestWholesaler invoke activity will be wired to the business rule in the ITSOPModule assembly diagram.
Has Best Wholesaler been found?	Choice	This activity determines if the best wholesaler has been found.
Best Wholesaler not found	Java Snippet	This activity prints to the log that the best wholesaler has not been found by the BestWholesaler business rule.

Activity name	Activity type	Additional information
Assign NotAvailableFault	Assign	This activity assigns the best wholesaler not found fault.
Throw NotAvailableFault	Throw	This activity throws the best wholesaler not found fault.
Create PlaceOrderRequest	Assign	This activity assigns the input to the Wholesaler A or Wholesaler B placeOrder Web service.
Order from selected Wholesaler	Choice	This activity determines which Wholesaler placeOrder Web service to call based on the output from the BestWholesaler business rule.
OrderFromA	Invoke	This activity invokes the Wholesaler A placeOrder Web service. This invoke activity will be wired to the Wholesaler A placeOrder Web service, in the ITSOPModule assembly diagram, using the selector-business rule pattern which enabled us to replace Wholesaler A at run time without changing the process.
OrderFromB	Invoke	This activity invokes the Wholesaler B placeOrder Web service.
Initialize Confirmation Code	Assign	This activity assigns the ConfirmationCode variable.
Approve Schedule Date	Invoke	This activity invokes the ApproveSchedule human task. The Approve Schedule Date Invoke activity is wired to the human task in the ITSOPModule assembly diagram.
Schedule Date Approval Scope	Scope	This activity is the scope that surrounds the Has Schedule Date been approved choice.
Has Schedule Date been approved?	Choice	This activity determines if the schedule date has been approved.
Notify Assembly Date	Visual snippet	This activity prints to the log that the assembly date has been approved.
Schedule Date Not Approved	Java snippet	This activity prints to the log that the assembly date has not been approved.
Assign NotApprovedFault	Assign	This activity assigns the schedule date is not approved fault.
Throw NotApprovedFault	Throw	This activity throws the schedule date is not approved fault.

Archived

Adding the Place Order Process Scope

Add the Place Order Process Scope to the PlaceOrderProcess.

1. Add a scope to the PlaceOrderProcess:
   a. On the Process Editor palette, click the **Scope** icon ( ).
   b. Click the arrow between the Receive activity and the process end node. Figure 13-22 shows how the process looks now.

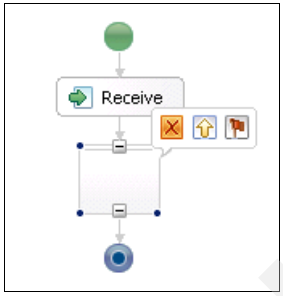


Figure 13-22 PlaceOrderProcess with the newly added scope

   c. Press Ctrl+S to save the process.

2. To change the properties of the newly created scope:
   a. Click the **Scope**.
   b. Click the **Properties** page.
   c. Click the **Description** subtab.
   d. In the Display Name field, type `Place Order Process Scope` (Figure 13-23).

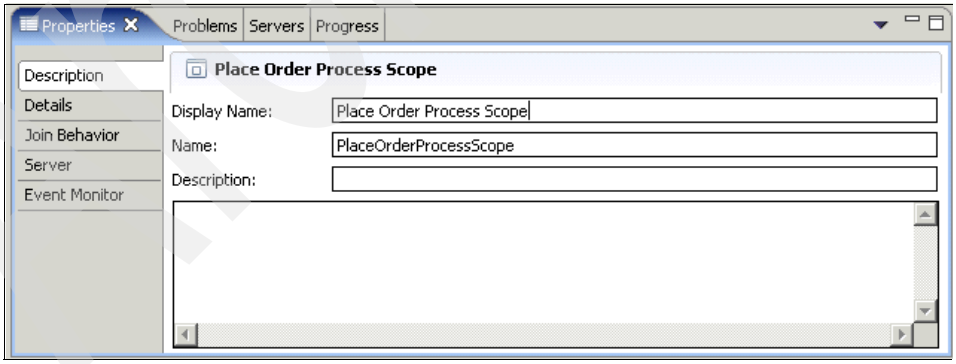


Figure 13-23 The Place Order Process Scope Properties page: Description tab

Archived

Adding the Place Order Process Scope event handler

Add an event handler to the Place Order Process Scope activity which enables the scope to handle incoming events such as cancel order at any time while the process is running. The Place Order Process Scope event handler contains the activities listed in Table 13-6.

Table 13-6 The Place Order Process Scope event handler activities

Activity name	Activity type	Additional information
Cancel Order Event received	Java snippet	Prints to the log that we have an event
Assign CancelledOrderFault	Assign	Assigns the order from the cancel order event to the CancelledOrderFault
Throw OrderCancelledFault	Throw	Throws the order cancelled fault

1. Right-click the **Place Order Process Scope** activity and click **Add Event Handler**.
2. Click **OnEvent**.
3. Click the **Properties** page.
4. Click the **Details** tab (Figure 13-24).
   a. Click **Browse**.
   b. In the Select a Partner window, select the **CancelOrderClient** partner, and click **OK**.
   c. Back on the Details tab, for Operation, select **cancelOrder**.
   d. In the Variable Name field, type `OrderToCancel`.

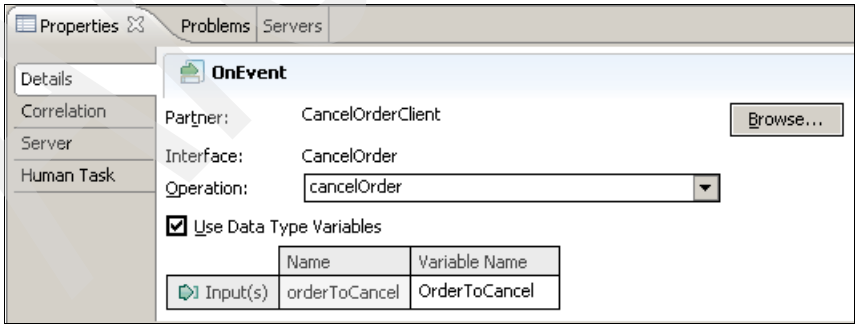


Figure 13-24 The onEvent Properties page: Details tab

Archived

5. Click the **Correlation** tab (Figure 13-25).
   a. Click **Add**.
   b. For Direction, select **Receive**.
   c. For Initiation, select **No**.
   d. For Correlation Set, select **OrderIdCorrelationSet**.

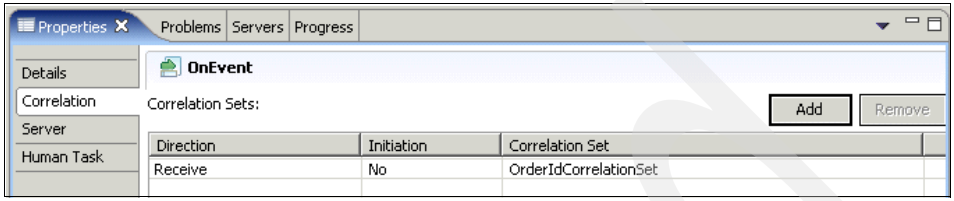


Figure 13-25 The onEvent Properties page: Correlation tab

6. Add a snippet to the Place Order Process Scope event handler:
   a. On the Process Editor palette, click the **Snippet** icon ( ).
   b. Under OnEvent, click the Place Order Process Scope **Event Handler** to add the new snippet.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Cancel Order Event received`.
   e. Click the **Details** tab (Figure 13-26).
      i. Click the **Java** radio button.
      ii. In the Question window, click **Yes**.
      iii. In the text field, enter the code shown in Example 13-1.

Example 13-1 Cancel Order Event received snippet Java code

```
System.out.println("We got an event");
```

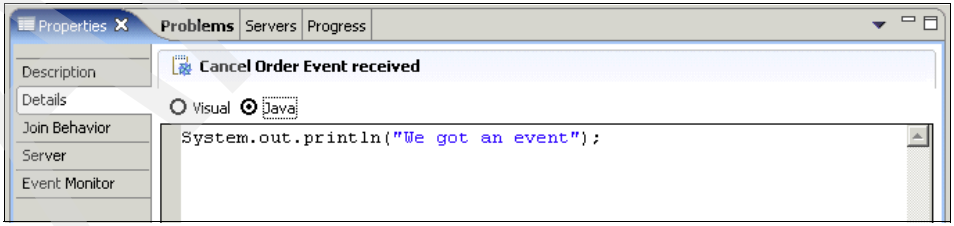


Figure 13-26 Cancel Order Event received Java Snippet Properties page: Details tab

Archived

7. Add an assign activity to the Place Order Process Scope event handler:
   a. On the Process Editor palette, click the **Assign** icon ( ).
   b. Under the Cancel Order Event received activity, click the Place Order Process Scope **Event Handler** to add the new assign.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Assign CancelledOrderFault`.
   e. Click the **Details** tab. Update the Assign CancelledOrderFault assign activity details as shown in Table 13-7.

Table 13-7 The Assign CancelledOrderFault Assign activity

Order	From		To	
1	Variable	OrderToCancel I	Variable	CancelledOrderFault → order

8. Add a Throw activity to the Place Order Process Scope Event Handler:
   a. On the Process Editor palette, click the **Throw** icon ( ).
   b. Under the Assign CancelledOrderFault activity, click the Place Order Process Scope **Event Handler** to add the new throw.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Throw OrderCancelledFault`.
   e. Click the **Details** tab (Figure 13-27).
      i. For Fault Type, select the **User-defined** radio button.
      ii. In the Fault Name field, type `OrderCancelledFault`.
      iii. Click **Browse**.
      iv. In the Select Fault Variable window, select the **CancelledOrderFault** variable, and click **OK**.

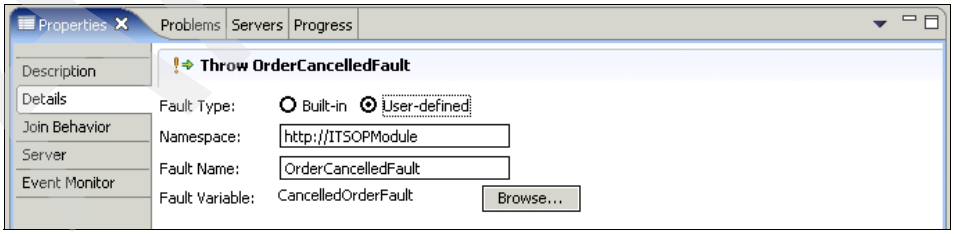


Figure 13-27 The Throw OrderCancelledFault Properties page: Details tab

Figure 13-28 shows the completed Place Order Process Scope Event Handler.

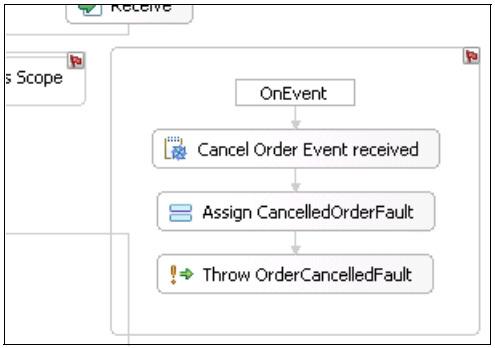


Figure 13-28 The Place Order Process Scope Event Handler

9. Press Ctrl+S to save the process.

Adding the Acknowledge Request visual snippet

In this section, you add the Acknowledge Request visual snippet to the PlaceOrderProcess. The Acknowledge Request activity description is shown in Table 13-8.

Table 13-8 The Acknowledge Request activity

Activity name	Activity type	Additional information
Acknowledge Request	Visual snippet	Prints an acknowledgement to the log. In the real world we might send an e-mail to acknowledge the order request.

1. Click the **Snippet** icon ( ) on the Process Editor palette, and then click **Place Order Process Scope** to add the new snippet.

   Figure 13-29 shows the process after completing this step.

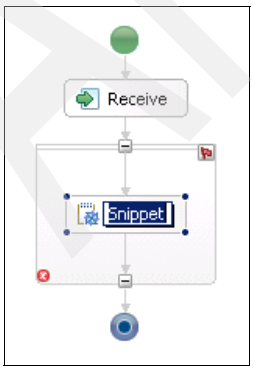


Figure 13-29 PlaceOrderProcess with the newly created snippet

2. Click the **Properties** page.
3. Click the **Description** tab. For Display Name, type `Acknowledge Request`.
4. Click the **Details** tab. Ensure that the **Visual** radio button is selected.
5. Build the Acknowledge Request visual snippet as shown in Figure 13-30.

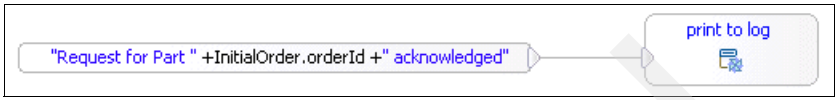


Figure 13-30 The Acknowledge Request Properties page: Details tab

6. Press Ctrl+S to save the process.

Adding the Parallel Activities flow

Next you add the Parallel Activities activity and the activities it contains to the PlaceOrderProcess (Figure 13-31) as explained in the sections that follow.

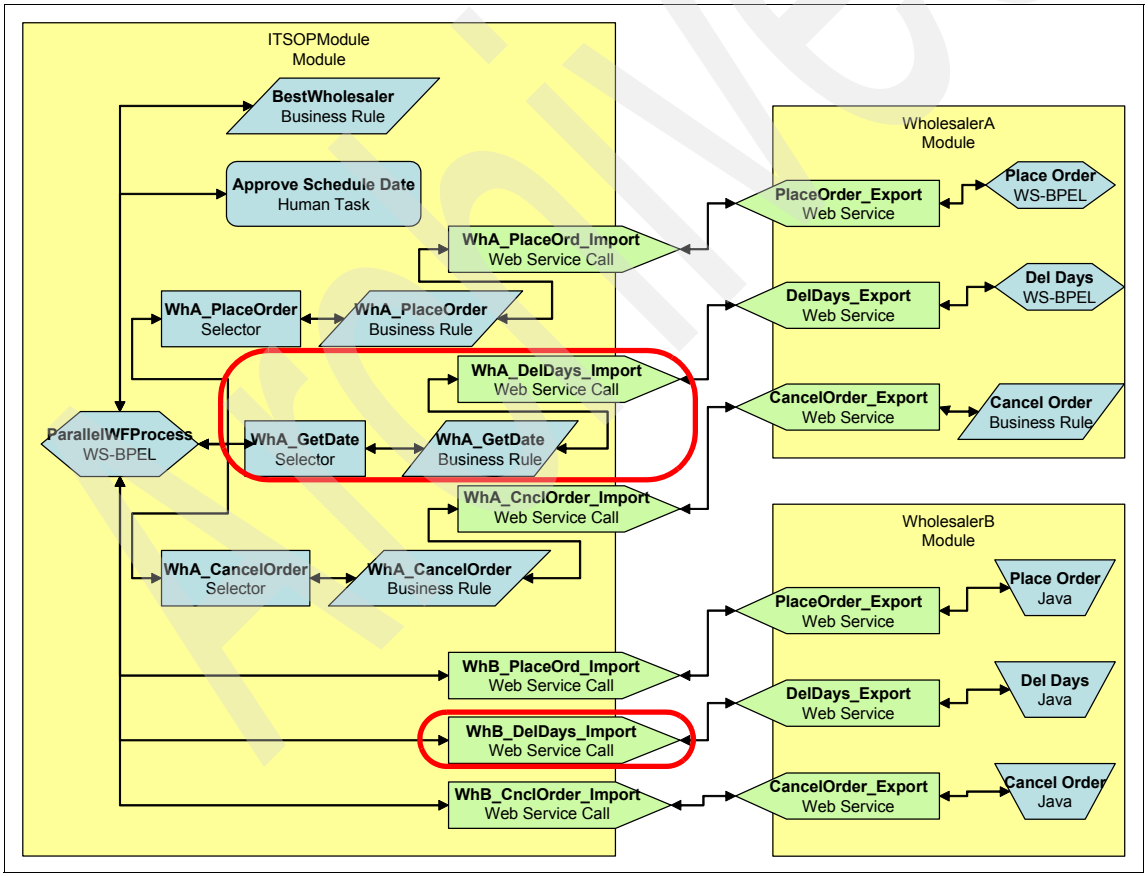


Figure 13-31 Overview of the scenario showing the interlinked components and modules

Archived

The Parallel Activities activity contains the activities listed in Table 13-9.

Table 13-9 The Parallel Activities activity and the activities it contains

Activity name	Activity type	Additional information
ParallelActivities	Parallel Activities	This activity ensures that the Wholesaler A and Wholesaler B getDeliveryDays Web services are invoked concurrently.
Create Wholesaler A Request	Assign	This activity assigns the input to the Wholesaler B getDeliveryDays Web service.
Wholesaler_A	Invoke	This activity invokes the Wholesaler A getDeliveryDaysWeb service. It is wired to the Wholesaler A getDeliveryDays Web service, in the ITSOPModule assembly diagram, using the selector-business rule pattern, which enabled us to replace Wholesaler A at run time without changing the process.
Create Wholesaler B Request	Assign	This activity assigns the input to the Wholesaler B getDeliveryDays Web service.
Wholesaler_B	Invoke	Invokes the Wholesaler B getDeliveryDays Web service.
Create WholesalerOrder	Assign	This activity assigns the Wholesaler A input to the BestWholesaler business rule. It is added after the BestWholesaler invoke activity is added.
Create WholesalerOrder	Visual snippet	This activity assigns the Wholesaler B input to the BestWholesaler business rule. It is added after the BestWholesaler invoke activity has been added.

Archived

Adding the Parallel Activities activity

Add the Parallel Activities activity to the PlaceOrderProcess.

1. On the Process Editor palette, click the **Parallel Activities** icon ( ).
2. Click the **arrow** after the Acknowledge Request snippet inside the Place Order Process Scope. Figure 13-32 shows how the process looks now.

Figure 13-32 PlaceOrderProcess with the parallel activities flow

3. Press Ctrl+S to save the process.

Adding the Parallel Activities flow invoke activities

This section describes the details of the invoke activities that will be added to the Parallel Activities flow. The invoke activities call the getDeliveryDays Web services, which Wholesaler A and Wholesaler B exposes.

1. Add the Wholesaler_A invoke activity to the Parallel Activities flow. On the Process Editor palette, click the **Invoke** icon ( ).
2. Click the **ParallelActivities** flow to add the Invoke activity.
3. Click the **Properties** page.
4. Click the **Description** tab. In the Display Name field, type `Wholesaler_A`.

Archived

5. Click the **Details** tab.
   a. Click **Browse**.
   b. In the Select a Partner window, select the **CompanyADeliveryDaysPartner** partner and click **OK** (Figure 13-33).
   c. On the Details tab, select the **getDeliveryDays operation**.
   d. In the Input(s) row, click **...**.
      i. In the Select Variable for getDeliveryDaysWholesalerRequest window, click **New**.
      ii. In the New Variable window, in the Variable Name field, type `WholesalerARequest`. Click **OK**.
      iii. Back on the Select Variable for getDeliveryDaysWholesalerRequest window, click **OK**.
   e. In the Output(s) row, click **...**.
      i. In the Select Variable for getDeliveryDaysWholesalerResponse window, click **New**.
      ii. In the New Variable window, in the Variable Name field, type `WholesalerAResponse`. Click **OK**.
      iii. Back in the Select Variable for getDeliveryDaysWholesalerResponse window, click **OK**.

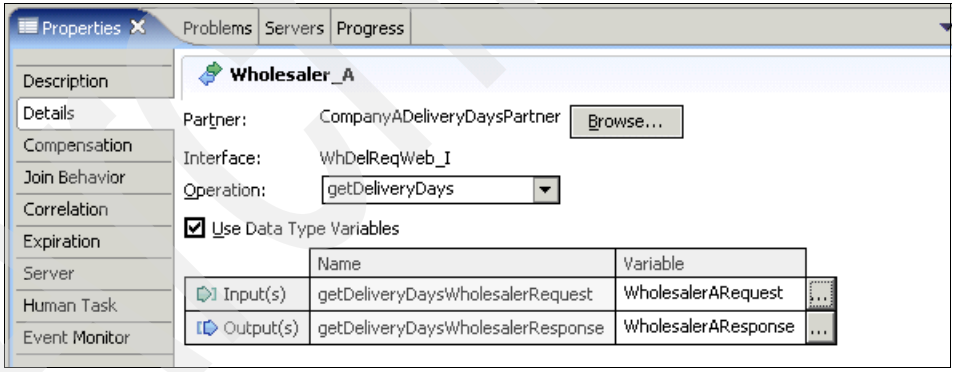


Figure 13-33 The Wholesaler_A Properties page: Details tab

Archived

6. Click the **Server** tab. For Transactional Behavior, select the **Requires Own** radio button (Figure 13-34).

> **Restriction:** To achieve true parallelism in a Business Process, each path must be included in a separate transaction. If the paths are included in the same transaction, serial behavior occurs.

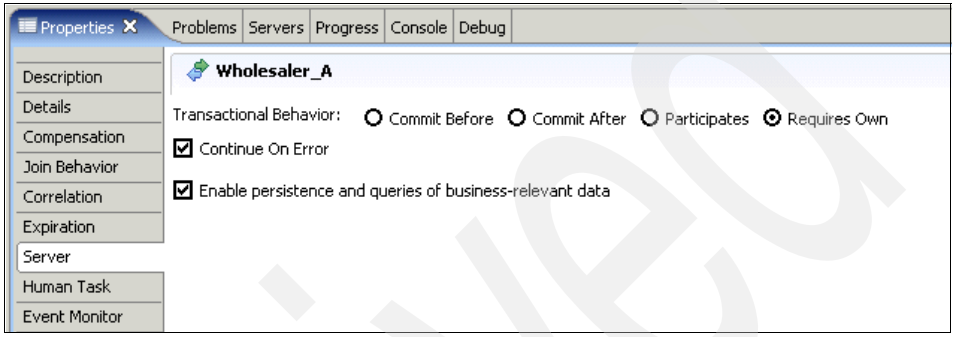


Figure 13-34 The Wholesaler_A Properties page: Server tab

7. Add the Wholesaler_B invoke activity to the Parallel Activities flow:
   a. On the Process Editor palette, click the **Invoke** icon ( ) and click the **ParallelActivities** flow to add the Invoke activity.
   b. Update the Invoke activity properties with the properties in Table 13-10.

Table 13-10 The Wholesaler_B invoke properties

Property name	Value
Description	
Display Name	Wholesaler_B
Details	
Partner	CompanyBDeliveryDaysPartner
Interface	WhDelReqWeb_I
Operation	getDeliveryDays
Input: getDeliveryDaysWholesalerRequest	WholesalerBRequest
Output: getDeliveryDaysWholesalerResponse	WholesalerBResponse

Archived

Property name	Value
Server	
Transactional Behavior	Requires Own

8. Press Ctrl+S to save the process.

Adding the input assign activities for the getDeliveryDays invokes

This section explains how to add the two input assign activities inside the Parallel Activities flow to assign the input for the two getDeliveryDays invokes Wholesaler_A and Wholesaler_B.

1. Click the **Assign** icon ( ) to add two assign activities from the Business Process Editor palette to the Parallel Activities flow. Rename them to:
   - Create Wholesaler A Request
   - Create Wholesaler B Request
2. Update the assign activities details with the properties in Table 13-11.

Table 13-11 The input assign activities for the getDeliveryDays invokes

Order	From		To	
Create Wholesaler A Request				
1	Variable	InitialOrder → partId	Variable	WholesalerARequest → partNo
2	Variable	InitialOrder → quantity	Variable	WholesalerARequest → quantity
Create Wholesaler B Request				
1	Variable	InitialOrder → partId	Variable	WholesalerBRequest → partNo
2	Variable	InitialOrder → quantity	Variable	WholesalerBRequest → quantity

3. Press Ctrl+S to save the process.

Archived

Adding the BestWholesaler activity

The following section explain how to add the BestWholesaler invoke activity to the PlaceOrderProcess (Figure 13-35).

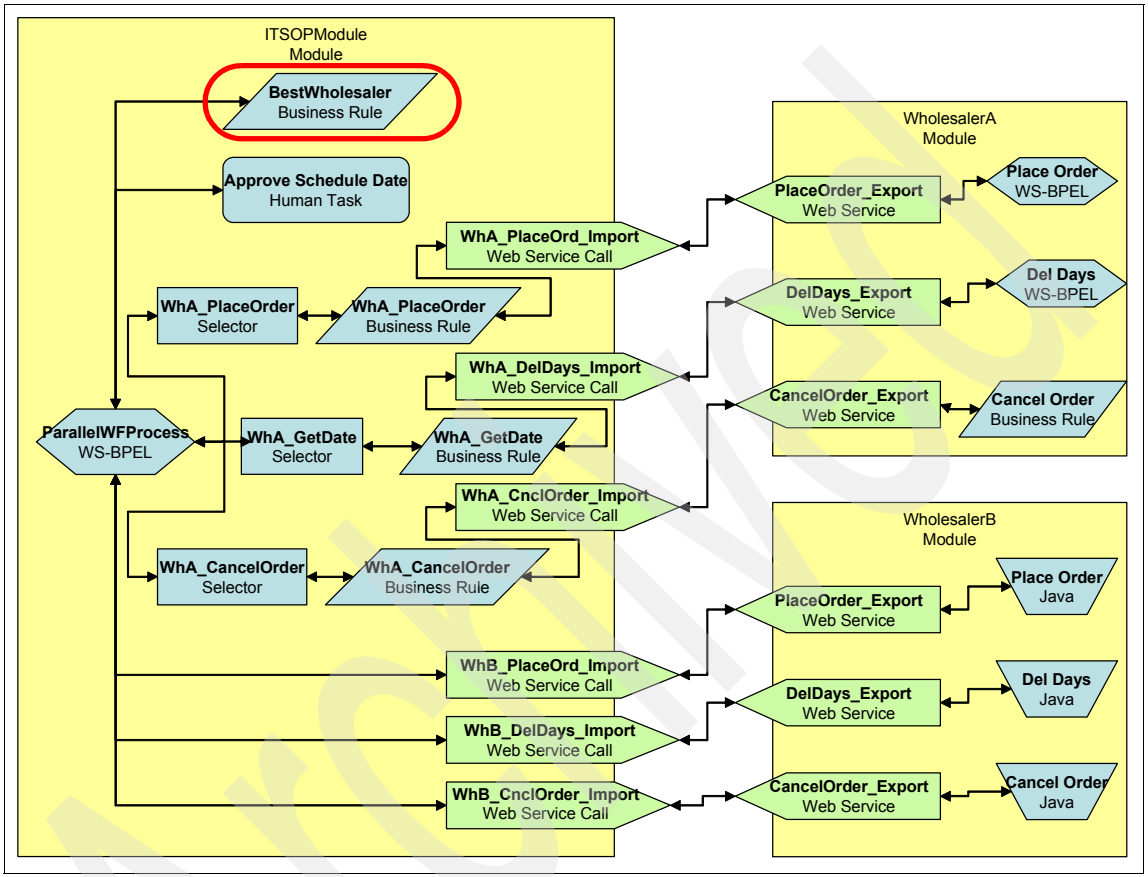


Figure 13-35 Overview of the scenario showing the interlinked components and modules

Table 13-12 describes the BestWholesaler activity.

Table 13-12 The BestWholesaler activity

Activity name	Activity type	Additional information
BestWholesaler	Invoke	This activity invokes the BestWholesaler business rule. It is wired to the business rule in the ITSOPModule assembly diagram.

Adding the BestWholesaler invoke activity

Add the BestWholesaler invoke activity to the Place Order Process Scope.

1. On the Process Editor palette, click the **Invoke** icon ( ), and click the **arrow** after the ParallelActivities flow inside the Place Order Process Scope.
2. Update the Invoke activity properties with the properties in Table 13-13.

Table 13-13 The BestWholesaler invoke properties

Property name	Value
Description	
Display Name	BestWholesaler
Details	
Partner	BestWholesalerPartner
Interface	BestWholesaler
Operation	getBetterWholesaler
Input: company1	WholesalerOrderA
Input: company2	WholesalerOrderB
Output: selectedWholesaler	SelectedWholesaler

3. Press Ctrl+S to save the process.

Adding the input assign for the BestWholesaler invoke activity

This section describes the details of the assign activity which will be added to the Parallel Activities flow to assign the WholesalerOrderA input for the BestWholesaler invoke activity.

1. From the Business Process Editor palette, click the **Assign** icon ( ) to add an assign activity to the Parallel Activities flow. Rename the activity to `Create WholesalerOrder`.

Archived

2. Update the Assign activity details using the properties in Table 13-14.

Table 13-14 The input assign for the BestWholesaler invoke activity

Order	From		To	
Create WholesalerOrder				
1	Variable	WholesalerAResponse → daysToDeliver	Variable	WholesalerOrderA → daysToDeliver
2	Variable	InitialOrder → partId	Variable	WholesalerOrderA → order → partId
3	Variable	InitialOrder → quantity	Variable	WholesalerOrderA → order → quantity
4	Variable	InitialOrder → orderId	Variable	WholesalerOrderA → order → orderId
5	Fixed Value	Wholesaler A	Variable	WholesalerOrderA → wholesalerName
6	Fixed Value	A	Variable	WholesalerOrderA → wholesalerId
7	Variable	WholesalerAResponse → price	Variable	WholesalerOrderA → price

3. Press Ctrl+S to save the process.

Adding the input visual snippet for the BestWholesaler invoke

Add the visual snippet activity to the Parallel Activities flow to assign the WholesalerOrderB input for the BestWholesaler invoke activity.

1. On the Process Editor palette, click the **Snippet** icon ( ).
2. Click the **ParallelActivities** flow to add the new snippet.
3. Click the **Properties** page.
4. Click the **Description** tab. In the Display Name field, type `Create WholesalerOrder`.
5. Click the **Details** tab. Ensure the **Visual** radio button is selected.

Archived

6. Build the Create WholesalerOrder visual snippet as shown in Figure 13-36.

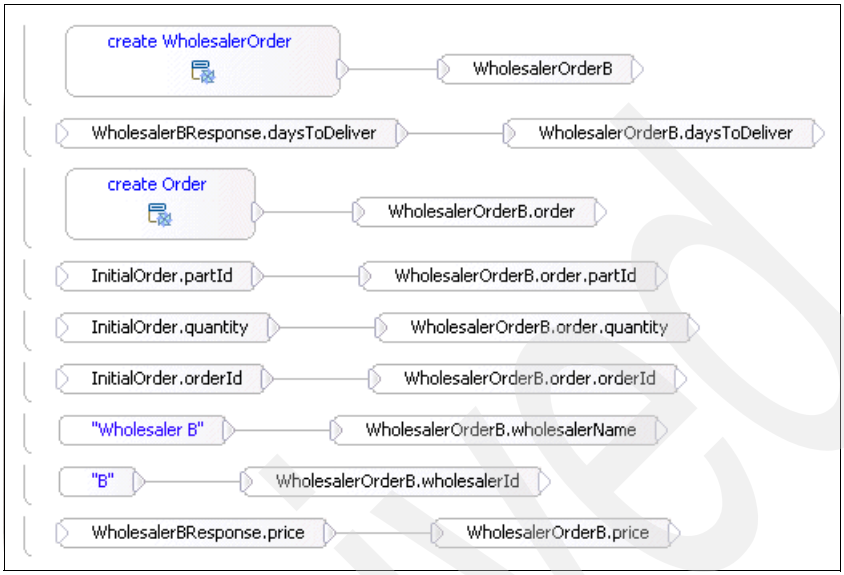


Figure 13-36 The Create WholesalerOrder Properties page: Details tab

7. Press Ctrl+S to save the process.

Adding links between the activities in the Parallel Activities flow

Link the components in the Parallel Activites flow as shown in Figure 13-37.

1. Right-click the **first activity**, select **Add Link**, and then click the **second activity**.

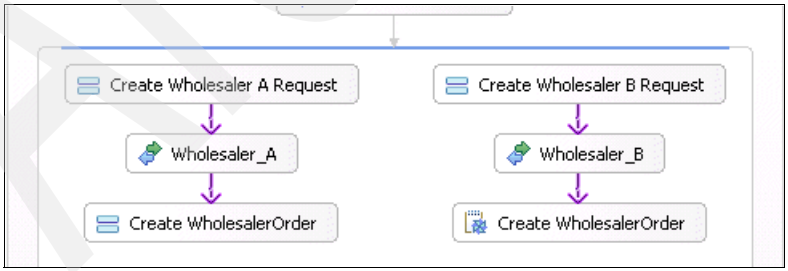


Figure 13-37 The completed Parallel Activities flow

2. Press Ctrl+S to save the process.

Archived

Creating the BestWholesaler business rule

The BestWholesaler business rule determines which wholesaler to place the order with, based on the number of days until delivery. The BestWholesaler business rule is wired to the PlaceOrderProcess BestWholesaler invoke activity in the ITSOPModule assembly diagram.

1. Right-click **ITSOPModule** and click **New** → **Rule Group**.
2. In the New Rule Group wizard, in the Name field, type `BestWholesalerRG`. Click **Next**.
3. In the Select an Interface window, select the **BestWholesaler** interface. Click **Finish**.
4. In the BestWholesalerRG Rule Group window (Figure 13-38), complete these steps:
   a. Click **getBetterWholesaler**.
   b. In the Active Destinations section, click **Enter Destination**.
   c. From the helper menu, click **New Ruleset**.

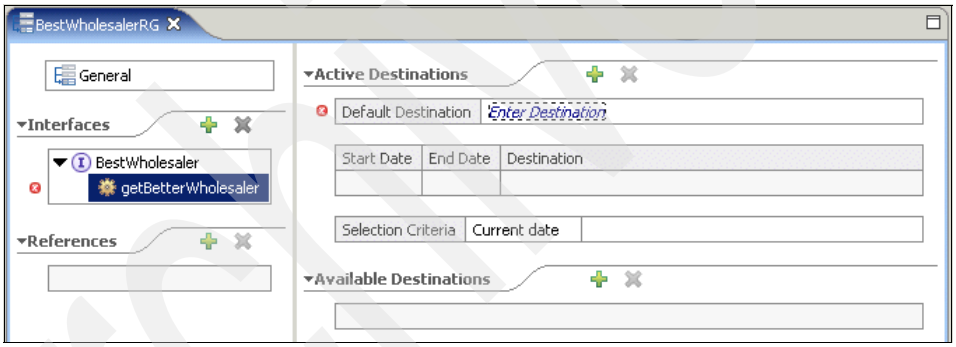


Figure 13-38 BestWholesalerRG Rule Group

5. In the New Rule Set window, in the Name field, type `BestWholesalerRS`. Then click **Finish**.
6. In the BestWholesalerRS Rule Set window (Figure 13-39), complete these steps:
   a. Click the **Add If-Then Rule** icon ( ).
      i. Click **Condition**.
      ii. Click **var1 is greater than var2**.
      iii. Click the first **Select_Variable**, and from the helper menu, select **company1.daysToDeliver**.
      iv. Click the second **Select_Variable?**, and from the helper menu, select **company2.daysToDeliver**.

Archived

v. Click **Action**.

vi. From the helper menu, click **Set var1 to var2**.

vii. Click the first **Select_Variable**, and from the helper menu, select **selectedWholesaler**.

viii. Click the second **Select_Variable?**, and, from the helper menu, select **company2**.

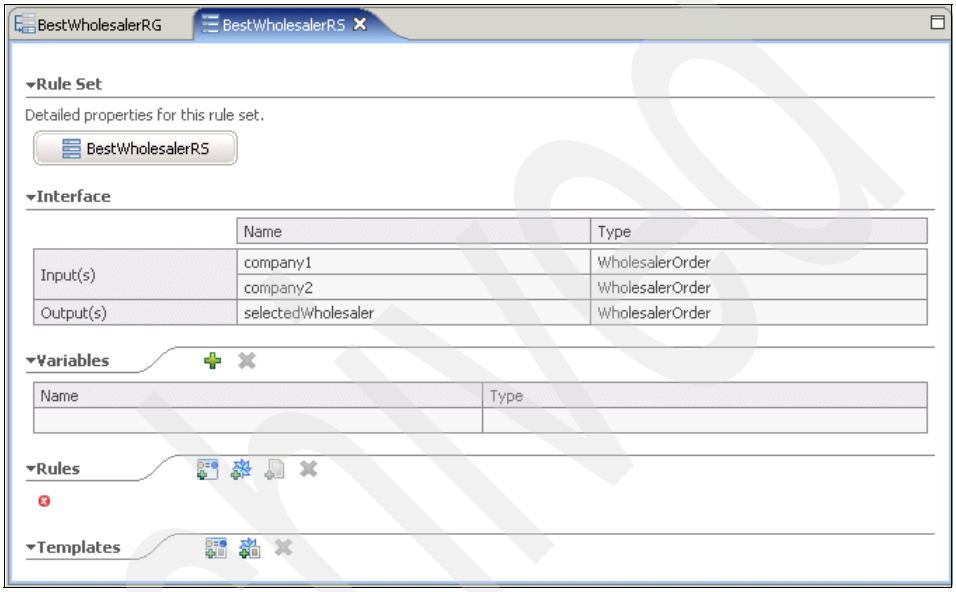


Figure 13-39 BestWholesalerRS Rule Set

b. Click the **Add If-Then Rule** icon .

i. Click **Condition**.

ii. From the helper menu, click **company1.daysToDeliver**.

iii. From the helper menu, click **<=**.

iv. From the helper menu, click **company2.daysToDeliver**.

v. Click **Action**.

vi. From the helper menu, click **Set var1 to var2**.

vii. Click the first **Select_Variable**, and from the helper menu, select **selectedWholesaler**.

viii. Click the second **Select_Variable?**, and from the helper menu, select **company1**.

Archived

7. Press Ctrl+S to save the BestWholesalerRS business rule set. Figure 13-40 shows the completed BestWholesalerRS business rule set.

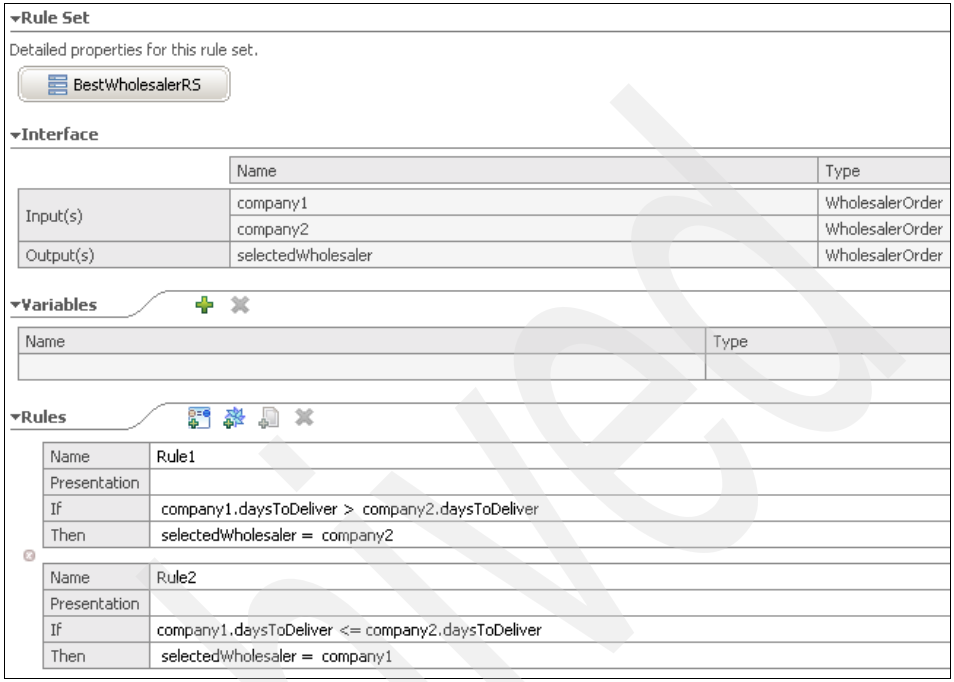


Figure 13-40 The completed BestWholesalerRS business rule set

8. Close the BestWholesalerRS business rule set editor.
9. Switch to the BestWholesalerRG Business Rule Group Editor. The completed BestWholesalerRG business rule group should contain the Available Destination BestWholesalerRS (Figure 13-41).

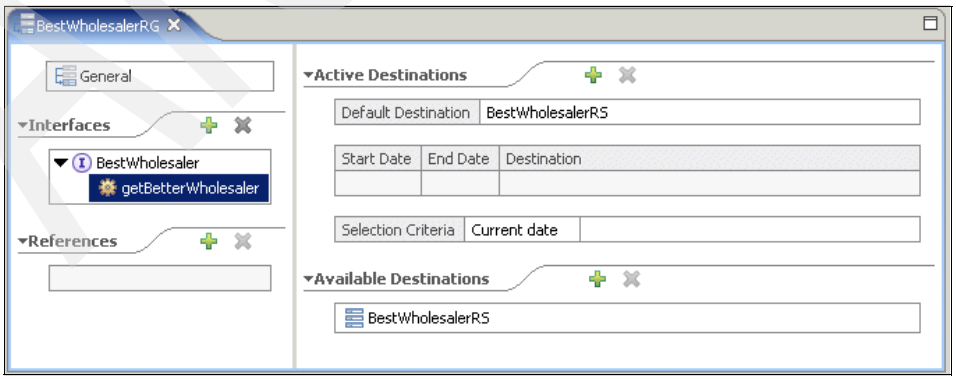


Figure 13-41 The completed BestWholesalerRG business rule group

Archived

10. Press Ctrl+S to save the BestWholesalerRG business rule group.

11. Close the BestWholesalerRG Business Rule Group Editor.

Adding the Has Best Wholesaler been found? choice

The following sections explain how to add the Has Best Wholesaler been found? choice activity and the activities it contains to the PlaceOrderProcess (Figure 13-42).

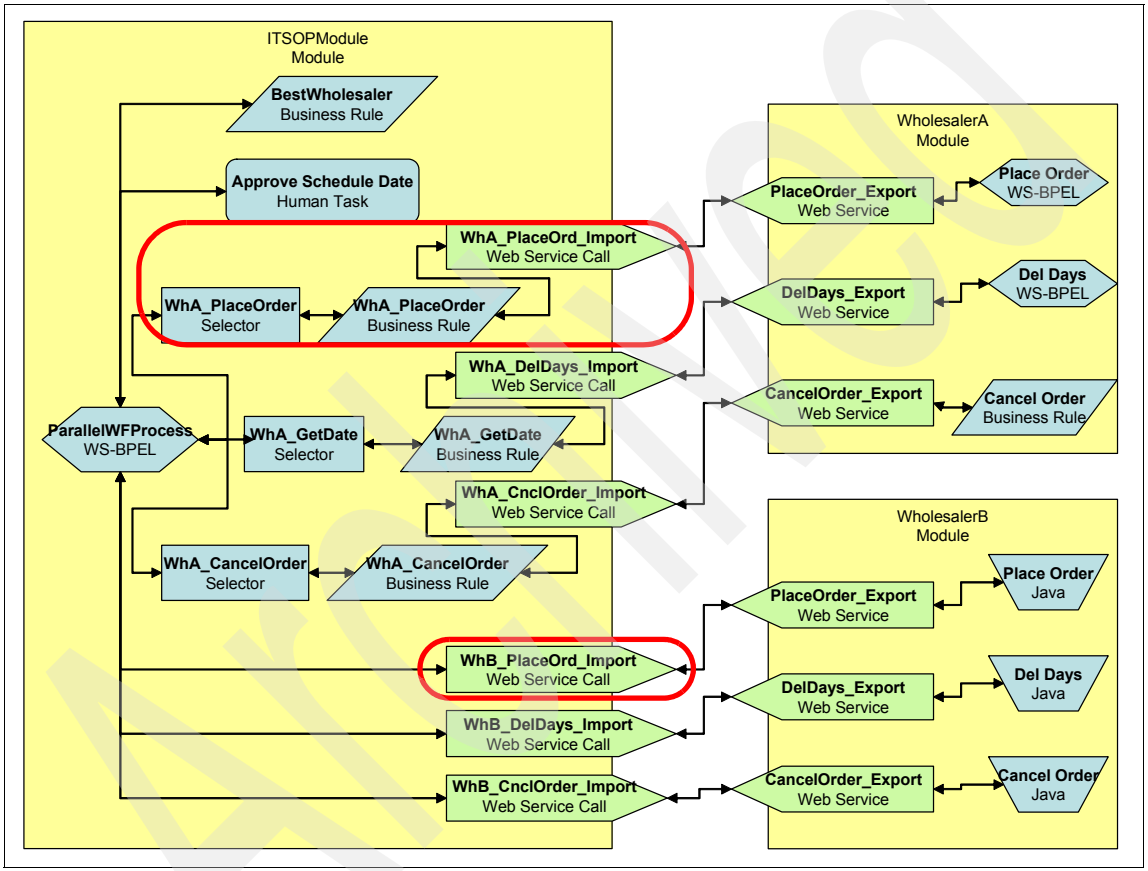


Figure 13-42 Overview of the scenario showing the interlinked components and modules

Archived

The Has Best Wholesaler been found? activity contains the activities listed in Table 13-15.

Table 13-15 The Has Best Wholesaler been found? activity and the activities it contains

Activity name	Activity type	Additional information
Has Best Wholesaler been found?	Choice	This activity determines if the best wholesaler has been found.
Assign NotAvailableFault	Assign	This activity assigns the best wholesaler not found fault.
Throw NotAvailableFault	Throw	This activity throws the best wholesaler not found fault.
Best Wholesaler not found	Java snippet	This activity prints to the log that the best wholesaler has not been found by the BestWholesaler business rule.
Create PlaceOrderRequest	Assign	This activity assigns the input to the Wholesaler A or Wholesaler B placeOrder Web service.
Order from selected Wholesaler	Choice	This activity determines which Wholesaler placeOrder Web service to call based on the output from the BestWholesaler business rule.
OrderFromA	Invoke	This activity invokes the Wholesaler A placeOrder Web service. It is wired to the Wholesaler A placeOrder Web service, in the ITSOPModule assembly diagram, using the selector-business rule pattern which enabled us to replace Wholesaler A at run time without changing the process.
OrderFromB	Invoke	This activity invokes the Wholesaler B placeOrder Web service.
Initialize Confirmation Code	Assign	This activity assigns the ConfirmationCode variable. This activity is added after the ApproveScheduleDate invoke activity is added.

Archived

Adding the Has Best Wholesaler been found? choice activity

Add the Has Best Wholesaler been found? activity to the PlaceOrderProcess.

1. On the Process Editor palette, click the **Choice** icon ( ).
2. Click the **arrow** after the BestWholesaler invoke inside the Place Order Process Scope. Figure 13-43 shows how the process looks now.

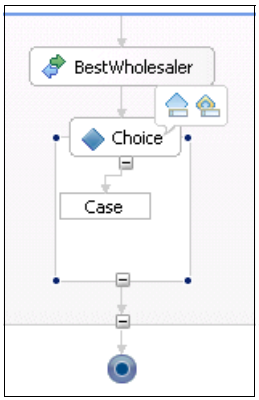


Figure 13-43 PlaceOrderProcess with the Has Best Wholesaler been found? choice

3. Click the **Properties** page.
4. Click the **Description** tab. In the Display Name field, type `Has Best Wholesaler been found?`
5. Click **Case**.
6. Click the **Properties** page.
7. Click the **Description** tab. In the Display Name field, type `OrderFromCompany`.
8. Click the **Details** tab.
   a. In the Expression language field, select **Same as Process (Java)**.
   b. For Expression Type, select the **Visual** radio button.
9. Build the OrderFromCompany visual snippet. In the Visual Snippet Editor, complete these steps:
   a. Click the **true** expression, and from the helper menu, click **SelectedWholesaler**.
   b. From the helper menu, click **!=**.
   c. Type `null`.

   Figure 13-44 shows the completed OrderFromCompany case visual snippet.

Archived

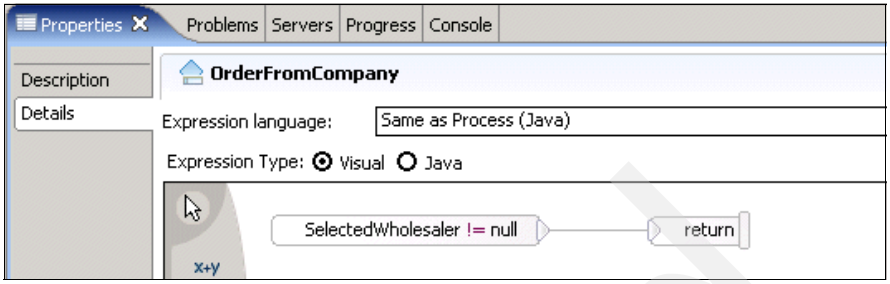


Figure 13-44 The OrderFromCompany case Properties page: Details tab

10. Right-click the **Has Best Wholesaler been found?** choice and click **Add Otherwise**.
11. Press Ctrl+S to save the process.

Adding the Order from selected Wholesaler choice activity

Add the Order from selected Wholesaler choice activity to the Has Best Wholesaler been found? choice.

1. On the Process Editor palette, click the **Choice** icon ( ).
2. Click below the OrderFromCompany case.
3. Click the **Properties** page.
4. Click the **Description** tab. In the Display Name field, type `Order from selected Wholesaler`.

   Figure 13-45 shows the Has Best Wholesaler been found? choice now.

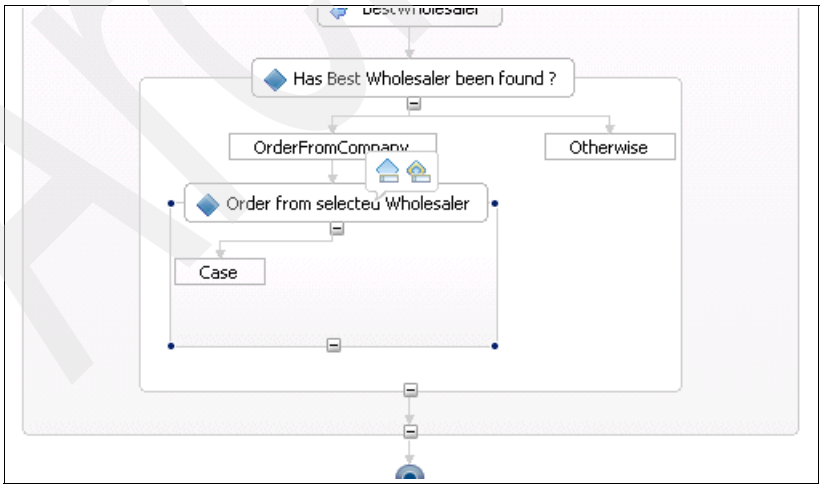


Figure 13-45 The Has Best Wholesaler been found? choice

Archived

5. Click **Case**.
6. Click the **Properties** page.
7. Click the **Description** tab. In the Display Name field, type `OrderFromCompanyA`.
8. Click the **Details** tab.
   a. In the Expression language field, select **Same as Process (Java)**.
   b. For Expression Type, click the **Visual** radio button.
9. Build the OrderFromCompanyA visual snippet. In the Visual Snippet Editor, complete the following steps:
   a. Click the **true** Expression, and from the helper menu, click **SelectedWholesaler.wholesalerName**.
   b. From the helper menu, click the dot (**.**).
   c. From the helper menu, click **equals**.
   d. From the helper menu, click **WholesalerOrderA.wholesalerName**.

   Figure 13-46 shows the completed OrderFromCompanyA case visual snippet.

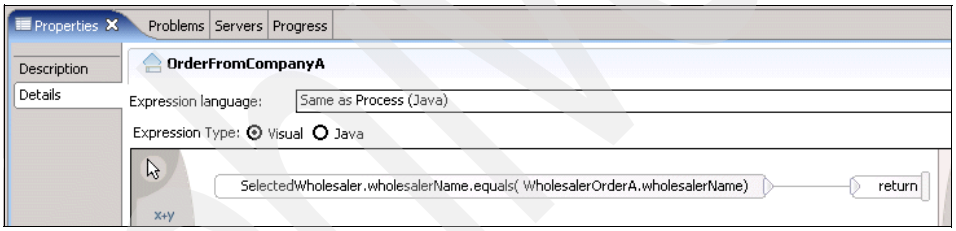


Figure 13-46 The OrderFromCompanyA case Properties page: Details tab

10. Right-click the **Order from selected Wholesaler** choice and click **Add Case**.
11. Click **Case**.
12. Click the **Properties** page.
13. Click the **Description** tab. In the Display Name field, type `OrderFromCompanyB`.
14. Click the **Details** tab.
    a. In the Expression language field, select **Same as Process (Java)**.
    b. Click the Expression Type **Visual** radio button.
15. Build the OrderFromCompanyB visual snippet. In the Visual Snippet Editor, complete the following steps:
    a. Click the **true** Expression, and from the helper menu, click **SelectedWholesaler.wholesalerName**.
    b. From the helper menud, click the dot (**.**).
    c. From the helper menu, click **equals**.

Archived

d. From the helper menu, click WholesalerOrderB.wholesalerName.

Figure 13-47 shows the completed OrderFromCompanyB case visual snippet.

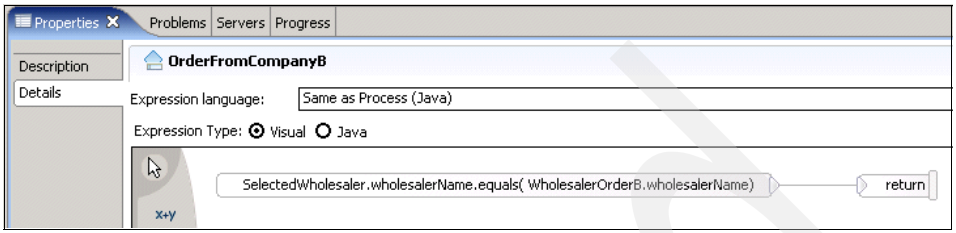


Figure 13-47 The OrderFromCompanyB case Properties page: Details tab

16. Press Ctrl+S to save the process.

Adding the Order from selected Wholesaler invoke activities

Add the Invoke activities to the Order from the selected Wholesaler choice activity. The invoke activities call the placeOrder Web services which Wholesalers A and B expose. We enable CEI event generation for the OrderFromA invoke activity on the Entry event.

1. Add the OrderFromA invoke activity to the Order from the selected Wholesaler choice:
   a. On the Process Editor palette, click the **Invoke** icon ( ).
   b. Click the **Order from selected Wholesaler** choice under the OrderFromCompanyA case.
   c. Update the invoke activity properties with the properties in Table 13-16.

Table 13-16 The OrderFromA invoke properties

Property name	Value
Description	
Display Name	OrderFromA
Details	
Partner	CompanyAOrderPartner
Interface	WhPlaceOrdWeb_I
Operation	placeOrder
Input: placeOrderRequest	PlaceOrderRequest
Output: placeOrderResponse	PlaceOrderResponse

Archived

d. Click the **Event Monitor** tab.

e. Click the **Selected** radio button.

f. Click the **Entry** check box.

g. In the Entry row, in the Event Content field select **Full**. Select the **On** check box.

2. Press Ctrl+S to save the process.

3. Add the **OrderFromB** Invoke activity to the Order from selected Wholesaler choice:

   a. On the Process Editor palette, click the **Invoke** icon ( ).

   b. Click the **Order from selected Wholesaler** choice under the OrderFromCompanyB case.

   c. Update the invoke activity properties with the properties in Table 13-17.

Table 13-17 The OrderFromB invoke properties

Property name	Value
Description	
Display Name	OrderFromB
Details	
Partner	CompanyBOrderPartner
Interface	WhPlaceOrdWeb_I
Operation	placeOrder
Input: placeOrderRequest	PlaceOrderRequest
Output: placeOrderResponse	PlaceOrderResponse

4. Press Ctrl+S to save the process.

Archived

Adding a compensation handler to the OrderFromA invoke activity

This section explains how to add a compensation handler to the OrderFromA invoke activity. Adding this handler enables the process to compensate the activity, by calling the Wholesaler A cancelOrder operation, if the order is cancelled or the schedule date is not approved (Figure 13-48).

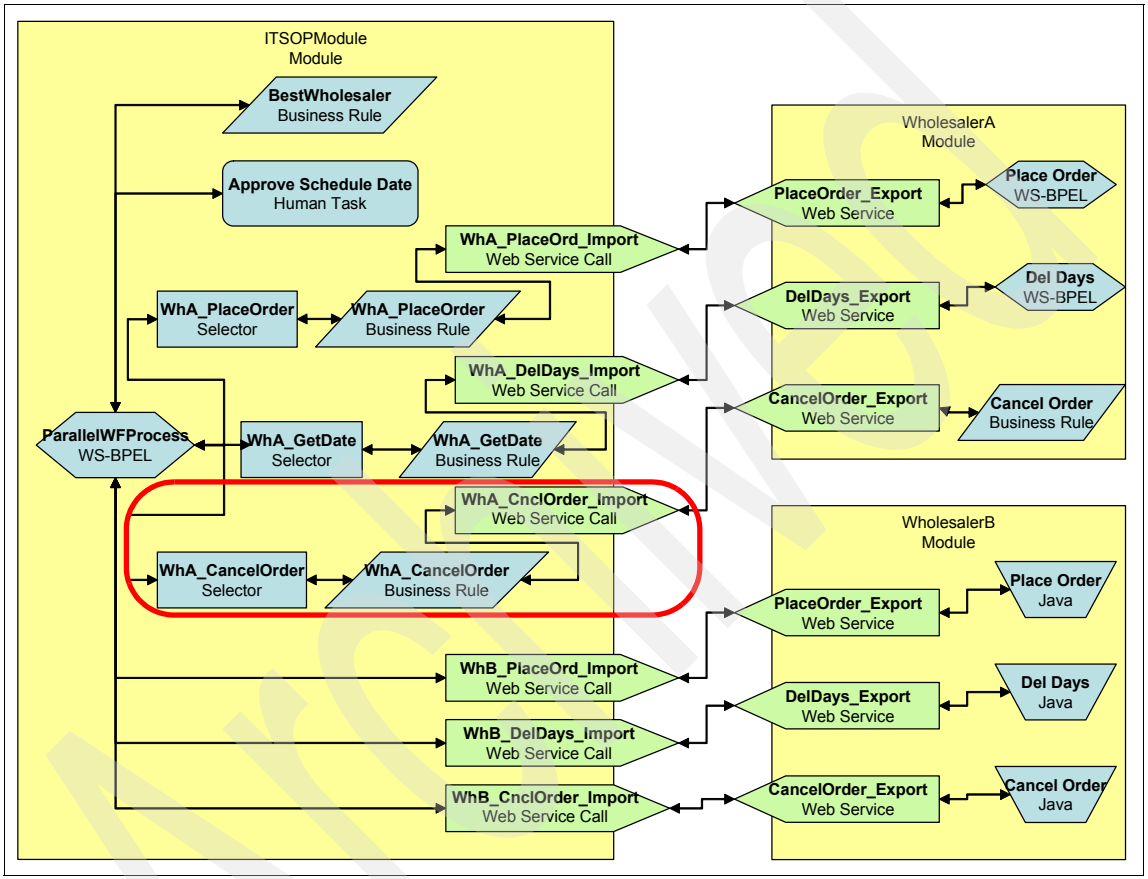


Figure 13-48 Overview of the scenario showing the interlinked components and modules

Archived

The OrderFromA compensation handler contains the activities listed in Table 13-18.

Table 13-18 The OrderFromA Compensation Handler activities

Activity name	Activity type	Additional information
Cancel Order A Sequence	Sequence	This activity refers to the sequence which contains the other activities in this compensation handler.
Assign CancelOrderRequest A	Assign	This activity assigns the input to the Wholesaler A cancelOrder Web service.
Compensating PlaceOrderA	Java snippet	This activity prints to the log that Wholesaler A placeOrder is being compensated.
CancelOrderA	Invoke	This activity invokes the Wholesaler A cancelOrder Web service. It is wired to the Wholesaler A cancelOrder Web service, in the ITSOPModule assembly diagram, using the selector-business rule pattern, which enabled us to replace Wholesaler A at run time without changing the process.

Add a compensation handler to the OrderFromA invoke activity:

1. Right-click the **OrderFromA** invoke activity and click **Add Compensation Handler**.
2. Add a sequence to the OrderFromA Compensation Handler:
   a. On the Process Editor palette, click the **Sequence** icon ( ).
   b. Click the **OrderFromA Compensation Handler** to add the new sequence.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Cancel Order A Sequence`.
3. Add an invoke to the Cancel Order A Sequence:
   a. On the Process Editor palette, click the **Invoke** icon ( ).
   b. Click inside the **Cancel Order A Sequence** sequence to add the new invoke.

Archived

c. Update the Invoke activity properties with the properties in Table 13-19.

Table 13-19 The CancelOrderA invoke properties

Property name	Value
Description	
Display Name	CancelOrderA
Details	
Partner	CompanyACancelOrderPartner
Interface	WhCnclOrdWeb_I
Operation	cancelOrder
Input: cancelOrderRequest	CancelOrderRequest
Output: cancelOrderResponse	CancelOrderResponse

4. Add an assign activity to the Cancel Order A Sequence:
   a. On the Process Editor palette, click the **Assign** icon ( ).
   b. Click inside the **Cancel Order A Sequence** sequence on the arrow above the CancelOrderA invoke to add the new assign.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Assign CancelOrderRequest A`.
   e. Click the **Details** tab. Update the Assign CancelOrderRequest A assign activity details as shown in Table 13-20.

Table 13-20 The Assign CancelOrderRequest A assign activity

Order	From		To	
1	Variable	PlaceOrderResponse → confirmCode	Variable	CancelOrderRequest → confirmationCode

5. Add a snippet to the Cancel Order A Sequence:
   a. On the Process Editor palette, click the **Snippet** icon ( ).
   b. Click inside the **Cancel Order A Sequence** sequence, on the arrow below the Assign CancelOrderRequest A assign activity, to add the new snippet.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Compensating PlaceOrderA`.
   e. Click the **Details** tab. Click the **Java** radio button.

f. In the Question window, click **Yes**.

g. In the text field, type the code shown in Example 13-2.

Example 13-2 Compensating PlaceOrderA snippet Java code

```
System.out.println("Compensating PlaceOrderA");
```

Figure 13-49 shows how the completed OrderFromA compensation handler looks.

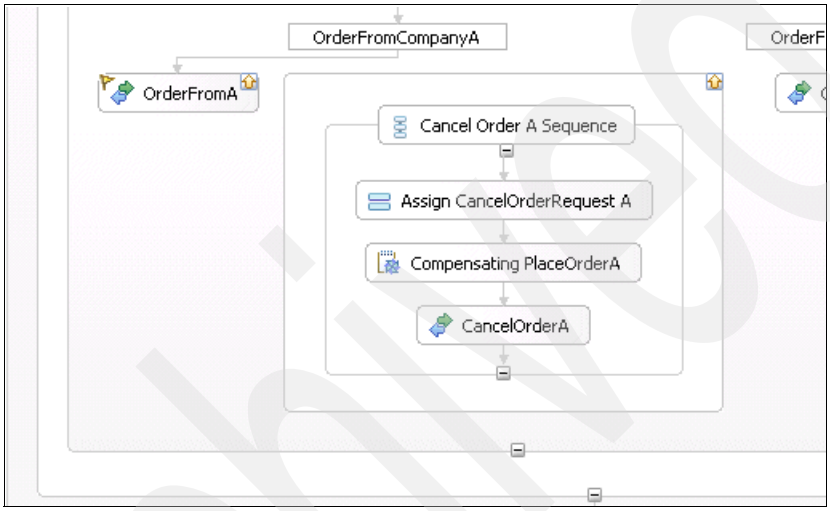


Figure 13-49 The OrderFromA compensation handler

6. Press Ctrl+S to save the process.

Archived

Adding a compensation handler to the OrderFromB invoke activity

This section explains how to add a compensation handler to the OrderFromB invoke activity. This handler enables the process to compensate the activity, by calling the Wholesaler B cancelOrder operation, if the order is cancelled or the schedule date is not approved (Figure 13-50).

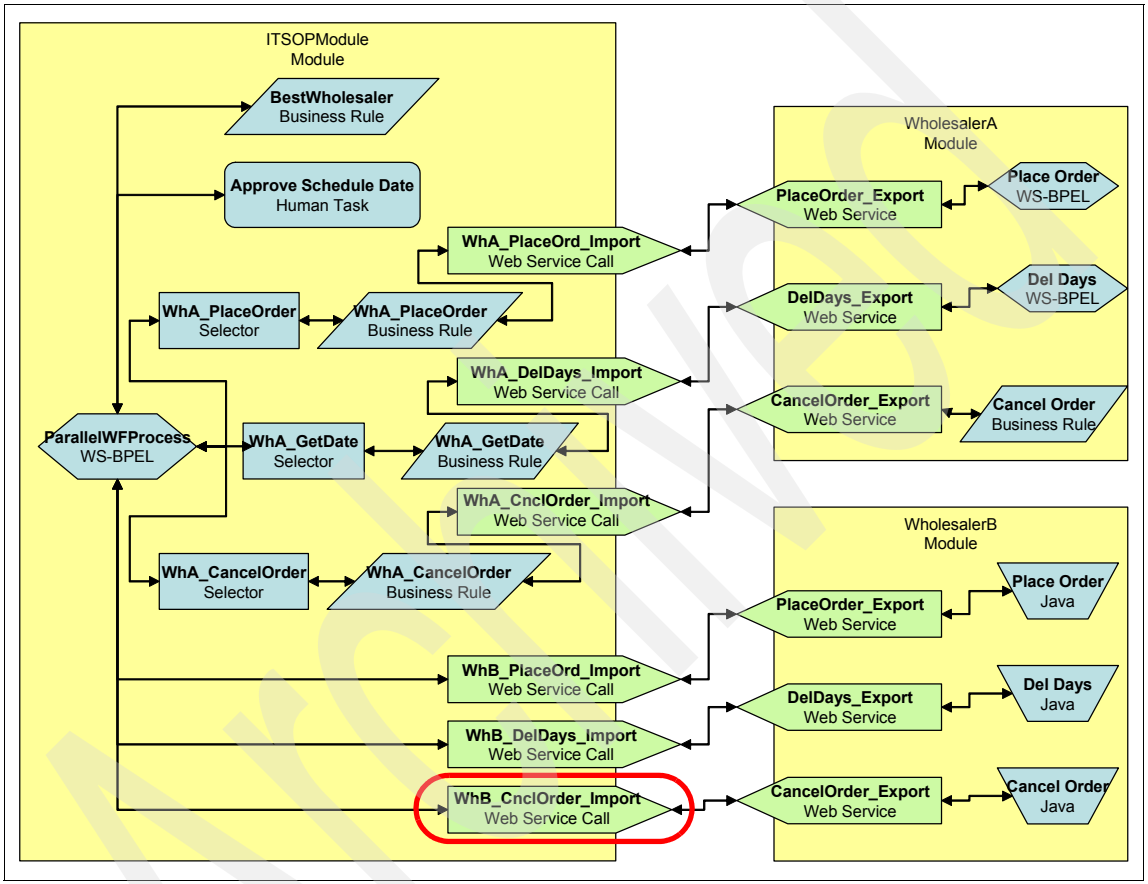


Figure 13-50 Overview of the scenario showing the interlinked components and modules

Archived

The OrderFromB compensation handler contains the activities listed in Table 13-21.

Table 13-21 The OrderFromB compensation handler activities

Activity name	Activity type	Additional information
Cancel Order B Sequence	Sequence	This refers to the sequence which contains the other activities in this compensatiion handler.
Assign CancelOrderRequest B	Assign	This activity assigns the input to the Wholesaler B cancelOrder Web service.
Compensating PlaceOrderB	Java snippet	This activity prints to the log that Wholesaler B placeOrder is being compensated.
CancelOrderB	Invoke	This activity invokes the Wholesaler B cancelOrder Web service.

Add a compensation handler to the OrderFromB Invoke activity:

1. Right-click the **OrderFromB** invoke activity and click **Add Compensation Handler**.
2. Add a sequence to the OrderFromB Compensation Handler:
   a. On the Process Editor palette, click the **Sequence** icon ( ).
   b. Click the **OrderFromB Compensation Handler** to add the new sequence.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Cancel Order B Sequence`.
3. Add an invoke to the Cancel Order B Sequence:
   a. On the Process Editor palette, click the **Invoke** icon ( ).
   b. Click inside the **Cancel Order B Sequence** sequence to add the new invoke.
   c. Update the invoke activity properties with the properties in Table 13-22.

Table 13-22 The CancelOrderB invoke properties

Property name	Value
Description	
Display Name	CancelOrderB
Details	
Partner	CompanyBCancelOrderPartner
Interface	WhCnclOrdWeb_I

Property name	Value
Operation	cancelOrder
Input: cancelOrderRequest	CancelOrderRequest
Output: cancelOrderResponse	CancelOrderResponse

4. Add an assign to the Cancel Order B Sequence:
   a. On the Process Editor palette, click the **Assign** icon ( ).
   b. Click inside the **Cancel Order B Sequence** sequence, on the arrow above the CancelOrderB invoke, to add the new assign.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Assign CancelOrderRequest B`.
   e. Click the **Details** tab. Update the Assign CancelOrderRequest B assign activity details as shown in the Table 13-23.

   Table 13-23 The Assign CancelOrderRequest B assign activity

Order	From		To	
1	Variable	PlaceOrderResponse → confirmCode	Variable	CancelOrderRequest → confirmationCode

5. Add a snippet to the Cancel Order B Sequence:
   a. On the Process Editor palette, click the **Snippet** icon ( ).
   b. Click inside the **Cancel Order B Sequence** sequence on the arrow, under the Assign CancelOrderRequest B assign activity, to add the new snippet.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Compensating PlaceOrderB`.
   e. Click the **Details** tab. Click the **Java** radio button.
   f. In the Question window, click **Yes**.
   g. In the text field, type the code shown in Example 13-3.

   Example 13-3 Compensating PlaceOrderB snippet Java code

```
System.out.println("Compensating PlaceOrderB");
```

Archived

Figure 13-51 shows the completed OrderFromB compensation handler.

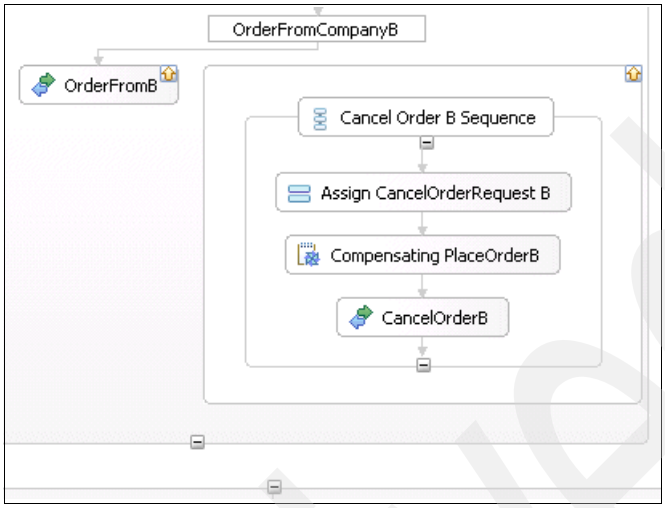


Figure 13-51 The OrderFromB compensation handler

6. Press Ctrl+S to save the process.

Adding the Has Best Wholesaler been found? assign activities

Add two of the assign activities to the Has Best Wholesaler been found? choice.

1. Click the **Assign** icon ( ) to add an assign activity from the Business Process Editor palette to the Has Best Wholesaler been found? choice, under the OrderFromCompany case (but before the Order from selected wholesaler choice), and rename it to `Create PlaceOrderRequest`.
2. Click the **Assign** icon ( ) to add an assign activity from the Business Process Editor palette to the Has Best Wholesaler been found? choice, under the Otherwise case, and rename it to `Assign NotAvailableFault`.
3. Update the Assign activities details shown in Table 13-24.

Table 13-24 The Has Best Wholesaler been found? choice assign activities

Order	From		To	
Create PlaceOrderRequest				
1	Variable	SelectedWholesaler → order → partId	Variable	PlaceOrderRequest → partNo
2	Variable	SelectedWholesaler → order → quantity	Variable	PlaceOrderRequest → qty

Archived

Order	From		To	
Assign NotAvailableFault				
1	Variable	InitialOrder	Variable	NotAvailableFault → order
2	Fixed Value	Best Wholesaler not found. Order cannot be fulfilled.	Variable	NotAvailableFault → reason

4. Press Ctrl+S to save the process.

Adding the Best Wholesaler not found Java Snippet

Add the Best Wholesaler not found Java snippet to the Has Best Wholesaler been found? choice.

1. On the Process Editor palette, click the **Snippet** icon ( ).
2. Click the **Has Best Wholesaler been found?** choice below the Otherwise case to add the new snippet.
3. Click the **Properties** page.
4. Click the **Description** tab. In the Display Name field, type `Best Wholesaler not found`.
5. Click the **Details** tab. Click the **Java** radio button.
6. In the Question window, click **Yes**.
7. In the text field, enter the code shown in Example 13-4.

Example 13-4 Best Wholesaler not found snippet Java code

```
System.out.println("Best Wholesaler not found !!");
```

Adding the Throw NotAvailableFault activity

Add the Throw NotAvailableFault activity to the Has Best Wholesaler been found? choice.

1. On the Process Editor palette, click the **Throw** icon ( ).
2. Click the **Has Best Wholesaler been found?** choice below the Assign NotAvailableFault activity to add the new throw.
3. Click the **Properties** page.
4. Click the **Description** tab. In the Display Name field, type `Throw NotAvailableFault`.

Archived

5. Click the **Details** tab (Figure 13-52).
   a. For Fault Type, select the **User-defined** radio button.
   b. In the Fault Name field, type `NotAvailableFault`.
   c. Click **Browse**.
   d. In the Select Fault Variable window, select the **NotAvailableFault** variable, and click **OK**.

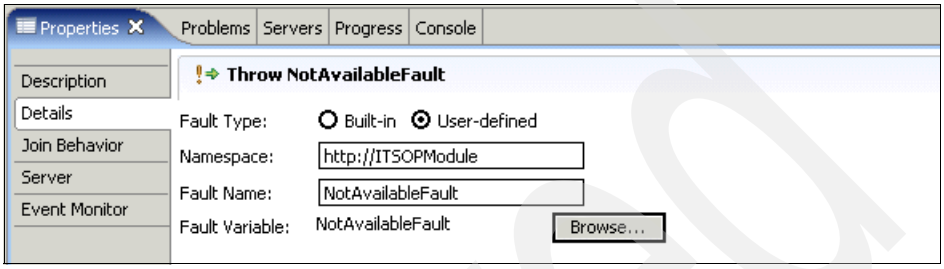


Figure 13-52 The Throw NotAvailableFault Properties page: Details tab

Figure 13-53 shows how the Has Best Wholesaler been found? choice looks after completing these steps.

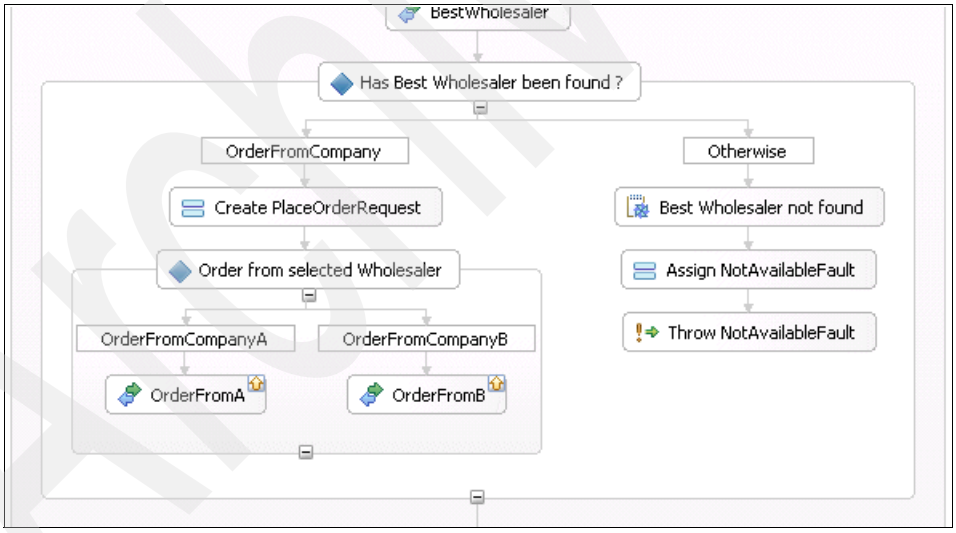


Figure 13-53 The Has Best Wholesaler been found? choice

6. Press Ctrl+S to save the process.

Archived

Adding the Place Order Process Scope fault handler

Add a fault handler to the Place Order Process Scope activity that will enable the scope to compensate specific activities or scopes and terminate the process if the order is cancelled, the schedule date is not approved, or the order cannot be fulfilled. The Place Order Process Scope fault handler contains the activities listed in Table 13-25.

Table 13-25 The Place Order Process Scope fault handler activities

Activity name	Activity type	Additional information
Fault caught... Lets compensate and terminate	Java snippet	This activity prints to the log that we caught a fault and will start the compensation and terminate the process.
Compensate the Acknowledgement	Visual snippet	This activity compensates the order acknowledgement by printing to the log that the order has been cancelled. In the real world, we might send an e-mail to the client that the order has been cancelled, the clerk has not approved the schedule date, or the order cannot be fulfilled.
Compensate PlaceOrderA	Compensate	This activity invokes the OrderFromA compensation handler.
Compensate PlaceOrderB	Compensate	This activity invokes the OrderFromB compensation handler.
Terminate Process	Terminate	This activity terminates the PlaceOrderProcess.

1. Right-click the **Place Order Process Scope** activity and click **Add Fault Handler**.
2. Right-click the Place Order Process Scope **Fault Handler** and click **Add Catch All**.
3. Right-click the Place Order Process Scope Fault Handler **Catch** and click **Delete**.
4. Add a snippet to the Place Order Process Scope Fault Handler:
   a. On the Process Editor palette, click the **Snippet** icon ( ).
   b. Click the Place Order Process Scope **Fault Handler** under the Catch All to add the new snippet.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Fault caught... Lets compensate and terminate`.
   e. Click the **Details** tab. Click the **Java** radio button.
   f. In the Question window, click **Yes**.

Archived

g. In the text field, enter the code shown in Example 13-5.

Example 13-5 Fault caught... Lets compensate and terminate snippet Java code

```
System.out.println("We caught a fault! Lets compensate and
terminate...");
```

5. Add a visual snippet activity to the Place Order Process Scope fault handler to compensate the order acknowledgement:
   a. On the Process Editor palette, click the **Snippet** icon ( ).
   b. Click the Place Order Process Scope **Fault Handler** under the Fault caught... Lets compensate and terminate activity to add the new snippet.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Compensate the Acknowledgement`.
   e. Click the **Details** tab. Ensure that the **Visual** radio button is selected and create the snippet shown in Figure 13-54.

Figure 13-54 The Compensate the Acknowledgement Properties page: Details tab

6. Add a Compensate activity to the Place Order Process Scope fault handler:
   a. On the Process Editor palette, click the **Compensate** icon ( ).
   b. Click the Place Order Process Scope **Fault Handler** under the Compensate the Acknowledgement Visual snippet to add the new compensate.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Compensate PlaceOrderA`.
   e. Click the **Details** tab. In the Target Activity field, select the **OrderFromA** activity.
7. Add a second Compensate activity to the Place Order Process Scope fault handler:
   a. On the Process Editor palette, click the **Compensate** icon ( ).
   b. Click the Place Order Process Scope **Fault Handler** under the Compensate PlaceOrderA activity to add the new compensate.
   c. Click the **Properties** page.

Archived

d. Click the **Description** tab. In the Display Name field, type `Compensate PlaceOrderB`.

e. Click the **Details** tab. In the Target Activity field, select the **OrderFromB** activity.

8. Add a Terminate activity to the Place Order Process Scope Fault Handler:

   a. On the Process Editor palette, click the **Terminate** icon ( ).

   b. Click the Place Order Process Scope **Fault Handler** under the Compensate PlaceOrderB activity to add the new terminate.

   c. Click the **Properties** page.

   d. Click the **Description** tab. In the Display Name field, type `Terminate Process`.

   Figure 13-55 shows the completed Place Order Process Scope Fault Handler.

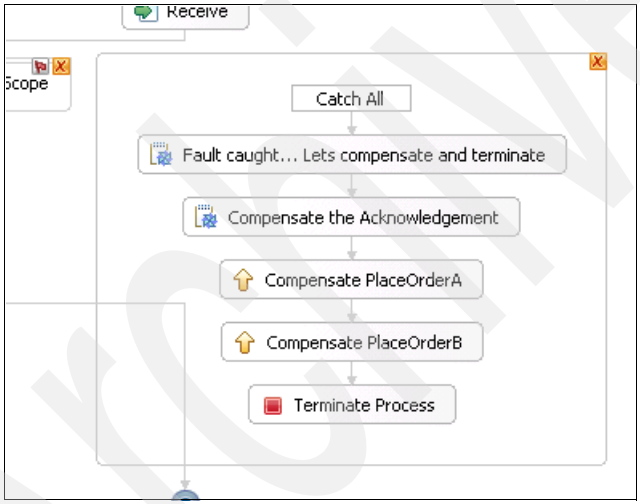


Figure 13-55 The Place Order Process Scope fault handler

9. Press Ctrl+S to save the process.

Archived

Adding the Approve Schedule Date activity

Add the Approve Schedule Date invoke activity to the PlaceOrderProcess (Figure 13-56) as explained in the following sections.

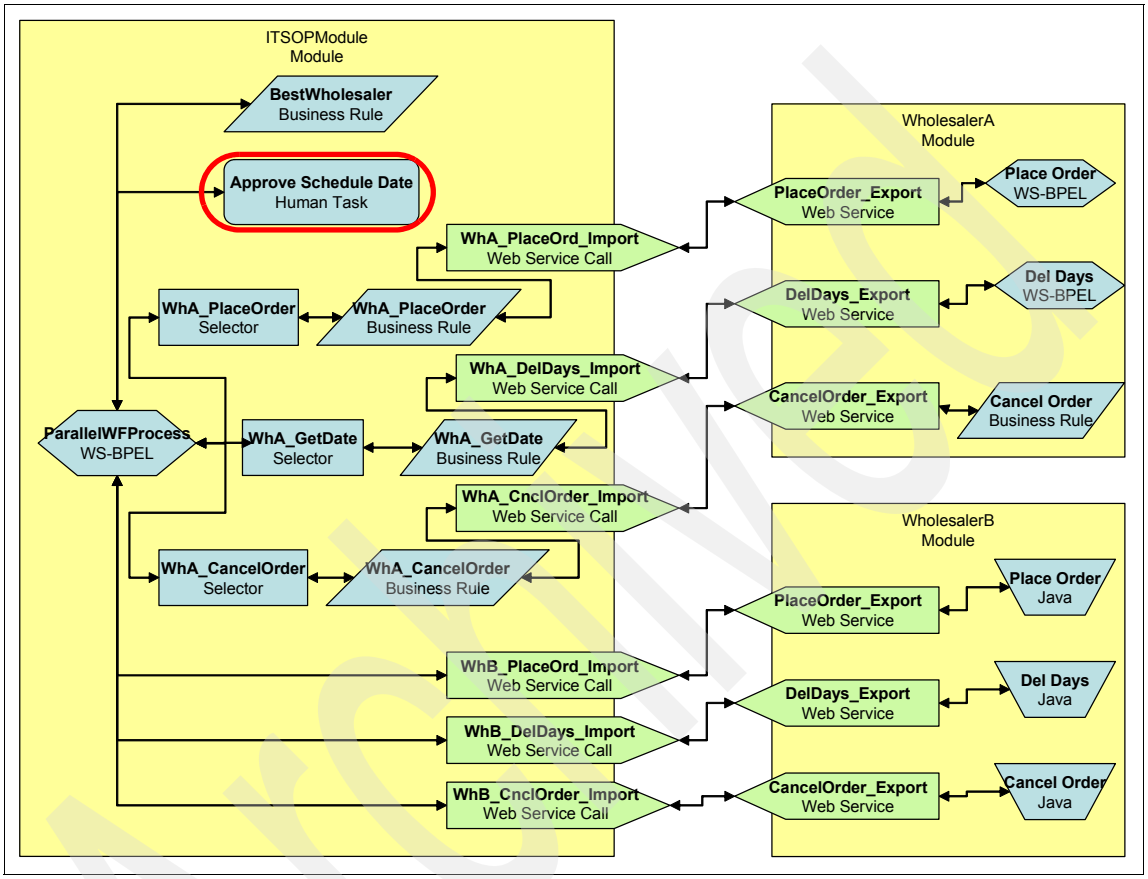


Figure 13-56 Overview of the scenario showing the interlinked components and modules

Table 13-26 describes the Approve Schedule Date activity.

Table 13-26 The Approve Schedule Date activity

Activity name	Activity type	Additional information
Approve Schedule Date	Invoke	This activity invokes the ApproveScheduleDate human task. The Approve Schedule Date invoke activity is wired to the human task in the ITSOPModule assembly diagram.

Adding the Approve Schedule Date invoke activity

Add the Approve Schedule Date invoke activity to the Place Order Process Scope.

1. On the Process Editor palette, click the **Invoke** icon ( ) and click the **arrow** after the Has Best Wholesaler been found? choice inside the Place Order Process Scope.
2. Update the Invoke activity properties with the properties in Table 13-27.

Table 13-27 The Approve Schedule Date invoke properties

Property name	Value
Description	
Display Name	Approve Schedule Date
Details	
Partner	ApproveSchedulePartner
Interface	ApproveSchedule
Operation	getOrderApproval
Input: wholesalerOrder	SelectedWholesaler
Input: confirmationCode	ConfirmationCode
Output: approved	IsApproved
Output: notApprovedReason	NotApprovedReason

3. Press Ctrl+S to save the process.

Archived

Adding the input assign for the Approve Schedule Date invoke

Add the assign activity to the Has Best Wholesaler been found? choice to assign the ConfirmationCode input for the Approve Schedule Date invoke activity.

1. Add an assign activity from the Business Process Editor palette. Click the **Assign** icon ( ) to the Has Best Wholesaler been found? choice under the Order from selected Wholesaler choice activity. Rename it to `Initialize Confirmation Code`.
2. Update the Assign activity details with the properties in Table 13-28.

Table 13-28 The input assign for the Approve Schedule Date invoke

Order	From		To	
Initialize Confirmation Code				
1	Variable	PlaceOrderResponse → confirmCode	Variable	ConfirmationCode

3. Press Ctrl+S to save the process.

Adding the Schedule Date Approval Scope

Add the Schedule Date Approval Scope scope activity and its fault handler to the PlaceOrderProcess as explained in the following sections. Table 13-29 describes the Schedule Date Approval Scope activity.

Table 13-29 The Schedule Date Approval Scope activity

Activity name	Activity type	Additional information
Schedule Date Approval Scope	Scope	The scope that surrounds the Has Schedule Date been approved? choice.

Adding the Schedule Date Approval Scope scope activity

Add the Schedule Date Approval Scope activity to the PlaceOrderProcess.

1. On the Process Editor palette, click the **Scope** icon ( ).
2. Click the **arrow** after the Approve Schedule Date invoke, inside the Place Order Process Scope.
3. Click the **Properties** page.
4. Click the **Description** tab. In the Display Name field, type `Schedule Date Approval Scope`.
5. Press Ctrl+S to save the process.

Archived

Adding the Schedule Date Approval Scope fault handler

Add a fault handler to the Schedule Date Approval Scope activity. This handler will enable the scope to do something useful with the fault or rethrow the fault if the order is cancelled or the schedule date is not approved. The Schedule Date Approval Scope fault handler contains the activities listed in Table 13-30.

Table 13-30 The Schedule Date Approval Scope Fault Handler activities

Activity name	Activity type	Additional information
Fault caught... Lets rethrow	Java snippet	This activity prints to the log that we caught a fault and that we are going to rethrow it.
Rethrow Fault	Rethrow	This activity rethrows the fault.

1. Right-click the **Schedule Date Approval Scope** activity and click **Add Fault Handler**.
2. Right-click the Schedule Date Approval Scope **Fault Handler** and click **Add Catch All**.
3. Right-click the Schedule Date Approval Scope Fault Handler **Catch** and click **Delete**.
4. Add a snippet to the Schedule Date Approval Scope Fault Handler:
   a. On the Process Editor palette, click the **Snippet** icon ( ).
   b. Click the Schedule Date Approval Scope **Fault Handler** under the Catch All to add the new snippet.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Fault caught... Lets rethrow`.
   e. Click the **Details** tab. Click the **Java** radio button.
   f. In the Question window, click **Yes**.
   g. In the text field, enter the code shown in Example 13-6.

   Example 13-6 Fault caught... Lets rethrow snippet Java code

   ```
   System.out.println("We caught a fault! Lets rethrow it...");
   ```

5. Add a rethrow activity to the Schedule Date Approval Scope fault handler:
   a. On the Process Editor palette, click the **Rethrow** icon ( ).
   b. Click the Schedule Date Approval Scope **Fault Handler** under the Fault caught... Lets rethrow Java snippet to add the new rethrow.

Archived

c. Click the **Properties** page.

d. Click the **Description** tab. In the Display Name field, type `Rethrow Fault`.

Figure 13-57 shows the completed Schedule Date Approval Scope Fault Handler.

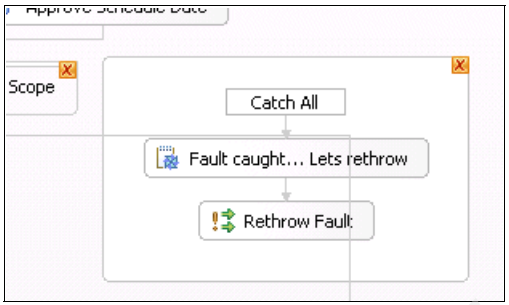


Figure 13-57 The Schedule Date Approval Scope fault handler

6. Press Ctrl+S to save the process.

Adding the Has Schedule Date been approved? choice

In the following sections, we add the Has Schedule Date been approved? choice activity and the activities it contains to the PlaceOrderProcess. The Has Schedule Date been approved? activity contains the activities listed in Table 13-31.

Table 13-31 The Has Schedule Date been approved? and the activities it contains

Activity name	Activity type	Additional information
Has Schedule Date been approved?	Choice	This activity determines if the schedule date has been approved.
Notify Assembly Date	Visual snippet	This activity prints to the log that the assembly date has been approved.
Schedule Date Not Approved	Java snippet	This activity prints to the log that the assembly date has not been approved.
Assign NotApprovedFault	Assign	This activity assigns the schedule date is not approved fault.
Throw NotApprovedFault	Throw	This activity throws the schedule date is not approved fault.

Archived

Adding the Has Schedule Date been approved? choice activity

Add the Has Schedule Date been approved? activity to the PlaceOrderProcess.

1. Add a choice activity to the Schedule Date Approval Scope:
   a. On the Process Editor palette, click the **Choice** icon ( ).
   b. Click inside the **Schedule Date Approval Scope**.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display Name field, type `Has Schedule Date been approved?`
   e. Click **Case**.
   f. Click the **Properties** page.
   g. Click the **Description** tab. In the Display Name field, type `Schedule Date Approved`.
   h. Click the **Details** tab.
      i. In the Expression language field, select **Same as Process (Java)**.
      ii. Click the Expression Type **Visual** radio button.
      iii. From the helper menu, click the **true** Expression and click **IsApproved**.

   Figure 13-58 shows the completed Schedule Date Approved case visual snippet.

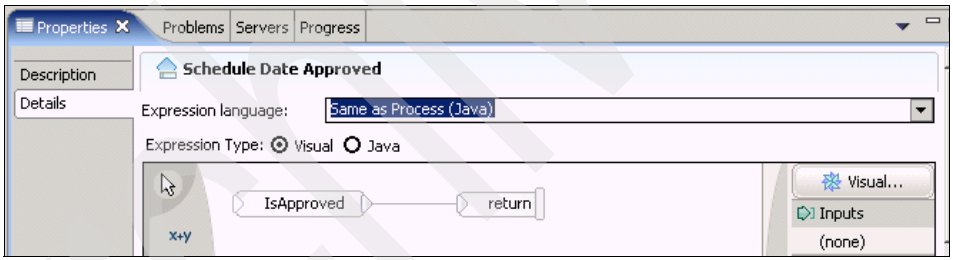


Figure 13-58 The Schedule Date Approved case Properties page: Details tab

2. Right-click the **Has Schedule Date been approved?** choice and click **Add Otherwise**.
3. Press Ctrl+S to save the process.

Adding the Notify Assembly Date visual snippet

Add the Notify Assembly Date visual snippet to the Has Schedule Date been approved? choice.

1. On the Process Editor palette, click the **Snippet** icon ( ).
2. Click the **Has Schedule Date been approved?** choice under the Schedule Date Approved case to add the new snippet.
3. Click the **Properties** page.

Archived

4. Click the **Description** tab. In the Display Name field, type `Notify Assembly Date`.
5. Click the **Details** tab. Ensure the **Visual** radio button is selected.
6. Build the Notify Assembly Date visual snippet as shown in Figure 13-59.

Figure 13-59 The Notify Assembly Date Properties page: Details tab

7. Press Ctrl+S to save the process.

Adding the Schedule Date Not Approved Java snippet

Add the Schedule Date Not Approved Java snippet to the Has Schedule Date been approved? choice.

1. On the Process Editor palette, click the **Snippet** icon ( ).
2. Click the **Has Schedule Date been approved?** flow under the Otherwise case to add the new snippet.
3. Click the **Properties** page.
4. Click the **Description** tab. In the Display Name field, type `Schedule Date Not Approved`.
5. Click the **Details** tab. Click the **Java** radio button.
6. In the Question window, click **Yes**.
7. In the text field, type the code shown in Example 13-7.

Example 13-7 Schedule Date Not Approved snippet Java code

```
System.out.println("The assembly date has not been approved. Throw
NotApprovedFault.");
```

8. Press Ctrl+S to save the process.

Adding the Assign NotApprovedFault assign activity

Add the Assign NotApprovedFault assign activity to the Has Schedule Date been approved? choice.

1. On the Process Editor palette, click the **Assign** icon ( ).
2. Click inside the **Has Schedule Date been approved?** choice under the Schedule Date Not Approved Java snippet to add the new assign.
3. Click the **Properties** page.

4. Click the **Description** tab. In the Display Name field, type `Assign NotApprovedFault`.
5. Click the **Details** tab. Update the **Assign NotApprovedFault** assign activity details as shown in Table 13-32.

Table 13-32 The Assign NotApprovedFault assign activity

Order	From		To	
1	Variable	NotApprovedReason	Variable	NotApprovedFault → reason
2	Variable	SelectedWholesaler	Variable	NotApprovedFault → order
3	Variable	ConfirmationCode	Variable	NotApprovedFault → confirmation

6. Press Ctrl+S to save the process.

Adding the Throw NotApprovedFault throw activity

Add the Throw NotApprovedFault throw activity to the Has Schedule Date been approved? choice.

1. On the Process Editor palette, click the **Throw** icon ( ).
2. Click inside the **Has Schedule Date been approved?** choice under the Assign NotApprovedFault assign to add the new throw.
3. Click the **Properties** page.
4. Click the **Description** tab. In the Display Name field, type `Throw NotApprovedFault`.
5. Click the **Details** tab (Figure 13-60).
   a. For Fault Type, click the **User-defined** radio button.
   b. In the Fault Name field, type `NotApprovedFault`.
   c. Click **Browse**.
   d. In the Select Fault Variable window, select the **NotApprovedFault** variable. Then click **OK**.

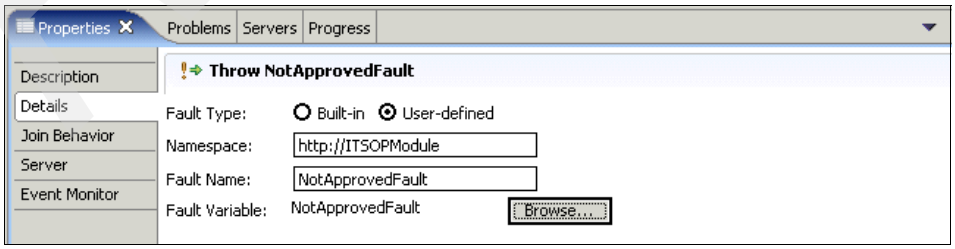


Figure 13-60 The Throw NotApprovedFault Properties page: Details tab

Figure 13-61 shows the completed Schedule Date Approval Scope.

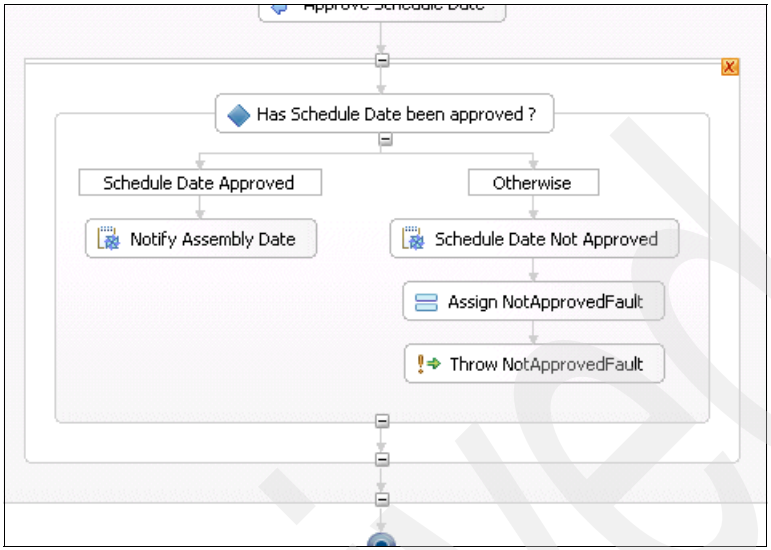


Figure 13-61 The completed Schedule Date Approval Scope

6. Press Ctrl+S to save the process.

13.3.5 Building the ITSOPModule assembly diagram

This section explains how to build the ITSOPModule PlaceOrderProcess Parallel Workflow assembly diagram. This diagram will wire the PlaceOrderProcess to the Web services it consumes and wire the Approve Schedule Date invoke to a human task. The WholesalerA Web services that the process consumes are invoked using the selector-business rule pattern which enable us to replace Wholesaler A at run time without changing the process.

Adding the Wholesaler A and B Web service imports

Add the Wholesaler A and Wholesaler B Web service imports to the ITSOPModule PlaceOrderProcess Parallel Workflow assembly diagram.

1. Copy the following Web service ports from the WholesalerA module to the ITSOPModule (you can find these in WholesalerA → Web Service Ports):
   - WholesalerA_CancelOrderExport_WhCnclOrdWeb_IHttpPort
   - WholesalerA_GetDeliveryDateExport_WhDelReqWeb_IHttpPort
   - WholesalerA_PlaceOrderExport_WhPlaceOrdWeb_IHttpPort

2. Copy the following Web service ports from the WholesalerB module to the ITSOPModule (you can find these in WholesalerB → Web Service Ports):
   - WhBExpCnclOrd_WhCnclOrdWeb_IHttpPort
   - WhBExpDelv_WhDelReqWeb_IHttpPort
   - WhBExpPlaceOrd_WhPlaceOrdWeb_IHttpPort
3. Open the ITSOPModule Assembly Diagram Editor.
4. Drag the **PlaceOrderProcess** from the ITSOPModule to the assembly diagram canvas.
5. Add the **WholesalerACancelOrder** import to the assembly diagram:
   a. Drag the **WholesalerA_CancelOrderExport_WhCnclOrd Web_IHttpPort** Web service port from the ITSOPModule to the assembly diagram canvas.
   b. In the Component Creation window that opens, select **Import with Web Service Binding**. Then click **OK**.
   c. Click the **Import1**.
   d. Click the **Properties** page.
   e. Click the **Description** tab. In the Display name field, type `WholesalerACancelOrder`.
6. Add five additional imports to the ITSOPModule assembly diagram with the properties listed in Table 13-33.

Table 13-33 Additional imports to add to the ITSOPModule assembly diagram

Web service port	Display name
WholesalerA_GetDeliveryDateExport_WhDelReq Web_IHttpPort	WholesalerAGetDate
WholesalerA_PlaceOrderExport_WhPlaceOrd Web_IHttpPort	WholesalerAPlaceOrder
WhBExpCnclOrd_WhCnclOrdWeb_IHttpPort	WholesalerBCancelOrder
WhBExpDelv_WhDelReqWeb_IHttpPort	WholesalerBGetDate
WhBExpPlaceOrd_WhPlaceOrdWeb_IHttpPort	WholesalerBPlaceOrder

7. Press Ctrl+S to save the assembly diagram.

Archived

Figure 13-62 shows how the assembly diagram looks after you add the imports.

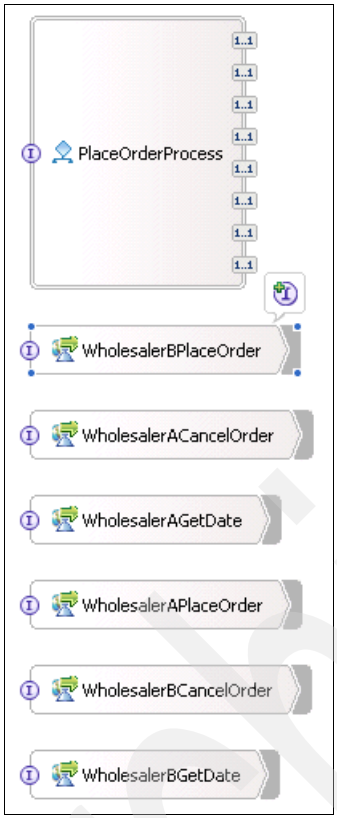


Figure 13-62 The ITSOPModule assembly diagram with the imports

Adding the Wholesaler A dynamicity business rule groups

Add the business rule groups needed to invoke the Wholesaler A Web services the process consumes, using the selector-business rule pattern to the ITSOPModule PlaceOrderProcess Parallel Workflow assembly diagram. The selector-business rule pattern enables us to replace Wholesaler A at run time without changing the process.

1. Add the WholesalerCancelOrderRG business rule group to the assembly diagram:
   a. On the Assembly Editor palette, click the **Rule Group** icon ( ).
   b. Click the **assembly diagram canvas** to add the new rule group.
   c. Click the **Properties** page.

d. Click the **Description** tab. In the Display name field, type `WholesalerCancelOrderRG`.

e. Click the **Details** tab.

i. Right-click **Interfaces** and click **Add Interface**.

ii. In the Add Interface window, select the **WhCnclOrdWeb_I interface**, and click **OK**.

2. Add two additional Rule Groups to the ITSOPModule assembly diagram with the properties listed in Table 13-34.

Table 13-34 Additional rule groups to add to the ITSOPModule assembly diagram

Display name	Interface
WholesalerGetDateRG	WhDelReqWeb_I
WholesalerPlaceOrderRG	WhPlaceOrdWeb_I

3. Add a wire from the WholesalerCancelOrderRG business rule group to the WholesalerACancelOrder import:

a. On the Assembly Editor palette, click the **Wire** icon ( ).
b. Click the **WholesalerCancelOrderRG** component.
c. Click the **WholesalerACancelOrder** import.
d. In the Add Wire window, click **OK**.

4. Add a wire from the WholesalerGetDateRG business rule group to the WholesalerAGetDate import.

5. Add a wire from the WholesalerPlaceOrderRG business rule group to the WholesalerAPlaceOrder import.

6. Press Ctrl+S to save the assembly diagram.

Archived

Figure 13-63 shows how the assembly diagram looks now after adding the rule groups.

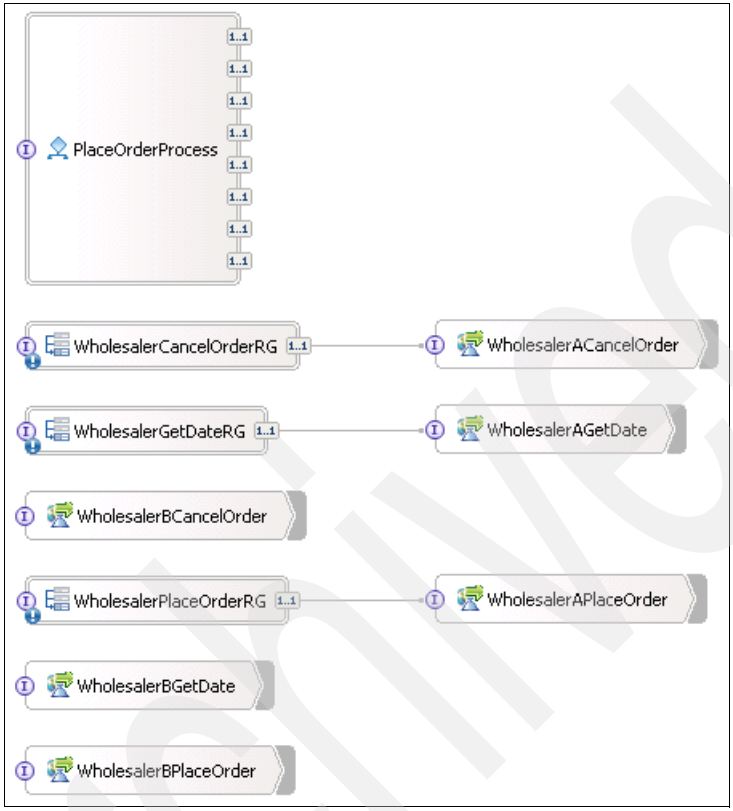


Figure 13-63 The ITSOPModule assembly diagram with the rule groups

Adding the Wholesaler A dynamicity selectors

Add the selectors needed to invoke the Wholesaler A Web service the process consumes, using the selector-business rule pattern to the ITSOPModule PlaceOrderProcess Parallel Workflow assembly diagram. The selector-business rule pattern enables us to replace Wholesaler A at run time without changing the process.

1. Add the WholesalerCancelOrderSelector selector to the assembly diagram:
   a. On the Assembly editor palette, click the **Selector** icon ( ).
   b. Click the **assembly diagram canvas** to add the new selector.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display name field, type `WholesalerCancelOrderSelector`.

e. Click the **Details** tab.

   i. Right-click **Interfaces** and click **Add Interface**.

   ii. In the Add Interface window, select the **WhCnclOrdWeb_I** interface, and click **OK**.

2. Add two additional selectors to the ITSOPModule assembly diagram with the properties listed in Table 13-35.

Table 13-35 Additional selectors to add to the ITSOPModule assembly diagram

Display name	Interface
WholesalerGetDateSelector	WhDelReqWeb_I
WholesalerPlaceOrderSelector	WhPlaceOrdWeb_I

3. Press Ctrl+S to save the assembly diagram.

Adding the Approve Schedule Date human task

Add the Approve Schedule Date human task to the ITSOPModule PlaceOrderProcess Parallel Workflow assembly diagram. The Approve Schedule Date human task enables the clerk to approve or decline the schedule date.

1. On the Assembly Editor palette, click the **Human Task** icon ( ).
2. Click the **assembly diagram canvas** to add the new human task.
3. Click the **Properties** page.
4. Click the **Description** tab. In the Display name field, type `ApproveScheduleDate`.
5. Click the **Details** tab.
6. Right-click **Interfaces** and click **Add Interface**.
7. In the Add Interface window, select the **ApproveSchedule** interface, and click **OK**.
8. In the Properties view, expand **Interfaces** and click the **ApproveSchedule** interface.
9. On the Details - Properties of Interface ApproveSchedule page, in the Preferred interaction style field, select **Asynchronous**.

Archived

> **Note:** SCA requires the preferred interaction style to be set to *asynchronous*. If the preferred interaction style is not set to asynchronous, we see the following error message when we save the assembly diagram:
>
> ```
> The interface 'ns1:ApproveSchedule' in the task component file
> '/ITSOPModule/ApproveScheduleDate.component' needs the
> preferredInteractionStyle attribute with a value of 'async'.
> ```

10. Click the **Qualifiers** tab.
11. In the Quality of Service (QOS) Qualifiers page, click **Add**.
12. In the Add Qualifier window, select **Join transaction**, and click **OK**.
13. Click **Join transaction**.
14. In the Value field, select **False**.

> **Note:** SCA requires the join transaction interface qualifier to be set to *false*. If the join transaction interface qualifier is not set to false, we see the following error message when we save the assembly diagram:
>
> ```
> The operation 'getOrderApproval' of the interface 'ns1:ApproveSchedule'
> in the task component file '/ITSOPModule/ApproveScheduleDate.component'
> does not contain the JoinTransaction interface qualifier with a value of
> 'false'.
> ```

Adding the Best Wholesaler business rule group

Add the BestWholesaler business rule group to the ITSOPModule PlaceOrderProcess Parallel Workflow assembly diagram. The BestWholesaler business rule determines which wholesaler to place the order with, based on the number of days to delivery.

1. Drag the **BestWholesalerRG** business rule group from the ITSOPModule to the assembly diagram canvas.
2. Press Ctrl+S to save the assembly diagram.

Archived

Figure 13-64 shows the human task and the BestWholesaler rule group the assembly diagram after you add the selectors.

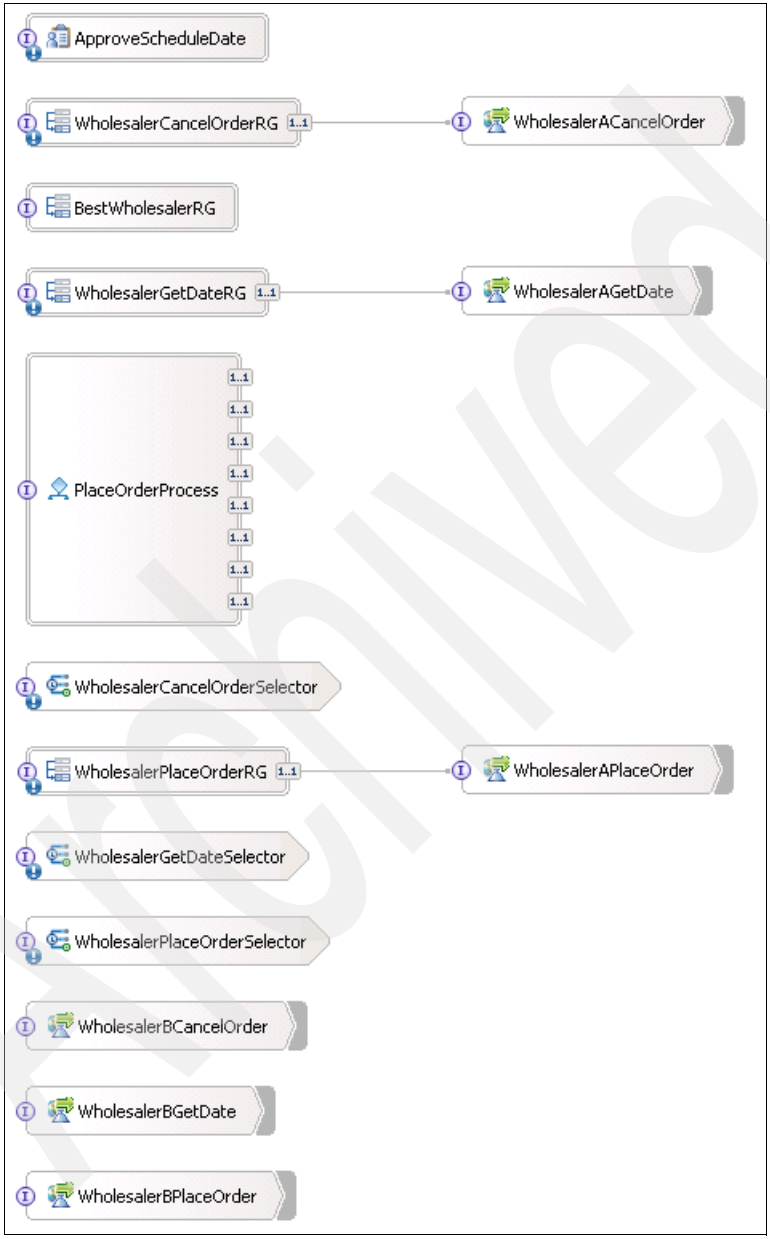


Figure 13-64 The ITSOPModule assembly diagram with the selectors and human task

Archived

Generating implementations for the Wholesaler A dynamicity business rule groups

This section describes how to generate implementations for the business rule groups needed to invoke the Wholesaler A Web services, the process consumes, using the selector-business rule pattern. The selector-business rule pattern allows us to replace Wholesaler A at run time without changing the process.

1. Generate an implementation for the WholesalerCancelOrderRG business rule group:
   a. Right-click the **WholesalerCancelOrderRG** component and click **Generate Implementation**.
   b. In the Generate Implementation window, click **OK**.
   c. In the WholesalerCancelOrderRG Rule Group window, click the **cancelOrder** operation. In the Active Destinations section, click **Enter Destination**, and from the helper menu, click **New Ruleset**.
   d. In the New Rule Set window, in the Name field, type `WholesalerCancelOrderRS`. Click **Finish**.
   e. In the WholesalerCancelOrderRS Rule Set window, click the **Add Action Rule** icon ( ).
      i. Click **Action**.
      ii. Click **Invoke**.
      iii. Click **Select Partner Link**, and from the helper menu, click **WhCnclOrdWeb_IPartner**.
      iv. Click **Select Operation**, and from the helper menu, click **cancelOrder**.
      v. On the Input row, click **Enter Value**, and from the helper menu, click **cancelOrderRequest**.
      vi. On the Output row, click **Enter Value**, and from the helper menu, click **cancelOrderResponse**.
   f. Press Ctrl+S to save the WholesalerCancelOrderRS business rule set.

Archived

Figure 13-65 shows the completed WholesalerCancelOrderRS business rule set.

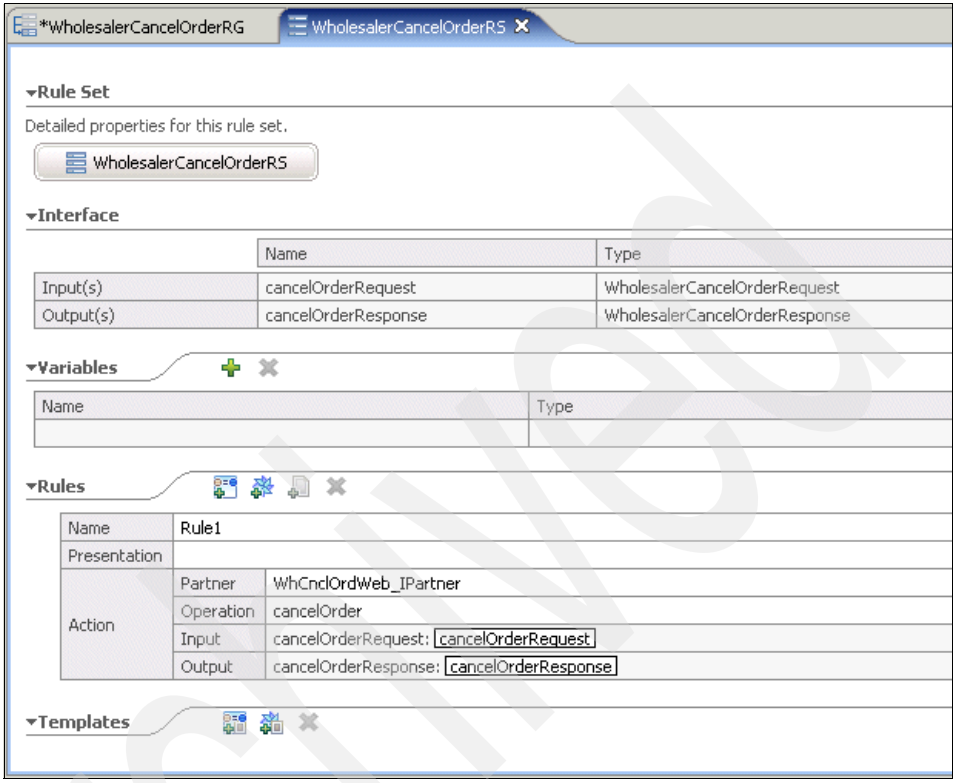


Figure 13-65 The WholesalerCancelOrderRS business rule set

g. Close the WholesalerCancelOrderRS Business Rule Set Editor.

h. Switch to the WholesalerCancelOrderRG Business Rule Group Editor.

Archived

The completed WholesalerCancelOrderRG business rule group should contain the Available Destination WholesalerCancelOrderRS as shown in Figure 13-66.

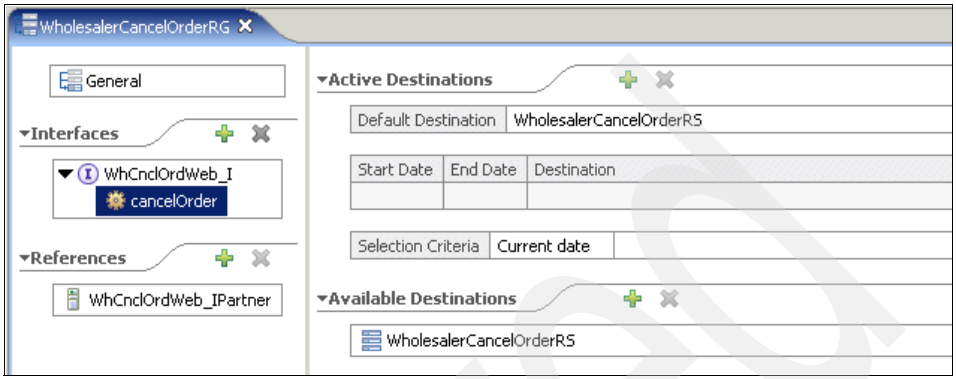


Figure 13-66 The WholesalerCancelOrderRG business rule group

i. Press Ctrl+S to save the WholesalerCancelOrderRG business rule group.

j. Close the WholesalerCancelOrderRG Business Rule Group Editor.

2. Generate an implementation for the WholesalerGetDateRG business rule group with the properties listed in Table 13-36.

Table 13-36 WholesalerGetDateRG Generate Implementation properties

Rule Set name	Partner	Operation	Input	Output
WholesalerGet DateRS	WhDelReqWeb_ IPartner	getDeliveryDays	getDeliveryDays Wholesaler Request	getDeliveryDays Wholesaler Response

3. Generate an implementation for the WholesalerPlaceOrderRG business rule group with the properties listed in Table 13-37.

Table 13-37 WholesalerPlaceOrderR Generate Implementation properties

Rule Set name	Partner	Operation	Input	Output
WholesalerPlace OrderRS	WhPlaceOrdWeb_ IPartner	placeOrder	placeOrderRequest	placeOrderResponse

Archived

Generating implementations for the Wholesaler A dynamicity selectors

Generate implementations for the selectors needed to invoke the Wholesaler A Web service the process consumes, using the selector-business rule pattern. The selector-business rule pattern enables us to replace Wholesaler A at run time without changing the process.

1. Generate an implementation for the WholesalerCancelOrderSelector component:
   a. Right-click the **WholesalerCancelOrderSelector** component, and click **Generate Implementation**.
   b. In the Generate Implementation window, click **OK**.
   c. In the WholesalerCancelOrderSelector window, click the **cancelOrder** operation. In the Active Destinations section, click **Enter SCA Destination**, and from the helper menu, click **WholesalerCancelOrderRG**.

      Figure 13-67 shows the completed WholesalerCancelOrderSelector that contains the Default Destination WholesalerCancelOrderRG.

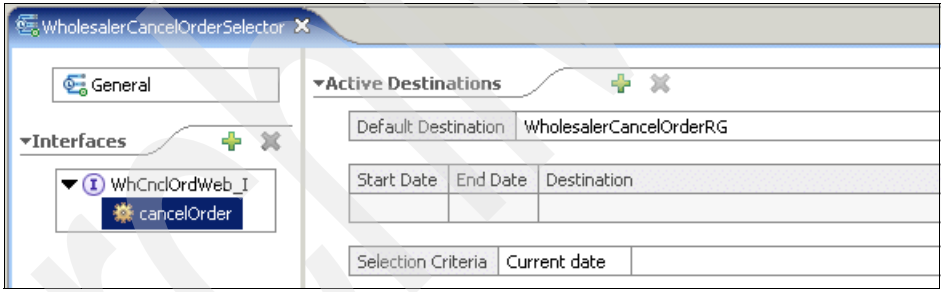


Figure 13-67 The WholesalerCancelOrderSelector

   d. Press Ctrl+S to save the WholesalerCancelOrderSelector.
   e. Close the WholesalerCancelOrderSelector editor.

2. Generate an implementation for the WholesalerGetDateSelector component with the properties listed in Table 13-38.

Table 13-38 WholesalerGetDateSelector Generate Implementation properties

Operation	SCA Destination
getDeliveryDays	WholesalerGetDateRG

Archived

3. Generate an implementation for the WholesalerPlaceOrderSelector component with the properties listed in Table 13-39.

Table 13-39 WholesalerPlaceOrderSelector Generate Implementation properties

Operation	SCA Destination
placeOrder	WholesalerPlaceOrderRG

Generating implementation for the Approve Schedule Date human task

This section explains how to generate an implementation for the Approve Schedule Date human task. The Approve Schedule Date human task enables the clerk to approve or decline the schedule date. We implement the ApproveScheduleDate human task with escalation by using three distinct user IDs: clerk, manager, and admin. If you do not have three distinct user IDs, create them now. If that is not possible or you do not want to implement escalation, you can skip the escalation part and use one user ID as the Potential Owner and Potential Instance creator.

Important: You must have set up security on the WebSphere Process Server test server before testing the ApproveScheduleDate Human Task. You can find instructions for configuring security on the test server in “Security” on page 603. To set up security on the WebSphere Process Server test server, the following steps are required:

1. “Setting up LocalOS security on WebSphere Process Server” on page 604
2. “Configuring the BPE Container” on page 611
3. “Configuring the Task Container” on page 614
4. “Configuring the WebSphere Integration Developer test server” on page 616
5. “Redeploying the file transfer application” on page 618

Generate an implementation for the ApproveScheduleDate human task:

1. Right-click the **ApproveScheduleDate** component and click **Generate Implementation**.
2. In the Generate Implementation window, click **OK**.
3. In the Human Task Component Handler window, in the name for the Human Task field, type `ApproveScheduleDate`. Then click **OK**.

4. In the ApproveScheduleDate window (Figure 13-68), complete these steps:
   a. Click the **Administrator** icon ( ).
   b. Click the **Properties** page.
   c. In the Verb field, select **Users** by user ID.
   d. In the Parameters field, on the User ID row in the Value column, type `admin`.

> **admin:** *admin* refers to the admin user ID. This ID makes the admin user the Administrator of this human task. An administrator has the authority to perform upper level duties such as to terminate the human task, delete the human task, manage work items associated with the human task, suspend the human task, and resume the human task.

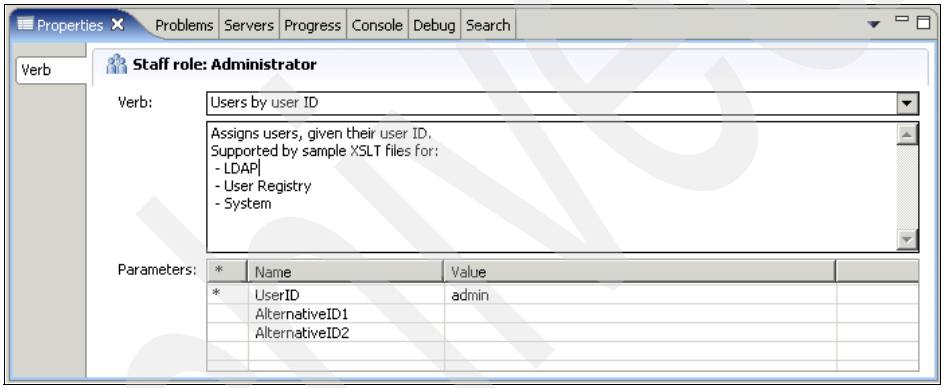


Figure 13-68 ApproveScheduleDate Receiver Settings Administrator properties

5. In the ApproveScheduleDate window in the *Receiver* settings section, click the **Potential Instance Creator** row.
   a. Click the **Properties** page.
   b. In the Verb field, select **Users by user ID**.
   c. In the Parameters field, type the following values:
      i. On the UserID row in the Value column, type `clerk`.
      ii. On the AlternativeID1row in the Value column, type `manager`.
      iii. On the AlternativeID2 row in the Value column, type `admin`.

> **Note:** A Potential Instance Creator can create an instance of the human task.

Archived

6. In the ApproveScheduleDate window in the *Receiver* settings section, click the **Potential Owner** row.

   a. Click the **Properties** page.
   b. In the Verb field, select **Users by user ID**.
   c. In the Parameters field, type the following values:

      i. In the UserID row in the Value column, type `clerk`.
      ii. In the AlternativeID1 row in the Value column, type `manager`.

   Note: A Potential Owner can claim, work on and complete tasks.

7. Add escalation to the human task. We escalate the task to the manager if the clerk has not claimed it within 2 minutes. When this happens, the manager is notified via a work item in the managers My Escalations view. The escalation is repeated only once. In the ApproveScheduleDate window in the *Escalation* settings section, complete the following steps:

   a. Click the **Ready** icon ( ).
   b. Click the **Escalation** icon ( ).
   c. Click the **Properties** page.
   d. Click the **Description** tab. Enter the following values:

      i. In the Name field, type `Assign_To_Manager`.
      ii. In the Display name field, type `Escalate To Manager`.

   e. Click the **Details** tab (Figure 13-69). Make the following selections:

      i. For Expected task state, select the **Claimed** radio button.
      ii. In the Duration until escalated field, type `2minutes`.
      iii. For Notification type, select the **Work item** radio button.
      iv. For Increase priority, select the **Increase this time only** radio button.

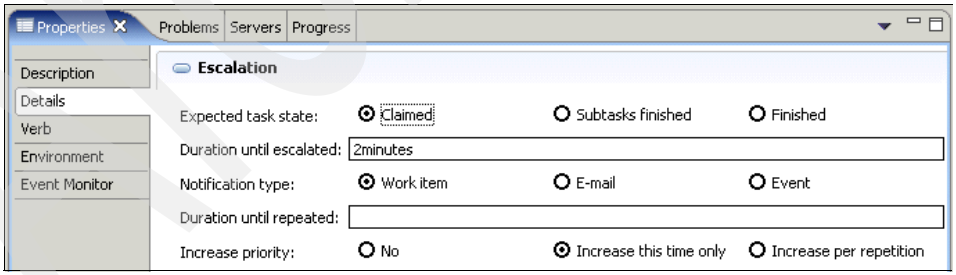


Figure 13-69 ApproveScheduleDate Escalation Properties: Details tab

Archived

f. Click the **Verb** tab and specify the following information:

   i. In the Verb field, select **Users by user ID**.
   ii. In the UserID row in the Value column, type `manager`.

> **Note:** All user IDs in this list are granted the Reader role for the human task. The reader is allowed to view tasks, but cannot work on them.

Figure 13-70 shows the completed ApproveScheduleDate.

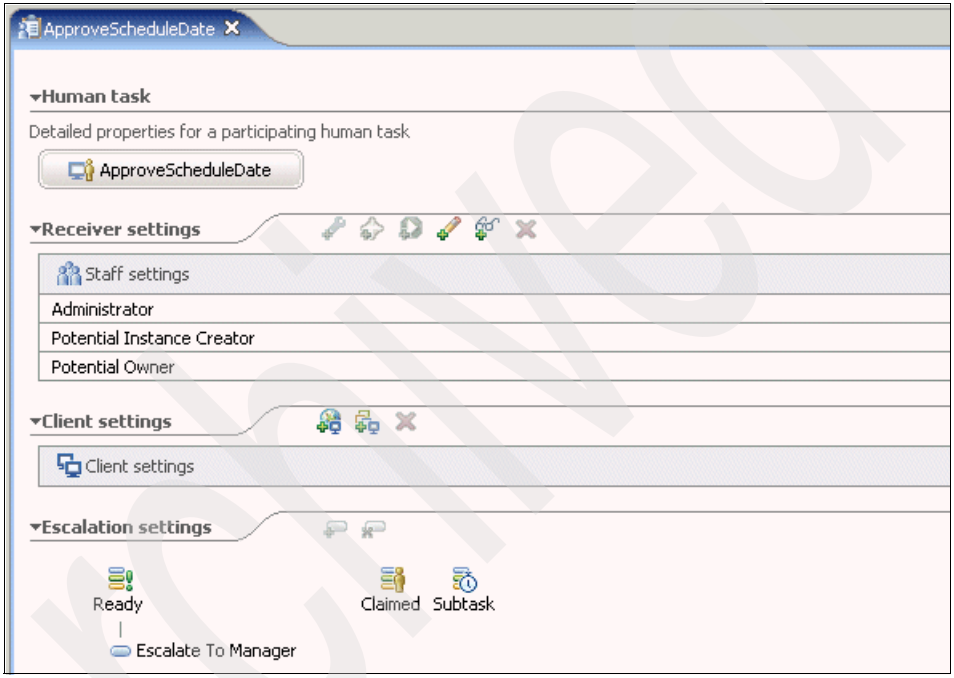


Figure 13-70 The ApproveScheduleDate human task

8. Press Ctrl+S to save the ApproveScheduleDate human task.
9. Close the ApproveScheduleDate Human Task Editor.

Wiring the PlaceOrderProcess reference partners

In the ITSOPModule assembly diagram editor, wire the PlaceOrderProcess reference partners to the following components:

- The Approve Schedule Date human task
- The Best Wholesaler business rule
- The Wholesaler A dynamicity selectors
- The Wholesaler B Web service imports

Complete these steps:

1. Right-click the **PlaceOrderProcess** component and click **Wire (Advanced)**.
2. In the Advanced Wiring window, complete the following steps (Figure 13-71):
   a. Ensure that the **Only show targets with matching interface types** check box is selected.
   b. In the Wire source list box, select the **ApproveSchedulePartner**. In the Wire target list box, click the **ApproveScheduleDate** check box.
   c. In the Wire source list box, select the **BestWholesalerPartner**. In the Wire target list box, click the **BestWholesalerRG** check box.
   d. In the Wire source list box, select the **CompanyACancelOrderPartner**. In the Wire target list box, click the **WholesalerCancelOrderSelector** check box.
   e. In the Wire source list box, select the **CompanyADeliveryDaysPartner**. In the Wire target list box, click the **WholesalerGetDateSelector** check box.
   f. In the Wire source list box, select the **CompanyAOrderPartner**. In the Wire target list box, click the **WholesalerPlaceOrderSelector** check box.
   g. In the Wire source list box, select the **CompanyBCancelOrderPartner**. In the Wire target list box, click the **WholesalerBCancelOrder** check box.
   h. In the Wire source list box, select the **CompanyBDeliveryDaysPartner**. In the Wire target list box, click the **WholesalerBGetDate** check box.
   i. In the Wire source list box, select the **CompanyBOrderPartner**. In the Wire target list box, click the **WholesalerBPlaceOrder** check box.
   j. Click **OK**.

Archived

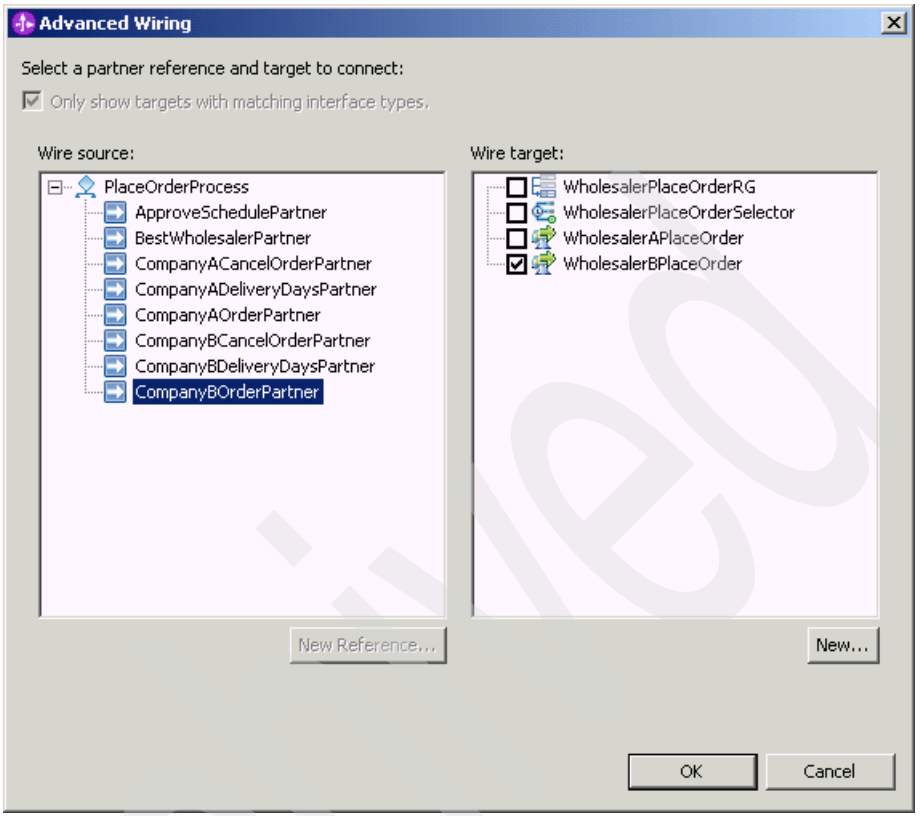


Figure 13-71 The PlaceOrderProcess Advanced Wiring window

3. Press Ctrl+S to save the assembly diagram.

Archived

Figure 13-72 shows the assembly diagram after wiring the components.

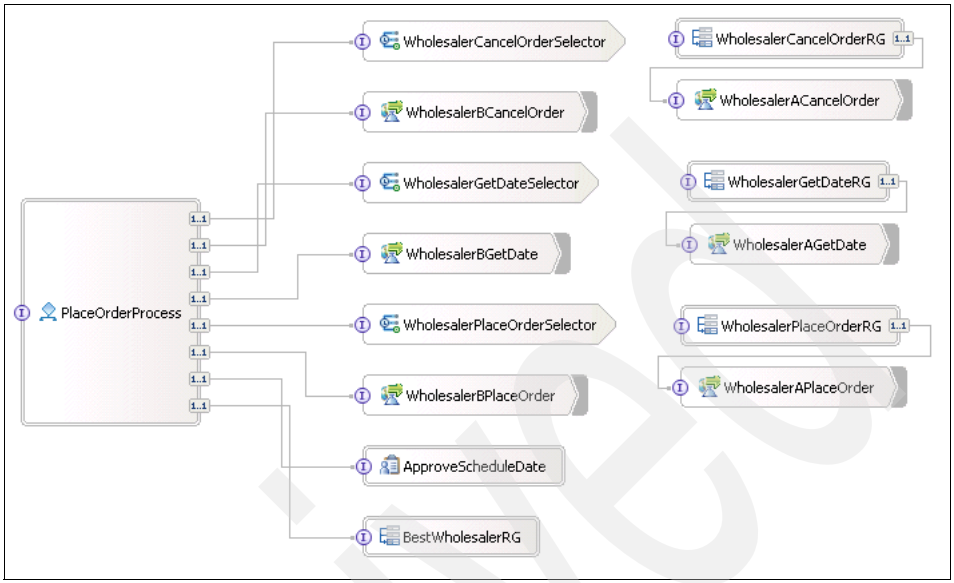


Figure 13-72 The ITSOPModule assembly diagram after wiring the components

13.4 Runtime guidelines

This section explains how to test the ITSO Electronics Parallel Workflow process. We show you how to test a number of features including:

- Fault handling
- Compensation
- Event handling
- Human tasks

See Appendix B, “Deploying and testing modules” on page 571, for a detailed description of the Integration Test Client.

13.4.1 Adding the required modules to the test server

In this section, we add the required modules to the WebSphere Integration Developer test server. The following modules are required to test the Parallel Workflow process:

- WholesalerA
- WholesalerB
- ITSOPModule (the Parallel Workflow version built in this chapter)

If the modules are not already installed on the test server, complete these steps:

1. Right-click the **WebSphere Process Server v6.0** server and click **Add and remove projects**.
2. Add the following modules to the WebSphere Integration Developer test server:
   - WholesalerAApp
   - WholesalerBApp
   - ITSOPModuleApp

   Click **Finish**.
3. Wait for the server to start and the applications to be installed.

If the modules are already installed on the test server, complete these steps:

1. Right-click the **WebSphere Process Server v6.0** server and click **Start**.
2. Wait for the server to start. Look for the `Server server1 open for e-business` message in the console.

13.4.2 Testing the Parallel Workflow process

To test the Parallel Workflow process using the Integration Test Client, follow these steps.

1. Open the ITSOPModule Assembly Diagram Editor.
2. Right-click the **PlaceOrderProcess** component and click **Test Component**. The Integration Test Client opens.
3. Click the **Configurations** tab.
   a. Remove the following emulators:
      - BestWholesalerRG
      - WholesalerCancelOrderRG
      - WholesalerCancelOrderSelector
      - WholesalerGetDateRG
      - WholesalerGetDateSelector
      - WholesalerPlaceOrderRG
      - WholesalerPlaceOrderSelector
      - WholesalerACancelOrder
      - WholesalerAGetDate
      - WholesalerAPlaceOrder
      - WholesalerBCancelOrder
      - WholesalerBGetDate
      - WholesalerBPlaceOrder

Archived

b. Ensure that the only remaining emulator is ApproveScheduleDate. We emulate this human task for now. Figure 13-73 shows the Integration Test Client configuration page after completing these steps.

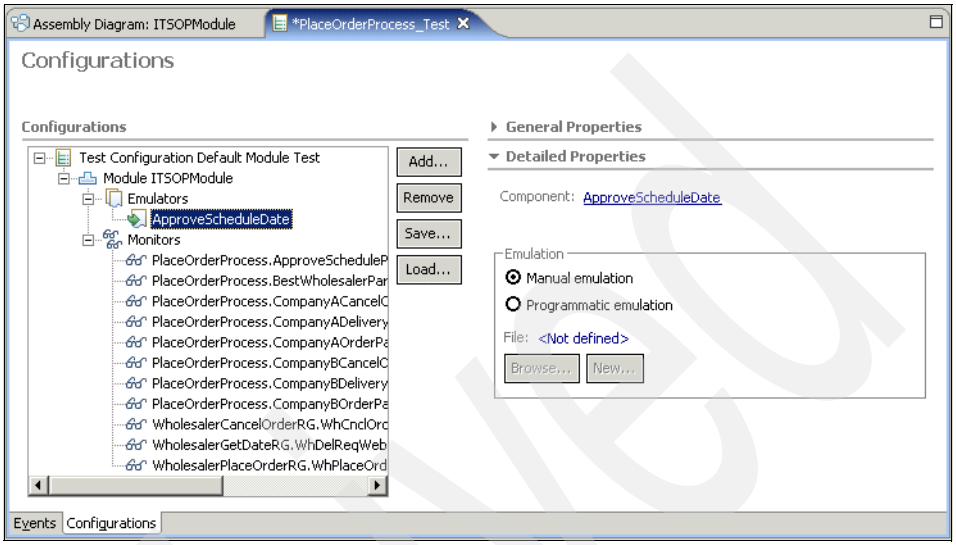


Figure 13-73 The PlaceOrderProcess Integration Test Client Configuration page

4. Click the **Events** tab.

   a. In the Module field, ensure that **ITSOPModule** is selected.
   b. In the Component field, ensure that **PlaceOrderProcess** is selected.
   c. In the Interface field, select the **OrderEntry** interface.
   d. In the Operation field, select the **createOrder** operation.
   e. In the Initial request parameters, enter the following values:

      i. In the orderId value field, type `123`.
      ii. In the partId value field, type `part1`.
      iii. In the quantity value field, type `3`.

      > **Note:** This input to the process successfully places the order with Wholesaler A.

   f. Click **Continue**.

5. In the Deployment Location window, click **Finish**.

Archived

6. If the User Login window opens (this only happens if you have security turned on), complete these steps:

   a. In the User ID field, type `wpsadmin`.
   b. In the password field, type the corresponding password.
   c. Click **OK**.

7. The process is now initiated. It pauses when it reaches the ApproveScheduleDate Human Task activity, which we decided to emulate. Figure 13-74 shows how the Integration Test Client events page looks now.

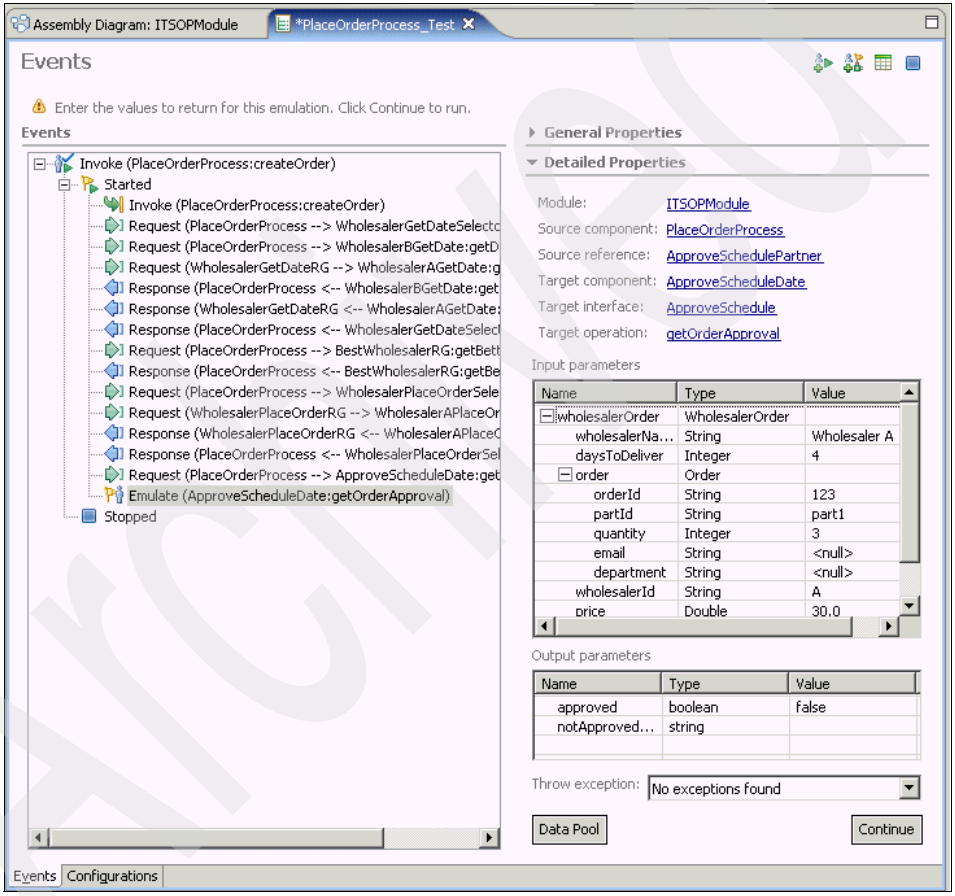


Figure 13-74 The PlaceOrderProcess Integration Test Client Events page

8. In the Output parameters approved value field, type `true`. Click **Continue**.

> **Note:** By entering *true*, the schedule date is approved, and the process finishes successfully.

The last part of the process is now executed. Example 13-8 shows how the server log looks when the process is finished.

Example 13-8 The PlaceOrderProcess server log

```
0000007b SystemOut     O Request for Part 123 acknowledged
...
...
...
00000062 ServletWrappe A   SRVE0242I: [WhBExpDelv_WhDelReqWeb_IHttpPort]: Initialization
successful.
00000063 ServletWrappe A   SRVE0242I:
[WholesalerA_GetDeliveryDateExport_WhDelReqWeb_IHttpPort]: Initialization successful.
00000062 SystemOut     O ****STARTING getDeliveryDays for WholesalerB****
00000062 SystemOut     O ****ENDING getDeliveryDays for WholesalerB****
...
...
...
00000077 ServletWrappe A   SRVE0242I: [WholesalerA_PlaceOrderExport_WhPlaceOrdWeb_IHttpPort]:
Initialization successful.
00000085 SystemOut     O The assembly date has been approved
```

13.4.3 Testing the Parallel Workflow compensation and fault handling

This section explains how to test compensation and fault handling in the Parallel Workflow process using the Integration Test Client.

1. Open the ITSOPModule Assembly Diagram Editor.
2. Right-click the **PlaceOrderProcess component** and click **Test Component**. The Integration Test Client opens.
3. Click the **Configurations** tab.
   a. Remove the following emulators:
      - BestWholesalerRG
      - WholesalerCancelOrderRG
      - WholesalerCancelOrderSelector
      - WholesalerGetDateRG
      - WholesalerGetDateSelector
      - WholesalerPlaceOrderRG
      - WholesalerPlaceOrderSelector
      - WholesalerACancelOrder
      - WholesalerAGetDate

Archived

- WholesalerAPlaceOrder
- WholesalerBCancelOrder
- WholesalerBGetDate
- WholesalerBPlaceOrder

b. Ensure that the only remaining emulator is ApproveScheduleDate.

4. Click the **Events** tab.

   a. In the Module field, ensure that **ITSOPModule** is selected.
   b. In the Component field, ensure that **PlaceOrderProcess** is selected.
   c. In the Interface field, select the **OrderEntry** interface.
   d. In the Operation field, select the **createOrder** operation.
   e. In the Initial request parameters, enter the following values:

      i. In the orderId value field, type `124`.
      ii. In the partId value field, type `part2`.
      iii. In the quantity value field, type `5`.

      Note: This input to the process places the order with Wholesaler B.

   f. Click **Continue**.

5. If the Deployment Location window opens, click **Finish**.

6. If the User Login window opens, complete the following steps:

   a. In the User ID field, type `wpsadmin`.
   b. In the password field, type the corresponding password.
   c. Click **OK**.

7. The process is now initiated. The process pauses when it reaches the ApproveScheduleDate Human Task activity, which we decided to emulate.

8. In the Output parameters approved value field, type `false`. Click **Continue**.

   Note: By entering *false*, the schedule date is declined.

Archived

Figure 13-75 shows how the Integration Test Client Events page Events section looks now.

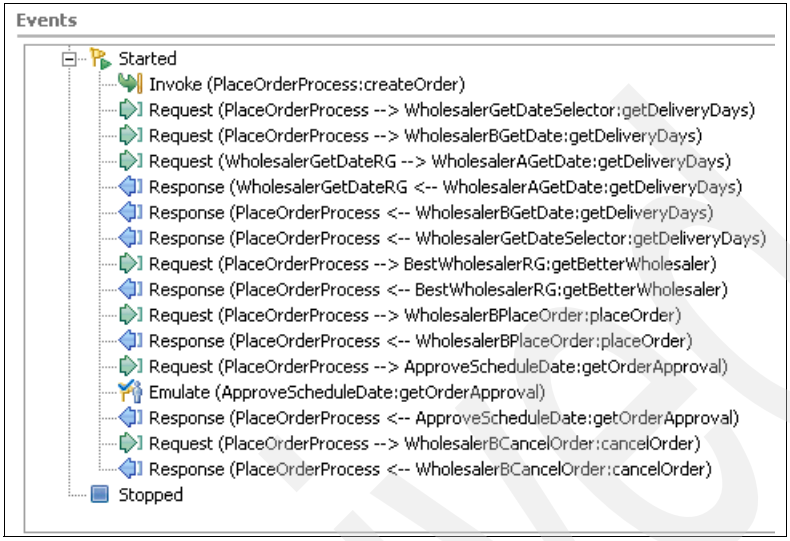


Figure 13-75 The PlaceOrderProcess Integration Test Client Events page Events section: Compensation and fault handling

9. The last part of the process is now executed. A NotApprovedFault is thrown, caught and rethrown, and then caught again. Compensation is done and the process is terminated. When the process is terminated, the server log contains the details shown in Example 13-9.

Example 13-9 The PlaceOrderProcess server log: compensation and fault handling

```
0000007b SystemOut     O Request for Part 124 acknowledged
...
...
00000063 SystemOut     O ****STARTING getDeliveryDays for WholesalerB****
00000063 SystemOut     O ****ENDING getDeliveryDays for WholesalerB****
...
...
00000077 ServletWrappe A   SRVE0242I: [WhBExpPlaceOrd_WhPlaceOrdWeb_IHttpPort]: Initialization
successful.
00000077 SystemOut     O ****STARTING placeOrder for WholesalerB****
00000077 SystemOut     O ****ENDING placeOrder for WholesalerB****
00000080 SystemOut     O The assembly date has not been approved. Throw NotApprovedFault.
0000007e SystemOut     O We caught a fault! Lets rethrow it...
00000085 SystemOut     O We caught a fault! Lets compensate and terminate...
00000085 SystemOut     O Request for Part 124 cancelled
0000007e SystemOut     O Compensating PlaceOrderB
```

```
00000077 ServletWrappe A   SRVE0242I: [WhBExpCnclOrd_WhCnclOrdWeb_IHttpPort]: Initialization
successful.
00000077 SystemOut     O ****STARTING cancelOrder for WholesalerB****
00000077 SystemOut     O ****ENDING cancelOrder for WholesalerB****
```

13.4.4 Testing the Parallel Workflow event handling

To test event handling in the Parallel Workflow process using the Integration Test Client, follow these steps.

1. Open the ITSOPModule Assembly Diagram Editor.
2. Right-click the **PlaceOrderProcess** component and click **Test Component**. The Integration Test Client opens.
3. Click the **Configurations** tab.
   a. Remove the following emulators:
      - BestWholesalerRG
      - WholesalerCancelOrderRG
      - WholesalerCancelOrderSelector
      - WholesalerGetDateRG
      - WholesalerGetDateSelector
      - WholesalerPlaceOrderRG
      - WholesalerPlaceOrderSelector
      - WholesalerACancelOrder
      - WholesalerAGetDate
      - WholesalerAPlaceOrder
      - WholesalerBCancelOrder
      - WholesalerBGetDate
      - WholesalerBPlaceOrder

   b. Ensure that the only remaining emulator is ApproveScheduleDate.
4. Click the **Events** tab.
   a. In the Module field, ensure that **ITSOPModule** is selected.
   b. In the Component field, ensure that **PlaceOrderProcess** is selected.
   c. In the Interface field, select the **OrderEntry** interface.
   d. In the Operation field, select the **createOrder** operation.

Archived

e. In the Initial request parameters, enter the following values:

   i. In the orderId value field, type `125`.
   ii. In the partId value field, type `part3`.
   iii. In the quantity value field, type `7`.

   Note: This input to the process places the order with Wholesaler B.

f. Click **Continue**.

5. If the Deployment Location window opens, click **Finish**.
6. If the User Login window opens, complete the following steps:

   a. In the User ID field, type `wpsadmin`.
   b. In the password field, type the corresponding password.
   c. Click **OK**.

7. The process is now initiated. The process pauses when it reaches the ApproveScheduleDate Human Task activity, which we decided to emulate.
8. Click the **Invoke** icon ( ).
9. Complete the following steps:

   a. In the Module field, ensure that **ITSOPModule** is selected.
   b. In the Component field, ensure that **PlaceOrderProcess** is selected.
   c. In the Interface field, select the **CancelOrder** interface.
   d. In the Operation field, select the **cancelOrder** operation.
   e. In the Initial request parameters, in the orderId value field, type `125`.
   f. Click **Continue**.

Archived

Figure 13-76 shows how the Integration Test Client Events page Events section looks after completing these steps.

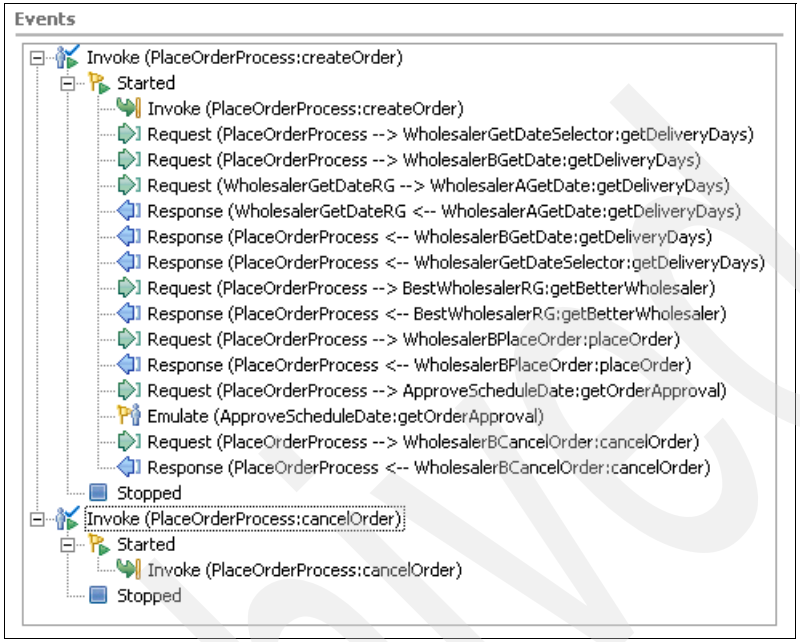


Figure 13-76 The PlaceOrderProcess Integration Test Client Events page Events section: Event handling

10. The process event handler receives the cancelOrder event. A n OrderCancelledFault is thrown and is caught in the fault handler. Compensation is done, and the process is terminated. When the process is terminated, the server log contains the output shown in Example 13-10.

Example 13-10 The PlaceOrderProcess server log: Event handling

```
0000007e SystemOut     O Request for Part 125 acknowledged
...
...
00000077 SystemOut     O ****STARTING getDeliveryDays for WholesalerB****
00000077 SystemOut     O ****ENDING getDeliveryDays for WholesalerB****
...
...
00000077 SystemOut     O ****STARTING placeOrder for WholesalerB****
00000077 SystemOut     O ****ENDING placeOrder for WholesalerB****
...
...
00000080 SystemOut     O We got an event
00000085 SystemOut     O We caught a fault! Lets compensate and terminate...
00000085 SystemOut     O Request for Part 125 cancelled
```

```
0000007e SystemOut     O Compensating PlaceOrderB
00000077 SystemOut     O ****STARTING cancelOrder for WholesalerB****
00000077 SystemOut     O ****ENDING cancelOrder for WholesalerB****
```

13.4.5 Testing Parallel Workflow ApproveScheduleDate Human Task

This section explains how to test the Human Task in the Parallel Workflow process using the Integration Test Client and the test server BPC Explorer. For more information, see “The Business Process Container Explorer” on page 586.

Important: You must ensure that security has been set up on the WebSphere Process Server test server before you test the ApproveScheduleDate Human Task. For instructions to help you configure security on the test server, see “Security” on page 603. The following steps are required to set up security on the WebSphere Process Server test server:

1. “Setting up LocalOS security on WebSphere Process Server” on page 604
2. “Configuring the BPE Container” on page 611
3. “Configuring the Task Container” on page 614
4. “Configuring the WebSphere Integration Developer test server” on page 616
5. “Redeploying the file transfer application” on page 618

1. Open the ITSOPModule Assembly Diagram Editor.
2. Right-click the **PlaceOrderProcess** component and click **Test Component**.
3. In the Test Client that opens, click the **Configurations** tab. Remove all of emulators, which are:
   - ApproveScheduleDate
   - BestWholesalerRG
   - WholesalerCancelOrderRG
   - WholesalerCancelOrderSelector
   - WholesalerGetDateRG
   - WholesalerGetDateSelector
   - WholesalerPlaceOrderRG
   - WholesalerPlaceOrderSelector
   - WholesalerACancelOrder
   - WholesalerAGetDate
   - WholesalerAPlaceOrder
   - WholesalerBCancelOrder
   - WholesalerBGetDate
   - WholesalerBPlaceOrder

4. Click the **Events** tab.
   a. In the Module field, ensure that **ITSOPModule** is selected.
   b. In the Component field, ensure that **PlaceOrderProcess** is selected.
   c. In the Interface field, select the **OrderEntry** interface.
   d. In the Operation field, select the **createOrder** operation.
   e. In the Initial request parameters, enter the following values:
      i. In the orderId value field, type `128`.
      ii. In the partId value field, type `part4`.
      iii. In the quantity value field, type `9`.

      Note: This input to the process places the order with Wholesaler B.

   f. Click **Continue**.
5. If the Deployment Location window opens, click **Finish**.
6. If the User Login window opens, complete the following steps:
   a. In the User ID field, type `wpsadmin`.
   b. In the password field, type the corresponding password.
   c. Click **OK**.
7. The process is now initiated. It pauses when it reaches the ApproveScheduleDate Human Task activity.
8. Click the **Servers** page.
9. Right-click the **WebSphere Process Server v6.0** server and click **Launch** → **BPC Explorer**.
10. In the Security Alert window, click **Yes**.
11. Login to the BPC Explorer using the `clerk` user ID and password.
12. Click **My Tasks**.

Archived

13. The My Tasks panel (Figure 13-77) is displayed on the right and shows the task instances assigned to the clerk.

   a. Click the **check box** to the left of the task.
   b. Click the **Work on** button to claim the ApproveScheduleDate Human Task.

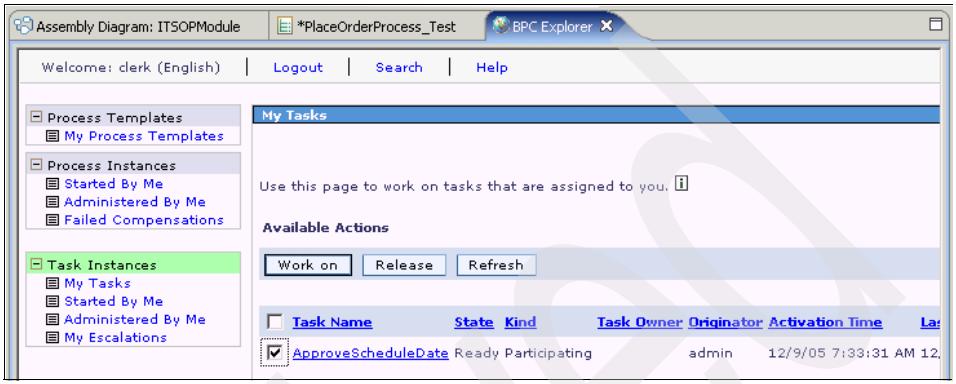


Figure 13-77 BPC Explorer My Tasks: Clerk

14. In the Task Message window (Figure 13-78), in the approved field, type `true`. Click **Complete**.

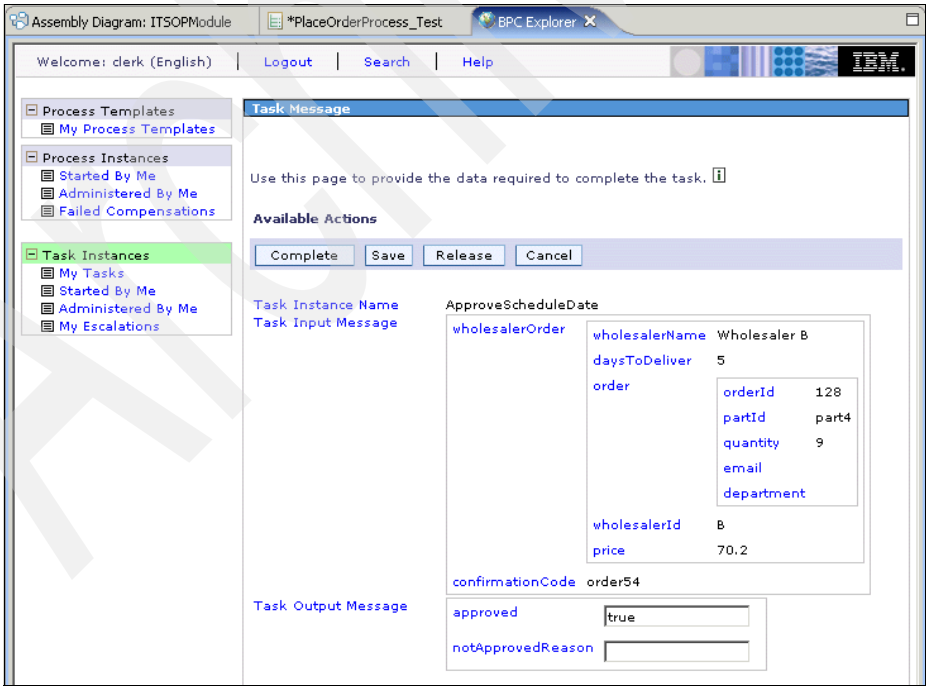


Figure 13-78 BPC Explorer Task Message: Clerk

The last part of the process is now executed. Figure 13-79 shows how the Test Client Events page Events section looks now.

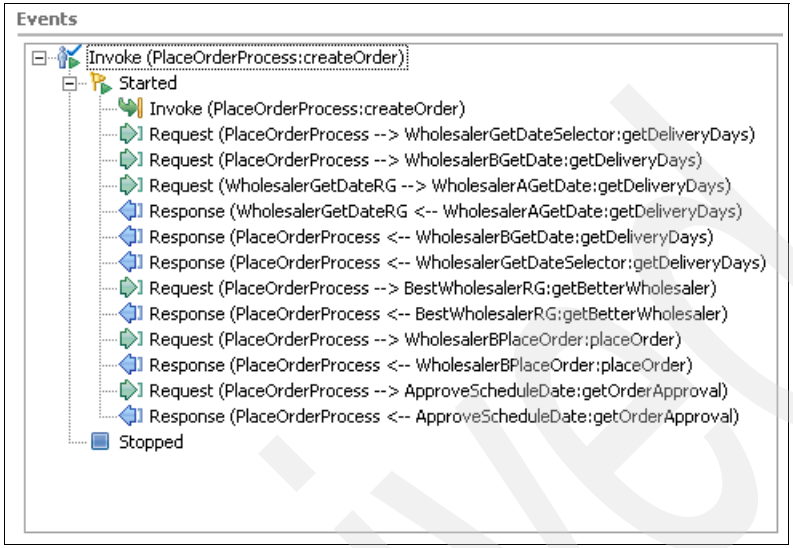


Figure 13-79 The PlaceOrderProcess Test Client Events page Events section: Human Task

When the process is finished, the server log shows the contents as displayed in Example 13-11.

Example 13-11 The PlaceOrderProcess server log: Human Task test

```
000000a5 SystemOut     O Request for Part 128 acknowledged
...
...
00000072 ServletWrappe A   SRVE0242I:
[WholesalerA_GetDeliveryDateExport_WhDelReqWeb_IHttpPort]: Initialization successful.
00000073 ServletWrappe A   SRVE0242I: [WhBExpDelv_WhDelReqWeb_IHttpPort]: Initialization
successful.
00000073 SystemOut     O ****STARTING getDeliveryDays for WholesalerB****
00000073 SystemOut     O ****ENDING getDeliveryDays for WholesalerB****
...
...
00000072 ServletWrappe A   SRVE0242I: [WhBExpPlaceOrd_WhPlaceOrdWeb_IHttpPort]: Initialization
successful.
00000072 SystemOut     O ****STARTING placeOrder for WholesalerB****
00000072 SystemOut     O ****ENDING placeOrder for WholesalerB****
...
...
00000072 ServletWrappe A   SRVE0242I: [/index.jsp]: Initialization successful.
00000073 ServletWrappe A   SRVE0242I: [/pages/Login.jsp]: Initialization successful.
```

```
000000a3 ServletWrappe A   SRVE0242I: [/pages/TaskInstanceListView.jsp]: Initialization
successful.
000000a3 ServletWrappe A   SRVE0242I: [/pages/layouts/Menubar.jsp]: Initialization successful.
000000a3 ServletWrappe A   SRVE0242I: [/pages/layouts/Navigator.jsp]: Initialization
successful.
000000a3 ServletWrappe A   SRVE0242I: [/pages/layouts/lists/TaskInstanceList.jsp]:
Initialization successful.
...
...
00000073 ServletWrappe A   SRVE0242I: [/pages/TaskMessageView.jsp]: Initialization successful.
00000073 ServletWrappe A   SRVE0242I: [/pages/layouts/TaskMessage.jsp]: Initialization
successful.
000000aa SystemOut     O The assembly date has been approved
```

Escalation of the ApproveScheduleDate Human Task

The ApproveScheduleDate Human Task is escalated to the manager if the clerk has not claimed the task within 2 minutes. The escalated task is in both the manager's My Tasks and My Escalations window (Figure 13-80). To see them, log in to the BPC Explorer using the `manager` user ID.

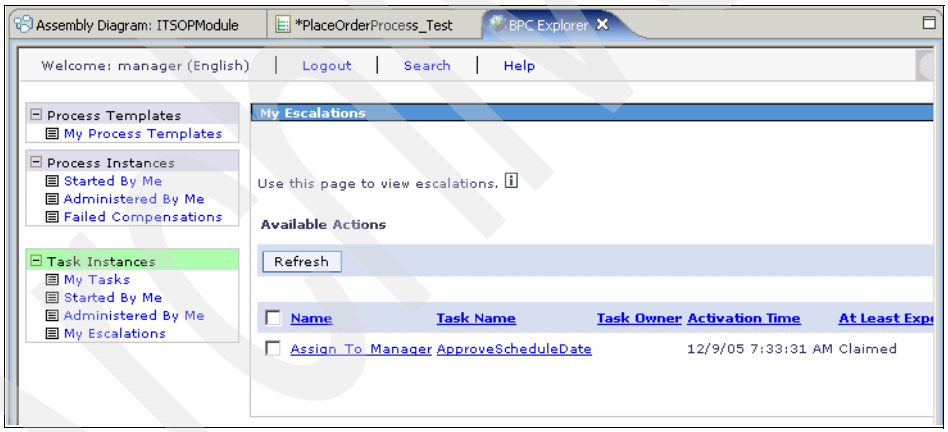


Figure 13-80 BPC Explorer My Escalations: Manager

Archived

Part 4

Appendixes

Archived

© Copyright IBM Corp. 2006. All rights reserved.

Archived

A

Building Wholesaler A and B service implementations

In this appendix, we explain how to build the service implementations for the Wholesaler A and Wholesaler B applications that are used through the scenario implementations in Part 3 of this IBM Redbook. This appendix is also a good starting point for those of you who are new to WebSphere Integration Developer and WebSphere Process Server. It provides step-by-step instructions for you to perform the following tasks:

- Create business objects and interfaces.
- Create Web Services Business Process Execution Language (WS-BPEL) processes.
- Create business rules.
- Expose the created services as a Web service.
- Build a service entirely in Java and expose it as a Web service.

If you do not want to build the Wholesaler implementations, you can import project interchange files that contain complete solutions into your workspace. You can find these project interchange files in the additional material supplied with this book in the \Wholesaler\WholesalerA and \Wholesaler\WholesalerB directories. For details about accessing the additional materials, see Appendix E, “Additional material” on page 689.

© Copyright IBM Corp. 2006. All rights reserved.

Scenario implementation overview

In this appendix, we build Web service implementations for Wholesaler A and Wholesaler B. These Web service implementations are used by the ITSO Electronics organization in Part 3 of this book.

Both Wholesaler A and Wholesaler B implement three business functions:

- Get Delivery Days
- Place Order
- Cancel Order

In Wholesaler A, this business logic is implemented in WS-BPEL processes and a business rule. In Wholesaler B, this logic is implemented in a Java class. The business logic in both wholesalers is exposed with a SOAP over HTTP (SOAP/HTTP) Web services interface. ITSO Electronics acts as a Web services consumer, sending SOAP/HTTP messages to communicate with the wholesalers.

Archived

In this appendix, we build the business logic for Wholesaler A and Wholesaler B, and then expose this business logic with a Web services interface. This is shown in Figure A-1.

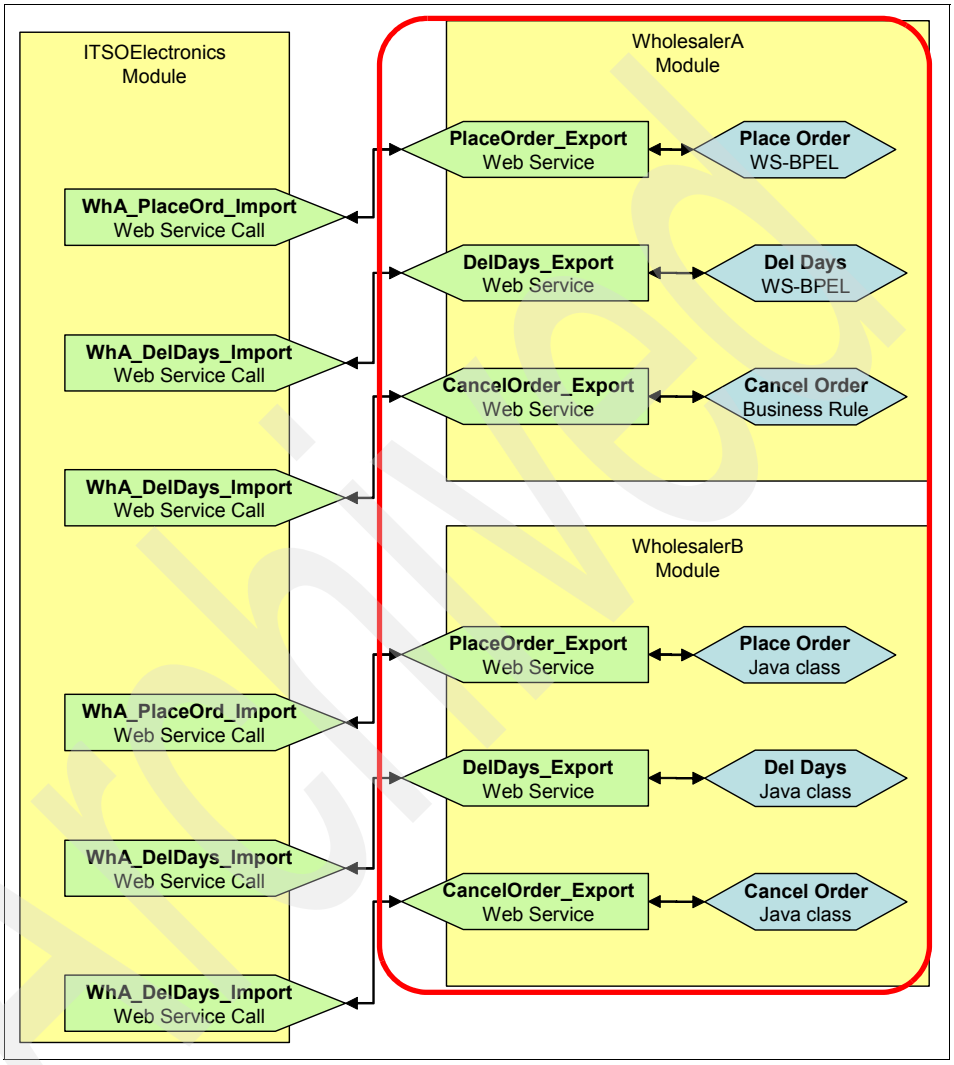


Figure A-1 Implementation of Wholesaler A and Wholesaler B

Archived

Creating the service implementation for Wholesaler A

We implement the business logic of Wholesaler A in this section. The business logic is exposed as a Web service. The Web service is the interface between Wholesaler A and ITSO Electronics. In a real life environment, Wholesaler A is a sophisticated application. In this scenario, we focus on the Web service capabilities of WebSphere Process Server as a Web service provider and consumer.

Wholesaler A implements three business functionalities:

- Get Delivery Days (implemented as a business process)
- Place Order (implemented as a business process)
- Cancel Order (implemented as a business rule)

Therefore Wholesaler A has three interfaces:

- Get delivery days
- Place an order
- Cancel an order

Each of these interfaces consists of an operation, an input and output, and a fault, as shown in Table A-1.

Table A-1 Wholesaler A interfaces

	Get delivery days	**Place order**	**Cancel order**
Name	WhDelReqWeb_I	WhPlaceOrdWeb_I	WhCnclOrdWeb_I
Operation	getDeliveryDays	placeOrder	cancelOrder
Input	getDeliveryDays WholesalerRequest	placeOrderRequest	cancelOrderRequest
Output	getDeliveryDays WholesalerResponse	PlaceOrderResponse	cancelOrder Response
Fault	cannotFulfillOrder	cannotFulfillOrder	N/A

Note: We recommend that you keep the interface names short, because Web Services Description Language (WSDL) files are created based on these interface names.

Each of the inputs, outputs, and faults are represented by a simple data type or a business object type. Before we can create the interfaces, we must create the business objects.

Table A-2, Table A-3, and Table A-4 list the business objects that are used in the interfaces listed in Table A-1.

Table A-2 Business objects of the get delivery interface WhDelReqWeb_I

	Name	Business object
Input	getDeliveryDaysWholesalerRequest	WholesalerDeliveryDaysRequest
Output	getDeliveryDaysWholesaler Response	WholesalerDeliveryDaysResponse
Fault	cannotFulfillOrder	WholesalerCannotFulfillOrder

Table A-3 Business objects of the place order interface WhPlaceOrdWeb_I

	Name	Business object
Input	placeOrderRequest	WholesalerPlaceOrderRequest
Output	PlaceOrderResponse	WholesalerPlaceOrderResponse
Fault	cannotFulfillOrder	WholesalerCannotFulfillOrder

Table A-4 Business objects of the cancel order interface WhCnclOrdWeb_I

	Name	Business object
Input	cancelOrderRequest	WholesalerCancelOrderRequest
Output	cancelOrderResponse	WholesalerCancelOrderResponse

Note: Although these interfaces and business objects are being created for Wholesaler A, the names do not include the text “Wholesaler A”. This is because these interfaces and business objects will be reused for other wholesalers as well.

Archived

Creating business objects for the WhDelReqWeb_I interface

Now we build the business objects specified in Table A-2. All Service Component Architecture (SCA) components communicate with each other using business objects. A *business object* is a data type that contains one or more simple or complex data types or arguments, similar to other business objects. A business object often represents a business entity such as an order.

1. Start a new workspace in WebSphere Integration Developer V6.0.1 or later.
2. In the Business Integration perspective, create the WholesalerA module project.
   a. Select **File** → **New** → **Project**.
   b. In the New Project window (Figure A-2), select **Module** and click **Next**.

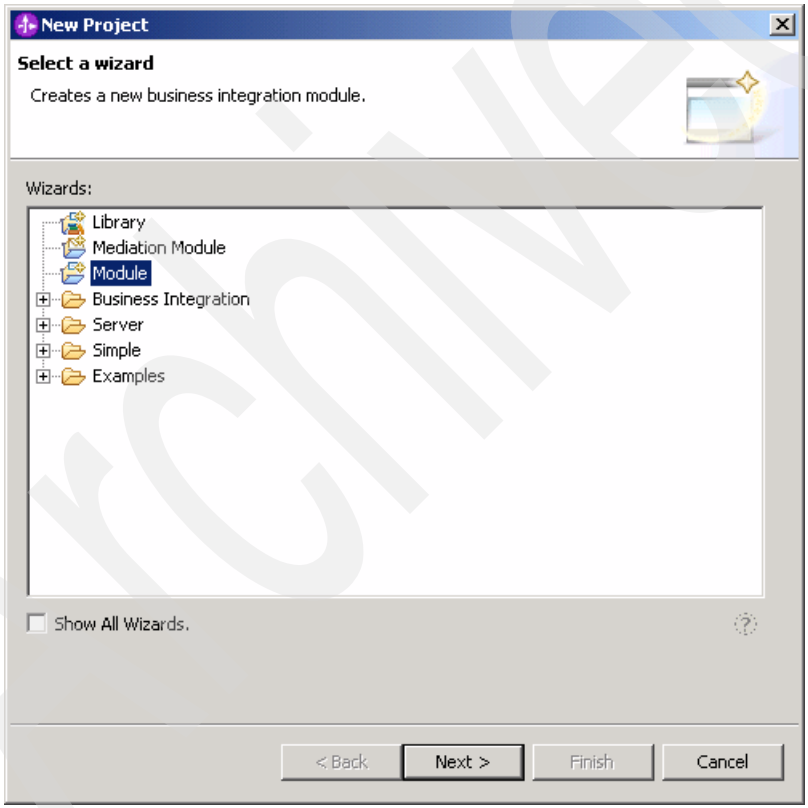


Figure A-2 New Project window

   c. In the next window, in the Module Name field, type `WholesalerA`. For Module Location Directory, select **Use default** and click **Finish**.

Archived

3. Create the business object *WholesalerDeliveryDaysRequest* in the module WholesalerA. The business object is sent to Wholesaler A through an interface and contains the request message (two arguments: partNo and quantity) from ITSO Electronics. Wholesaler A processes the business object in its business logic and generates another business object that contains the response message for ITSO Electronics.

   a. In the Business Integration view, expand the **WholesalerA** module project (Figure A-3).

Figure A-3 Expanding the WholesalerA module project

   b. Right-click **Data Types** and select **New** → **Business Object**.

Archived

c. In the New Business Object window (Figure A-4), complete these steps:

i. Deselect the **Default** check box.

ii. Change the Namespace field from `http://WholesalerA` to `http://Wholesaler`.

> **Note:** We changed the Namespace field to hold the business objects for all wholesalers in a common namespace. We do not place it in a library project because each wholesaler is a self-contained entity.

iii. In the Name field, type `WholesalerDeliveryDaysRequest`.

iv. Click **Finish**.

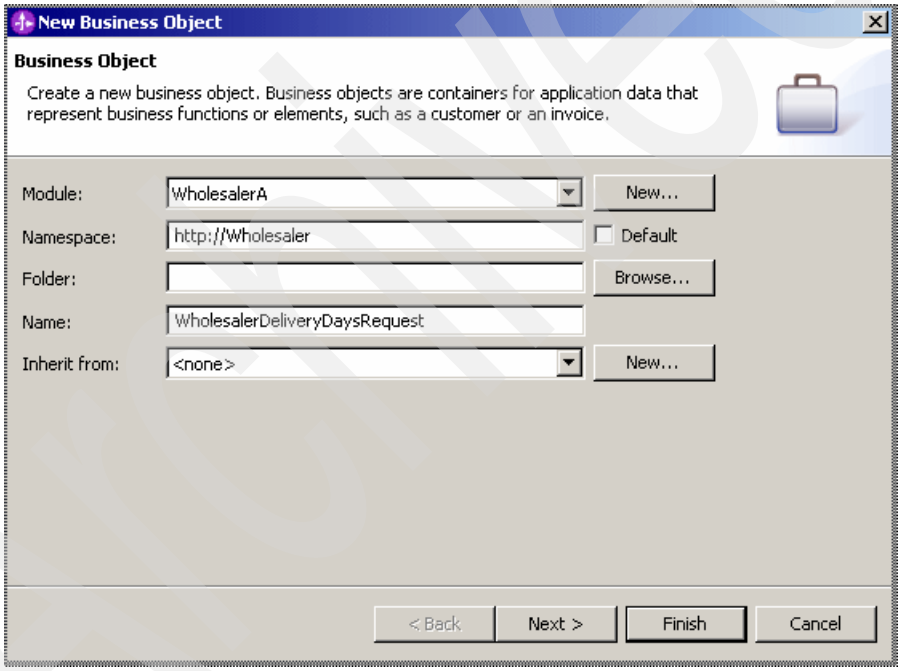


Figure A-4 Creating the WholesalerDeliveryDaysRequest business object

d. The Business Object Editor appears (Figure A-5). Click the icon to add a new attribute and rename it to `partNo`.

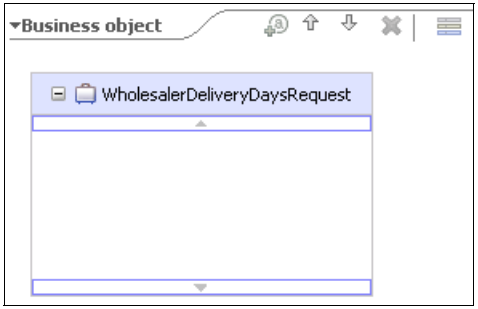


Figure A-5 The Business Object Editor

e. Set the data type as **string** (Figure A-6). This should be the default.

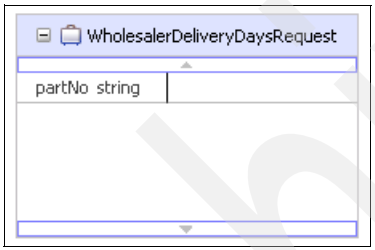


Figure A-6 Attribute partNo

f. Click the **Add attribute** ( ) icon to add another attribute and name it `quantity`.

Archived

g. Change the data type to *int* by clicking **string**. In the pop-up menu that is displayed (Figure A-7), select **int**.

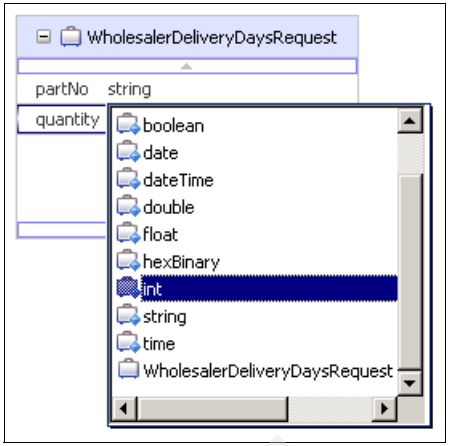


Figure A-7 Selecting the data type

Make sure that the completed business object looks like the example shown in Figure A-8.

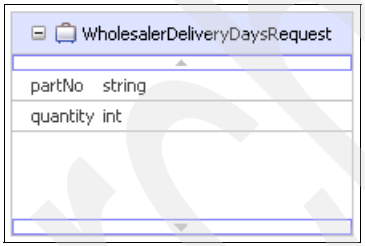


Figure A-8 Completed WholesalerDeliverDaysRequest business object

h. Press Ctrl+S to save the business object and close the Business Object Editor.

4. Create the business object *WholesalerDeliveryDaysResponse* in the module WholesalerA.

a. Name the business object `WholesalerDeliveryDaysResponse`.

> **Important:** Remember to check that the namespace for this and all Wholesaler A business objects is `http://Wholesaler`.

b. Add the attribute `daysToDeliver` of type `int`.

Archived

c. Add the attribute `price` of type `double`.

   Make sure the completed business object looks like the example in Figure A-9.

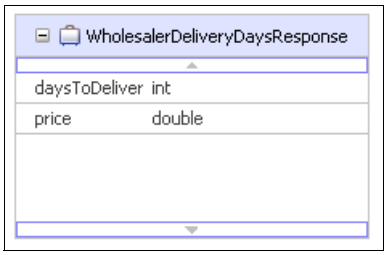


Figure A-9 Completed WholesalerDeliverDaysResponse business object

d. Press Ctrl+S to save the business object and close the Business Object Editor. Your Business Integration view should now look like the example in Figure A-10.

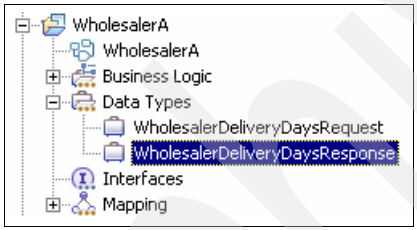


Figure A-10 Business Integration view showing first two business objects

5. In the Business Integration view, click the **Show Namespace** icon ( ) to check the business object namespaces. Because namespaces are important, WebSphere Integration Developer provides a facility to show the namespaces of artefacts. You should see a structure like the one shown in Figure A-11.

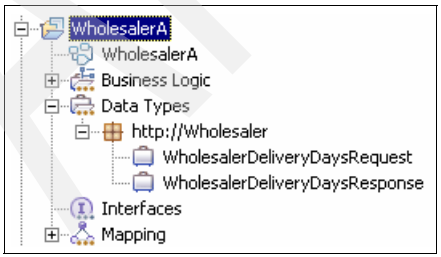


Figure A-11 Namespace view

Archived

6. Create the business object *WholesalerCannotFulfillOrder* in the module WholesalerA.

   a. Name the business object `WholesalerCannotFulfillOrder`.
   b. Set the namespace to `http://Wholesaler`.
   c. Add the attribute `message` of type `string`.
   d. Add the attribute `errorCode` of type `int`.

      Make sure the completed business object looks like the example in Figure A-12.

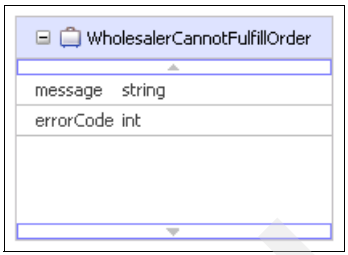


Figure A-12 The completed WholesalerCannotFulfillOrder business object

   e. Press Ctrl+S to save the business object and close the Business Object Editor.

We have now created the business objects, so we can create the interfaces that use the business objects.

Creating the WhDelReqWeb_I interface

In this section, we explain how to build the WhDelReqWeb_I interface (as specified in Table A-2 on page 505) using the business objects:

- WholesalerDeliverDaysRequest
- WholesalerDeliverDaysResponse
- WholesalerCannotFulfillOrder

An *interface* defines operations that contain an input and optionally an output data type. Each SCA component needs at least one interface to communicate to other SCA components. If the interfaces of SCA components are different, additionally an interface map is required.

The interface receives, as input, a request message containing a part number and quantity of the part. ITSO Electronics is looking for, and will response with, a message that contains the amount of delivery days and price of the part.

Archived

Create the *WhDelReqWeb_I* interface in the module WholesalerA.

1. In the Business Integration view, right-click **Interfaces** and select **New** → **Interface**.
2. In the New Interface Wizard window (Figure A-13), complete these steps:
   a. Deselect the **Default** check box.
   b. Set the Namespace field to `http://Wholesaler/WhDelReqWeb_I`.
   c. In the Name field, type `WhDelReqWeb_I` and click **Finish**.

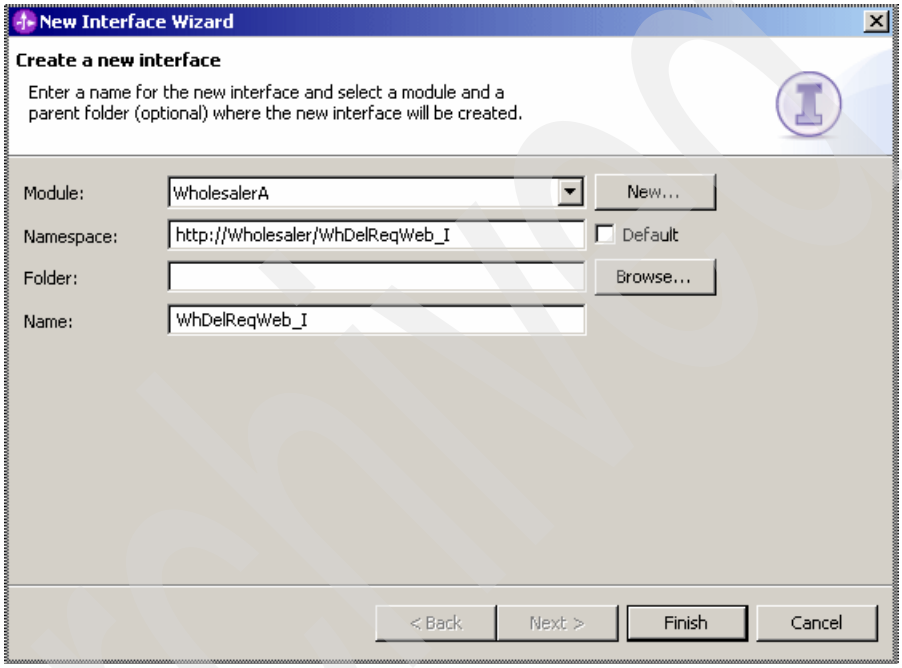


Figure A-13 Creating the WhDelReWeb_I interface

3. In the Interface Editor (Figure A-14), click the **Add Request Response Operation** ( ) icon to add an operation, and name the operation `getDeliveryDays`.

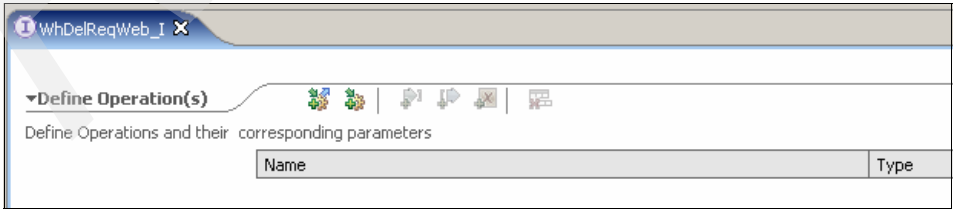


Figure A-14 The Interface Editor

4. Now the getDeliveryDays operation is displayed under Define Operations as shown in Figure A-15. Click **getDeliveryDays**.

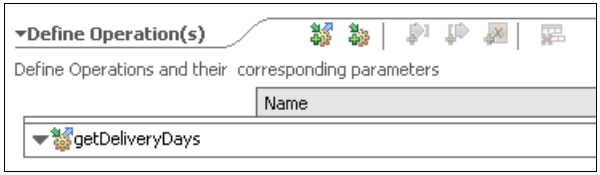


Figure A-15 getDeliveryDays operation

5. The Add Input ( ), Add Output ( ) and Add Fault ( ) icons are displayed. Click each icon to add an input, an output and a fault operation to the Interface Editor (Figure A-16).

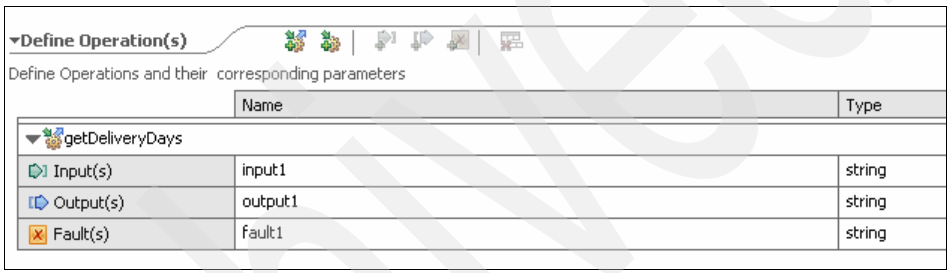


Figure A-16 Interface Editor with input, output, and fault operations

6. In the Name column of the Input(s) row, click **input1**, and rename it to `getDeliveryDaysWholesalerRequest`.
7. In the Name column of the Output(s) row, click **output1**, and rename it to `getDeliveryDaysWholesalerResponse`.
8. In the Name column of the Fault(s) row, click **fault1**, and rename it to `cannotFulfillOrder`. Figure A-17 shows how the Interface Editor looks now.

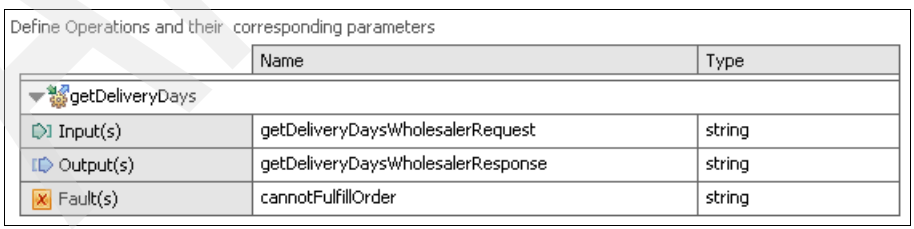


Figure A-17 Interface Editor with a renamed input, output, and fault

Archived

9. In the Type column of the Input(s) row, click **string** to change the type of the input data type to `WholesalerDeliveryDaysRequest`, as displayed in Figure A-18.

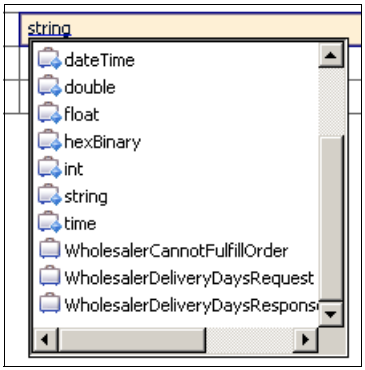


Figure A-18 Changing the type to wholesalerDeliverDaysRequest

10. In the Type column of the Output(s) row, click **string** and change the output data type to `WholesalerDeliveryDaysResponse`.
11. In the Type column of the Fault(s) row, click **string** and change the fault data type to `WholesalerCannotFulfillOrder`.

    Figure A-19 shows how the Interface Editor looks after making these changes.

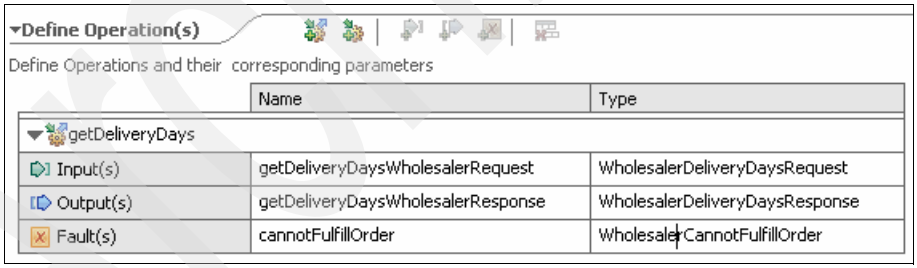


Figure A-19 The completed WhDelReqWeb_I interface

12. Press Ctrl+S to save the interface, and then close the Interface Editor.

We have now created the first interface for the Get Delivery Days functionality. We continue to create business objects and interfaces for Place Order and Cancel Order functionalities in the sections that follow.

Archived

Creating the business objects for the remaining interfaces

In this section, we build the required business objects for the interfaces:

- WhPlaceOrdWeb_I
- WhCnclOrdWeb_I

The interfaces and business objects are used to place and cancel an order with Wholesaler A and respond to ITSO Electronics with a confirm message.

In the WholesalerA module, create four business objects:

> **Important:** Remember to change the namespace for the business objects and all Wholesaler A business objects from http://WholesalerA to http://Wholesaler.

1. Create the business object *WholesalerPlaceOrderRequest* (Figure A-20).

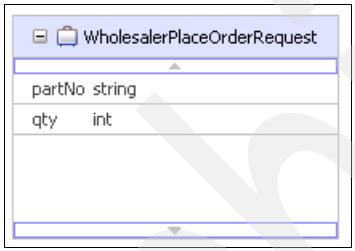


Figure A-20 WholesalerPlaceOrderRequest business object

2. Create the business object *WholesalerPlaceOrderResponse* (Figure A-21).

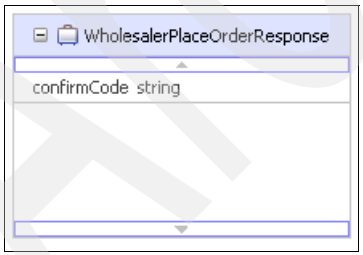


Figure A-21 WholesalerPlaceOrderResponse business object

Archived

3. Create the business object *WholesalerCancelOrderRequest* (Figure A-22).

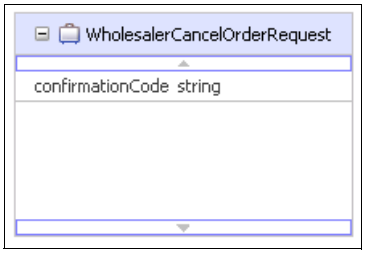


Figure A-22 WholesalerCancelOrderRequest business object

4. Create the business object *WholesalerCancelOrderResponse* (Figure A-23).

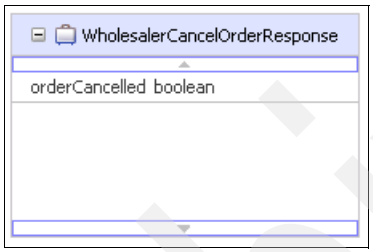


Figure A-23 WholesalerCancelOrderResponse business object

Figure A-24 shows how your Business Integration view should look. Notice that we show the namespace view to confirm that the business object namespaces are all the same.

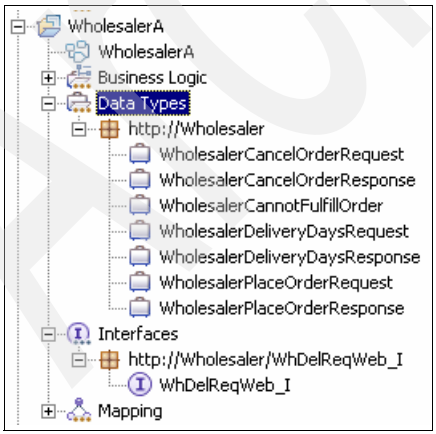


Figure A-24 Business Integration view with all business objects

Archived

Tip: You can change the namespace of a business object by double-clicking its name and clicking the Properties tab and Description tab (Figure A-25).

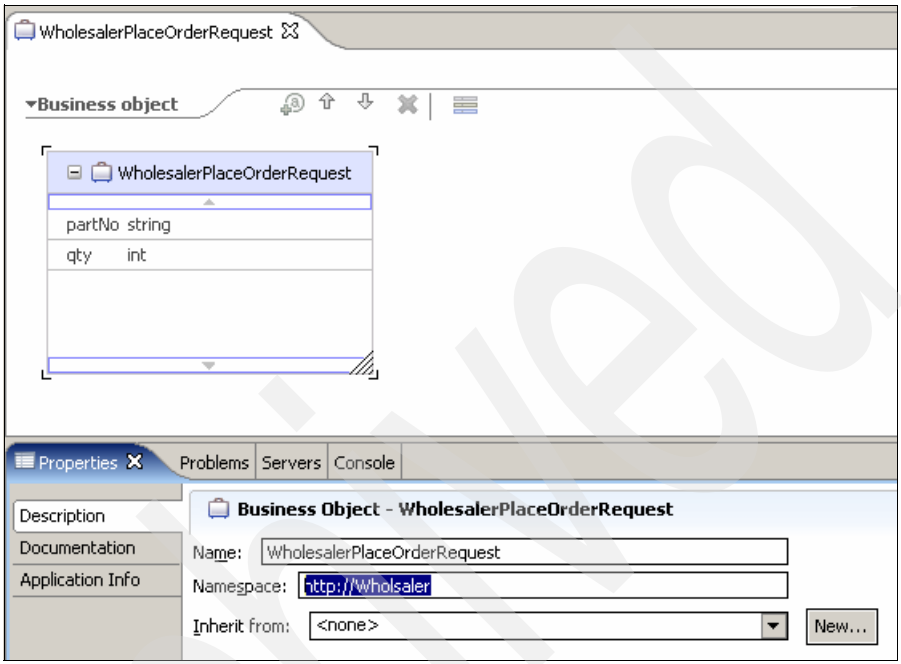


Figure A-25 Changing a namespace

Archived

Creating the remaining interfaces

In this section, we build the remaining interfaces for Wholesaler A. The interfaces are used to place and cancel an order at Wholesaler A and respond to ITSO Electronics with a confirmation message.

In the WholesalerA module, create the following interfaces:

1. Create the interface *WhPlaceOrdWeb_I* with the namespace `http://Wholesaler/WhPlaceOrdWeb_I` as shown in Figure A-26.

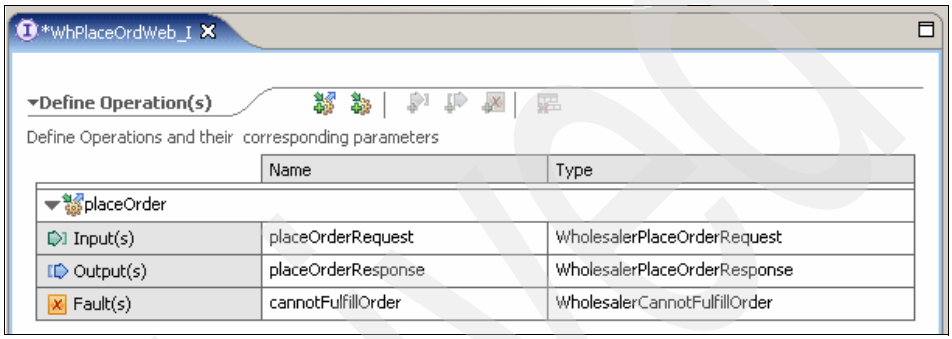


Figure A-26 WhPlaceOrdWeb_I interface

2. Create the interface *WhCnclOrdWeb_I* with the namespace `http://Wholesaler/WhCnclOrdWeb_I` as shown in Figure A-27.

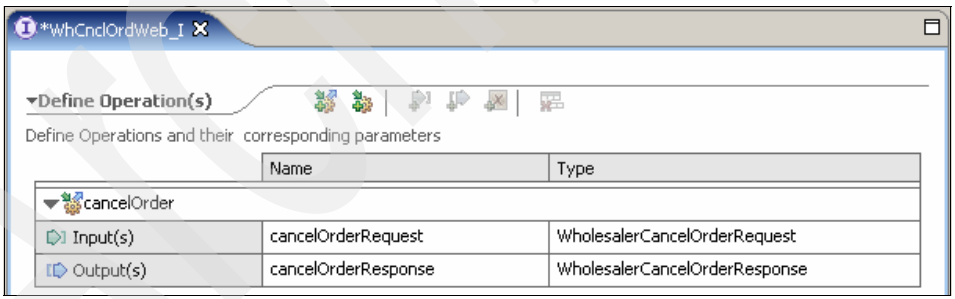


Figure A-27 WhCnclOrdWeb_I interface

Archived

Figure A-28 shows how your Business Integration view should look after creating the interfaces.

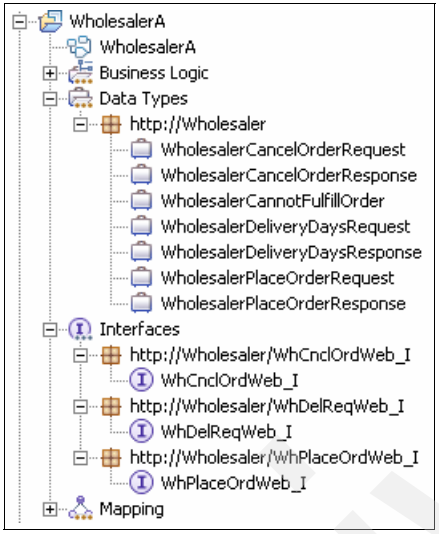


Figure A-28 Business Integration view with all interfaces

Archived

Creating the business logic

After you create the business objects and interfaces, you need to create the simple business logic for Wholesaler A. Wholesaler A implements three business functionalities (Figure A-29):

- Place Order (implemented as business process)
- Get Delivery Days (implemented as business process)
- Cancel Order (implemented as business rule)

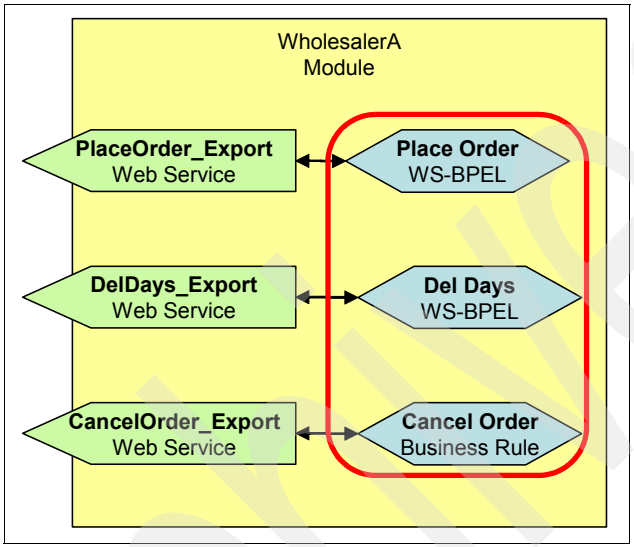


Figure A-29 Business logic for Wholesaler A

Creating the WholesalerA_PlaceOrder WS-BPEL business process

The business process places an order to Wholesaler A. The logic is kept simple. The logic receives a place order request message and returns a confirmation code using the WhPlaceOrdWeb_I interface. To simplify the implementation, the confirmation code is a static value.

Complete the following steps:

1. In the WholesalerA module, expand the **Business Logic** folder in the Business Integration view, right-click and select **Processes** → **New** → **Business Process**.

Archived

2. In the New Business Process window (Figure A-30), leave Namespace set to `http://WholesalerA`. In the Name field, type `WholesalerA_PlaceOrder`. Click **Next**.

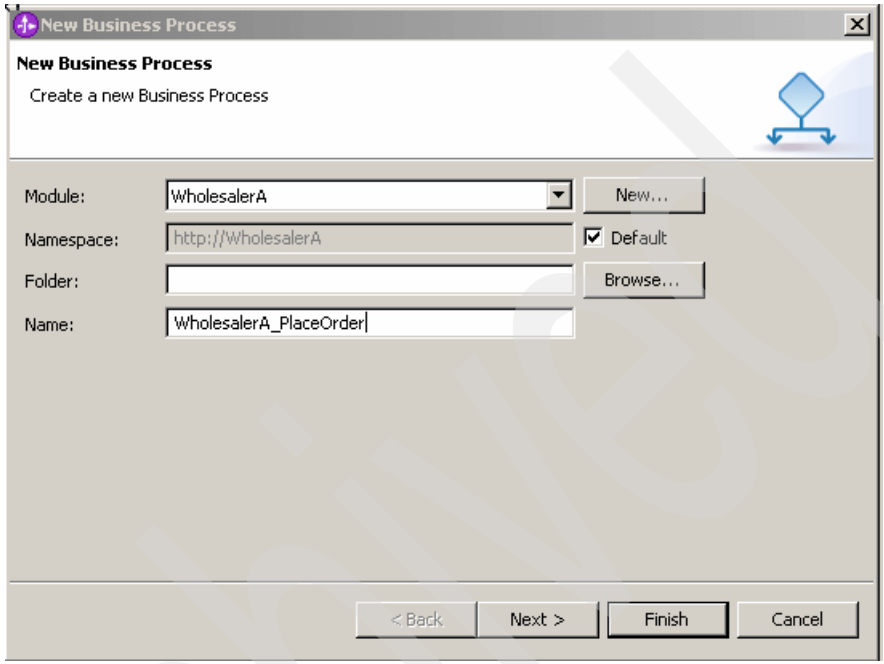


Figure A-30 New Business Process window

3. In the next window, select **Select an existing Interface** and click **Browse**.

Archived

4. In the Interface Selection window (Figure A-31), select **WhPlaceOrdWeb_I** and click **OK**.

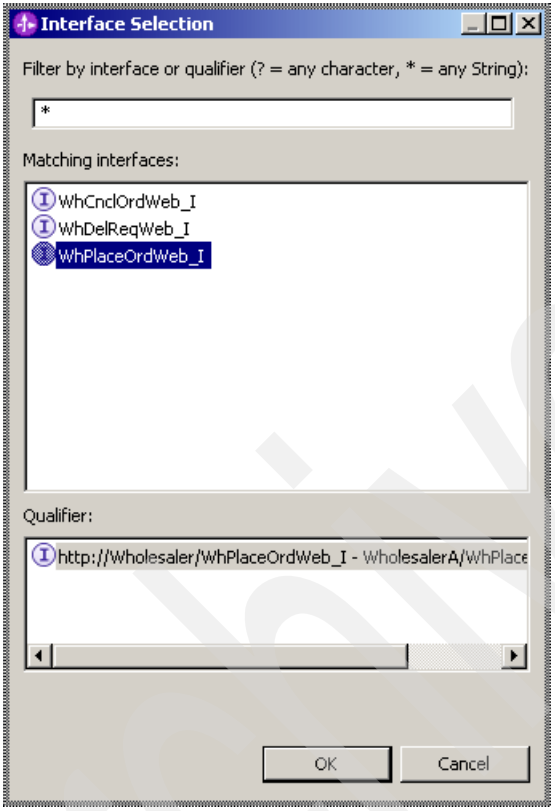


Figure A-31 Interface Selection window

Archived

5. In the New Business Process window (Figure A-32), click **Next**.

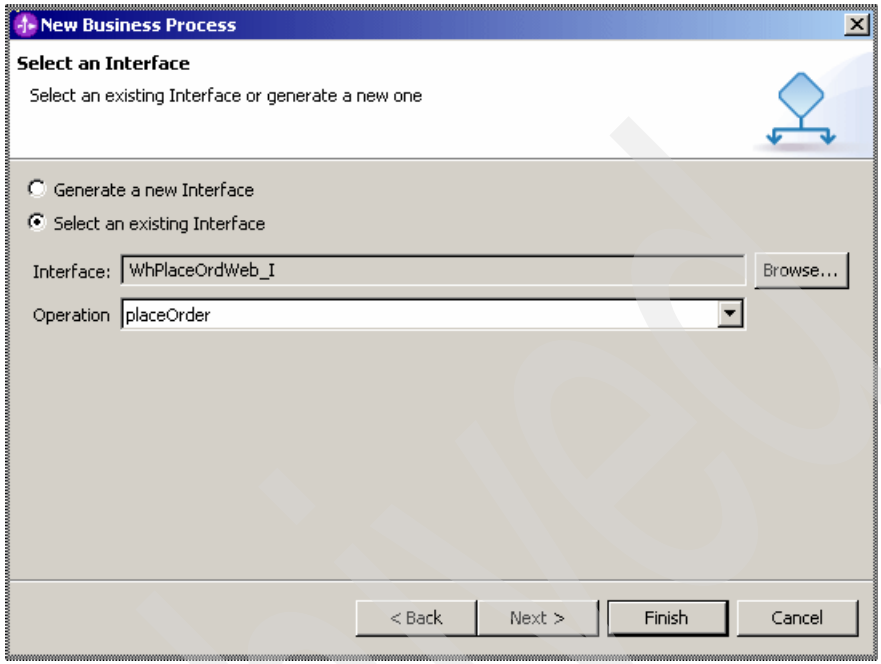


Figure A-32 New Business Process window

6. The Advanced settings prompt you to specify whether you want to disable WS-BPEL extensions. WebSphere Process Server extends the WS-BPEL specification to include Java snippets and human tasks. If you want to run this WS-BPEL process outside of WebSphere Process Server, you need to turn off these features. Click **Finish**.

7. The Business Process Editor (Figure A-33) opens. This editor allows you to work with and construct WS-BPEL processes. Click the **Receive** activity, click the **Properties** tab, and then click the **Details** tab.

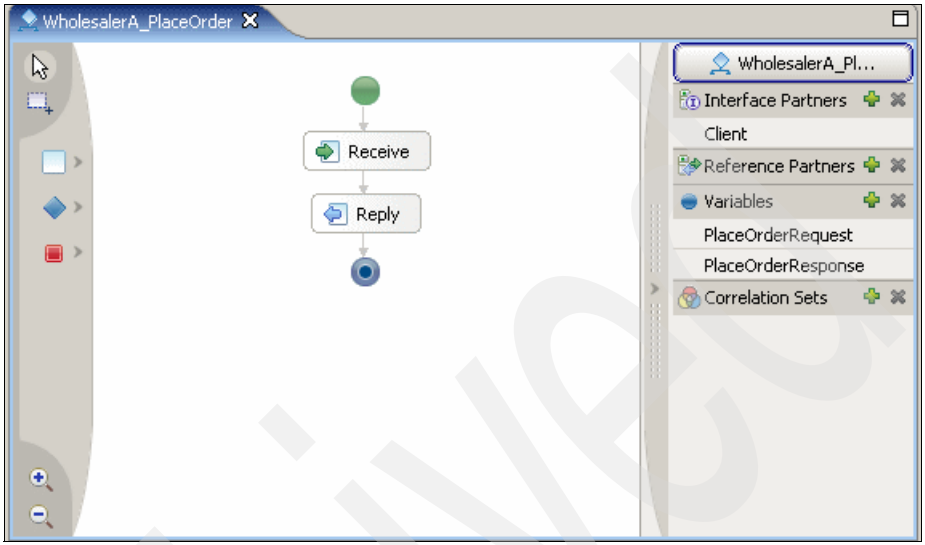


Figure A-33 Business Process Editor

Archived

8. Verify that your results looks like the example in Figure A-34. Note that we assigned the process to the placeOrder operation and that the Receive activity receives the placeOrderRequest interface input.

   Click the **Reply** activity, the **Properties** tab, and then the **Details** tab.

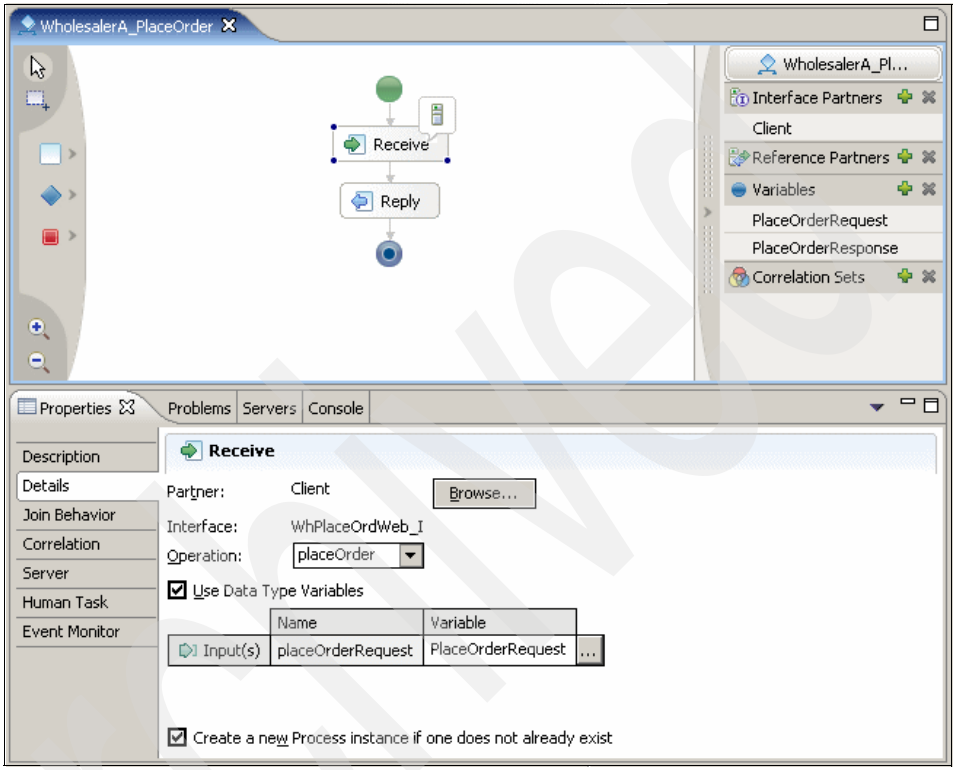


Figure A-34 Detail properties of the Receive activity

Archived

9. Verify that your results looks like the example in Figure A-35. The Reply activity is assigned the placeOrderResponse interface output.

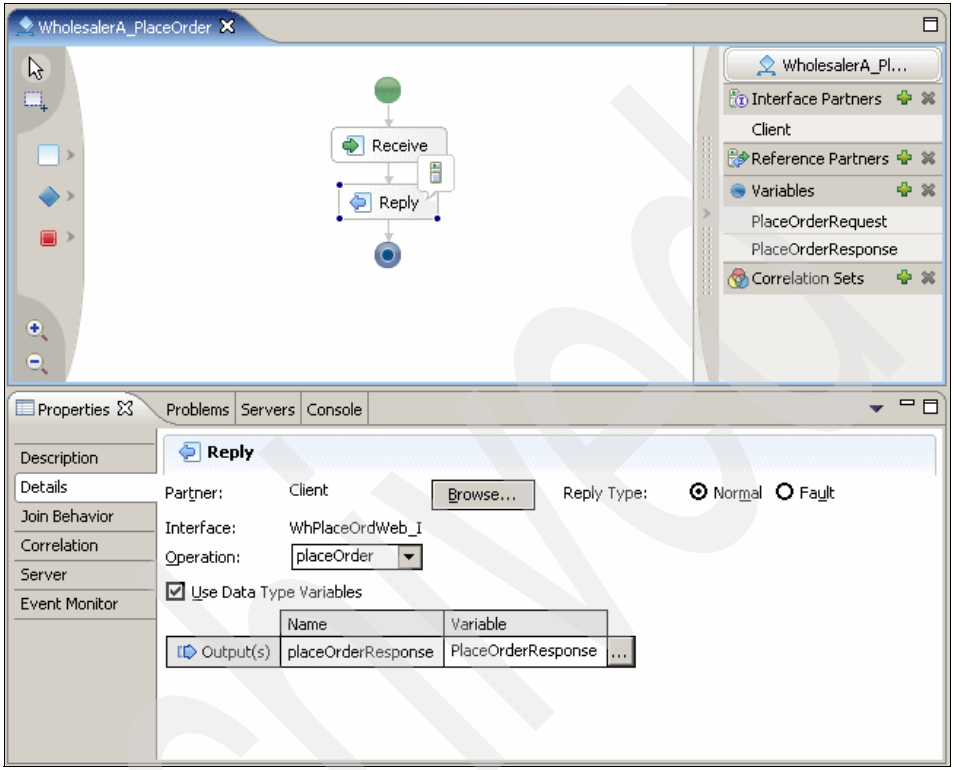


Figure A-35 Detail properties of the Reply activity

10. Back in the Business Process Editor, click the **Empty Action** icon ( ) to expand the toolbar.
11. Click the **Snippet** icon ( ).
12. Click between Receive and Reply to insert the snippet (Figure A-36).

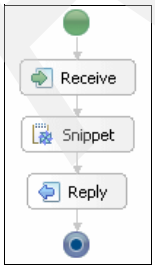


Figure A-36 Insert Snippet activity

13. Click the **Snippet** activity. Click the **Properties** tab and the **Details** tab to open the Visual Snippet Editor.

14. Ensure that the **Visual** radio button is selected (Figure A-37).

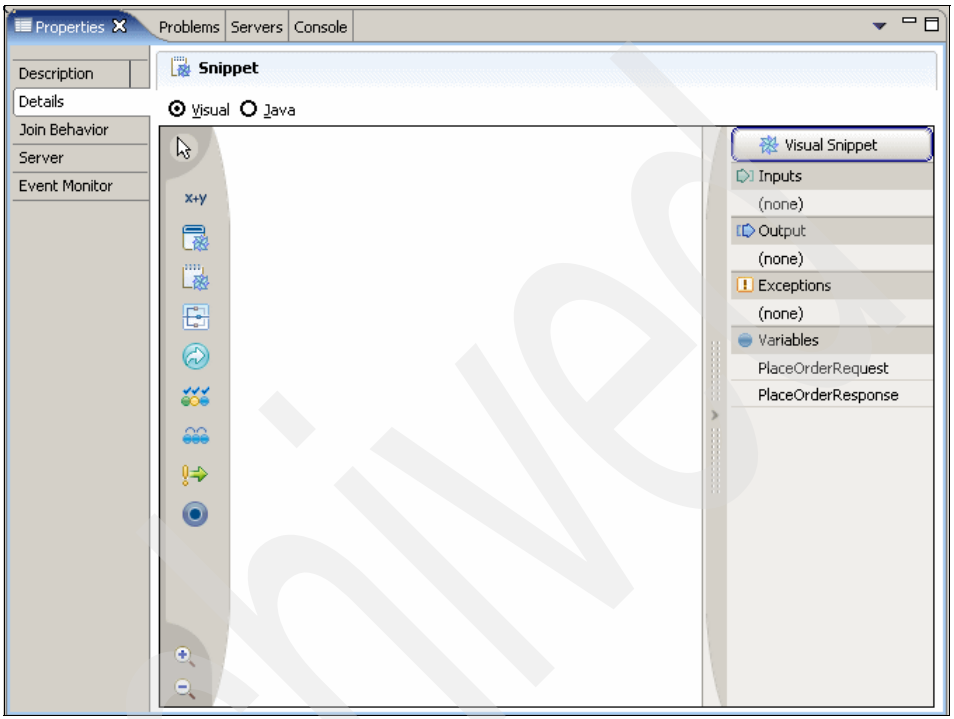


Figure A-37 Visual Snippet Editor

Archived

15. We use the Visual Snippet Editor to create a new instance of the PlaceOrderResponse business object and then set confirmCode to be a fixed value. The alternative to a visual snippet is a Java snippet, which uses Java code to implement the same logic.

   a. Click the **Standard** icon ( ).

   b. In the Add a Standard Visual Snippet window (Figure A-38), select **SCA services** → **create specific BO** and click **OK**.

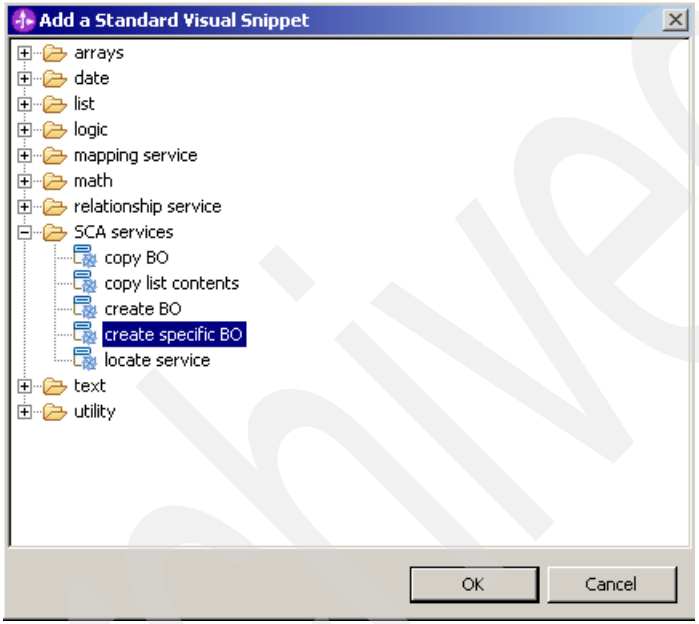


Figure A-38 Add a Standard Visual Snippet window

Archived

c. In the Data Type Selection window (Figure A-39), select **WholesalerPlaceOrderResponse** and click **OK**.

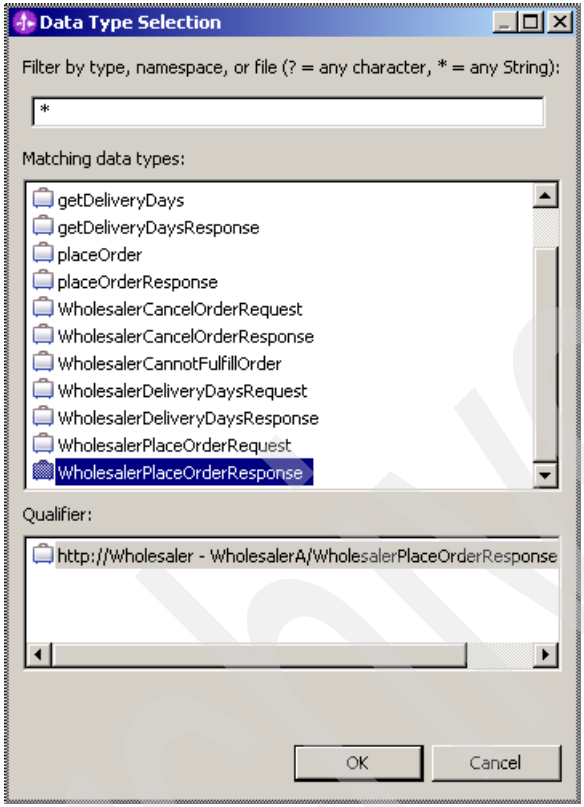


Figure A-39 Data Type Selection window

d. Click the **Visual Snippet Editor canvas** to add the WholesalerPlaceOrderResponse business object instance (Figure A-40).

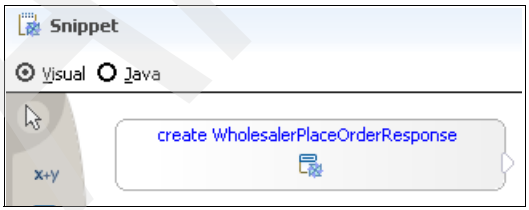


Figure A-40 WholesalerPlaceOrderResponse business object in the Snippet Editor

Archived

e. Drag the variable **PlaceOrderResponse** from the Variables section to the Visual Snippet Editor canvas (Figure A-41).

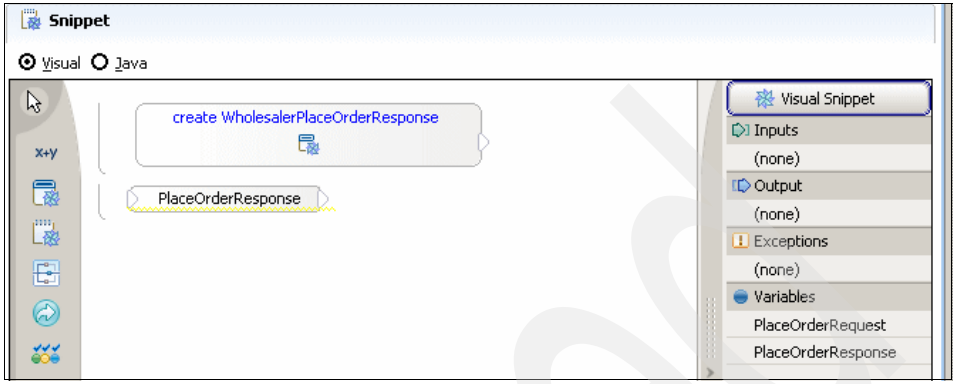


Figure A-41 Dragging the variable to the canvas

f. Drag the link ( )of **WholesalerPlaceOrderResponse** (A) to the link of **PlaceOrderResponse** (B in Figure A-42).

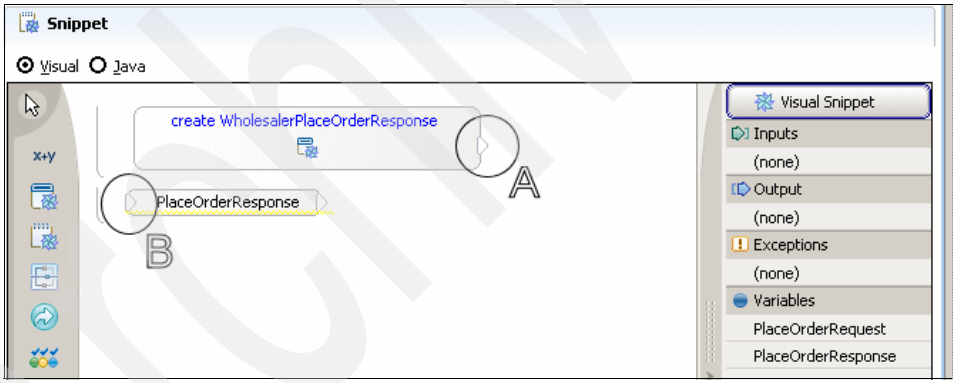


Figure A-42 Dragging link A to link B

Archived

Figure A-43 shows how the Visual Snippet Editor looks after you create the link.

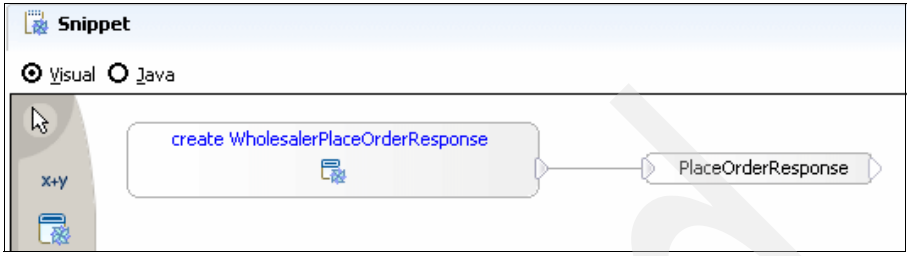


Figure A-43 WholesalerPlaceOrderResponse linked to PlaceOrderResponse

g. Click the **Expression** icon (x+y).

h. Click the **Visual Snippet Editor canvas** to add the new expression.

i. Click the **Expression** activity on the canvas and type **"**`order54`**"** (including the quotation marks; see Figure A-44). Alternatively you can select **String** in the visual assist menu and type `order54` (without the quotation marks).

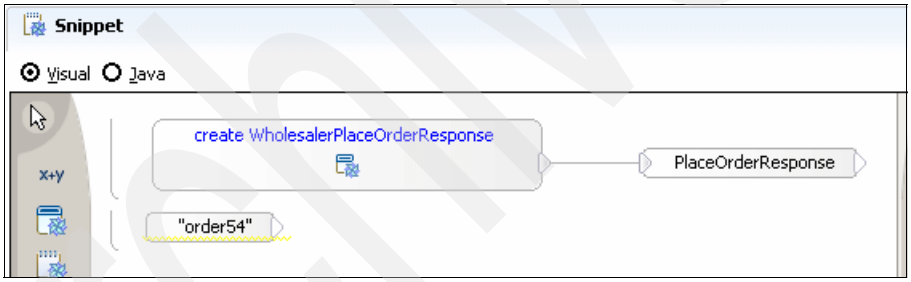


Figure A-44 Adding the Expression to the Visual Snippet Editor canvas

j. Drag another instance of the variable **PlaceOrderResponse** from the Variables section to the Visual Snippet Editor canvas.

Archived

k. Click the **PlaceOrderResponse** in the canvas to open the visual assist menu (Figure A-45).

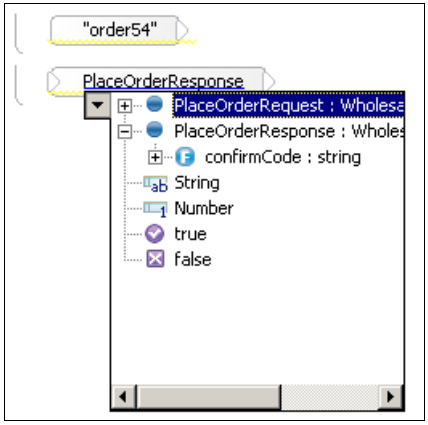


Figure A-45 Visual assist menu

l. Expand **PlaceOrderResponse** and click **confirmCode** to select it.

Figure A-46 shows how the Visual Snippet Editor canvas looks.

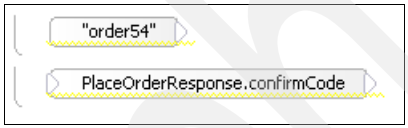


Figure A-46 Visual Snippet Editor canvas

m. Link the output of the “order54” expression to the input of the PlaceOrderResponse.confirmCode variable. Figure A-47 shows how the Visual Snippet Editor canvas looks with this change.

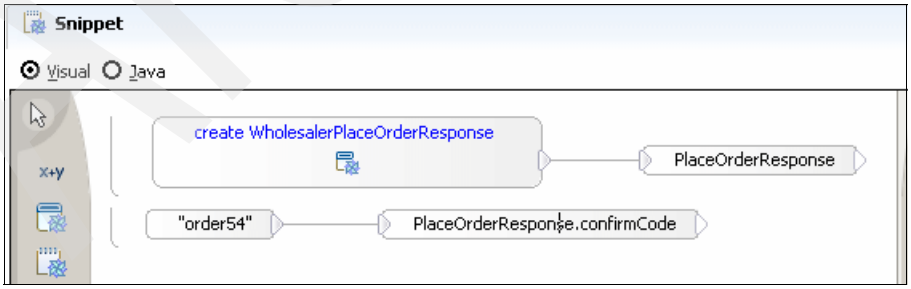


Figure A-47 Competed Visual Snippet Editor canvas

Archived

n. The visual snippet is now completed. Click the **Snippet Activity** in the Business Process Editor and rename it to `SetOrderResponse` (Figure A-48).

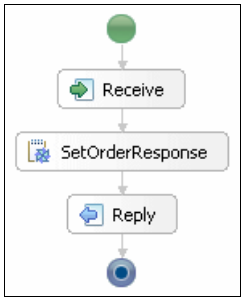


Figure A-48 Competed business process

16. Click Ctrl+S to save the business process, and close the business process editor.

Creating the getDeliveryDays WS-BPEL business process

This business process receives a request message, including a part number of an item, and returns the amount of delivery days and price for the item. The business process calculates the price using a simple formula in a visual snippet. For the item part1, it returns a delivery time of 4 days; for all other items, it returns a delivery time of 10 days.

Archived

Complete these steps:

1. Create a new Business Process named *WholesalerA_GetDeliveryDate*.
2. In the New Business Process window (Figure A-49), click the **Select and existing Interface** option. For Interface, select **WhDelReqWeb_I**. For Operation, select **getDeliveryDays**. Click **Next**.

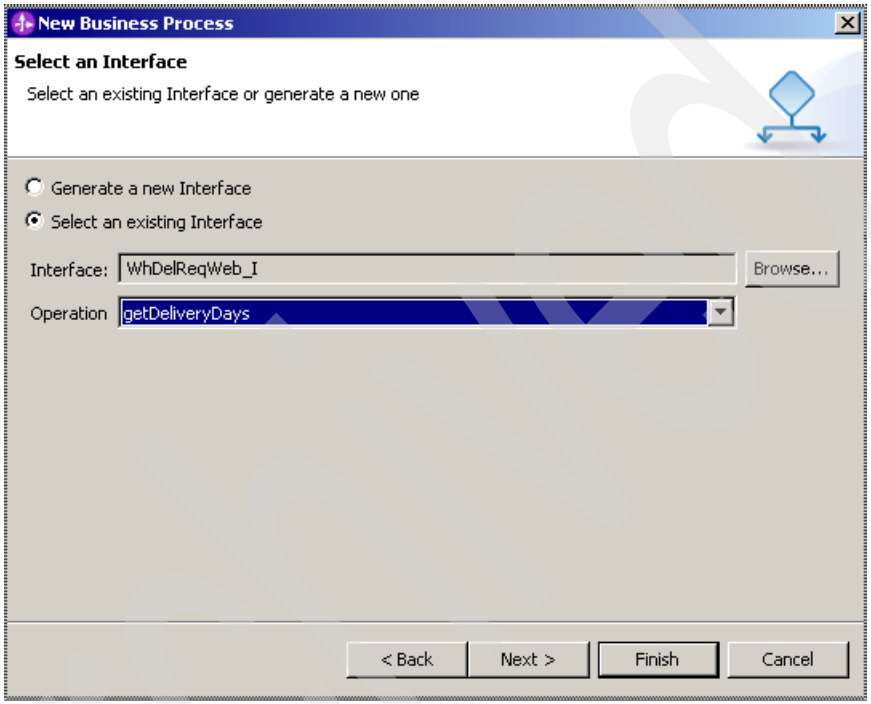


Figure A-49 Selecting the interface and operation

3. Do not disable the WebSphere Process Server BPEL Extensions.

Archived

4. Add a snippet activity to the Business Process Editor canvas between the Receive and the Reply activities, and rename it to `Calculate Price` (Figure A-50). This snippet is used to set the price value in GetDeliveryDaysWholesalerResponse.

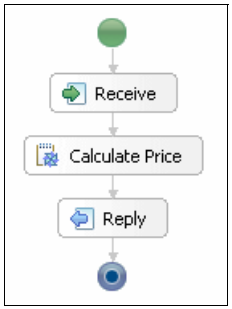


Figure A-50 Business Process Editor canvas

5. Click the **Calculate Price** snippet. Then click the **Properties** tab and **Details** tab to open the Visual Editor.
6. Click the **Standard** icon to add a standard visual snippet.
7. Select **SCA services** → **create specific BO** and click **OK**.
8. Select **WholesalerDeliveryDaysResponse** and click **OK**.
9. Click the **Visual Snippet Editor canvas** to add the create WholesalerDeliveryDaysResponse snippet.
10. Drag the variable **GetDeliveryDaysWholesalerResponse** from the Variables area to the Visual Snippet Editor canvas.
11. Link the output of the create WholesalerDeliveryDaysResponse snippet to the input of the GetDeliveryDaysWholesalerResponse variable (Figure A-51).

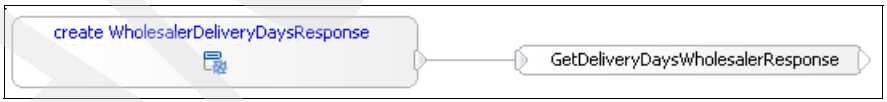


Figure A-51 Linked snippet and variable

12. Click the **Expression** icon, and click the **Visual Snippet Editor canvas** to create a new expression.
13. For Expression, type `10*` and select **GetDeliveryDaysWholesalerRequest** → **quantity** from the visual assist menu.

Archived

14. Drag another instance of the variable **GetDeliveryDaysWholesalerResponse** from the Variables area to the Visual Snippet Editor canvas.

15. Click **GetDeliveryDaysWholesalerResponse** and select **GetDeliveryDaysWholesalerResponse** → **price** from the visual assist menu.

16. Link the output of **10*GetDeliveryDaysWholesalerRequest.quantity** to the input of GetDeliveryDaysWholesalerResponse.price. See Figure A-52.

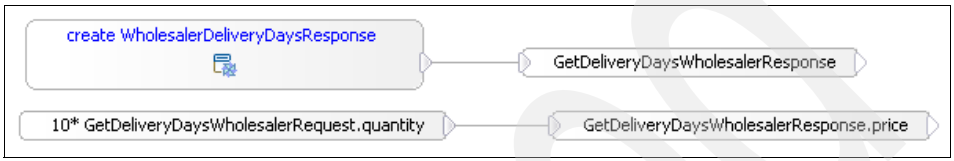


Figure A-52 Completed Visual Snippet Editor canvas

17. In the Business Process Editor, click the **Choice** activity and place it between the Calculate Price and Reply activities (Figure A-53). This choice activity is used to determine the delivery days value to return.

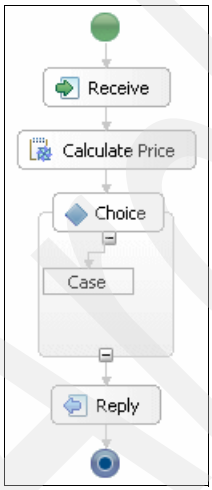


Figure A-53 Choice activity

18. Right-click the **Choice** activity and select **Add Otherwise**.

Archived

19. Click **Case** and click the **Description** tab. For Display Name, type `part1` (Figure A-54).

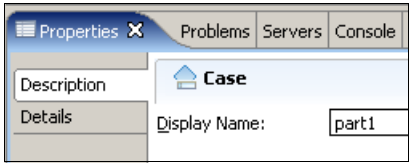


Figure A-54 Renaming the case to part1

20. Click the **Details** tab and click **Create a new Condition**.
21. Check that the Expression Language is set to **Same as Process (Java)**.
22. Click the **true snippet** and select **GetDeliveryDaysWholesalerRequest → partNo.**
23. Click **==**.
24. Click **String** and type `part1` (note lowercase and without quotation marks) as shown in Figure A-55.

Figure A-55 Completed visual snippet

> **Note for Java programmers**: The == operator used here is translated into the java.lang.String method equals(), to avoid the problem of comparing Strings using ==. To verify this, click the **Java** radio button and view the generated code from the visual snippet. Do not make any changes to the code at this point or you will not be able to return to the visual representation.

25. In the Business Process Editor, click the **Assign activity** icon ( ) and insert it below part1. We configure this assign activity to assign the value `4` to the daysToDeliver attribute.

Archived

26. Rename the assign activity to `4days` (Figure A-56).

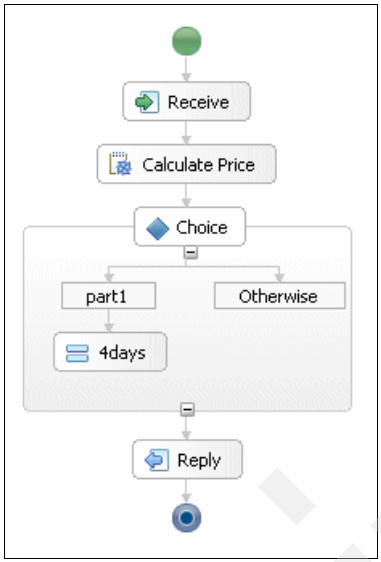


Figure A-56 Adding the Assign activity

27. In the Detail tab of the Assign activity, from the From list box, select **Fixed Value**, and from the To list box, select **Variable** (Figure A-57).

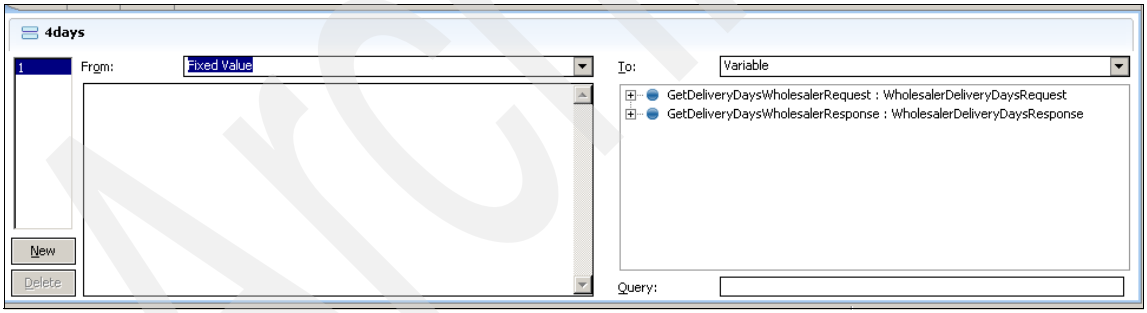


Figure A-57 Assigning fixed values to the variables

Archived

28. As shown in Figure A-58, in the To side, expand **GetDeliveryDaysWholesalersResponse** and highlight **daysToDeliver**. On the From side, type 4.

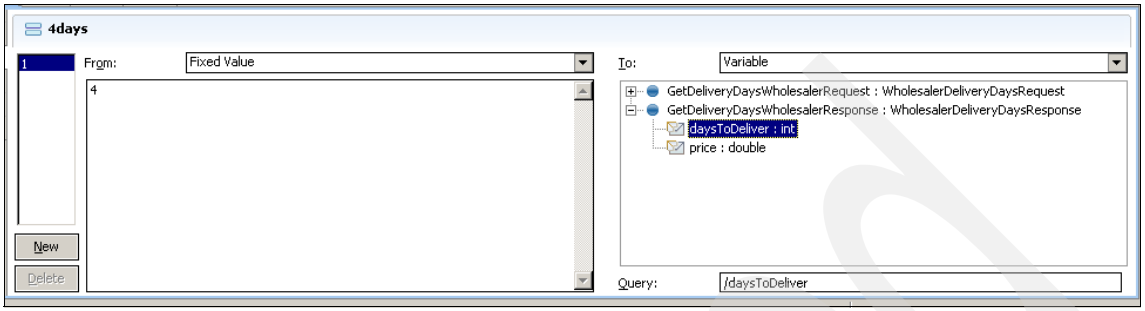


Figure A-58 The completed assign activity

Important: When using an assign activity, there is no OK or Apply button to click to indicate that the assign is complete. The indication that the assign has been made is the fact that the To field is highlighted. Remember that if a different field is highlighted, it changes the operation of the assign node.

Hint: Look at the Query box at the bottom right, to check which field is being assigned.

Archived

29. In the Business Process Editor, click the **Assign activity** icon ( ) and insert it under Otherwise, and rename the assign activity to `10days` (Figure A-59).

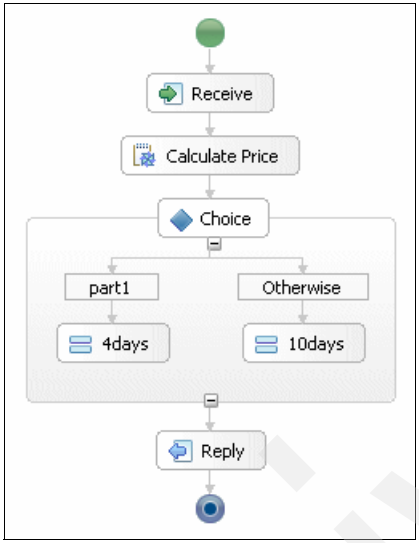


Figure A-59 Adding an additional assign activity

30. Assign the fixed value of `10` to the variable GetDeliveryDaysWholesalersResponse.daysToDeliver (Figure A-60).

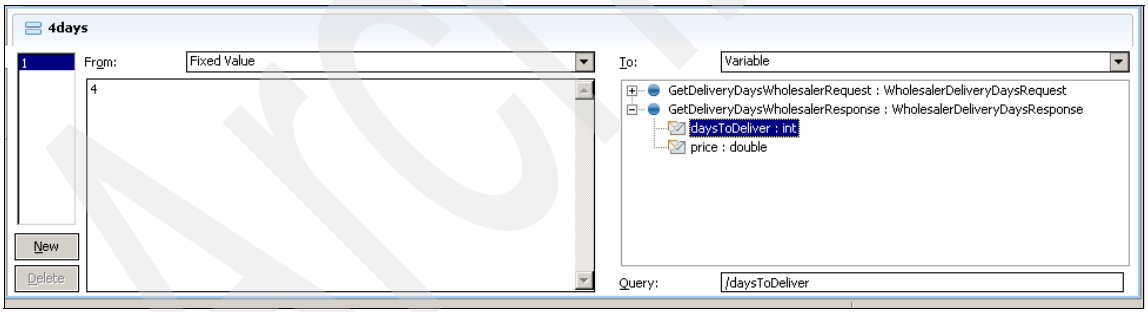


Figure A-60 Completed assign activity

31. Click the **Business Process Editor canvas**, and then press Ctrl+S to save the completed business process.

Archived

Creating the WholesalerA_CancelOrder business rule

The Order Cancellation functionality is implemented as a business rule. A *business rule* is anything that imposes structure upon, or controls the behavior of, a business practice. A rule can enforce business policy, establish common guidelines within an organization, or control access in a business environment. A business rule is used to help abstract the client from implementation.

Use business rules to officiate over frequently changing business practices that can come from within a business or mandated from outside a business. Typical uses for business might be, for example, determining special groups such as wholesalers or preferred customers.

Business rules allow business analysts to change business rules on demand, without restarting the project or application server using a Business Rule Editor Web tool.

In the following steps, we create a business rule to set the value of orderCancelled to *true*.

1. In the WholesalerA module, expand **Business Logic**, right-click, and select **Rule Groups** → **New** → **Rule Group**.
2. In the New Rule Group window (Figure A-61), for Name field, type `WholesalerA_CancelOrder` and then click **Next**.

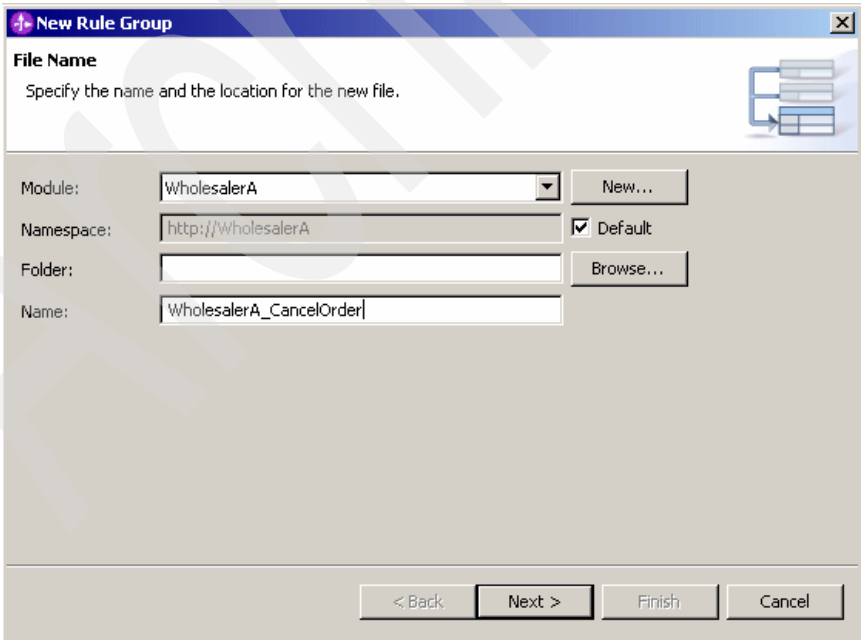


Figure A-61 New Rule Group window

3. In the next window (Figure A-62), select the interface **WhCnclOrdWeb_I** and click **Finish**.

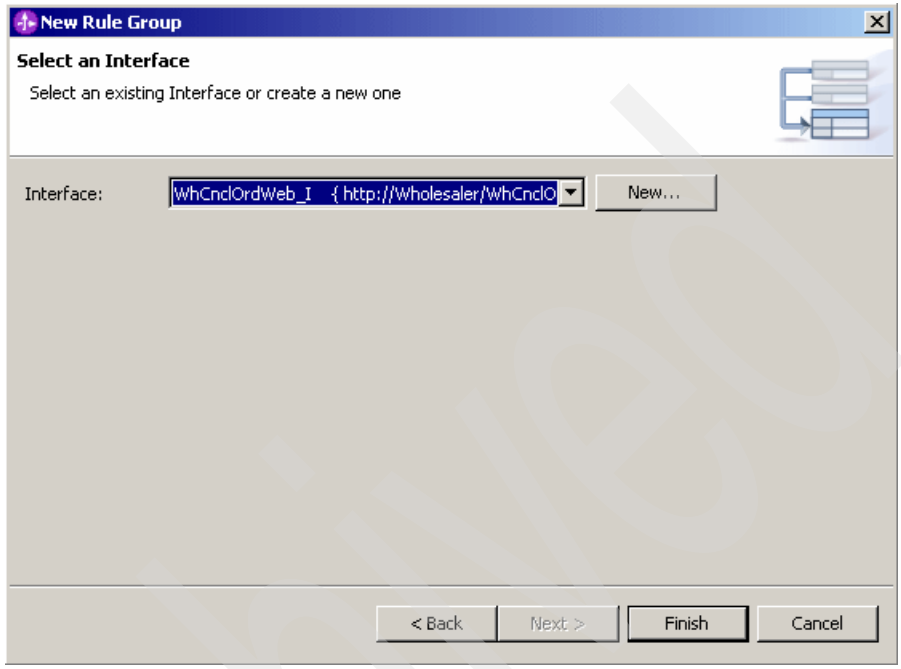


Figure A-62 Selecting the interface WhCnclOrdWeb_I

4. The Rule Group Editor (Figure A-63) opens. Click the **cancelOrder** operation.

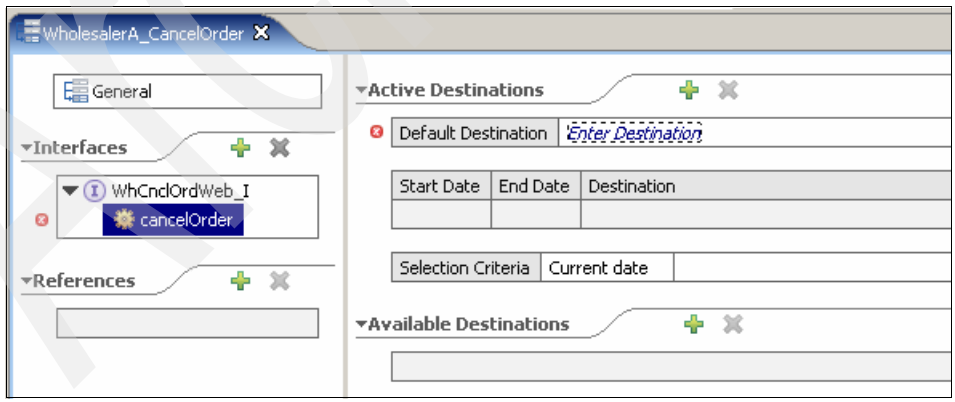


Figure A-63 The cancelOrder operation

5. Under Active Destinations, click **Enter Destination**, and then select **New Ruleset** (Figure A-64).

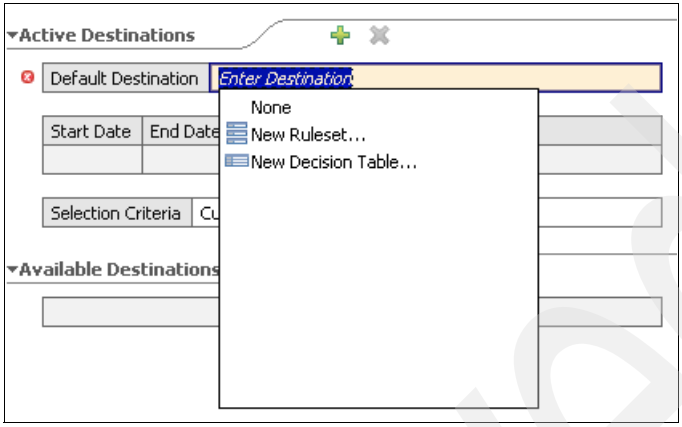


Figure A-64 Selecting New Ruleset

6. In the New Rule Set window (Figure A-65), in the Name field, type `cancelOrderRS`. Then click **Finish**. This creates a new ruleset.

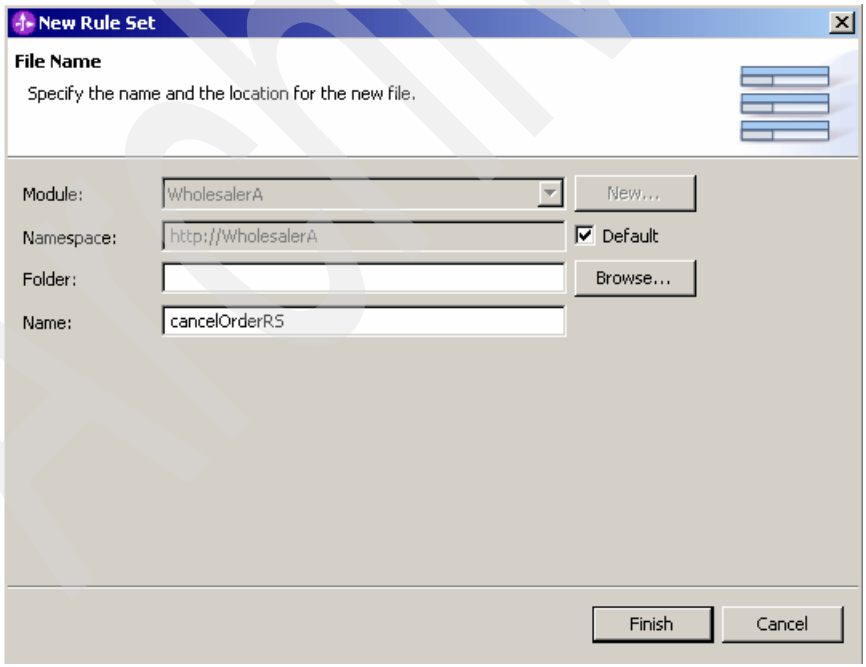


Figure A-65 New Rule Set window

7. In the Ruleset Editor that opens, click the **Add Action Rule** icon ( ).
8. Click **Action** and select **Set var1 to var2**.
9. Click the first **Select_Variable** and select **cancelOrderResponse** → **orderCancelled** from the visual assist menu.
10. Click the second **Select_Variable?** and select **true** from the visual assist menu. Figure A-66 shows the completed ruleset.

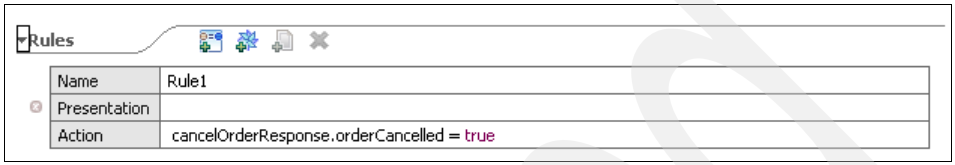


Figure A-66 Completed ruleset

11. Press Ctrl+S to save the ruleset and close the Ruleset Editor.
12. Return to the business rule group WholesalerA_CancelOrder. The business rule group should now contain the Available Destination cancelOrderRS as shown in Figure A-67.

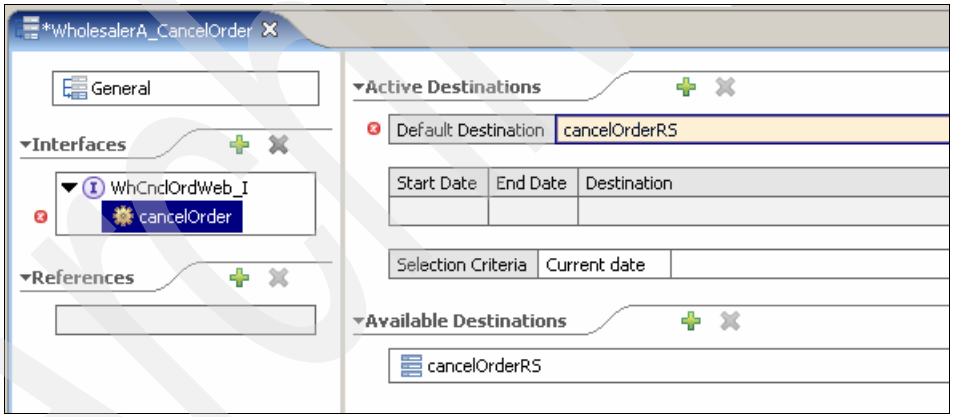


Figure A-67 Completed business rule group

13. Press Ctrl+S to save the business rule group, and close the Business Rule Group Editor.

Archived

Figure A-68 shows how the Business Integration view should look now.

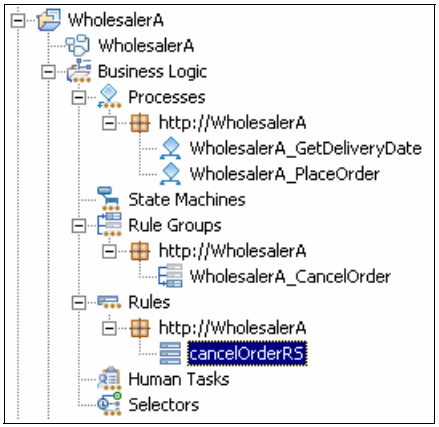


Figure A-68 Business Integration view

Creating the assembly diagram

We have now created the business objects, the interfaces, and the business logic for each of the services that Wholesaler A provides. In the assembly diagram, we need to expose these services as Web services, so they can be called by service consumers such as ITSO Electronics. This creates three Web services exports, as shown in Figure A-69.

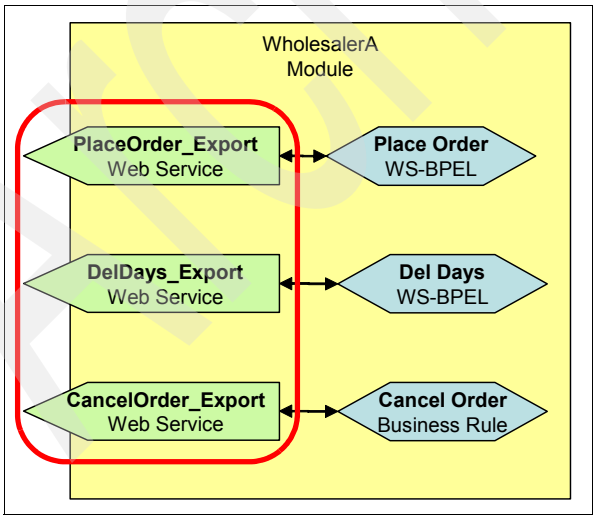


Figure A-69 Web services exports for Wholesaler A

Complete the following steps:

1. Double-click **WholesalerA** (as shown in Figure A-70) to open the assembly diagram in the Assembly Editor.

Figure A-70 Opening the assembly diagram

2. Expand **WholesalerA** → **Business Logic** → **Processes**, and from the Processes folder, drag the following items onto the assembly diagram:
   - WholesalerA_GetDeliveryDate
   - WholesalerA_PlaceOrder
3. Expand **WholesalerA** → **Business Logic** → **Rule Groups**, and from the Rule Groups folder, drag **WholesalerA_CancelOrder** onto the assembly diagram. Figure A-71 shows the changes to the assembly diagram.

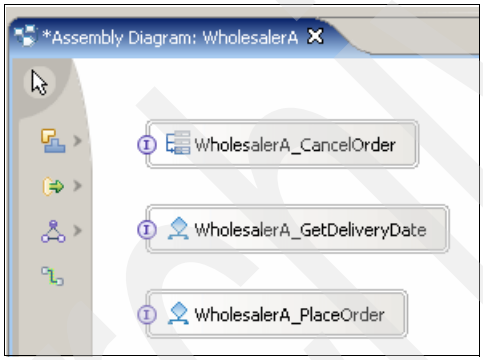


Figure A-71 Assembly diagram with components

> **Tip:** Right-click the **canvas** and select **Arrange Contents Automatically** to align the content of the assembly diagram. The order of the components is not important.

4. We are now ready to create Web services exports for these components. Right-click the **WholesalerA_CancelOrder** component, and select **Export** → **Web Service Binding**.

Archived

5. In the Add Binding File Generation window (Figure A-72), click **Yes**. This ensures a WSDL file is generated.

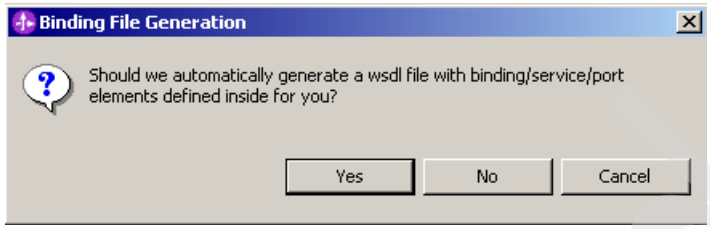


Figure A-72 Binding File Generation window

6. When you create a Web service, you have two choices for the type of Web service you can create, SOAP over HTTP or SOAP over JMS. In the Select Transport window, select **soap/http** and click **OK**. A Web service binding is now added to the assembly diagram as shown in Figure A-73.

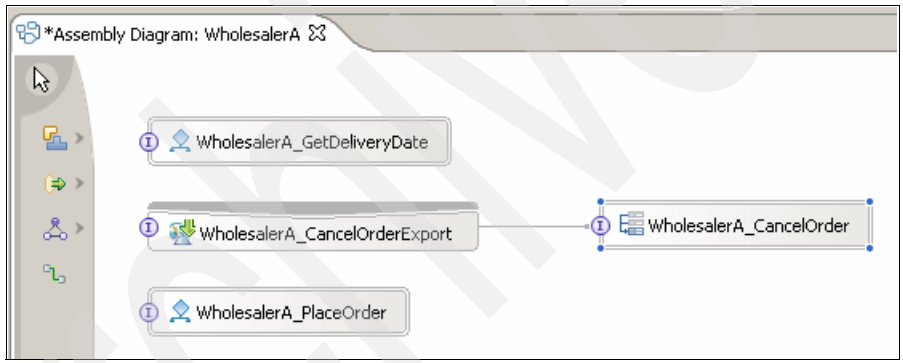


Figure A-73 New Web services binding

Archived

7. Create SOAP/HTTP Web service bindings for WholesalerA_GetDeliveryDate and WholesalerA_Place Order using the same method. Figure A-74 shows the completed assembly diagram.

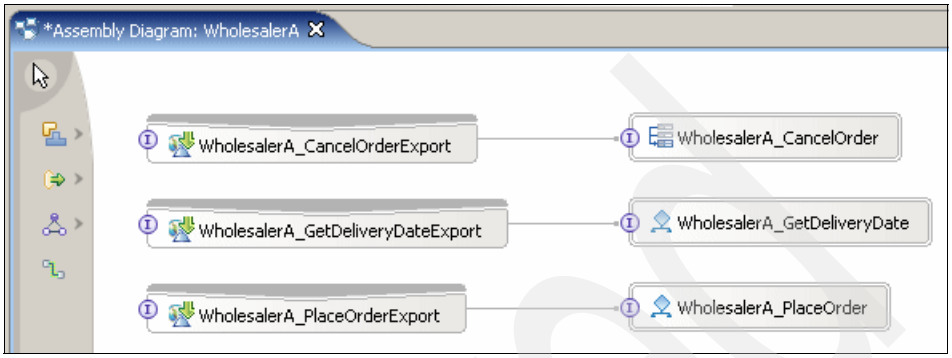


Figure A-74 Completed assembly diagram

8. Press Ctrl+S to save the assembly diagram, and close the Assembly Editor.

You now see a new folder Web Service Ports where the generated Web service WSDL files are stored. A Web service consumer can use these files later to invoke the Web services.

Archived

Figure A-75 shows the complete artifacts for Wholesaler A.

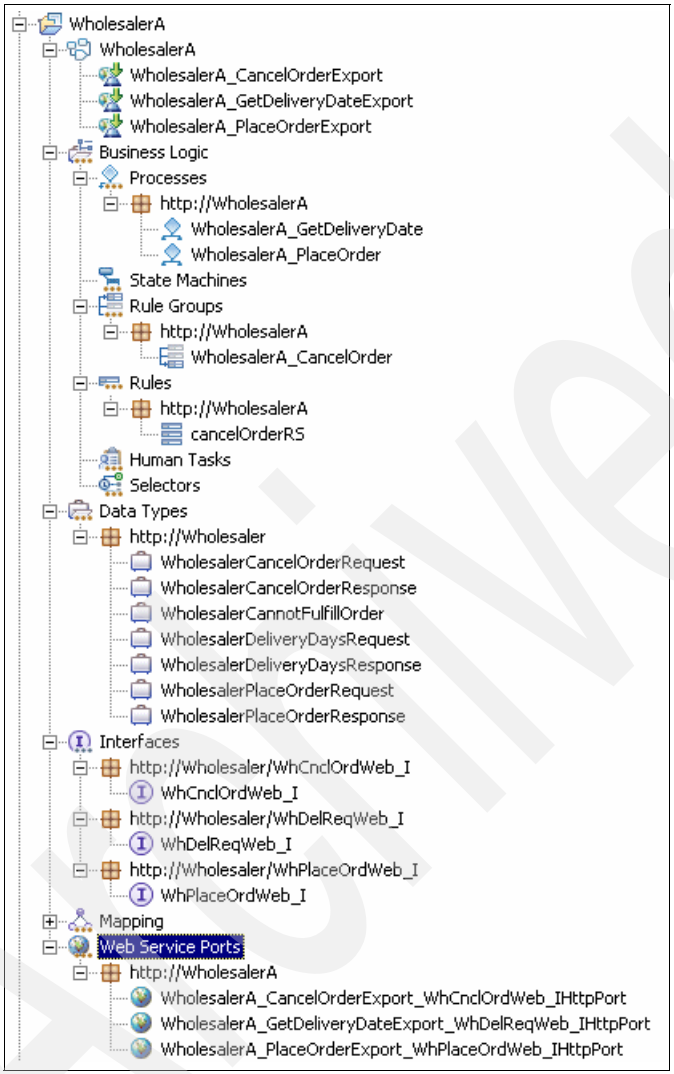


Figure A-75 Wholesaler A artifacts

Archived

Examining the generated WSDL files

Web services consumers typically invoke Web services by examining one or more WSDL files. This examination tells the Web service consumer the format of the SOAP message to construct, the protocol to send the SOAP message over, and the network address to send the SOAP message to.

You can view the WSDL files which the Web service consumer will use to contact the three Web services we have built by performing the following steps:

> **Optional:** These steps are optional. You can skip them if you are not interested in viewing the generated WSDL files.

1. A WSDL file has been generated for each Web service created in the previous section:
   - WholesalerA_CancelOrderExport_WhCnclOrdWeb_IHttp_Service.wsdl
   - WholesalerA_GetDeliveryDateExport_WhDelReqWeb_IHttp_Service.wsdl
   - WholesalerA_PlaceOrderExport_WhPlaceOrdWeb_IHttp_Service.wsdl

   In the Business Integration view, expand **Web Services Ports**, right-click **WholesalerA_PlaceOrderExport_WhPlaceOrdWeb_IHttpPort** and select **Open With** → **XML Source Page Editor**. The WholesalerA_PlaceOrder Export_WhPlaceOrdWeb_IHttp_Service.wsdl file opens.
2. A Web service consumer uses this WSDL file to construct a Web service SOAP call over HTTP to the Web service. Notice that this WSDL file imports the interface that you created earlier:

   ```
   <wsdl:import namespace="http://Wholesaler/WhPlaceOrderWeb_I"
   location="WhPlaceOrdWeb_I.wsdl"/>
   ```

   Also notice that this WSDL file states the HTTP URL where the SOAP message from the Web service consumer should be sent:

   ```
   <soap:address
   location="http://localhost:9080/WholesalerAWeb/sca/WholesalerA_PlaceOrderEx
   port"/>
   ```

Archived

Testing Wholesaler A

You can test any component within the assembly diagram using the Integration Test Client. Perform the following to test the WholesalerA_GetDeliveryDate process, and its associated Web service binding:

> **Note:** You can find more information about using the Integrated Test Client in "Starting run time and deploying modules to the test environment using the Integration Test Client" on page 572.

1. Open the assembly diagram for Wholesaler A in the Assembly Editor.
2. Right-click **WholesalerA_GetDeliveryDateExport** and select **Test Component**.
3. In the Integration Test Client, as shown in Figure A-76, notice that the WholesalerA_GetDeliveryDateExport component, and the getDeliveryDays operation are both selected. Click the **Configurations** tab.

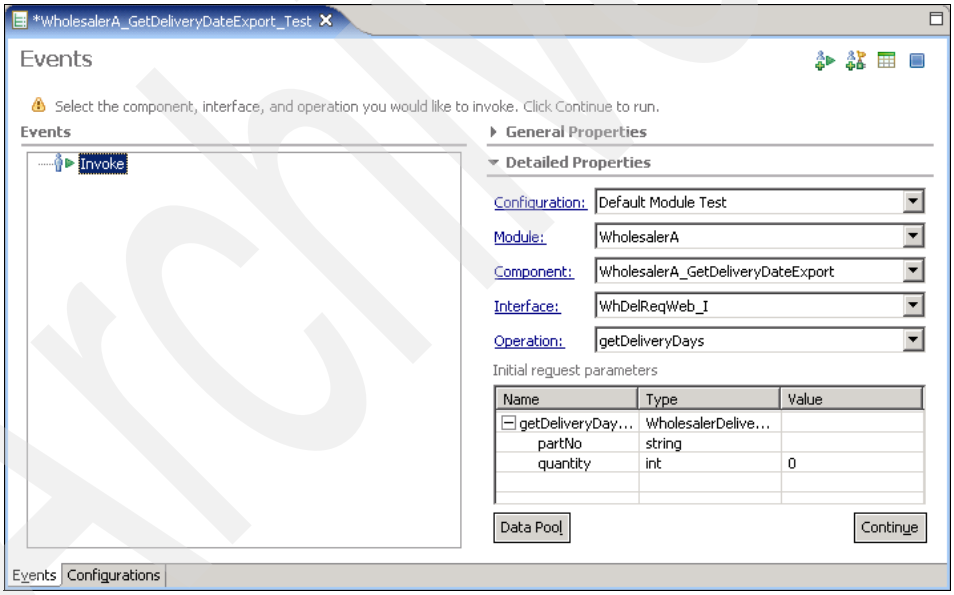


Figure A-76 Integration Test Client

Archived

4. As shown in Figure A-77, with the current settings, the call to the WholesalerA_GetDeliveryDate component (the WS-BPEL business process) is emulated. Click **WholesalerA_GetDeliveryDate** and then click **Remove**. This ensures that we call the real WS-BPEL process during testing rather than an emulator.

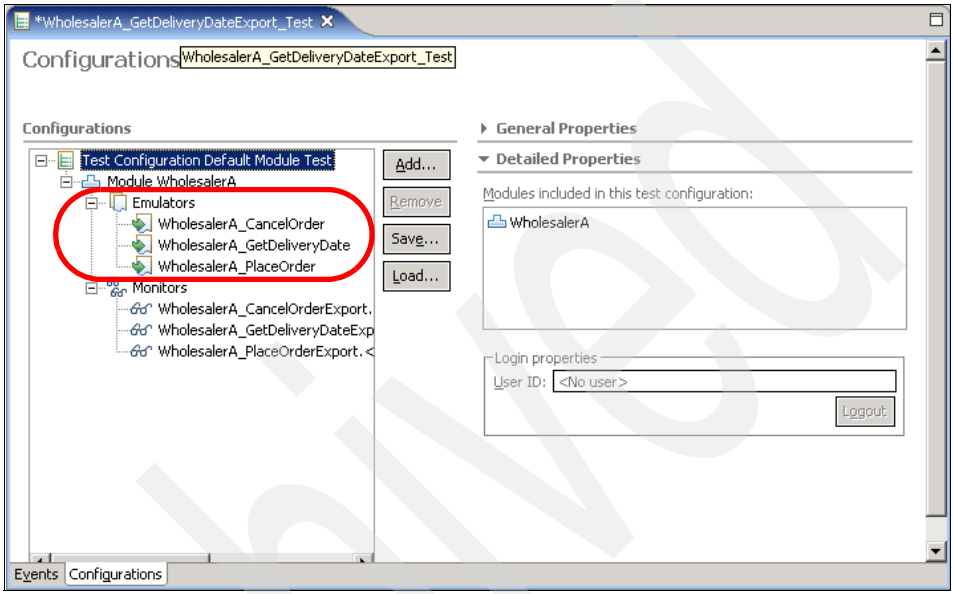


Figure A-77 Emulations in the Configurations view

5. Click the **Events** tab again. In the Initial requests parameters table, set partNo to `part1` and set quantity to `10` (Figure A-78). Click **Continue** to start the test.

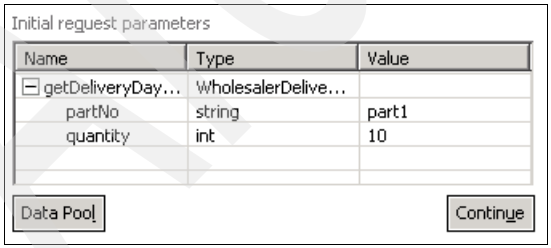


Figure A-78 Setting the input values

Archived

6. In the Deployment Location window (Figure A-79), select WebSphere Process Server v6.0 and click **Finish**. The Integration Test Client starts, by starting an instance of WebSphere Process Server and deploying the WholesalerA module to this server.

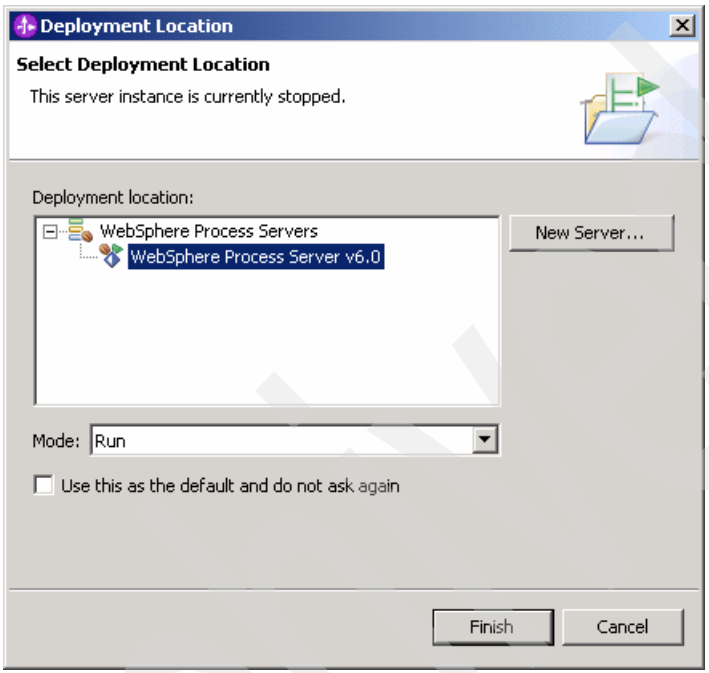


Figure A-79 Deployment Location window

Archived

7. The Web service is invoked, which in turn invokes the WS-BPEL business process (Figure A-80). Check for this flow in the Events section. You should also receive a response where daysToDeliver is set to 4 and price is set to 100. You can see this in the Return parameters table.

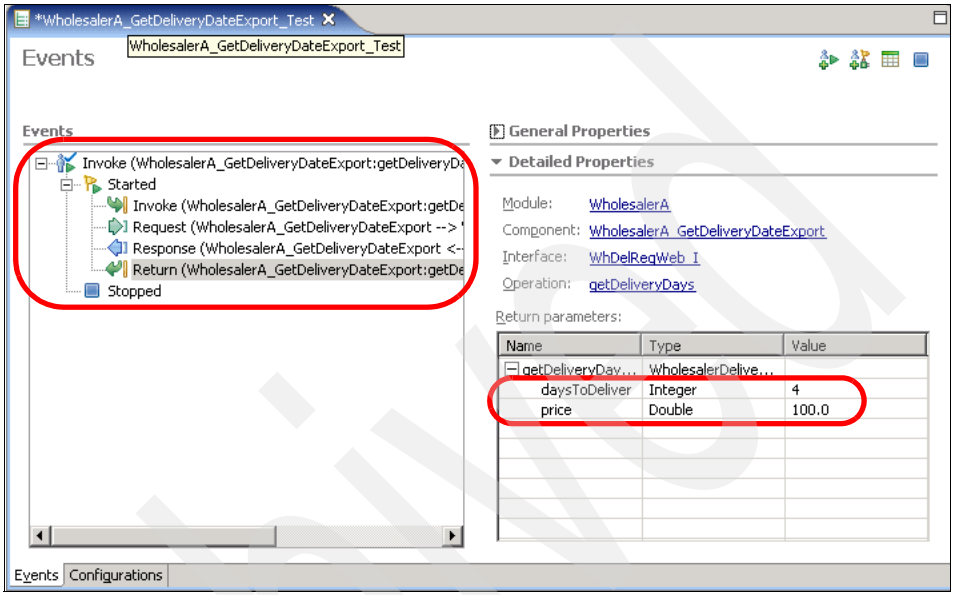


Figure A-80 Successful test of the Get Delivery Date functionality

Archived

8. Test this process again, this time specifying different input criteria.

   a. Click the **Invoke** icon to create a new test case.
   b. Specify a partNo of `part2` and a quantity of `5`.
   c. Click **Continue**.

   The component should this time respond with a daysToDeliver value of 10 and a quantity of 50 (Figure A-81).

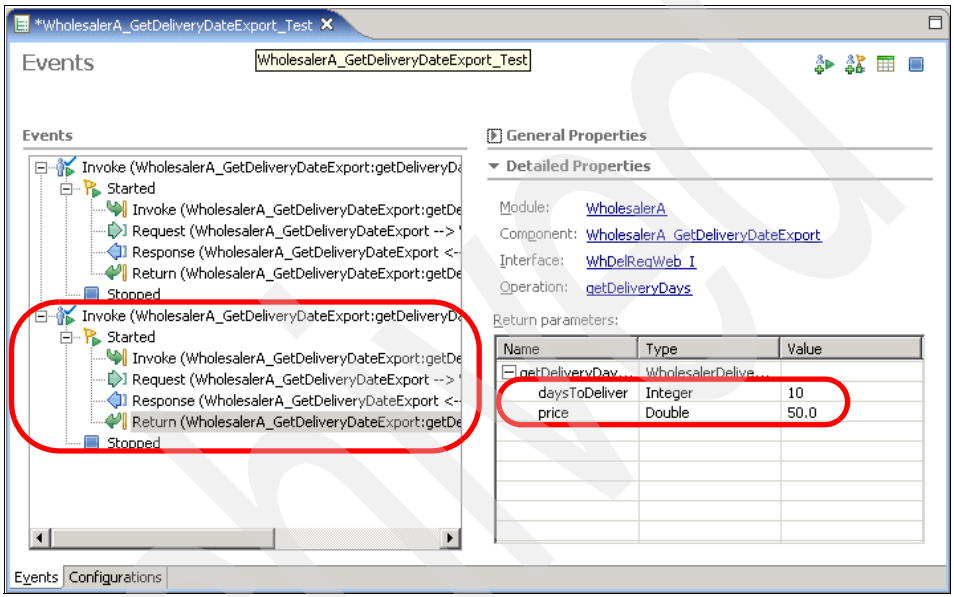


Figure A-81 A second test of the Get Delivery Date functionality

9. Optionally, you can test the other components the same way. Right-click them in the Assembly Editor and select **Test Component**. Remember to remove the emulators before testing them.

Creating the service implementation for Wholesaler B

Now we implement Wholesaler B. The business logic for this wholesaler is as a Web service. Wholesaler B implements the same three business functionalities as Wholesaler A, but it implements them in Java, not as a WS-BPEL process or business rule:

- Place Order (implemented as Java class)
- Get Delivery Days (implemented as Java class)
- Cancel Order (implemented as Java class)

Wholesaler B also implements the same interfaces as Wholesaler A:

- Get delivery days
- Place an order
- Cancel an order

We can reuse the interfaces from Wholesaler A shown in Table A-5.

Table A-5 Wholesaler B interfaces

	Get delivery days	**Place order**	**Cancel order**
Name	WhDelReqWeb_I	WhPlaceOrdWeb_I	WhCnclOrdWeb_I
Operation	getDeliveryDays	placeOrder	cancelOrder
Input	getDeliveryDays WholesalerRequest	placeOrderRequest	cancelOrderRequest
Output	getDeliveryDays WholesalerResponse	PlaceOrderResponse	cancelOrderResponse
Fault	cannotFulfillOrder	cannotFulfillOrder	N/A

Table A-6, Table A-7, and Table A-8 show the business objects that are used in the interfaces listed in Table A-5. The interfaces and business objects are *identical* to Wholesaler A.

Table A-6 Business objects of the get delivery interface WhDelReqWeb_I

	Name	**Business object**
Input	getDeliveryDaysWholesalerRequest	WholesalerDeliveryDaysRequest
Output	getDeliveryDaysWholesalerResponse	WholesalerDeliveryDaysResponse
Fault	cannotFulfillOrder	WholesalerCannotFulfillOrder

Table A-7 Business objects of the place order interface WhPlaceOrdWeb_I

	Name	**Business object**
Input	placeOrderRequest	WholesalerPlaceOrderRequest
Output	PlaceOrderResponse	WholesalerPlaceOrderResponse
Fault	cannotFulfillOrder	WholesalerCannotFulfillOrder

Archived

Table A-8 Business objects of the cancel order interface WhCnclOrdWeb_I

	Name	Business object
Input	cancelOrderRequest	WholesalerCancelOrderRequest
Output	cancelOrderResponse	WholesalerCancelOrderResponse

Copying the business objects and interfaces from WholesalerA

We need to copy the following business objects from WholesalerA to WholesalerB:

- WholesalerCancelOrderRequest
- WholesalerCancelOrderResponse
- WholesalerCannotFulfillOrder
- WholesalerDeliveryDaysRequest
- WholesalerDeliveryDaysResponse
- WholesalerPlaceOrderRequest
- WholesalerPlaceOrderResponse

We need to copy the following interfaces from WholesalerA to WholesalerB:

- WhDelReqWeb_I
- WhPlaceOrdWeb_I
- WhCnclOrdWeb_I

Note: Because WholesalerA and WholesalerB are separate entities, we copy the artifacts from the WholesalerA module to the WholesalerB module. Alternatively we could use a shared library project to avoid duplication.

1. In the Business Integration perspective of WebSphere Integration Developer create a new module called `WholesalerB`. This module has no dependencies.
2. To copy the business objects, highlight the business objects in the WholesalerA module, right-click, and select **Copy**.
3. Right-click the **Data Types** folder in the WholesalerB module and select **Paste**. Notice that the Namespaces are the same as for WholesalerA.
4. Repeat this process to copy the interfaces from the WholesalerA module and paste them into the Interfaces folder of the WholesalerB module.

Creating the business logic using the assembly diagram

Wholesaler B implements its services in Java. Using the assembly diagram, WebSphere Integration Developer generates the Java class and method signatures for our service implementation (Figure A-82).

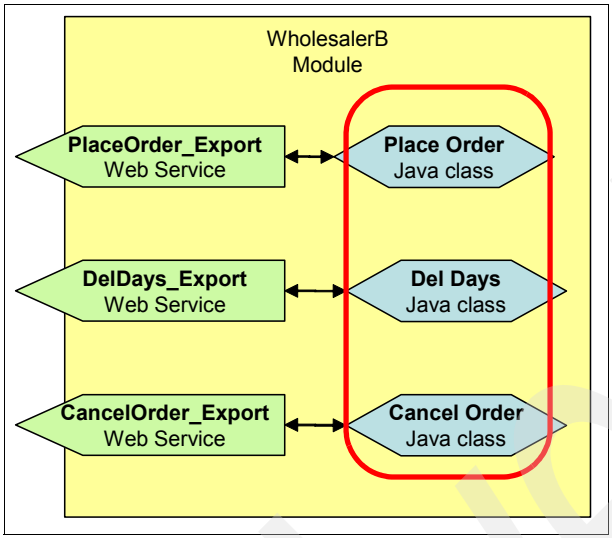


Figure A-82 Business logic for Wholesaler B

Note: You do not need to know Java to create this component. We provide step-by-step instructions and pre-built code for you to use.

Creating the Java component

Add a Java component to the assembly diagram as follows:

1. Open the assembly diagram for Wholesaler B.
2. Drag the **Java Component** icon ( ) from the palette to the assembly diagram canvas. This creates a Java component named *Component1*.
3. Rename Component1 to `WholesalerB` (Figure A-83).

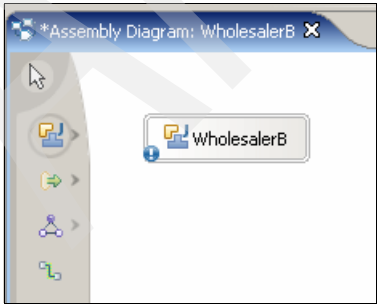


Figure A-83 Java component WholesalerB

Archived

Adding the interfaces to the Java component

Follow these steps:

1. Right-click the Java component **WholesalerB** and select **Add** → **Interface**.
2. In the Add Interface window (Figure A-84), you see all WSDL and Java interfaces.
   a. Because our interfaces are WSDL interfaces, click the **Show WSDL** radio button to display only the WSDL interfaces.
   b. Select **WhDelReqWeb_I** and click **OK**.
   c. Using the same method, add the following interfaces to the WholesalerB Java component:
      - WhPlaceOrdWeb_I
      - WhCnclOrdWeb_I

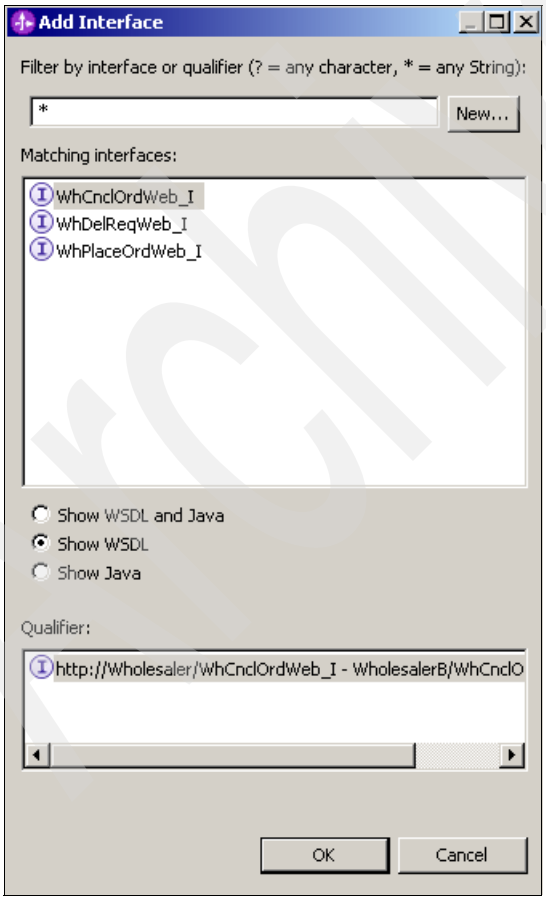


Figure A-84 Add Interface window displaying only WSDL interfaces

Archived

3. Click the **Properties** tab and the **Details** tab to verify that the interfaces are added to the Java component WholesalerB (Figure A-85).

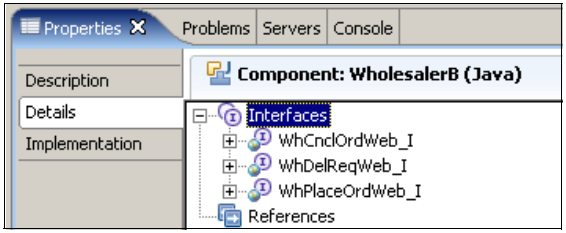


Figure A-85 Interfaces of the Java component WholesalerB

Generating the implementation for the Java component

Now follow these steps:

1. Right-click the **WholesalerB** Java component and select **Generate Implementation**.
2. In the Generate Implementation window, complete these steps:
   a. Click **New Package**.
   b. In Package name field, type `com.ibm.itso.wholesalerB` and click **OK**.
   a. In the Generate Implementation window (Figure A-86), select **com.ibm.itso.wholesalerB** and click **OK**.

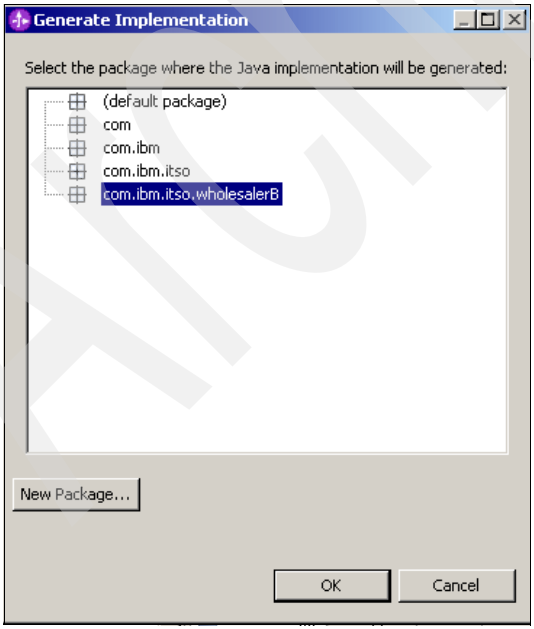


Figure A-86 Selecting the com.ibm.itso.WholesalerB package

Archived

3. A Java class WholesalerBImpl.java is generated and should open in the Java Editor. This Java class contains three methods:
   - getDeliveryDays
   - placeOrder
   - cancelOrder

   Populate this Java class with business logic. The complete code for this Java class is provided in the \Wholesalers\WholesalerB\ directory in the additional material supplied with this book. To add the code, use one of the following options:
   - Paste the code from the additional material into WholesalerBImpl.java.
   - Implement the getDeliveryDays method as shown in Example A-1. This code sets deliveryDays to a value of 10 if part1 is ordered. Otherwise it sets deliveryDays to a value of 5.

Example: A-1 getDeliveryDays Java code

```
public DataObject getDeliveryDays(
       DataObject getDeliveryDaysWholesalerRequest)
       throws ServiceBusinessException {
   System.out.println("****STARTING getDeliveryDays for WholesalerB****");

   int deliveryDays; //Holds the delivery days to be returned
   String partNo=getDeliveryDaysWholesalerRequest.getString("partNo"); //holds the part number

   //If part1, 10 days to deliver, else 5 days to deliver
   if(partNo.equalsIgnoreCase("part1"))
   {
      deliveryDays=10;
   }
   else
   {
      deliveryDays=5;
   }

   //Create a new Business Object to hold result
   commonj.sdo.DataObject deliveryDateBO;

   //Create a factory for new business objects
   com.ibm.websphere.bo.BOFactory factory =
         (com.ibm.websphere.bo.BOFactory) new
com.ibm.websphere.sca.ServiceManager().locateService("com/ibm/websphere/bo/BOFactory");
   //Create an instance of type "WholesalerDeliveryDaysResponse" - note the namespace
   deliveryDateBO = factory.create("http://Wholesaler", "WholesalerDeliveryDaysResponse");

   //set the result fields - daysToDeliver and price in the returned BO. Price is 7.8 times
the quantity
```

```
   deliveryDateBO.setInt("daysToDeliver", deliveryDays);
   deliveryDateBO.setDouble("price", 7.8*getDeliveryDaysWholesalerRequest.getInt("quantity"));

   System.out.println("****ENDING getDeliveryDays for WholesalerB****");

   //Return the delivery days back to the service consumer
   return deliveryDateBO;
}
```

4. Implement the placeOrder method as shown in Example A-2. This code sets the confirmCode to a string of order54.

Example: A-2 placeOrder Java code

```
public DataObject placeOrder(DataObject placeOrderRequest)
       throws ServiceBusinessException {
   System.out.println("****STARTING placeOrder for WholesalerB****");

   //Create a new Business Object to hold the result
   commonj.sdo.DataObject placeOrderNotification;

   //Create a factory for new business objects
   com.ibm.websphere.bo.BOFactory factory =
         (com.ibm.websphere.bo.BOFactory) new
com.ibm.websphere.sca.ServiceManager().locateService("com/ibm/websphere/bo/BOFactory");

   //Create an instance of type "WholesalerPlaceOrderResponse"
   placeOrderNotification = factory.create("http://Wholesaler",
"WholesalerPlaceOrderResponse");

   //set the result field to a hard-coded order number of 'order54'
   placeOrderNotification.setString("confirmCode","order54");

   System.out.println("****ENDING placeOrder for WholesalerB****");

   //Return the confirmation code back to the service consumer
   return placeOrderNotification;
}
```

Archived

5. Implement the cancelOrder method as shown in Example A-3. This sets the orderCancelled boolean to *true*.

Example: A-3 cancelOrder Java code

```
public DataObject cancelOrder(DataObject cancelOrderRequest) {
   System.out.println("****STARTING cancelOrder for WholesalerB****");

   //Create a factory for new business objects
   com.ibm.websphere.bo.BOFactory boFactory = (com.ibm.websphere.bo.BOFactory)
ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/BOFactory");

   //Create an instance of type "WholesalerCancelOrderResponse" - note the namespace
   DataObject response = boFactory.create("http://Wholesaler",
"WholesalerCancelOrderResponse");

   //Set the response to 'true' i.e. we can always cancel the order
   //Note that we have to use the Boolean class, not the boolean base type here
   response.set(0, new Boolean(true));

   System.out.println("****ENDING cancelOrder for WholesalerB****");

   //Return the response code back to the service consumer
   return response;
}
```

6. Press Ctrl+S to save WholesalerBImpl.java, and then close the Java Editor.
7. Press Ctrl+S to save the WholesalerB assembly diagram.

In the Business Logic folder of the WholesalerB module, there is now a new Java folder. Inside the Java folder, you can now see the WholesalerBImpl.java file (Figure A-87).

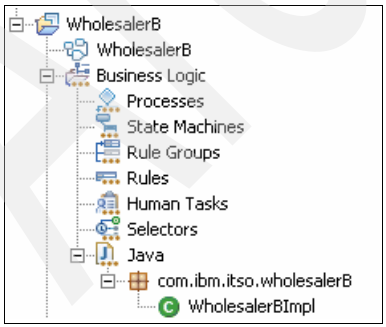


Figure A-87 File WholesalerBImpl.java

Creating the Web services using the assembly diagram

We now provide Web services interfaces for the generated Java class. These Web services exports are used by consumers to invoke the Java business logic (Figure A-88).

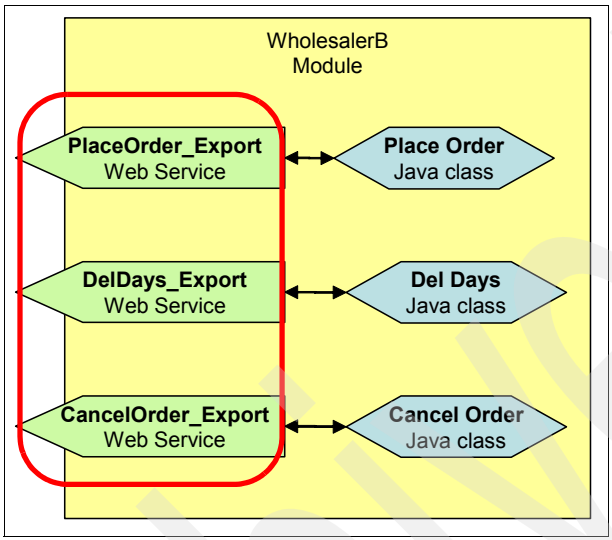


Figure A-88 Web services exports for WholesalerB

1. Because we want three separate WSDL bindings for our Web service, we create three exports. Right-click the **assembly diagram canvas** and select **Add Node → Export**.
2. Rename the Export1 component to `WhBExpDelv` (Figure A-89).

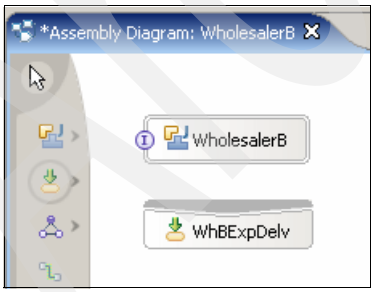


Figure A-89 Export component WhBExpDelv

3. Right-click **WhBExpDelv** and select **Add Interface**.
4. Select the WSDL interface **WhDelReqWeb_I** and click **OK**.

Archived

5. Right-click **WhBExpDelv** and select **Generate Binding** → **Web Service Binding**.
6. In the Binding File Generation window (Figure A-90), select **Yes**.

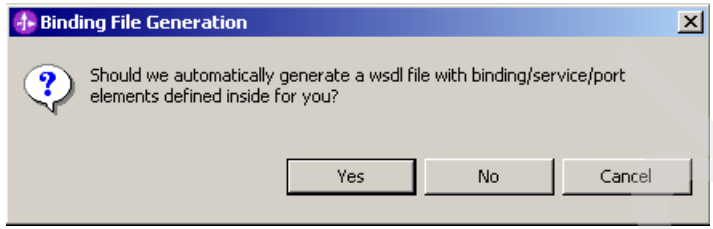


Figure A-90 Binding File Generation window

7. In the Select Transport window (Figure A-91), select **soap/http** and click **OK**.

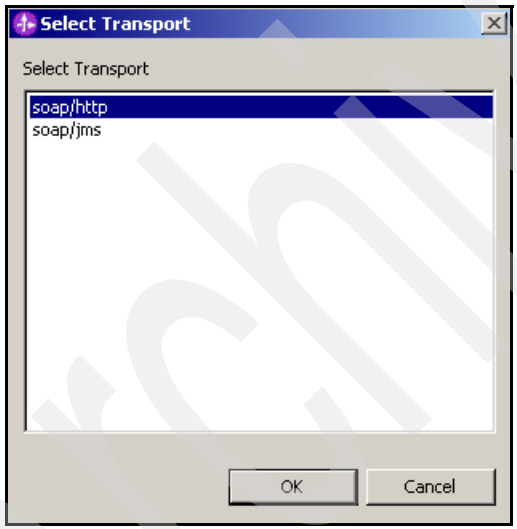


Figure A-91 Select Transport window

8. Right-click **WhBExpDelv** and select **Wire to Existing**.

Archived

Figure A-92 shows how the assembly diagram looks after making these changes.

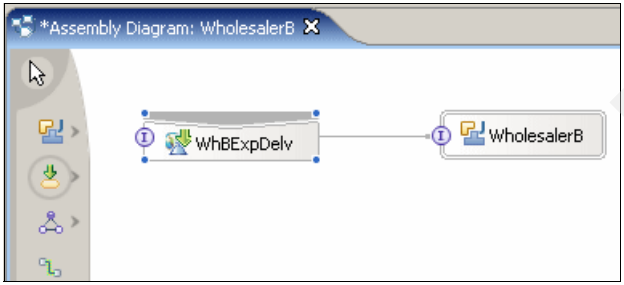


Figure A-92 Wired components

Notice that a WSDL file is generated in the folder Web Service Ports (Figure A-93).

Figure A-93 Generated WSDL file

9. Add another Export node to the assembly diagram canvas, and name it `WhBExpPlaceOrd`.
10. Add the **WhPlaceOrdWeb_I** `interface` to the assembly diagram canvas.
11. Generate a Web service binding using the SOAP/HTTP transport, and wire it to the WholesalerB component (Figure A-94).

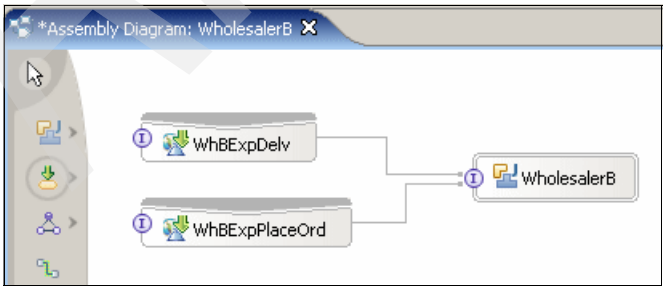


Figure A-94 Wired components

Archived

12. Add one more Export node called `WhBExpCnclOrd`. Add the **WhCnclOrdWeb_I** to it, generate a Web service binding using the SOAP/HTTP transport, and wire it to the WholesalerB component (Figure A-95).

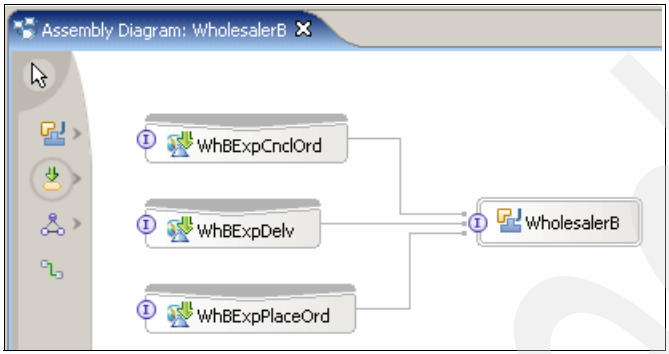


Figure A-95 Wired components

13. Press Ctrl+S to save the assembly diagram, and close the Assembly Editor.

Archived

Notice that three WSDL files are generated in the Web Service Ports folder. All artifacts for the WholesalerB component are shown in Figure A-96. You can optionally test these three Web services using the Integration Test Client, in a similar way as described in “Testing Wholesaler A” on page 552.

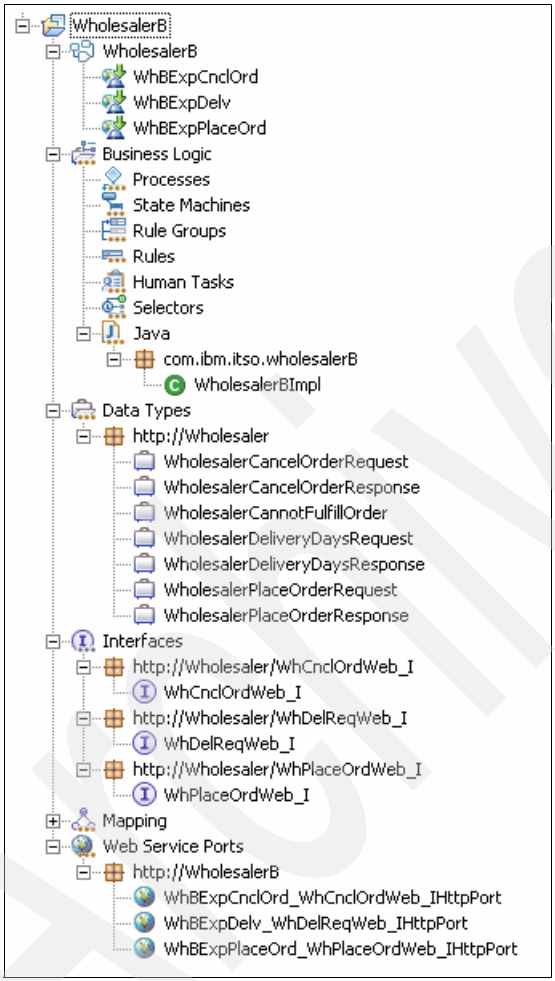


Figure A-96 The WholesalerB component with its artifacts

Archived

Archived

B

Deploying and testing modules

In this appendix, we discuss multiple ways to deploy and test your solution using WebSphere Integration Developer and WebSphere Process Server. There are a few equivalent ways to deploy a module to the server:

- The Integration Test Client
- The Add/Remove programs wizard
- The administration console

The tool also provides a few ways to invoke components to test the solution:

- The Integration Test Client
- The Business Process Explorer
- The Web Services Explorer

Five managers in WebSphere Process Server can be used with specific deployed components:

- The Business Rules Manager
- The administrative console (for selectors)
- The Common Base Event browser
- The Failed Event Manager
- The Relationship Manager

© Copyright IBM Corp. 2006. All rights reserved.

In this appendix, we also discuss server configurations for security and the Java Database Connectivity (JDBC) J2EE Connector architecture (JCA) 1.5 Adapter. Plus we explain component support for debugging.

Deployment

There are various ways that you can deploy modules, from tooling to run time, as explained in the following sections.

Important: In Version 6.0, a single server profile is used to for all workspaces that use the test environment. This means that projects that were added to the server, but never removed in a previous workspace, could cause your console to show FileNotFound errors when starting. To avoid these messages, always remove your applications from the server when closing an old workspace and starting a new one.

Starting run time and deploying modules to the test environment using the Integration Test Client

Use the Integration Test Client to start your run time and deploy your modules to the test environment. You can do this by choosing one of the following actions:

- In the Business Integration view, right-click one or more module projects and click **Test** → **Test Module**.
- In the Assembly Editor, right-click the **Canvas** and click **Test Module**.
- In the Assembly Editor, right-click one or more components and click **Test Component**.

Archived

Regardless of the option you choose, the test client (Figure B-1) opens. You select a **module**, **component**, **interface**, and **operation**. Enter the data to use for the test and then click **Continue**.

> **Important:** Projects are not deployed to the run time until you click the **Continue** button.

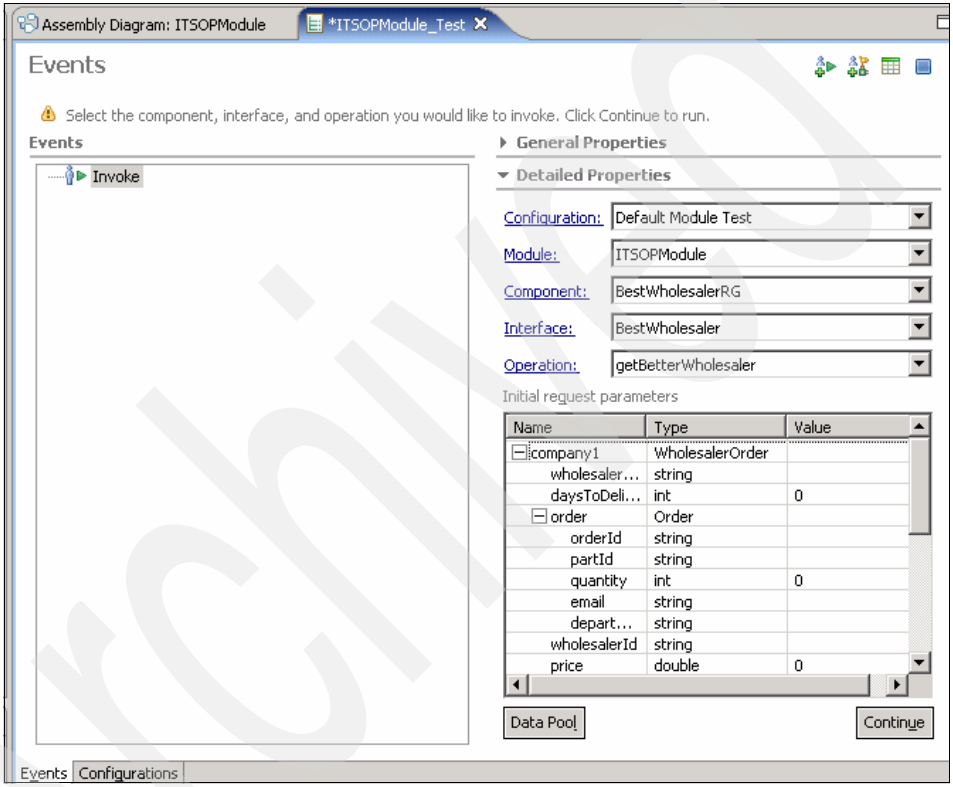


Figure B-1 The Integration Test Client

Archived

In the Deployment Location window (Figure B-2), you can choose the server that the modules will be deployed on. You can also choose the mode in which the run time will start. In the case where multiple modules are selected, each module can be deployed onto a different server.

Tip: In order for multiple servers to be available for deployment, multiple servers must exist in the Servers view.

Click the **Finish** button to begin the deployment.

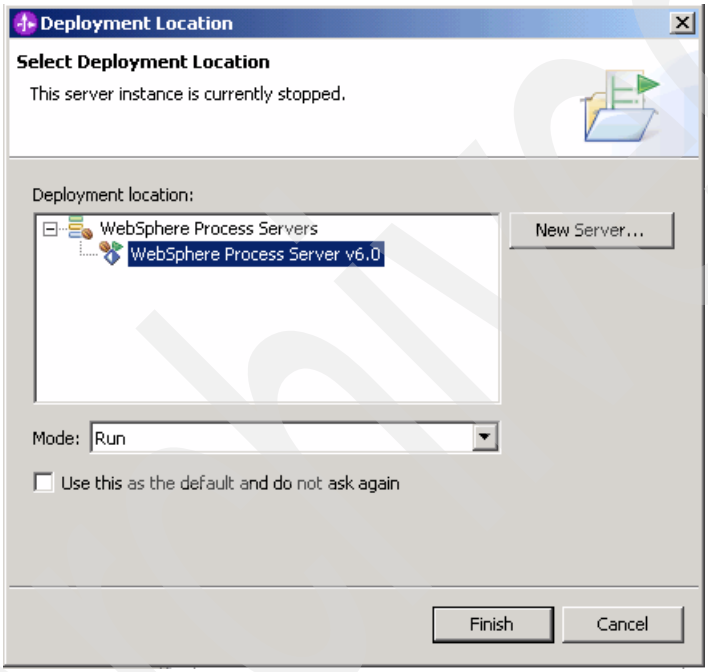


Figure B-2 The Deployment location window

Tip: To start the server manually:

1. Switch to the Business Integration perspective.
2. Open the **Servers** view.
3. Right-click the ***server*** and click **Run**.

After the progress bars are complete, the modules are deployed and available.

Tips:

- It is possible that publishing to the server will fail. Should that occur, consult the server's console output. If the error does not make sense, try the following workaround:

  a. Remove your application from the run time.
  b. Shut down the run time.
  c. Restart WebSphere Integration Developer.
  d. From the menu toolbar, click **Projects** → **Clean**. Click **OK**.
  e. Start your run time.
  f. Add your projects back to the run time.

- The default location of the Test Environment system log is WID_INSTALL_PATH/prof/logs/server1/SystemOut.log.

This appendix addresses the behavior of the test client later. For now, it is sufficient to understand that the test client can automatically start your run times and deploy your modules.

Important: It is not possible to uninstall an application from the Test Client. You have to either use the Add/Remove wizard or the administrative console to uninstall them.

Adding and removing projects to the server

You can use the Add/Remove Projects wizard to publish to the run time. This wizard is available in the Servers view which is shown by default in the Business Integration perspective. To start the wizard:

1. Right-click the ***server instance*** you want to deploy to, and select click **Add/Remove Projects**.

Archived

2. The Add and Remove Projects wizard (Figure B-3) opens in which you can see all deployable projects. Select the project or projects and click **Add** or **Add All** to move them under the Configured Projects section. All projects listed under Configured Projects are deployed to the server.

> **Important:** Projects are only uninstalled from the server if the wizard has listed them in the Configured Projects section first.

When you want deployment to begin, click **Finish**. A progress window opens and publishing begins. If the run time has not already been started, then it starts for you.

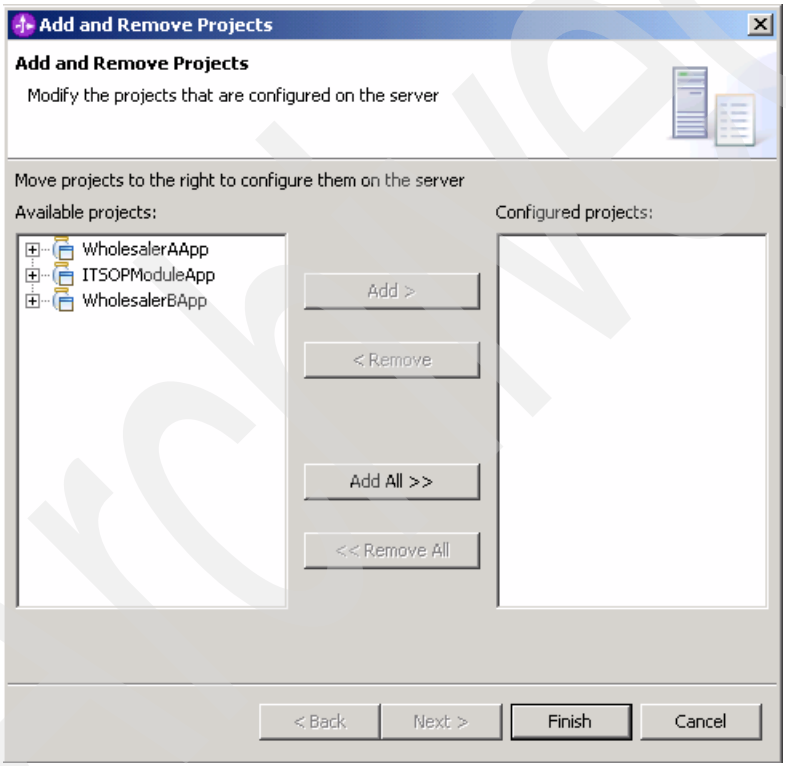


Figure B-3 The Add and Remove Projects wizard

> **Tip:** Sometimes the server may start in debug mode rather than in run mode. Debug mode is slow and should only be used when debugging functionality is required.

Installing an EAR file using the administrative console

The WebSphere administrative console is built into the server and can be accessed in the tool. It is a Web-based front end that is used for configuring a multitude of parameters on the server. To access the administrative console from WebSphere Integration Developer:

1. In the servers view of the Business Integration perspective, right-click the ***server***.
2. Click **Run Administrative Console**.
3. In the Web browser window that opens, showing the administrative console, log in with your user name and password. Then you see the page shown in Figure B-4.

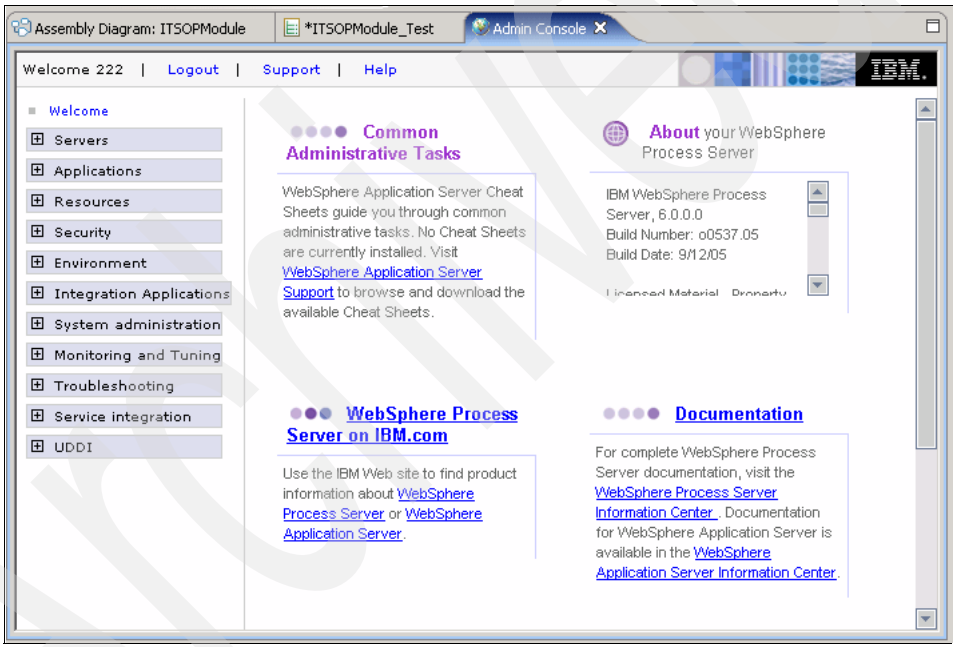


Figure B-4 The Administrative Console

The administrative console only supports installations from EAR files. There is no direct link between the administrative console and the projects with your WebSphere Integration Developer workspace. The following steps presume that you have exported your applications into EAR files:

Archived

1. Expand the **Applications** heading, and click **Install New Application**.
2. Click the **Browse** button and select the ***EAR file*** from the file system. Click **Next**.
3. Configure any custom settings required for your EAR. Unless you know that you need to change some settings, accept the defaults settings. Click **Next**.

> **Tip:** You can click directly on the final step (Summary) hyperlink to accept defaults for all the previous steps. There is no need to continue clicking Next to iterate through all the steps.

4. After you enter the necessary information, in the last window, click **Finish**. The application begins the installation.
5. Check for the `Application has installed successfully` message to confirm that the installation was a success.

Testing

Now that you know how to deploy a set of modules to a run time, you can see how to invoke components to determine if everything is working properly.

> **Tip:** The server system log can become very long. By default, WebSphere Integration Developer only displays the last 80,000 lines. You can turn off this restriction by using the following steps:
>
> 1. Click **Window** → **Preferences**.
> 2. Expand **Run/Debug** and click **Console**.
> 3. Deselect the **Limit console output** check box.
>
> To prevent the console from slowing your system now, periodically clear the output manually.

Archived

The Integration Test Client

The Integration Test Client is an Eclipse-based test client, which allows us to invoke our module at many different stages in its development. We can use it when we have nothing more than a set of unimplemented components in the Assembly Editor. We can also use it when we have a fully implemented solution with multiple import calls using various bindings. All invocations are conducted at the Service Component Architecture (SCA) level, so that both imports and exports can be tested regardless of their bindings.

You can start the Test Client in three different ways, each of which requires you to be in the Business Integration perspective.

- In the Business Integration view, right-click one or more module projects and click **Test** → **Test Module**.
- In the Assembly Editor, right-click the **Canvas** and click **Test Module**.
- In the Assembly Editor, right-click one or more components and clicking **Test Component**.

Running a test

After the test client is open, select a **module**, **component**, **interface**, and **operation**. The appropriate parameters are displayed in the Initial Invocation Parameters section. Enter the appropriate values into the editor and click **Continue** to begin testing.

Tip: There is no input validation in the Integration Test Client. Therefore, the values that you enter must be valid or you will receive a runtime exception.

Important: If you are running tests on a remote (non-WebSphere Integration Developer installed) server, you must apply the Test Client patch to the server. Otherwise, you see the following error message:

```
Error: Apply the integration test client update to the server.
```

The patch is available for download from the WebSphere Process Server support site:

http://www-306.ibm.com/software/integration/wps/support/

Note: If the server goes into production, you must uninstall the Test Client patch. This patch allows the rerouting of messages between SCA components and clients and is a security risk.

Archived

Rerunning a test

You can rerun completed tests. Right-click the **Invoke** event and click **Re-Run** or **Re-Run with Auto Emulate**.

Re-run reinvokes your operation with the specified values and reprompts you for the data to enter should any emulation events occur. Re-run with Auto Emulate does the same thing as Re-Run except it also uses the previously entered values for any emulation events.

Saving execution traces

You can save your test results by using the standard eclipse methods (pressing Ctrl+S). To view test results, double-click the **.wbiexetrace** file or right-click a ***module*** and click **Test** → **Load Execution Trace**.

Entering values

You can enter values in the Integration Test Client in the following ways:

- Click the **Value** cell for the row you want to enter. Use the keyboard to highlight the row you want to enter a value for and press Enter.
- Right-click the row and click **Set Value**.

The value cell is a combination box with three built-in functions (Table B-1).

Table B-1 Integration Test Client operations

Operation name	Explanation
<null>	This operation sets the value in the business object to be null.
<unset>	This operation unsets the given property in the business object.
<new>	This operation is used in the circular dependency case. It creates a new business object for one that is by default set to null in the editor so that you can enter values for it.

Archived

Tips:

- If you have a list, right-click the **row** and click **Add Element** to create a child of the proper type.
- If your operation has an abstract type as input, click the ***type field*** for that row and then click the **...** button. This allows you to choose an implementation type for the abstract type. The Test Client does not restrict you from choosing incorrect implementation types.
- To enter unicode characters, you can specify the \uxxxx escape code and the unicode equivalent is used.

Important: By default, the Test Client sets all values found in the editor on the resultant business object that passes through the wires. This can create a problem for components that expect empty properties to be in the *unset* state. If you require that your fields not be set, be sure to specify **unset**.

Emulators

The Test Client introduces the concept of emulators. *Emulators* allow you to programatically or manually return the value for an invoke as though the actual implementation was used. This is convenient in that you can return various edge cases without creating real implementations to return them.

Emulators are automatically created when:

- There is an unwired reference (case 1).
- There is an import with no binding (case 2).
- There is a wired component with no implementation (case 3).

When a test is started using case 1 or case 2, only those emulators are automatically created. When a test is started via case 3, an emulator is created for every reference that has a selected component.

Important: The rationale behind case 3 is that you want to explicitly test those selected components in isolation. All references on selected components are emulated. When you want to test your entire solution with no emulation (aside from the automatic ones when your wiring diagram is incomplete), choose **Test Module**.

Emulators can also be used for components that are wired and have implementations. You open the Integration Test Client. Then switch the from the Runtime page to the Configurations page.

Figure B-5 shows where emulators can be added and removed. An emulator can be created on a reference and for a component. An emulation created on a reference is used only when that reference is invoked in the components implementation. A component emulation is invoked any time an operation on the component is invoked.

Emulators can also be specified as *manual* or *scripted*. A *manual emulation* creates an event that blocks the invocation while it waits for the user to type the response values. A *scripted emulation* allows the user to create a visual snippet or a Java snippet that will be run.

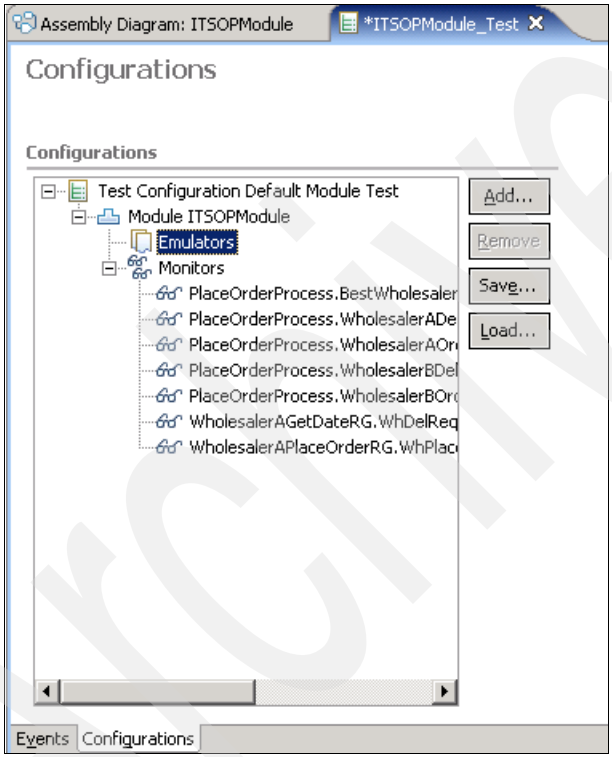


Figure B-5 Emulators on the Integration Test Client Configurations page

Archived

Monitors

The Test Client also introduces the concept of monitors. *Monitors* show the data that flows over a wire between two components. They can show both the outgoing request and the incoming response. You can use this information to debug your application to see, if given a certain request, the component is returning the proper response. By default, monitors are created for all wires in the assembly diagram.

If we look at a completed test, we can see the monitors that appear in the test client. They are the entries in the Events tree, prefaced with *Request* and *Response* (Figure B-6).

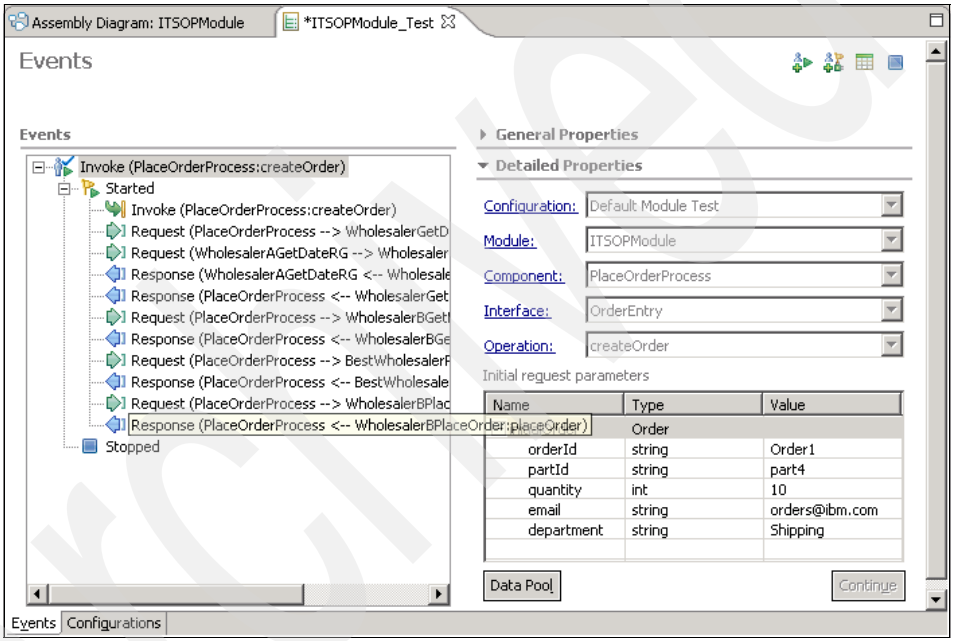


Figure B-6 An example of monitors

Archived

Datapools

Datapools allow you to store commonly keyed values and then use them in manual events. The datapool is global to the entire workspace. Multiple instances of the Integration Test Client can read and write to the pool. Figure B-7 shows an example of the datapool editor.

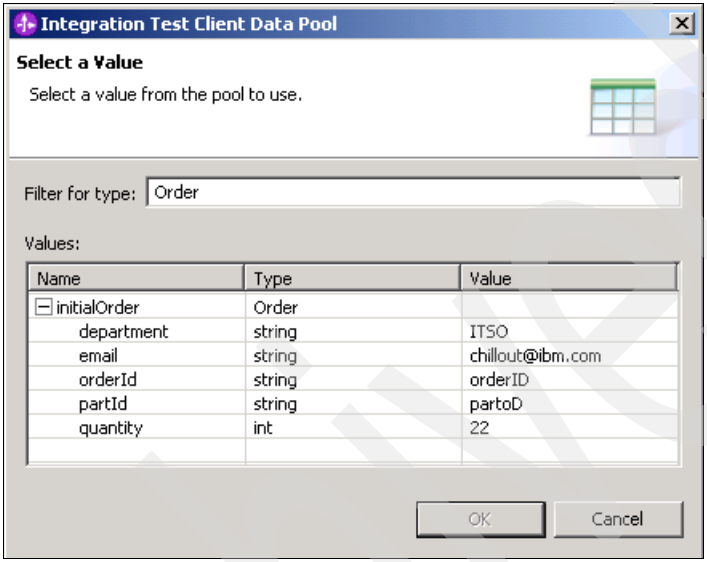


Figure B-7 The datapool editor

To store a value into the datapool editor, right-click the value from the invocation parameters section and click **Add Value To Pool**.

To use a value from the datapool, right-click the row you want to use and click **Use Value From Pool**. Click the value to use and click **OK**. The values are now updated in the event.

> **Restriction:** Values can be added only to the datapool from the Invocation Event and the Emulation Output parameters section.

Archived

Attaching to a module

The Integration Test Client supports the notion of *attach*. Attaching to a module means to connect to an active run time and allow monitors and emulations to catch events. Attach is best used to debug an application that it already deployed and running in your test environment. It then shows you all the communications between modules and allows you to catch requests and emulate them.

To start the attach on a module:

1. In the Business Integration view, right-click a **module** and click **Test** → **Attach**. The Test Client opens.
2. The test configuration that is created by default is the same if the test module was run. If emulators and monitors are created as specified above, you can switch to the configuration page and add or remove emulators as required.

   With attach monitoring, the Test Client is not required to start the test. Any SCA compliant client can start the original invoke. The attach acts in the same manner as though it started the test. It allows the testing of adapters and other asynchronous invocations. As emulation and monitors occur, they are displayed in the client.
3. Click the **Continue** button to begin attach testing for the given module or modules.

Archived

The Business Process Container Explorer

You can use the Business Process Container (BPC) Explorer to start and work with a business process and human tasks. It can start any process template and is not dependent on SCA. The BPC Explorer is also used to handle human tasks and can assign and complete tasks along with dealing with escalations. The BPC Explorer included in WebSphere Integration Developer is intended only as an example of how to use the application programming interfaces (APIs). Do not use it in a production environment.

To start the BPC Explorer:

1. Switch to the Business Integration perspective.
2. Click the **Servers** view tab, and right-click the ***server*** and click **Launch** → **BPC Explorer**.

The BPC Explorer launches a Web browser in WebSphere Integration Developer (Figure B-8).

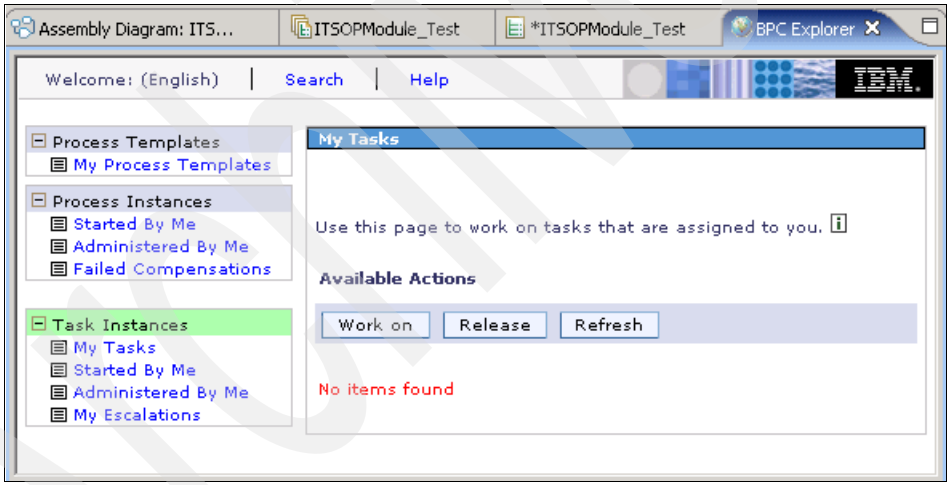


Figure B-8 The BPC Explorer

Archived

Starting a process instance

Follow these steps to start a process instance:

1. Click **My Process Templates**. A list of process templates is displayed.
2. Enable the check box next to the process instance you want to start and click **Start Instance**.
3. As shown in Figure B-9, enter the values to use for invocation and click **Submit**. The process instance is created and the page refreshes with the output.

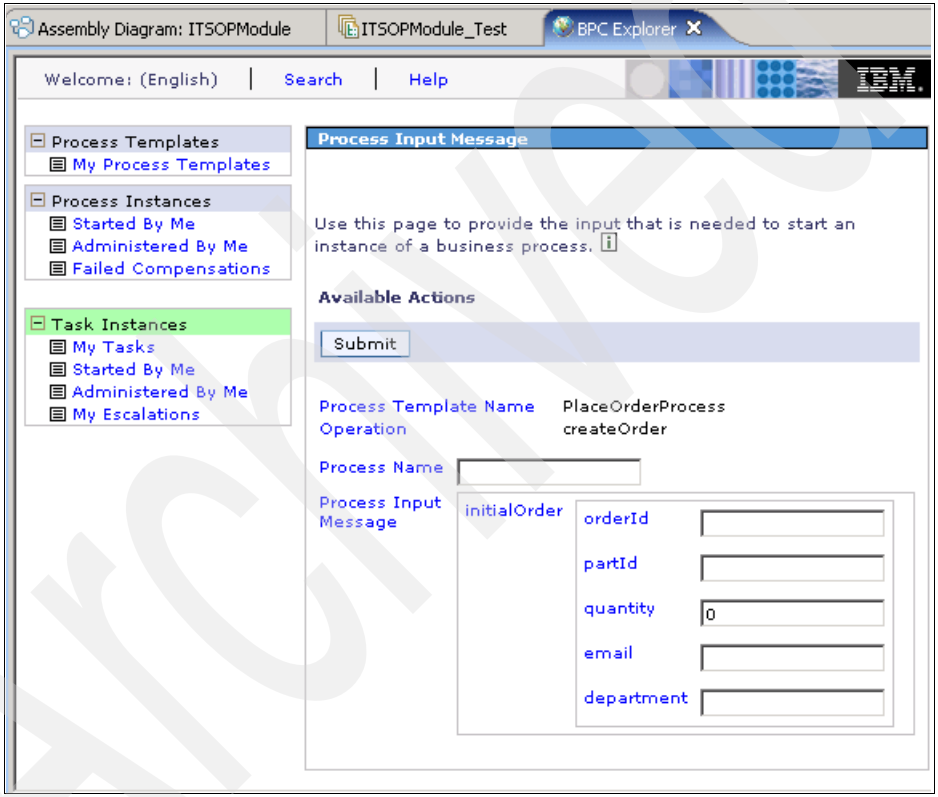


Figure B-9 Input parameters in the BPC Explorer

4. If the process interface is asynchronous, you return to the Process Template page. To see running process instances, click **Started By Me**.

Archived

Working with a human task

The BPC Explorer also allows you to claim human task work items.

1. Optionally log in to the BPC Explorer with the appropriate user credentials to claim the task.
2. Click the **My Tasks** hyperlink.
3. Select the check box next to the work item to claim and click the **Work On** button.

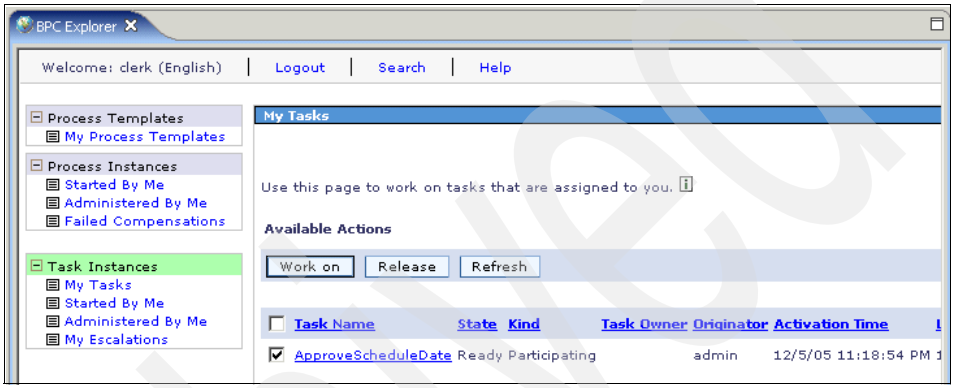


Figure B-10 The My Tasks view

Archived

4. Enter the values for the response objects and click the **Complete** button (Figure B-11).

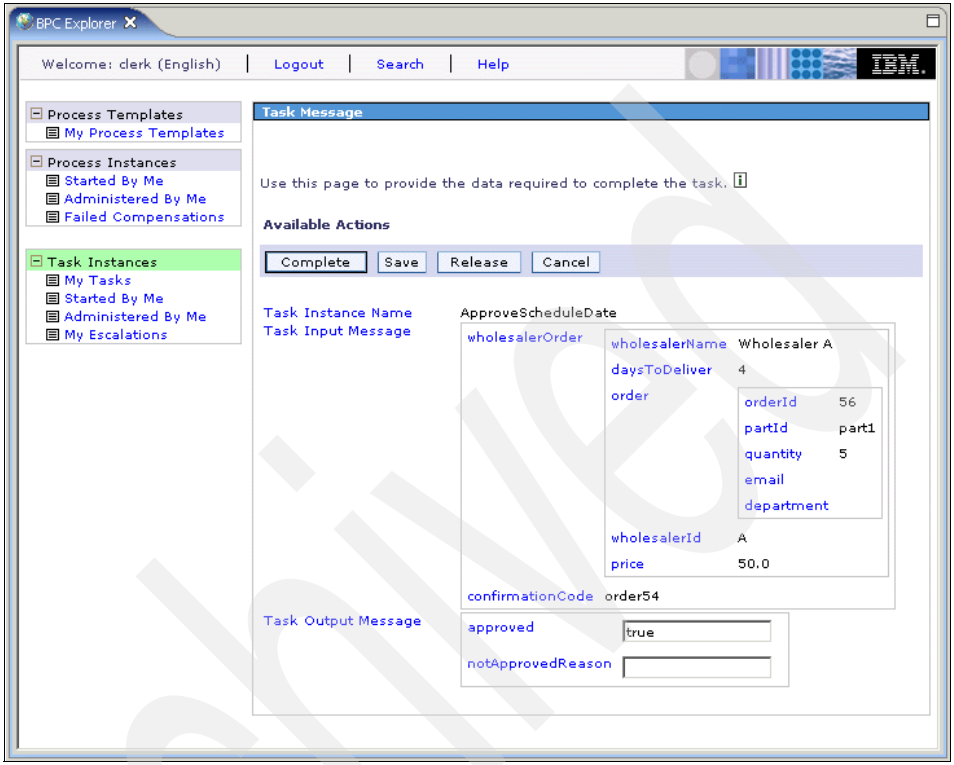


Figure B-11 Completing a work item

The work item is now complete. If a component or process was waiting on the result, it now continues to the next step.

Archived

Managing deployed components

Several other Web-based explorers are provided by WebSphere Process Server as explained in the following sections.

The Business Rules Manager

The Business Rules Manager is a Web-based tool that allows business analysts to dynamically modify deployed business rules on the a running server. This is an optional tool that is not installed by default.

Installing the Business Rules Manager

The Business Rules Manager is not installed to the server by default. It is a powerful Web client that can change runtime behavior. Therefore, server administrators should ensure that it is required in the business scenario and that access to it is restricted.

1. Open a console into the *WID_INSTALL*/runtimes/bi_v6/bin directory.
2. Run the following command to install the Business Rules Manager into the Test Environment. The command conforms to the following specification:

   ```
   wsadmin -f installBRManager.jacl -s servername -n nodename -c cellname -r
   rootname -a applicationname
   ```

   To install into the Test Environment, enter the following command:

   ```
   wsadmin -f installBRManager.jacl
   ```

This script uses the defaults shown in Table B-2.

Table B-2 The defaults used in the installBRManager script

Variable	Value
Server Name	server1
Context Root	br/webclient
Application name	BusinessRulesManager
Target File	brmanager

If the installation was successful, you see the following line:

```
ADMA5013I: Application BusinessRulesManager install successfully.
```

Archived

Starting and using the Business Rules Manager

Start the Business Rules Manager in WebSphere Integration Developer:

1. In the Business Integration perspective, click the **Servers** tab.
2. Right-click the ***server*** and click **Launch** → **Business Rules Manager**.
3. The client starts and shows all the rule groups that are defined on the run time (Figure B-12).

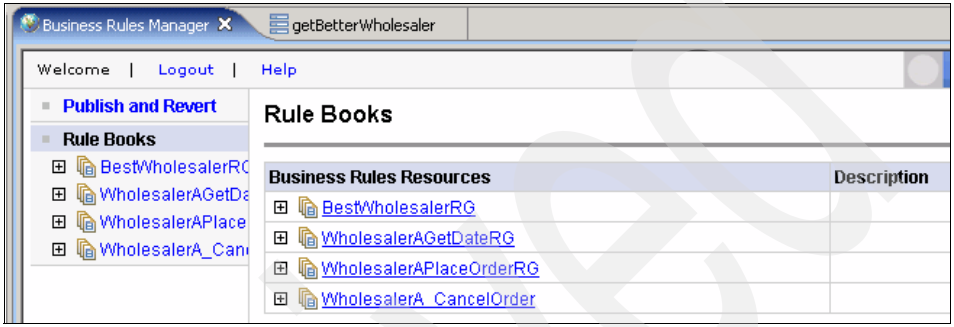


Figure B-12 The Business Rules Web Client

> **Important:** To update the deployed rule sets and decision tables, the rules must be defined using templates. If templates are not used, then these artifacts cannot be updated.

4. Expand a **rule book** to view its operations and any rulesets or decision tables it defines (Figure B-13).

Figure B-13 Listing the operations and rules for a rule book

Archived

5. Click an operation to view details like those shown in Figure B-14.

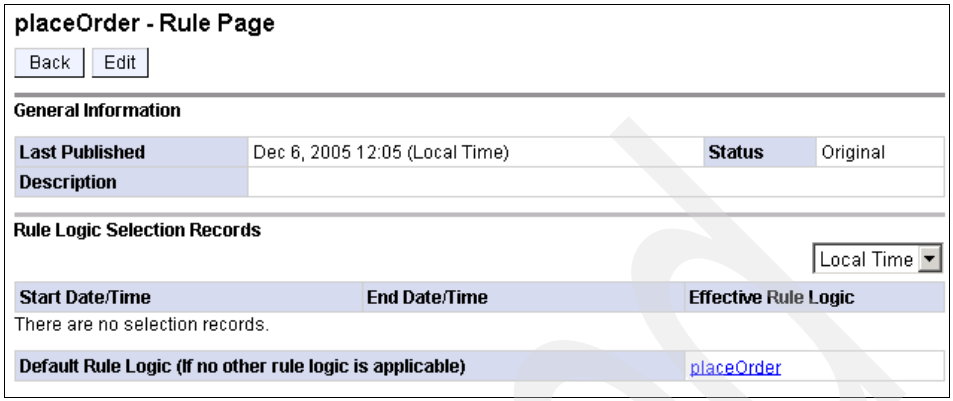


Figure B-14 The operation information page

6. Click the **Edit** button to turn on edit mode where you can make changes to the Selection Records and the Rule Logic (Figure B-15).

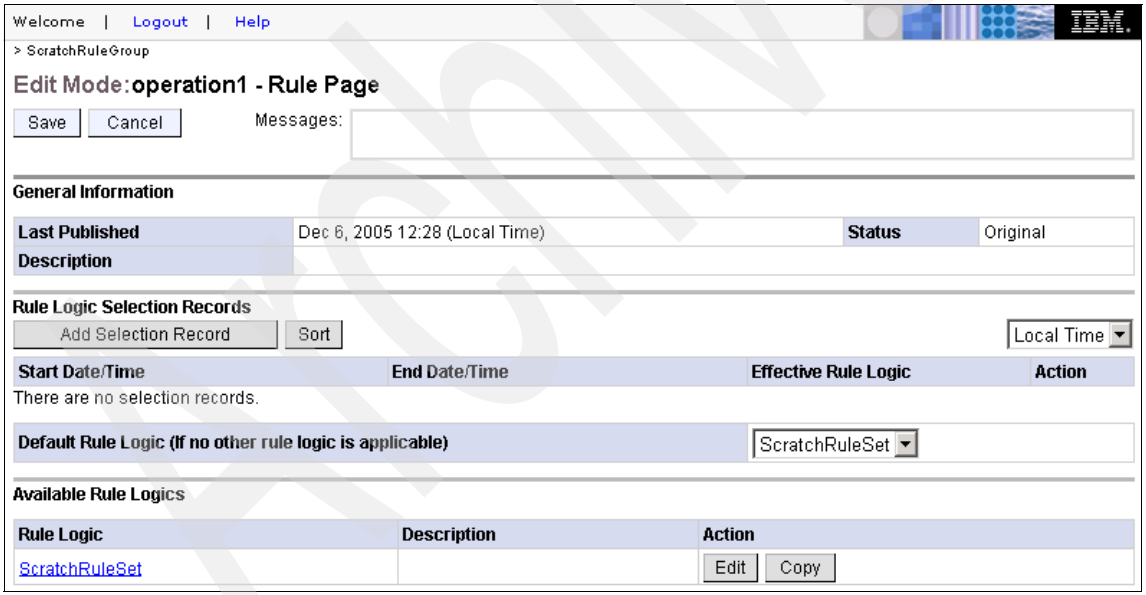


Figure B-15 Operation edit mode

Archived

7. Click **Rule Set** (which has a template) to view information like the example in Figure B-16. This rule uses a template for the 30 parameter. The Manager shows the parameter in bold, which means it is changeable.

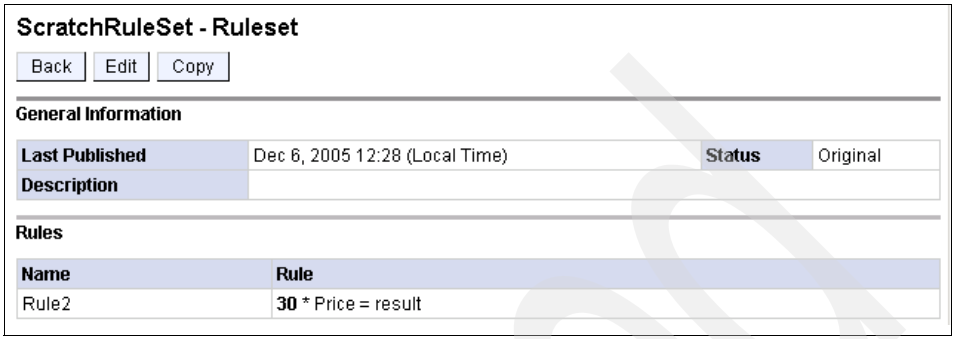


Figure B-16 The Rule Set Editor

8. Click the **Edit** button to update the value (Figure B-17).

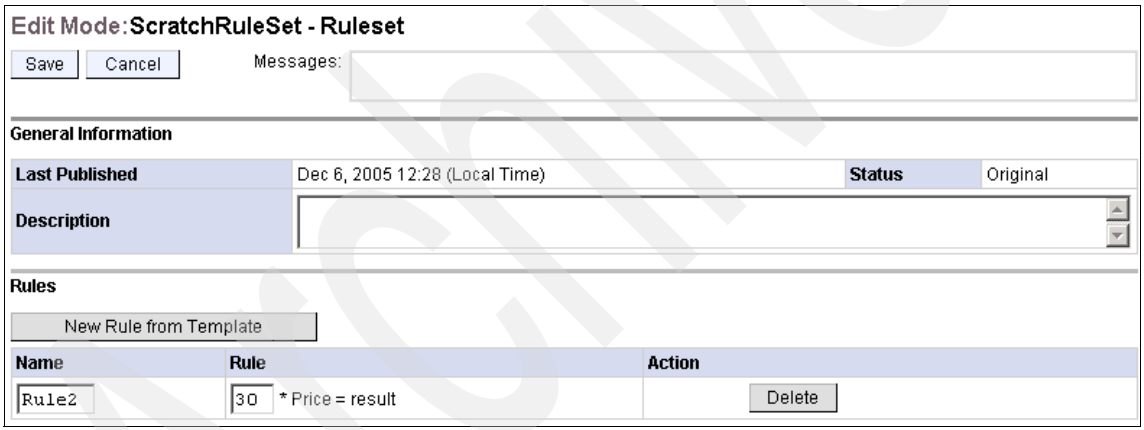


Figure B-17 The Rule Set in edit mode

9. Make any modifications and click **Save**.
10. Click the **Publish and Revert** hyperlink.
11. Click the **Publish** button to commit the changes or click the **Revert** button to discard them.

> **Restriction:** Any changes made to the Rule Group, Rule Set, or Decision Table are not reflected into the artifacts in WebSphere Integration Developer. This means that, if you redeploy your module to the server, you lose any updates made at run time.

Archived

After the changes are published, invoke the component, and you see the new values being used in the run time.

The administrative console for selectors

Selectors have the ability to be dynamically updated on a running server. By default a selector makes a decision based on a date and time. In the administrative console, we can change the behavior to either change the selection date or change the component (or export) that is invoked.

To update the selector in your module:

1. In the Business Integration perspective, click the **Servers** tab.
2. In the Servers view, right-click the **WebSphere Process Server v6.0** server and click **Run administrative console**. The administrative console Web browser opens in WebSphere Integration Developer.
3. In the navigation pane, click **Servers** and click **Application servers**.
4. Click **server1**.
5. On the **Configuration** tab in the Business Integration section, click **Selectors**.
6. From the list of available selectors (Figure B-18), click the selector to work with, and click the selector table to work with.

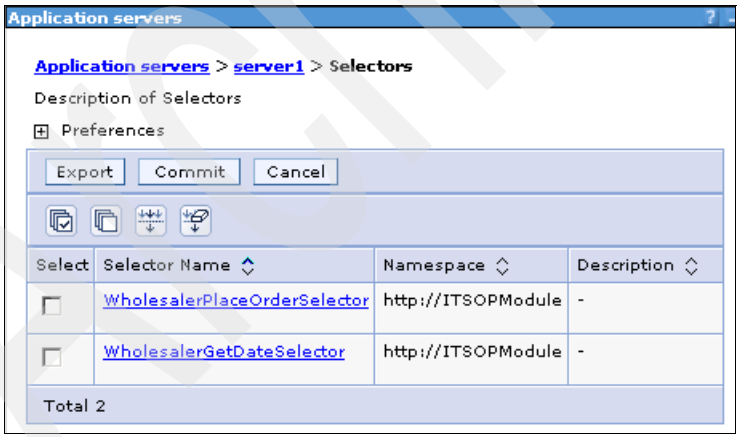


Figure B-18 Selector list in the administrative console

Archived

7. Click the **New** button to create a new destination (Figure B-19).

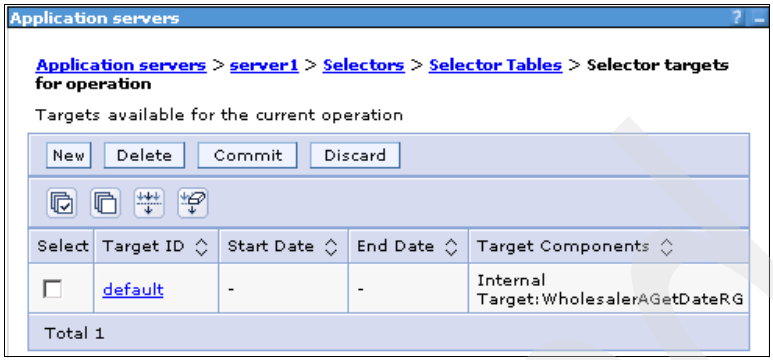


Figure B-19 Defined destinations for a selection

8. As shown in Figure B-20, the form prompts you for a start date and end date and the target component to invoke.

> **Tip:** Selectors can call any export defined on the server. This includes any exports outside the module the selector resides in. This technique is used in Chapter 13, “Parallel Workflow scenario” on page 393, where a new Wholesaler is introduced while the server is still running.

Click the **Apply** button to put the new destination into effect.

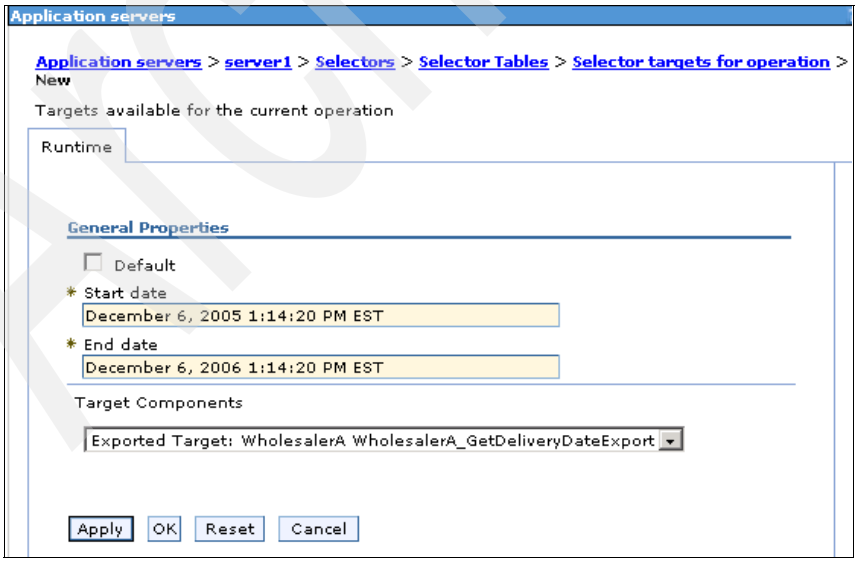


Figure B-20 Required information for a new destination

The ability to update selectors at run time is another powerful tool provided by WebSphere Process Server. The ability to dynamically change the component used, combined with the ability to invoke completely new components, allows staged application deployment without affecting currently executing processes.

> **Restriction:** Any updates made to a selector at run time are not reflected back into the artifact in the tool. You will lose any runtime specific changes if you redeploy your module to the server.

The Common Base Event Event Browser

Common Base Event (CBE) Event Browser allows you to view all the Common Event Infrastructure (CEI) and CBE events that are generated by the server.

Opening the CBE Event Browser

Follow these steps:

1. In the Business Integration perspective, click the **Servers** view.
2. Right-click the ***server*** and click **Launch** → **CBE Event Browser** (Figure B-21).

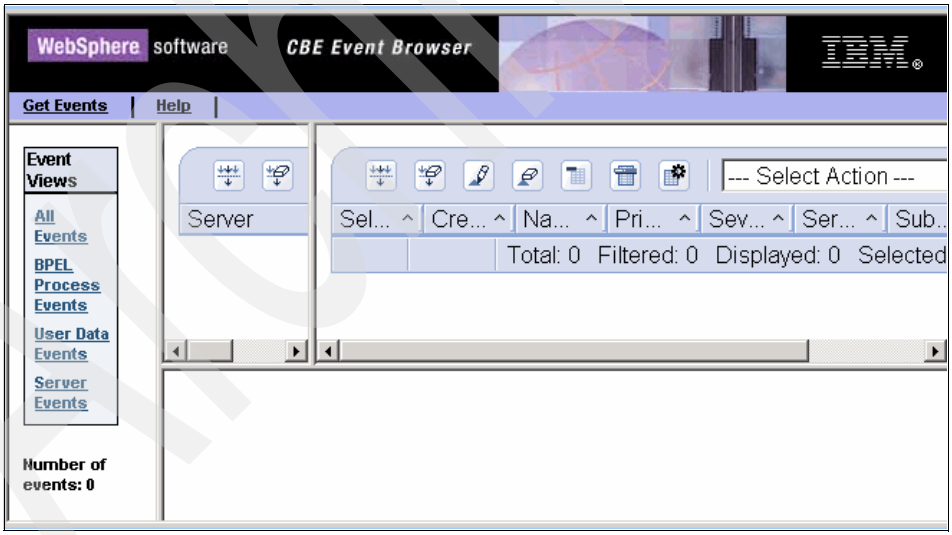


Figure B-21 The CBE Event Browser

Archived

Attention: The CBE Event Browser may be empty even with CEI events enabled in your module. CEI has to be turned on explicitly in the runtime containers of the component implementations.

Enabling CEI events in a business processes

CEI events generated by business processes must be enabled manually in the administrative console.

1. Open the administrative console, and log in.
2. Browse to the **Servers** → **Application Servers** → **server1** → **Business process container settings** → **Business process container** page.
3. Click the **Enable Common Event Infrastructure logging** check box.
4. Click **OK** and save your changes to the master configuration.
5. Restart the server.

Enabling CEI events for human tasks

CEI events generated by human tasks must be enabled manually in the administrative console.

1. Open the administrative console.
2. Browse to the **Servers** → **Application Servers** → **server1** → **Human task container settings** → **Human task container** page.
3. Click the **Enable Common Event Infrastructure logging** check box.
4. Click **OK** and save your changes to the master configuration.
5. Restart the server.

For all other component implementation types, the server automatically turns on CEI logging if it is used.

Working with the CBE Event Browser

To work with the CBE Event Browser and use it to view events that are generated, follow these steps:

1. Invoke a module that generates some events.
2. Open the CBE Event Browser.
3. Click the **Get Events** hyperlink. A page is displayed that allows you to filter the events to retrieve by a variety of methods.
4. Click the **Get Events** button. Notice that the number of events increases after clicking Get Events if any events are found.

Archived

> **Important:** In a business process, any node that is has the *Enable persistence and queries of business-relevant data* check box enabled logs a CEI event when it is executed. By default, all Invoke and Receive/Reply nodes have this check box enabled.

5. Click any of the hyperlinks under Event Views (Figure B-22) to further filter the list of events. Click the view that pertains to the event your module generated.

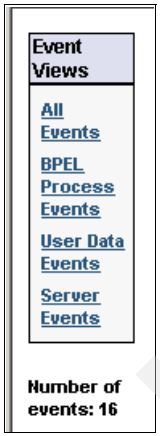


Figure B-22 The Event Views pane

6. Continue to click hyperlinks from left to right to eventually display the events captured in the database (Figure B-23).

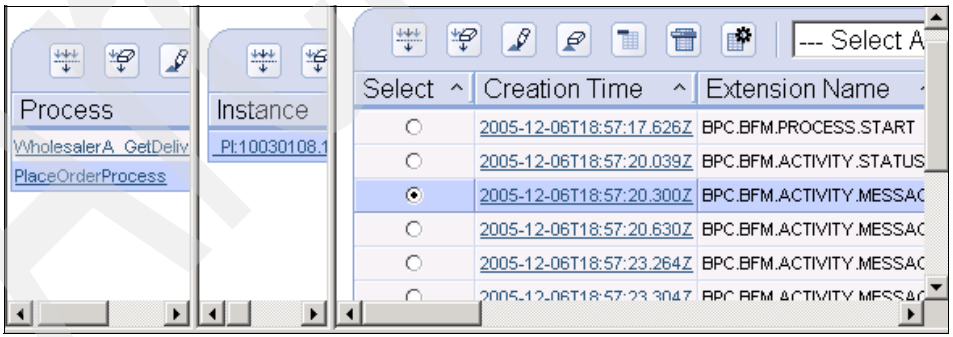


Figure B-23 A sample of the events generated for a process instance

7. Under Select, click the radio button for the event to view it.

Archived

The CBE Event Browser does not offer any way to remove or change the data contained in the events. It is strictly used for viewing the contents of the event database.

The Failed Event Manager

You use the Failed Event Manager to view SCA messages that could not be delivered to their intended destinations. This includes the times where a Web service invoke could not be completed because the service is offline or a Java Message Service (JMS) queue is full. Rather than discard the messages, SCA places them here. With the messages in the Failed Event Manager, the events can be resubmitted or discarded as the system administrator sees fit.

Important: Failed events are created only when asynchronous invocations fail. If your invocation is synchronous, then an exception is thrown but the event is not placed in the failed event manager.

To open the Failed Event Manager:

1. In the Business Integration perspective, click the **Servers** tab.
2. Right-click the ***server*** and click **Launch** → **WPS Failed Event Manager**.
3. In the Failed Event Manger page that opens in a Web browser (Figure B-24), click **Get all failed events**.

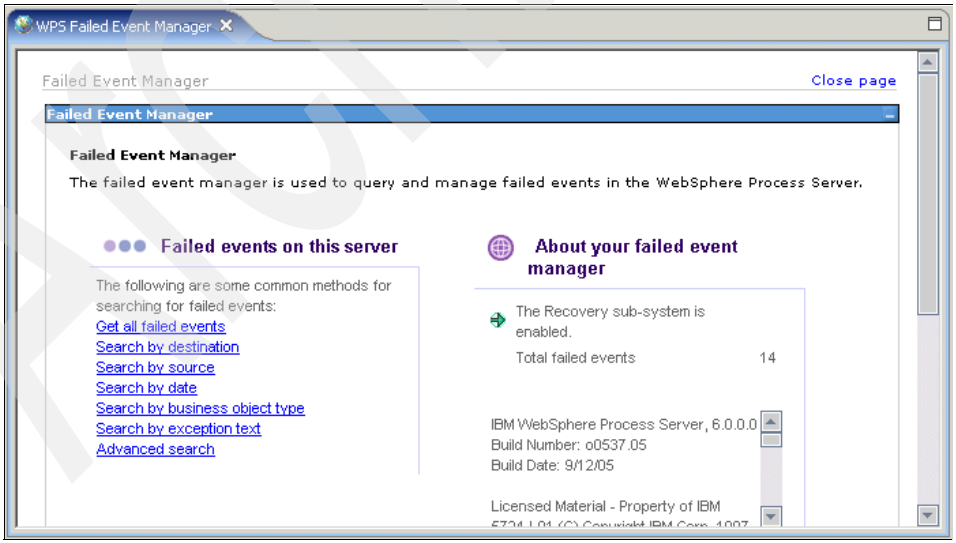


Figure B-24 The WPS Failed Event Manager

4. In the search results page (Figure B-25) that opens, under Message ID, click one of the links.

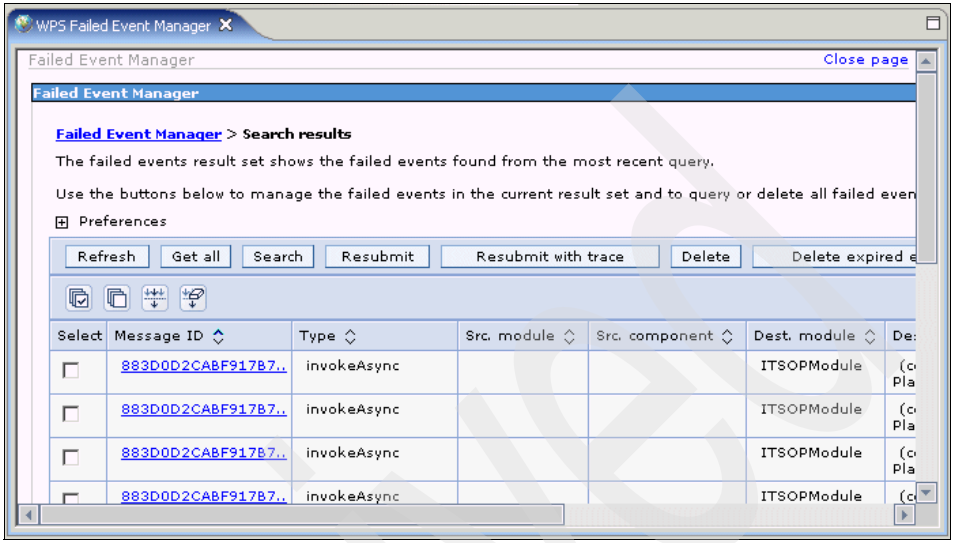


Figure B-25 The WPS Failed Event Manager Search results

Archived

As shown in Figure B-26, you see the details about a failed event.

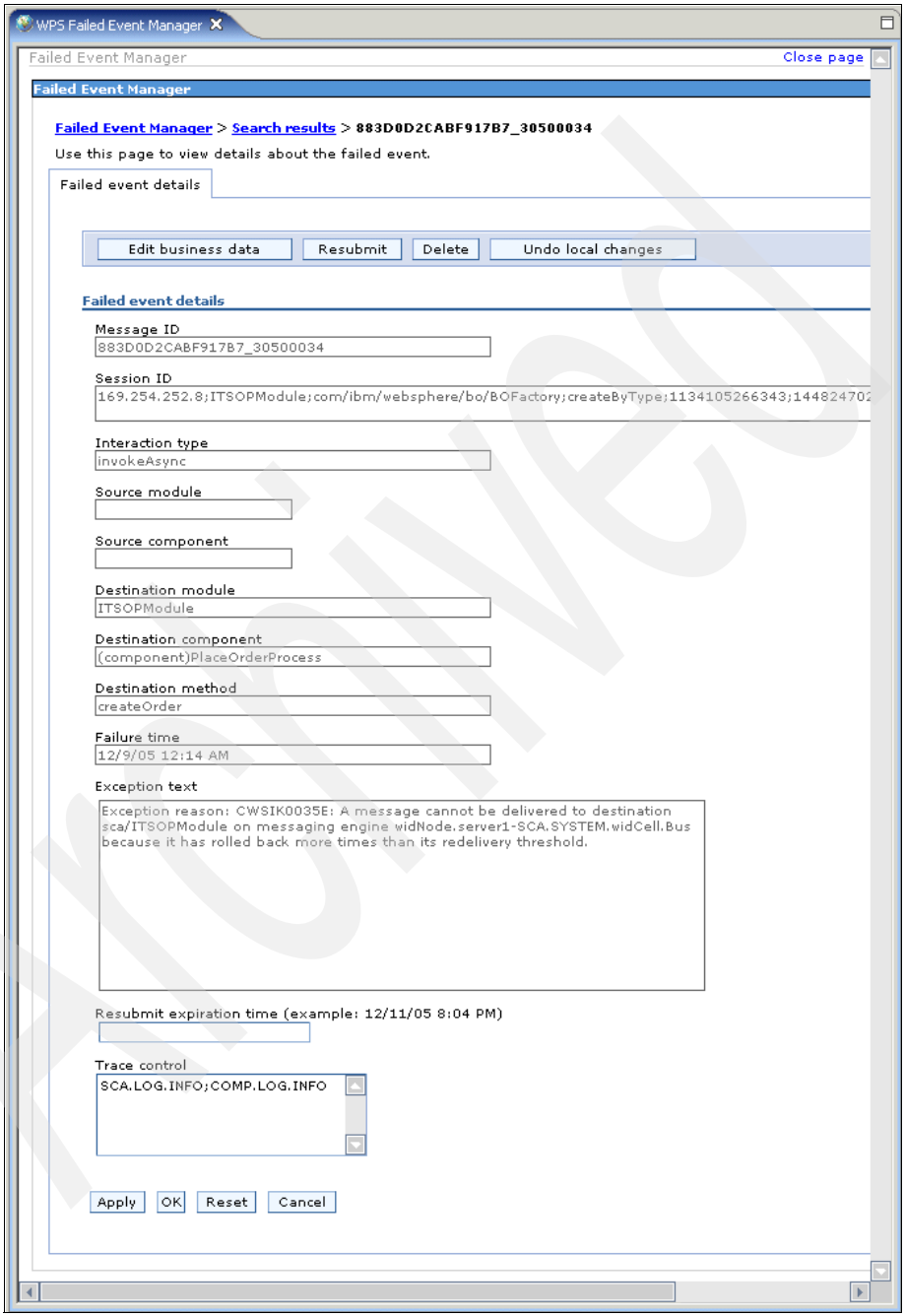


Figure B-26 The WPS Failed Event Manager Event details

The Relationship Manager

The Relationship Manager is used to manage relationships on the server. It is a Web-based client where you can configure relationships.

To open the Relationship Manager:

1. From the Business Integration perspective, click the **Servers** tab.
2. Right-click the ***server*** and click **Launch** → **Relationship Manager**.

The Relationship Manager opens in a browser window (Figure B-27).

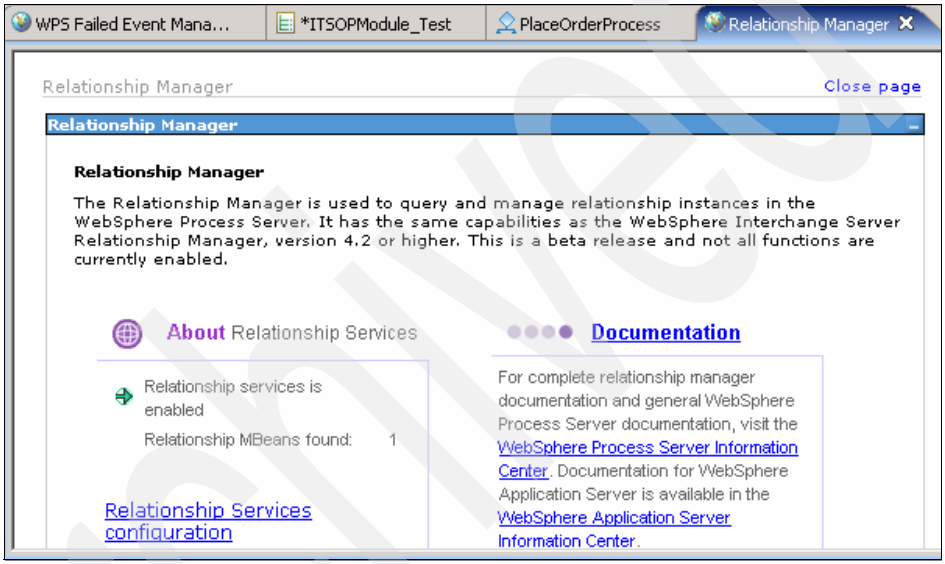


Figure B-27 The Relationship Manager

Archived

Configuring the server

In this section, we guide you through the steps that are required to enable security on the server. We also outline the steps required to connect to DB2 using the WebSphere Adapter for JDBC, which is JCA 1.5 compliant.

Security

To begin, we must to turn on authentication for the run time using LocalOS. We require the user group in Table B-3 on LocalOS.

Table B-3 Required LocalOS user group

Group name	Additional information
wpsadmins	Members of this group are granted the BPESystemAdministrator role and the TaskSystemAdministrator role.

We also require the user accounts in Table B-4 on LocalOS.

Table B-4 Required LocalOS user accounts

User ID	Additional information
wpsadmin	This is used to set up security on the test server. Add wpsadmin as a member of the wpsadmins user group.
clerk	This is used to test the ApproveScheduleDate human task in the Parallel Workflow process.
manager	This is used to test the ApproveScheduleDate human task in the Parallel Workflow process.
admin	This is used to test the ApproveScheduleDate human task in the Parallel Workflow process.

Archived

Setting up LocalOS security on WebSphere Process Server

In this section, we set up LocalOS security on the WebSphere Integration Developer test server.

1. In WebSphere Integration Developer, click the **Servers** page (Figure B-28).

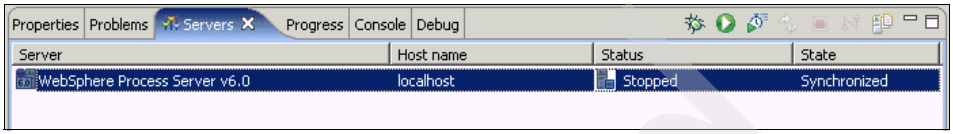


Figure B-28 The Servers page

2. Right-click the **WebSphere Process Server v6.0** server and click **Start**.
3. Wait for the server to start. Look for the `Server server1 open for e-business` message in the console.
4. When the server has started, right-click the **WebSphere Process Server v6.0** server and click **Run administrative console**.
5. In the Admin Console logon window (Figure B-29), enter any user ID and click **Log in**.

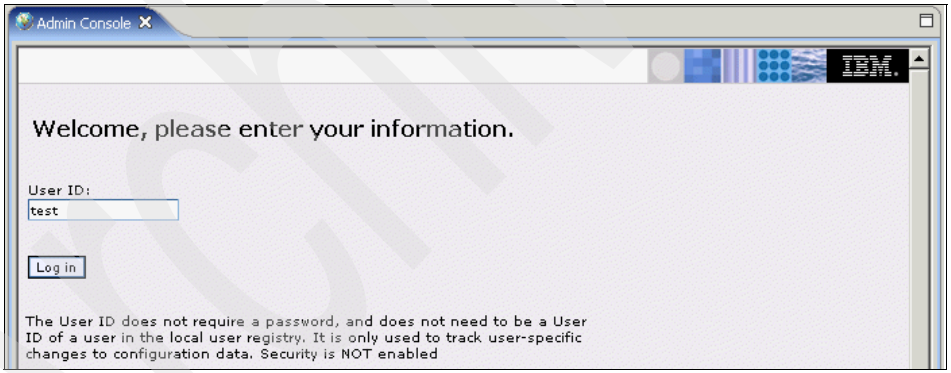


Figure B-29 The Admin Console login without security enabled

6. Click **Security**. Then click **Global security**.

Archived

7. On the right side of the Global security page (Figure B-30), under Authentication, click **JAAS Configuration**.

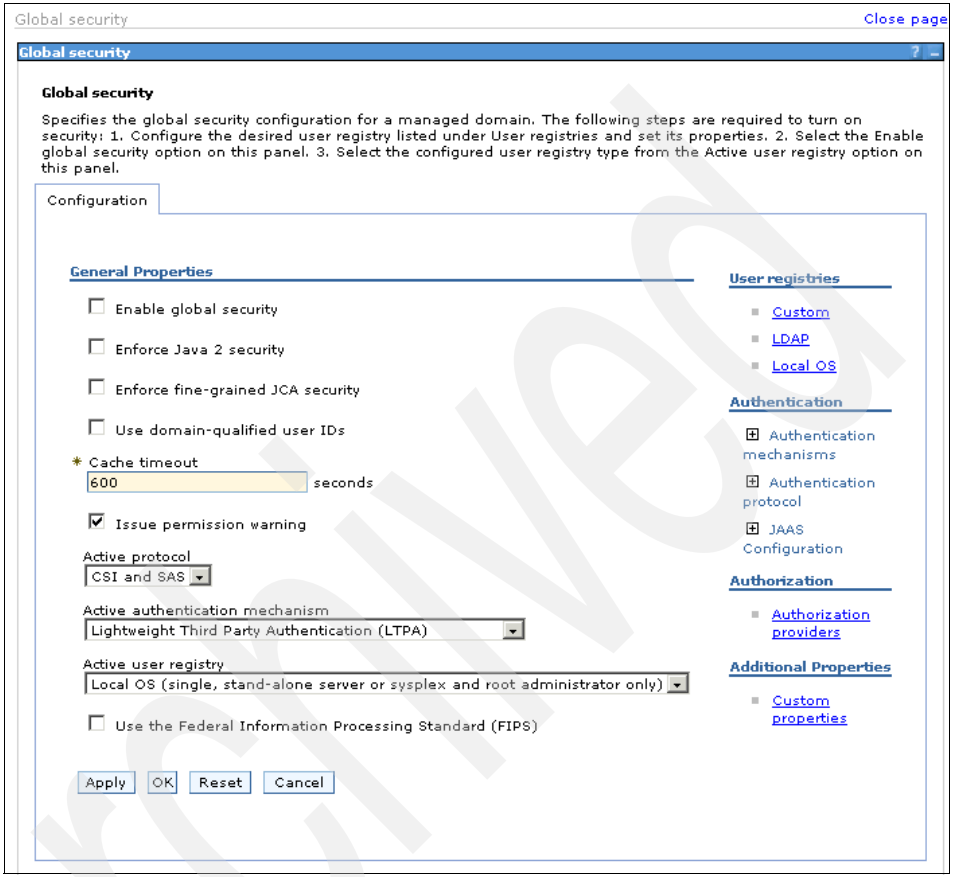


Figure B-30 The administrative console Global security page

Archived

8. Click **J2C Authentication data**. This shows the J2EE Connector Architecture (J2C) authentication data entries page (Figure B-31).
9. You should see at least three aliases with the *wid* user ID. Click each Alias link that has a user ID of wid. First click **SCA_Auth_Alias**

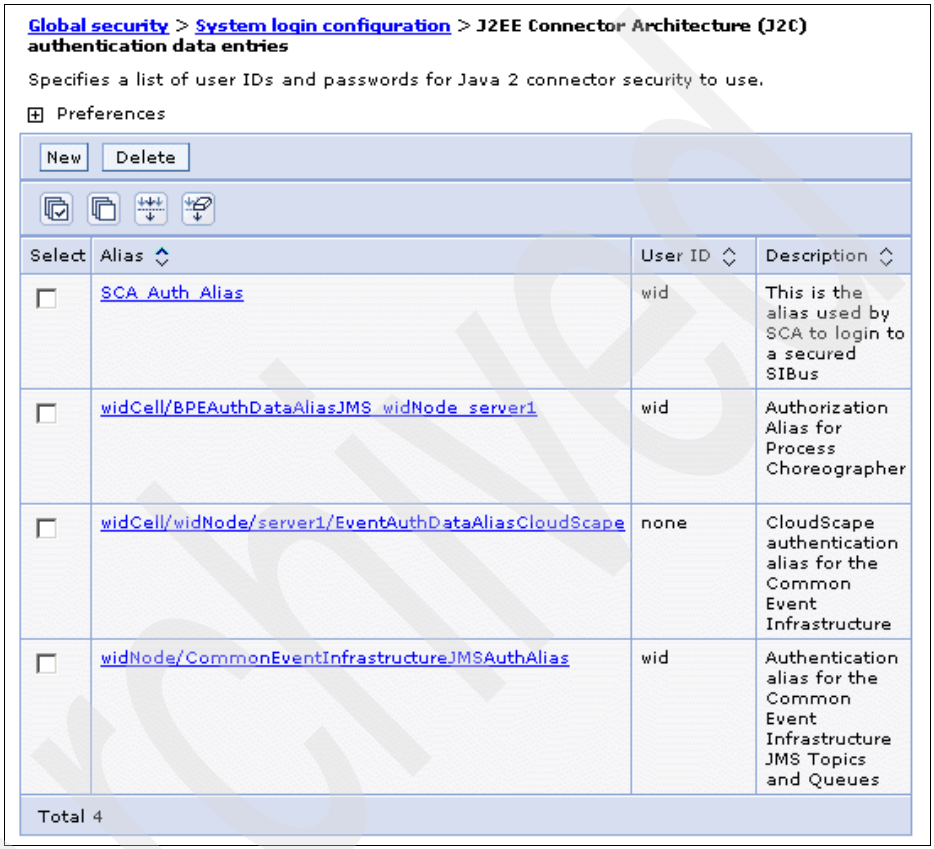


Figure B-31 The J2C authentication data entries page

Archived

10. In the SCA Auth Alias panel (Figure B-32), complete these steps:
    a. In the User ID field, type `wpsadmin`.
    b. In the Password field, type the corresponding password.
    c. Click **OK**.

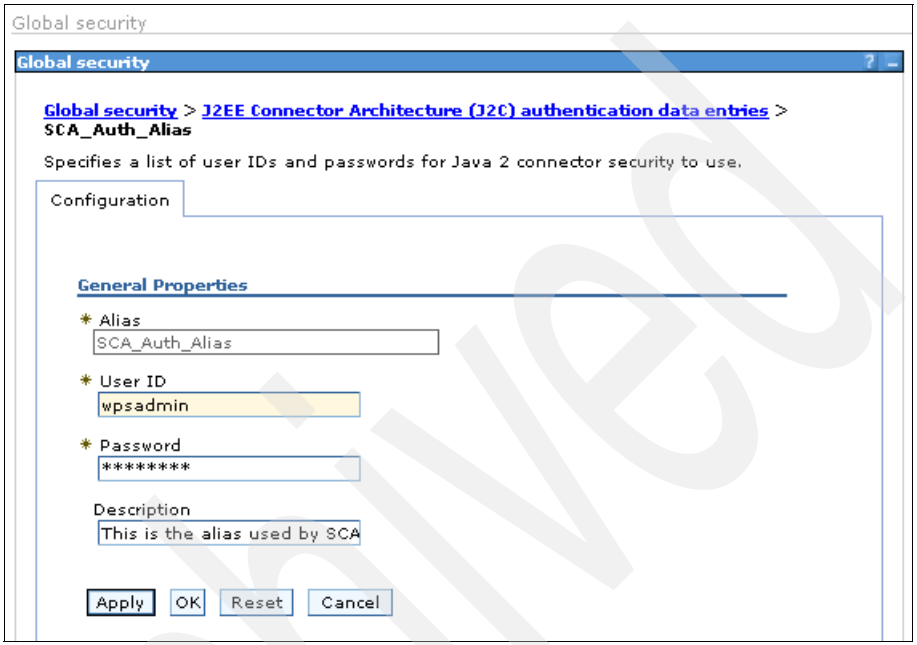


Figure B-32 The SCA Auth Alias page

11. Click **widCell/BPEAuthDataAliasJMS_widNode_server1**.
12. In the widCell/BPEAuthDataAliasJMS_widNode_server1 window, complete these steps:
    a. In the User ID field, type `wpsadmin`.
    b. In the Password field, type the corresponding password.
    c. Click **OK**.
13. Click **widNode/CommonEventInfrastructureJMSAuthAlias**.
14. In the widNode/CommonEventInfrastructureJMSAuthAlias window, complete these steps:
    a. In the User ID field, type `wpsadmin`.
    b. In the Password field, type the corresponding password.
    c. Click **OK**.
15. Click **Security**.
16. Click **Global security**.

Archived

17. On the Global security page, click **Local OS**.

18. In the Local OS user registry window (Figure B-33), complete these steps:

    a. In the Server user ID field, type `wpsadmin`.
    b. In the Server user password field, type the corresponding password.
    c. Click **OK**.

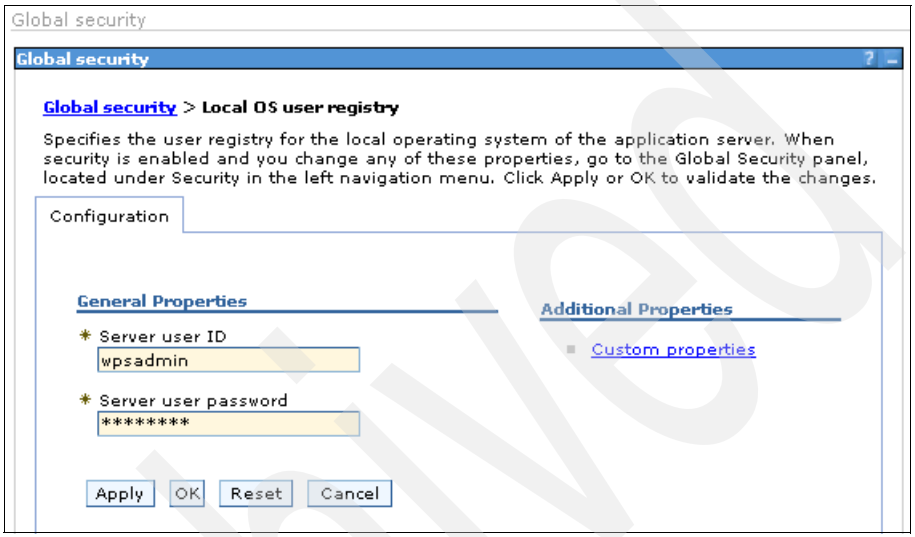


Figure B-33 The Local OS user registry page

19. Click **Global Security**.

20. On the right side of the Global security page, under Authentication, click **Authentication mechanisms**.

21. Click **LTPA**.

Archived

22. In the Lightweight Third Party Authentication (LTPA) configuration settings window (Figure B-34), complete these steps:

   a. In the Password field, type the password for the Server user ID, which was entered in the Local OS user registry window, for example the password for wpsadmin.

   b. In the Confirm password field, type the password again.

   c. Click **OK**.

Figure B-34 The LTPA configuration settings page

Archived

23. In the Global security window (Figure B-35), complete these steps:
    a. Select **Enable global security**.
    b. Deselect **Enforce Java 2 security**.
    c. In the Active protocol box, select **CSI and SAS**.
    d. In the Active authentication mechanism box, select **Lightweight Third Party Authentication (LPTA)**.
    e. In the Active user registry, select **Local OS (single, stand-alone server or sysplex and root administrator only)**.
    f. Click **OK**.
    g. Click **Save** to apply changes to the master configuration.

Figure B-35 The Global security page

Archived

24. In the Global security save window (Figure B-36), click **Save**.

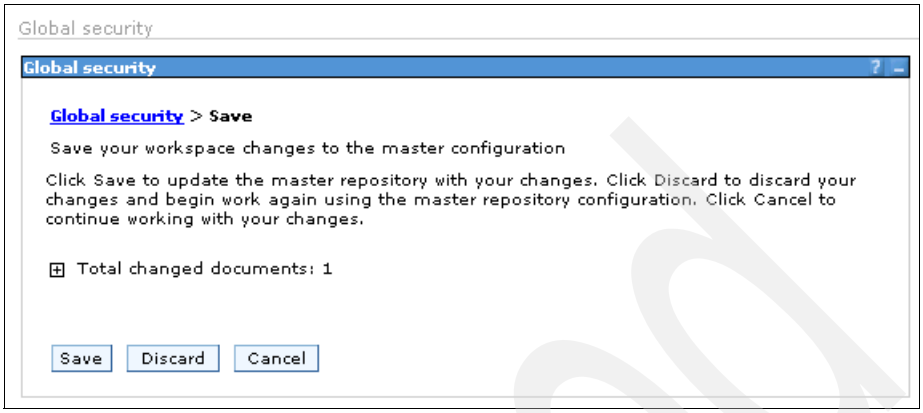


Figure B-36 The Global security save page

Configuring the BPE Container

Ensure that the BPE Container is configured correctly, meaning that it has valid security information.

1. In the navigation pane, click **Applications** and click **Enterprise Applications**.
2. Click the **BPEContainer_widNode_server1** application.
3. In the Additional Properties section, click **Map RunAs roles to users**.

Archived

4. In the Mapping RunAs roles to users window (Figure B-37), complete these steps:

   a. Select the **JMSAPIUser** role.
   b. In the username field, type `wpsadmin`.
   c. In the password field, type the corresponding password.
   d. Click **Apply**.
   e. Click **OK**.

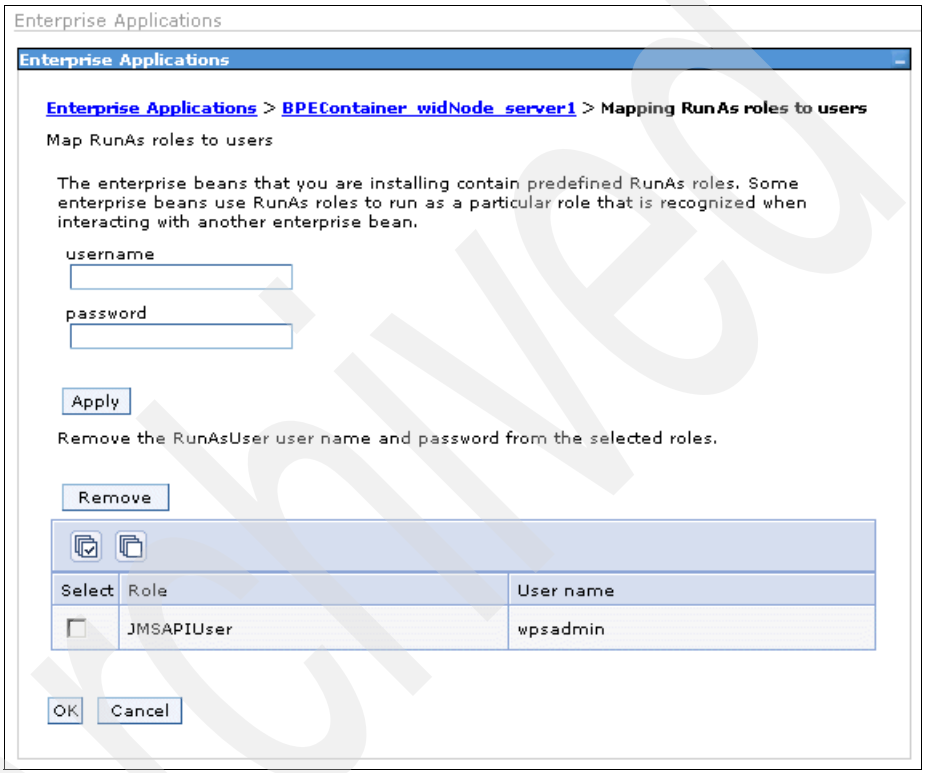


Figure B-37 The BPEContainer_widNode_server1 Map RunAs roles to users page

5. Click **Save** to apply changes to the master configuration. Click **Save** again.
6. In the navigation pane, click **Applications** and click **Enterprise Applications**.
7. Click the **BPEContainer_widNode_server1** application.
8. In the Additional Properties section, click **Map security roles to users/groups**.
9. Select the **BPESystemAdministrator** role. Click **Look up users**.

Archived

10. In the Look up window, click **Search**.
    a. Select **wpsadmin** from the Available list box.
    b. Click **>>** to add it to the Selected list.
    c. Click **OK**.
11. Select the **BPESystemAdministrator** role again. Click **Look up groups**.
12. In the Look up window, click **Search**.
    a. Select the **wpsadmins** group from the Available list box.
    b. Click **>>** to add it to the Selected list.
    c. Select **wid** from the Selected list.
    d. Click **<<** to remove it from the Selected list (Figure B-38).
    e. Click **OK**.

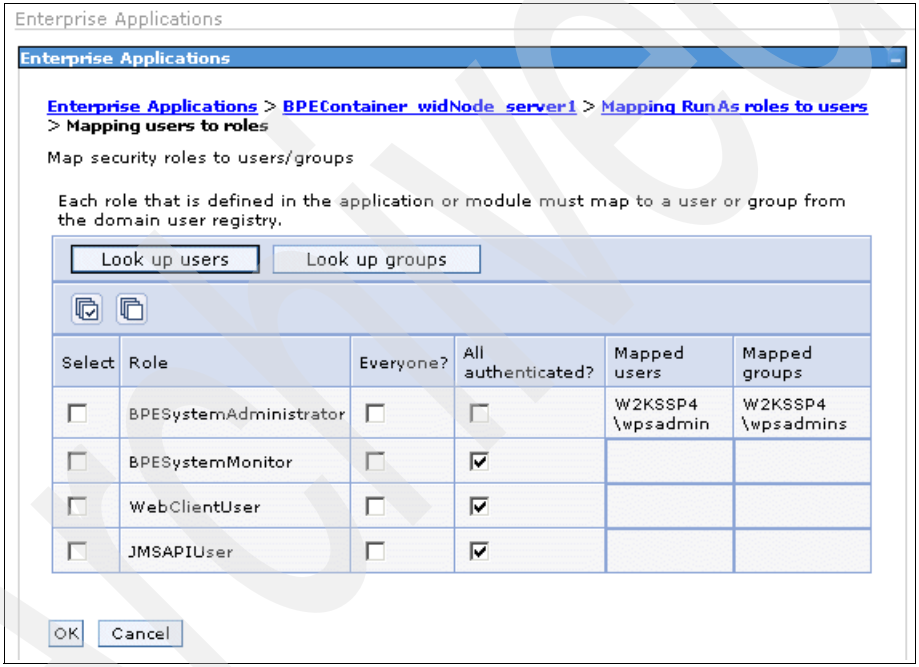


Figure B-38 The BPEContainer_widNode_server1 Map roles to users/groups page

13. Click **OK**.
14. Click **Save** to apply changes to the master configuration. Click **Save** again.

Archived

Configuring the Task Container

Ensure that the Task Container is configured correctly, meaning that it has valid security information.

1. In the navigation pane, click **Applications** and click **Enterprise Applications**.
2. Click the **TaskContainer_widNode_server1** application.
3. In the Additional Properties section, click **Map RunAs roles to users**.
4. In the Mapping RunAs roles to users window:
   a. Select the **EscalationUser** role.
   b. In the username field, type `wpsadmin`.
   c. In the password field, type the corresponding password.
   d. Click **Apply**.
   e. Click **OK**.

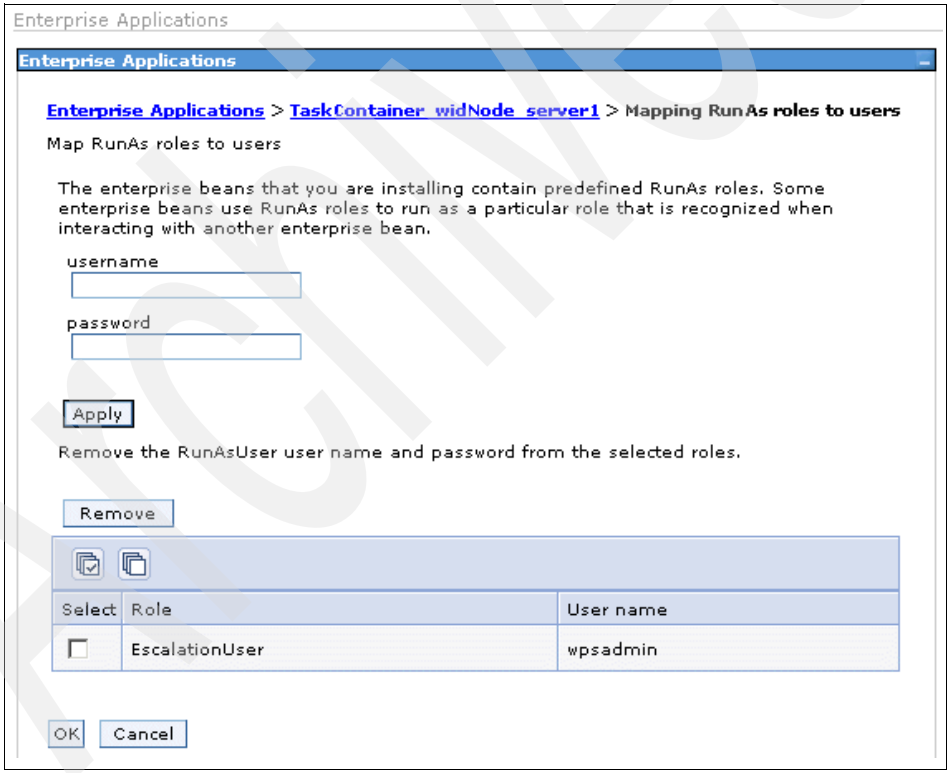


Figure B-39 The TaskContainer_widNode_server1 Map RunAs roles to users page

5. Click **Save** to apply changes to the master configuration. Click **Save** again.

Archived

6. In the navigation pane, click **Applications** and click **Enterprise Applications**.
7. Click the **TaskContainer_widNode_server1** application.
8. In the Additional Properties section, click **Map security roles to users/groups**.
9. Select the **TaskSystemAdministrator** role. Click **Look up users**.
10. In the Look up window, click **Search**.
    a. From the Available list box, select **wpsadmin**.
    b. Click **>>** to add it to the Selected list.
    c. Click **OK**.
11. Select the **TaskSystemAdministrator** role again. Click **Look up groups**.
12. In the Look up window, click **Search**.
    a. From the Available list box, select the **wpsadmins** group.
    b. Click **>>** to add it to the Selected list.
    c. From the Selected list box, select **wid**.
    d. Click **<<** to remove it from the Selected list.
    e. Click **OK**.

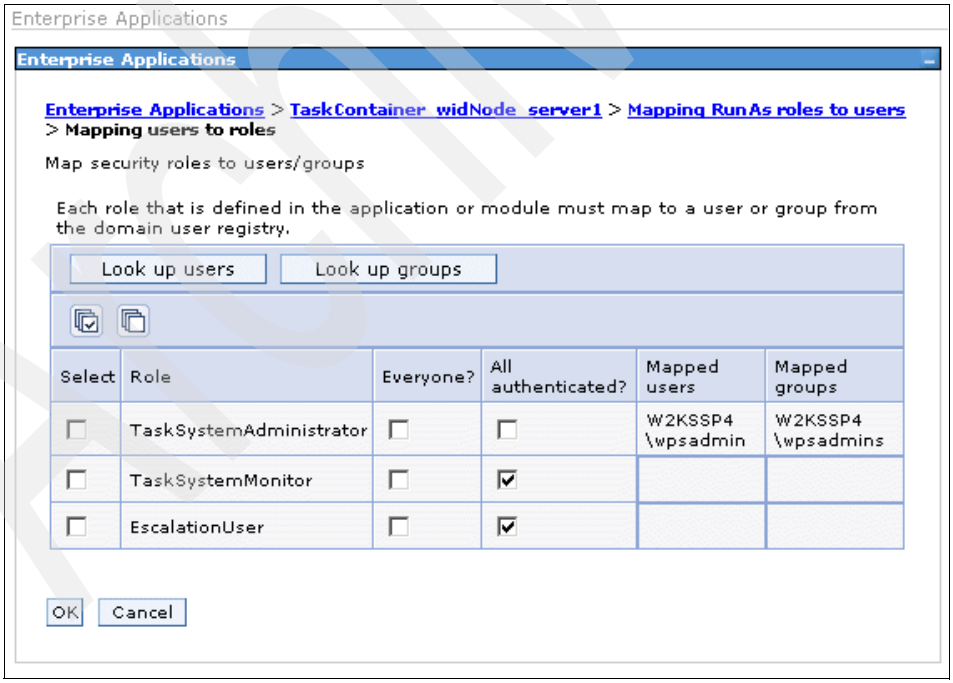


Figure B-40 The TaskContainer_widNode_server1 Map roles to users/groups page

Archived

13. Click **OK**.
14. Click **Save** to apply changes to the master configuration. Click **Save** again.
15. Click **Logout**.

Figure B-41 The Admin console window

16. Close the Admin Console window.

Configuring the WebSphere Integration Developer test server

Now we set up the WebSphere Integration Developer test server configuration to communicate with a secure server.

1. Click the **Servers** page.
2. Right-click the **WebSphere Process Server v6.0** server and click **Open**.
3. In the WebSphere Process Server v.6.0 Server overview window (Figure B-42), click **Security** to expand the Security settings area.

Archived

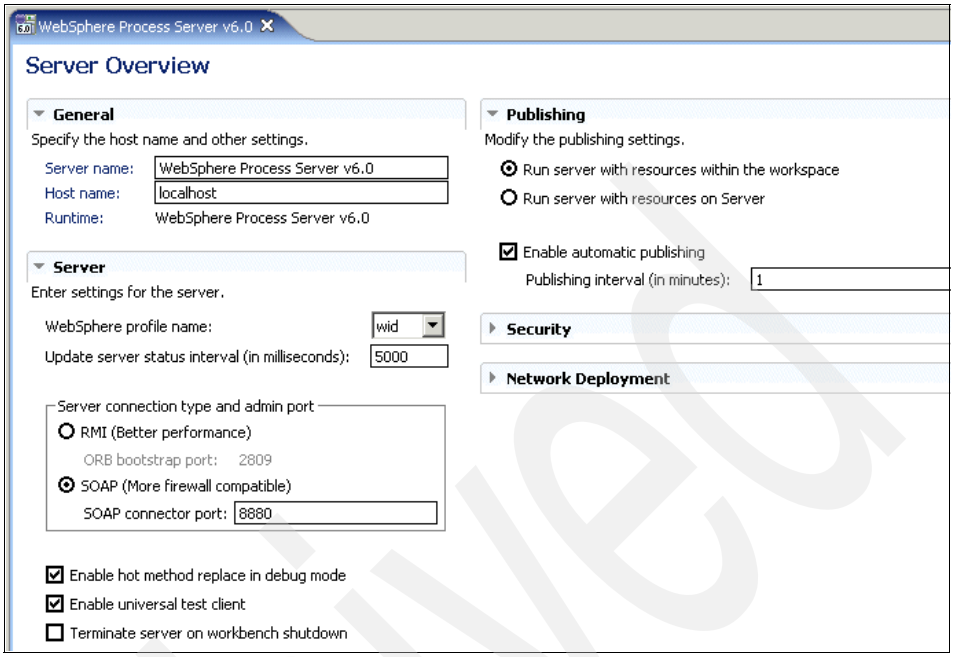


Figure B-42 The WebSphere Process Server v.6.0 Server overview window

4. Select the **Security is enabled on this server** check box.
5. Press Ctrl+S to save the server configuration.
6. Close the WebSphere Process Server v.6.0 Server overview window.
7. Right-click the **WebSphere Process Server v6.0** server and click **Open**.
8. Click **Security** to expand the Security settings area.
9. In the User ID field, type `wpsadmin`. In the Password field, type the corresponding password.
10. Press Ctrl+S to save the server configuration.
11. Close the WebSphere Process Server v.6.0 Server overview window.
12. Right-click the **WebSphere Process Server v6.0** server and click **Stop**.
13. Wait for the server to stop.

Archived

Redeploying the file transfer application

We must now re-deploy the file transfer application to the WebSphere Process Server v.6.0 test server, to ensure that we can continue to deploy applications to the test server after security is setup on the server.

1. If the server is not already started, right-click the **WebSphere Process Server v6.0** server and click **Start**.
2. Wait for the server to start.
3. After the server starts, open a Windows Command Prompt window.
4. Change to the *WID_HOME*\runtimes\bi_v6\bin directory.
5. In the Command Prompt window, type:

   ```
   wsadmin -profile redeployFileTransfer.jacl -lang jacl -c
   “fileTransferAuthenticationOn widCell widNode server1“ -user User ID
   -password Password
   ```

   Here *User ID* is `wpsadmin` and *Password* is the corresponding password.
6. Look for the message `Application filetransfer installed successfully`.
7. Close the Command Prompt window.

WebSphere Adapter for JDBC

In this section, we take you through the steps to configure the server to communicate with the WebSphere Adapter for JDBC using DB2 as a backend.

J2C Authentication Alias

In order for the adapter to communicate with DB2, we have to set up a J2C Authentication Alias in the administrative console.

> **Tip:** If you do not set up a J2C authentication alias, the following exception occurs at run time:
>
> ```
> javax.resource.spi.ResourceAllocationException: PasswordCredential not
> found.
> ```

1. In the administrative console, expand the **Security** tab, and click **Global Security**.
2. Expand the **JAAS Configuration** tab, and click **J2C Authentication data**.

Archived

3. Click **New**.
4. In the New J2C authentication data page (Figure B-43):
   a. Set the alias to use. We used `db2admin`.
   b. Specify the user ID to connect to DB2 Universal Database.
   c. Specify the password to connect to DB2 Universal Database.
   d. Click **OK**.

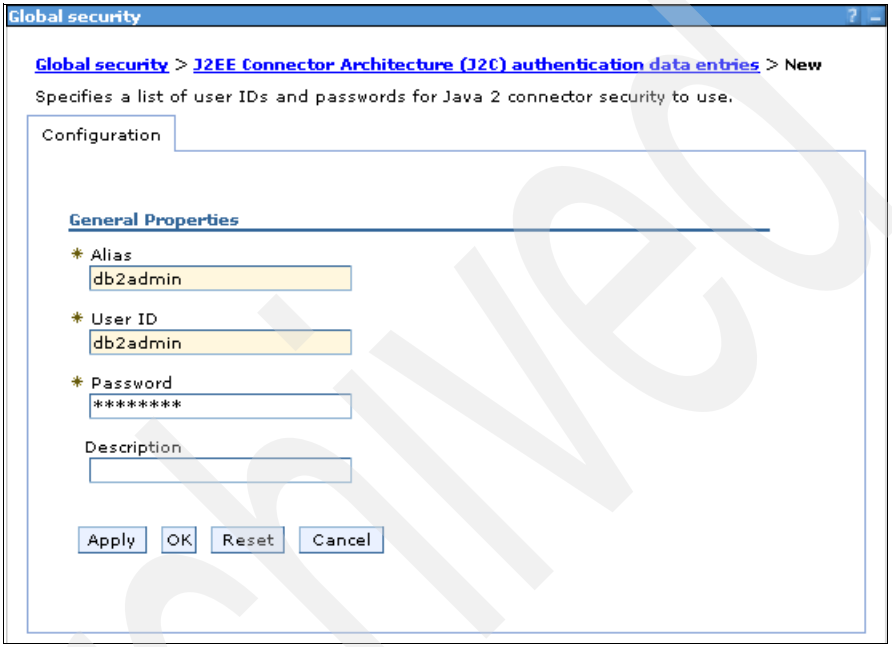


Figure B-43 Configuring a J2C authentication alias

5. Save your changes. Note that the format of the alias name is *nodeName/alias name*, so in our case it is widNode/db2admin.

Transaction resource

If you are attempting to use the adapter in a long-running business process, you may encounter the following exception:

```
java.lang.NullPointerException at
com.ibm.j2ca.jdbc.JDBCManagedConnection.getXAResource(...)
```

This exception occurs because the resource adapter was unable to obtain a transaction resource. In a long-running business process, multiple transactions are created. If you see this message, then the server was unable to create an instance because it is not configured automatically.

Archived

Use the following steps to configure the server to use DB2 Universal Database as a JDBC transaction resource and to configure the adapter to use it.

1. In the administrative console, expand **Resources** and click **JDBC Providers**.
2. Click the **New** button
3. Define a new JDBC provider as follows (Figure B-44).

   a. Select **DB2** as the database type.
   b. Select **DB2 Universal JDBC Driver Provider** as the provider type.
   c. Select **XA data source** as the implementation type.
   d. Click **Next**.

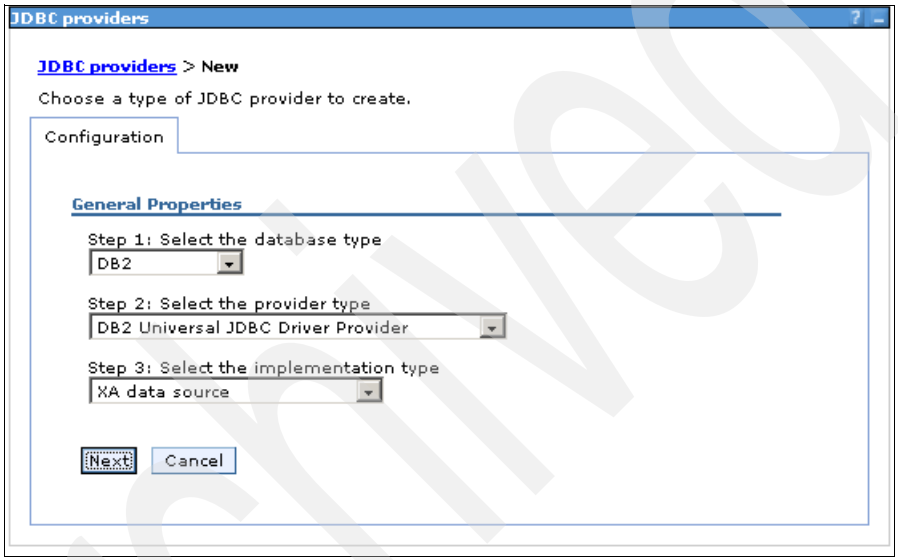


Figure B-44 Defining a new DB2 JDBC provider

4. Click **OK** and save the changes to the master configuration.

Configure the server variables to point to the DB2 installation location.

1. In the administrative console, expand **Environment** and click **WebSphere Variables**.
2. Click the **DB2UNIVERSAL_JDBC_DRIVER_PATH** variable.
3. Set the value of this variable to point to the location of the DB2 java directory. For example, depending on your installation, you may set it to `C:/SQLLIB/java`.

> **Note:** Use forward slashes for directory separators, even on a Windows system.

Archived

4. Click **OK** and save your changes.

   We need to return to the DB2 XA JDBC Provider and configure the transaction data source.

5. In the administrative console, expand **Resources** and click **JDBC Providers**.
6. Click **DB2 Universal JDBC Driver Provider (XA)** and then click **Data sources**.
7. Click **New**.
8. Define a new data source as follows:
   a. Set the Component-managed authentication alias to **widNode/db2admin**.
   b. Set the Database Name to the name of the database you want to connect to. We specified `ITSO_E`.
   c. Click **OK** and save your changes.
9. Test the connection by placing a check mark next to **DB2 Universal JDBC Driver XA DataSource** and clicking **Test Connection**. If the connection is successful, you see a message similar to the one in Figure B-45.

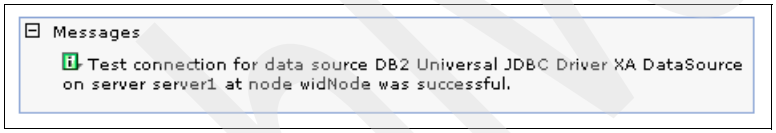


Figure B-45 A successful result of Test Connection

The final step requires us to update the resource adapter deployment descriptor to configure the location of the transaction resource.

1. In the administrative console, expand **Applications** and click **Enterprise Applications**.
2. Click the application project which contains the JDBC adapter. We clicked **ITSOPModuleApp**.
3. Click **Connector Modules**, and then click **CWYBC_JDBC.rar**.
4. Click **Resource Adapter**, and then click **J2C connection factories**.
5. Click the connection factory that you want to configure. We clicked **ITSOPModule.JDBCOutboundInterface_CF**.
6. Click **Custom properties**.
7. Click the **XADataSourceName** custom property and specify a value of `COM.ibm.db2.jdbc.DB2XADataSource`. Click **OK**.
8. Click the **XADatabaseName** custom property and specify the name of the database to connect to. We specified `ITSO_E`. Click **OK**.

Archived

Figure B-46 shows how the custom properties should look.

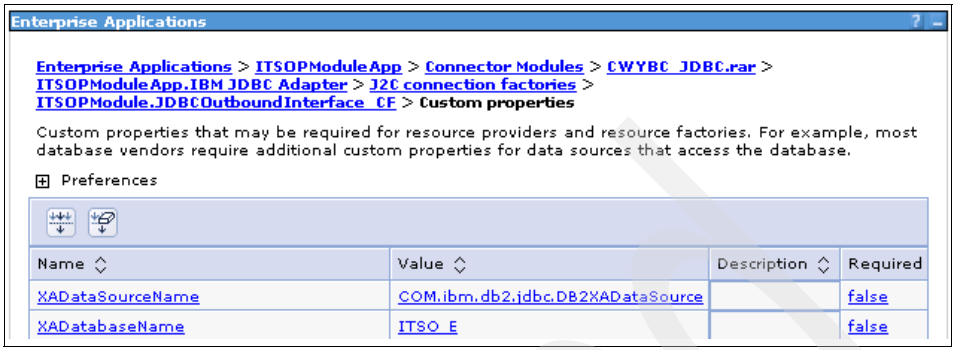


Figure B-46 Setting the required XA custom properties

9. Save your changes to the master configuration.

Attention: If the application is republished to the server, you need to re-enter these two properties. To workaround this problem, you can modify the values, highlighted in bold in below, directly in the ra.xml file in the connector project.

```
<config-property>
<config-property-name>XADataSourceName</config-property-name>
<config-property-type>java.lang.String</config-property-type>
<config-property-value>COM.ibm.db2.jdbc.DB2XADataSource
</config-property-value>
</config-property>
<config-property><config-property-name>XADatabaseName</config-property-name>
<config-property-type>java.lang.String</config-property-type>
<config-property-value>ITSO_E</config-property-value>
</config-property>
```

This forces any other modules that use this resource adapter to use the same database for transactions.

The WebSphere Adapter for JDBC is now completely configured.

ResourceAdapterInternalException

When the server starts, you may see the following exception in the log:

```
javax.resource.spi.ResourceAdapterInternalException: One or more adapter
configuration properties are invalid.
```

Located further along in the trace, you see the following exception:

```
Caused by: javax.resource.spi.InvalidPropertyException: DatabaseVendor cannot
be null
```

This error occurs because the property specified in the resource adapter in the assembly diagram did not propagate to the server. To apply this property manually:

1. In the administrative console, click **Applications** → **Enterprise Applications**, and then select the application project that contains the resource adapter.
2. Select **Connector Modules** → **CWYBC_JDBC.rar** → **Resource Adapter** → **Custom properties**.
3. Set the DatabaseVendor property to `DB2`, save your changes, and restart the server.

Debugging

WebSphere Process Server includes support for debugging both Java and certain component implementations. In this section, we explain how to start the server in debug mode, set breakpoints, and show the Debug perspective.

Starting the server in debug mode

To start the server in debug mode:

1. Open the ***servers*** view.
2. Right-click the ***server instance*** and click **Debug** (see Figure B-47).

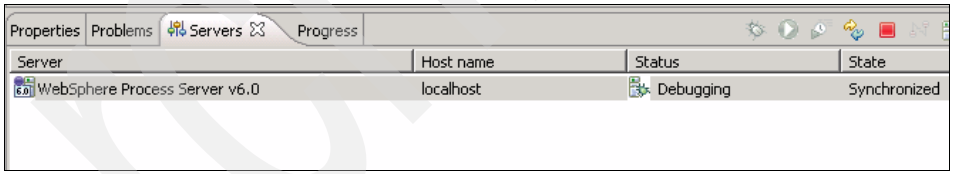


Figure B-47 The server started in debug mode

The server starts and when finished, the status reads *Debugging...*.

> **Tip:** Starting the server in debug mode makes the server less responsive and requires more memory than Run mode. Start the server in this mode only when required.

Archived

When a breakpoint is encountered, WebSphere Integration Developer switches to the Debug perspective (Figure B-48).

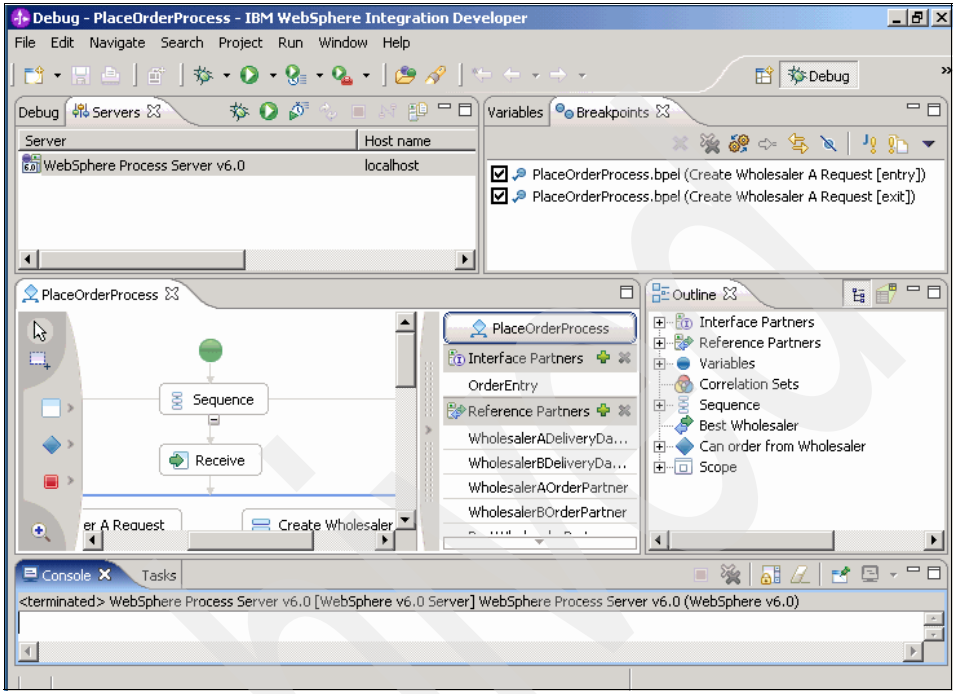


Figure B-48 The Debug perspective

Attention: When using debugging and the Integration Test Client, start the server manually from the Servers view. If you use the Test Client to start the server, the breakpoints may not be installed by the time the invocation of your component begins.

Adding breakpoints to components

You can add a breakpoint to the various components in WebSphere Integration Developer that support them. All of these breakpoints behave exactly like standard Java breakpoints. You can step over, step into, and view the values of the objects. The visual editor used to create the breakpoint is also used for debugging. This means that you can use the editor as a frame of reference for the part of the code that is executing.

Tips:

- If a change is made to an implementation while the server is in debug mode, it is likely that you need to restart your test for your change to occur.
- Breakpoints are ignored when the server is started in Run mode.

Business processes

In a business process, you can add an entry or exit breakpoint to any activity. Entry breakpoints are reached before the activity is run and Exit breakpoints start after the activity is run.

To add a breakpoint, right-click any ***activity*** in your business process and select **Debug** → **Add Entry Breakpoint** or **Debug** → **Add Exit Breakpoint**. The activity is changed to show that the breakpoint has been added (Figure B-49).

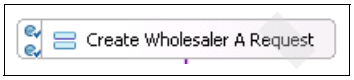


Figure B-49 An assign node with an entry and exit breakpoint specified

Rule groups

Right-click a ***rule*** and select **Debug** → **Add Breakpoint** (Figure B-50).

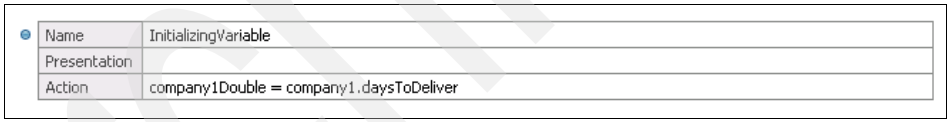


Figure B-50 A rule with a breakpoint added

Decision tables

Right-click a ***cell*** and select **Debug** → **Add Breakpoint** (Figure B-51).

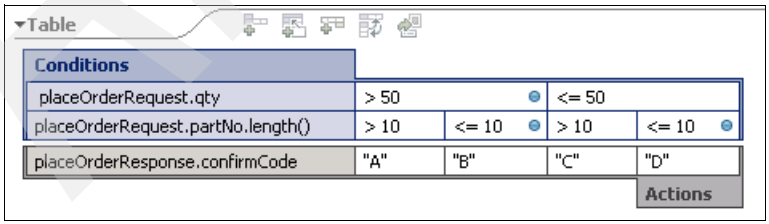


Figure B-51 Breakpoints on a decision table

Archived

Business state machines

Right-click a ***state*** and select **Debug** → **Add Entry Breakpoint** or **Debug** → **Add Exit Breakpoint** (Figure B-52).

Figure B-52 A state with a set breakpoint

Maps

Right-click a ***transformation*** and select **Debug** → **Add Entry Breakpoint** or **Debug** → **Add Exit Breakpoint**.

> **Restriction:** It is not possible to visually distinguish between an entry breakpoint and an exit breakpoint. Look at the breakpoint view to determine which one is being started.

Visual Snippet Editor

Right-click an **element** and select **Debug** → **Add/Remove Breakpoint**. This adds a breakpoint marker (Figure B-53).

Figure B-53 A breakpoint set on a visual snippet

Java components and Java snippets have breakpoints set in the standard method.

> **Restriction:** Step Into and Step Out Of cannot be used when debugging visual snippets. Use extra breakpoints instead.

Components without debugging support

Components that do not support setting a breakpoint are:

- Selectors
- Rule Groups
- The Assembly Editor
- Relationships
- Human Tasks

Archived

C

Relationships and cross-referencing

WebSphere Process Server provides the capability to cross-reference data identifiers between different systems and services. In this appendix, we explain this through example of our business process using a new wholesaler service, provided by a new wholesaler (WholesalerF), which uses different identifiers for part numbers. Where our business process uses part1 and part2, WholesalerF uses item1 and item2.

This is a common scenario, especially when using the exposed or extended-enterprise variations of the patterns to link to external partners. WebSphere Process Server provides facilities for dynamically cross-referencing identifiers that refer to the same entity in different system. This is known as *relationship services* or *relationships*.

In theory, we can use a simple database table to perform the lookups. However, WebSphere Process Server relationships provide a built-in, more flexible method of cross-referencing than a simple database table, with tools which hide the implementation of the relationship services from the user.

Relationships can be used to cross-reference multiple systems which all use different identifiers to refer to the same entity. It is easy to add new systems to an existing relationship as they are added into our business process or as more partners are added to our extended enterprise. In an on demand environment

© Copyright IBM Corp. 2006. All rights reserved.

where WebSphere Process Server is used to automatically synchronize entities across multiple systems, the cross-reference data is maintained by WebSphere Process Server with no need for human intervention.

Relationships: What they do and how they do it

In business integration scenarios, accessing the same data in various back-end systems is often a requirement. A common problem for keeping data in sync is that different back-end systems use different keys to represent the same data. The relationship service in WebSphere Process Server is used to relate these disparate data sources. As a business object is converted from one application specific representation into another, WebSphere Process Server can dynamically maintain a database of keys, which enables the mapping of one data record into another between disparate back-end systems.

Effectively, through the use of the relationship instance ID, WebSphere Process Server provides a master key which you can use to reference the same entity, regardless of the representation in different systems.

Archived

Figure C-1 illustrates the concept of a dynamic identity relationship with a customer identifier being maintained in four different applications. These four systems use different identifiers and store customer information in a different format or object model. A typical integration requirement is to propagate new or changed records in all four systems. In conventional middleware solutions, this is an extremely difficult solution to develop and, more importantly, to manage continuously. WebSphere Process Server automates this activity through a set of capabilities that provide an integrated framework for managing dynamic relationships.

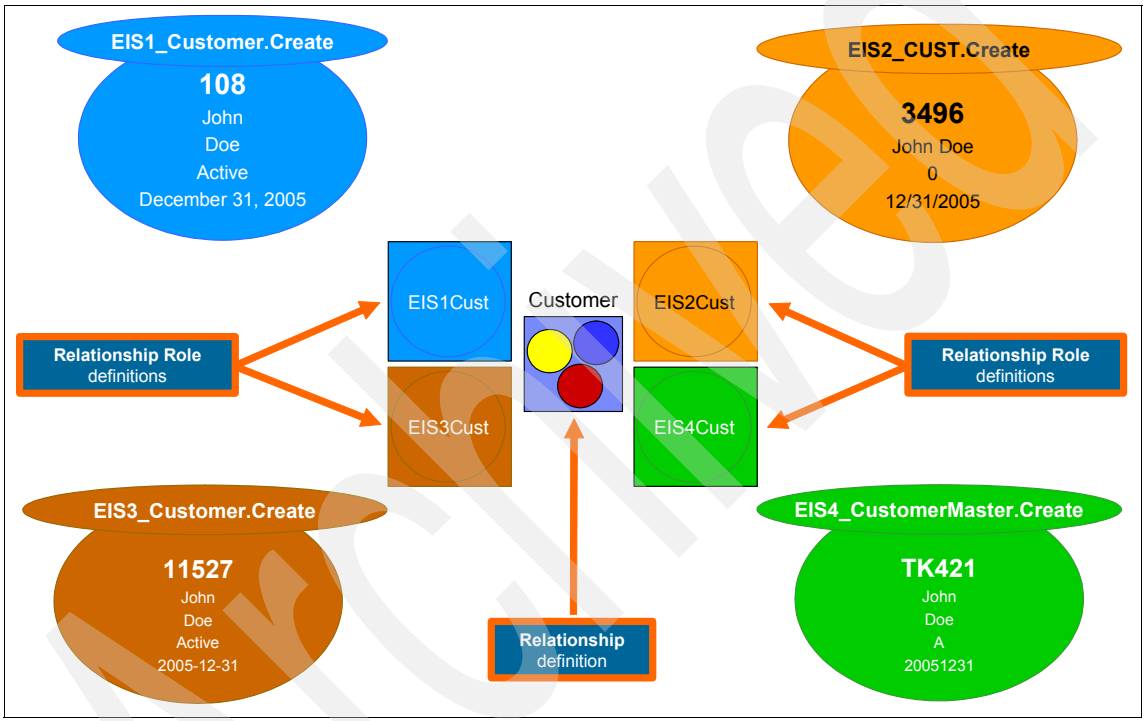


Figure C-1 Multiple sources of customer record information

Archived

Figure C-2 shows how WebSphere Process Server maintains information about each application. Basically, a new identifier is created for each customer and that identifier refers to the specific identifiers in each application. When information is received from the application that is carrying identifier 108, then that identifier is replaced with the generic identifier 42 during processing in WebSphere Process Server. When the business object is sent to one or more target applications, the generic identifier 42 is replaced with the appropriate identifier for that system, for example 3496 in the second application.

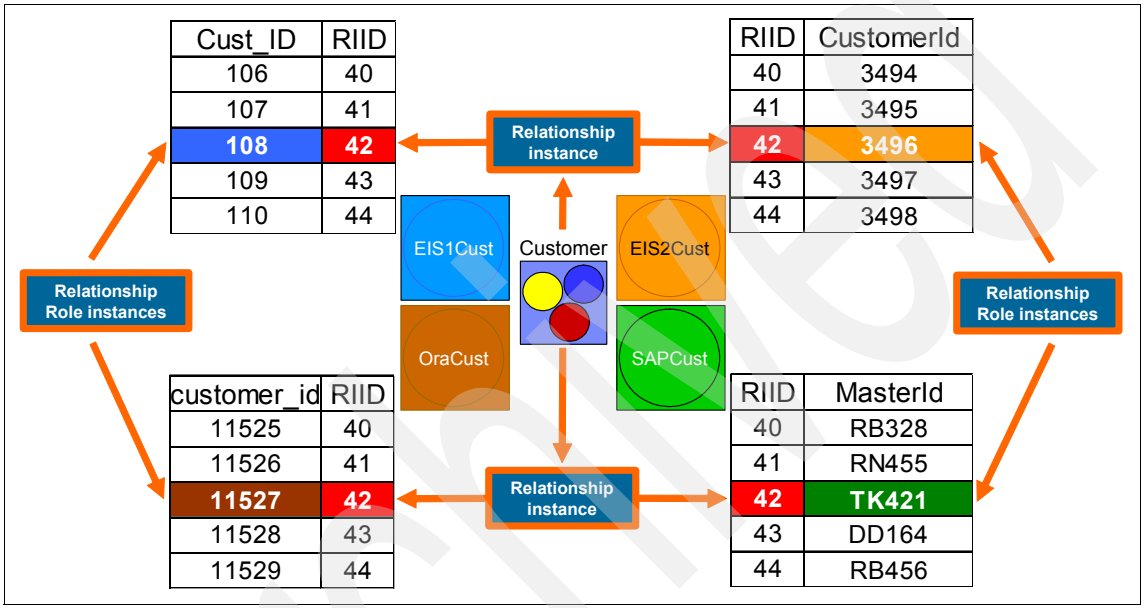


Figure C-2 Relationship instances linking multiple customer identifiers

Creating and maintaining these links between identifiers is completely automatic in WebSphere Process Server. Relationships are available directly from business object maps and simplify the management of disparate back-end data representations.

Example usage of relationships

Consider the example shown in Figure C-3. A new customer is created in EIS1. This is an application event that is delivered to the integration solution. The business object that holds the new customer record contains the new identifier and the verb *Create*.

Within the context of event delivery, the mediation component is called to transform the EIS1 specific business object to a generic object. The map then

uses the services of the relationship manager. Given the context of event delivery and the verb, the relationship manager creates a new relationship instance and generates a new relational identifier. For example, a new record is added to the relationship tables that link EIS1 customer 108 to generic customer 42.

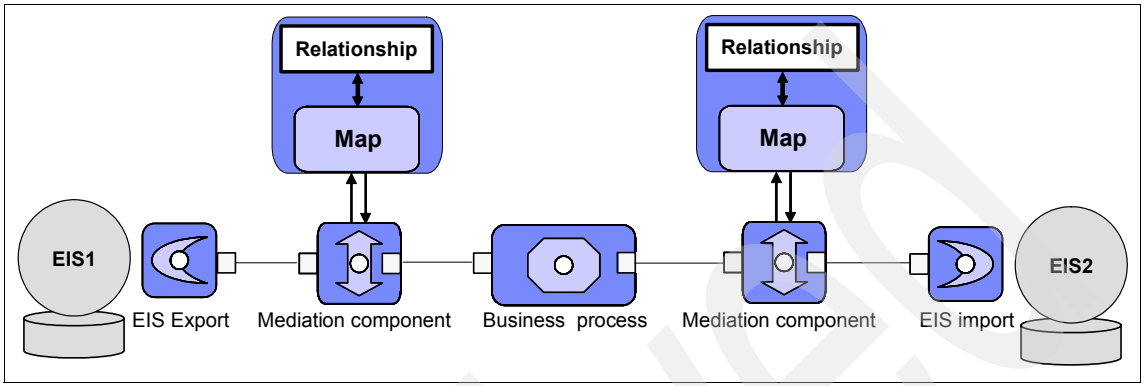


Figure C-3 Relationship invoked during the execution of a map

After the business process is executed, the generic business object is transformed to the business object specific for EIS2. The creation of a new customer record in EIS1 should trigger the creation of a new customer record in EIS2. A service call request is issued to achieve this. Upon return, a new identifier for this customer record in EIS2 is passed back to the business process.

The new customer record for EIS2 is mapped back to the generic object. During the mapping, in the context of the service call response, the relationship manager creates a new relationship instance to link the generic identifier to the newly created identifier for EIS2. For example, identifier 3496 is linked to the generic customer identifier 42. A similar process takes place when an event is passed to the business process for updated or deleted customer records.

Similarly, the relationship service can be called from a data map to perform a lookup of a value or other static information. In addition to dynamic identity relationship management, WebSphere Process Server also supports lookup, or static, relationships. This feature can be used to define static lookup tables where one entry always represents another entry, and this relationship never changes for example, LT = liter, KM = kilometer, and so forth.

Both dynamic and static relationships are created using the relationship editor in WebSphere Integration Developer. Relationships are stored in a database and are maintained automatically or administered through the relationship manager.

Using relationships

In this appendix, we use relationships to cross-reference between two data representations:

- The ITSO Electronics process representation (part1, part2)
- The Wholesaler F representation (item1, item2)

We build a relationship with two roles:

- WholesalerF Part Number Role
- ITSOElectronics Part Number Role

Any new processes that we build now have the option to implement the standard relationship usage or copy this usage. If the processes implement the standard usage, they can use the relationship ID internally within their processes and use the ITSOElectronics role to cross-reference to the existing ITSOElectronics processes. If they copy the usage described in this chapter, they can add their new role to the Part Number Relationship and use their own internal representation of the part number data.

Table C-1 shows the cross-referencing, where each ITSOElectronics and WholesalerF represent a role that we implement.

Table C-1 Relationship roles showing different data representations

RelationshipID	ITSOElectronics	WholesalerF
1	part1	item1
2	part2	item2
3	part3	item3

We replace Wholesaler A with Wholesaler F in our implementation to demonstrate how to replace a service provider that uses different data representations, without changing our business process.

Archived

Setting up the workspace

We use the Serial Process described in Chapter 10, “Serial Process scenario” on page 145, as the basis for this chapter. If you already built this process, you can reuse it here. If not, complete the following steps:

1. Open a new workspace in WebSphere Integration Developer.
2. Select **File** → **Import** → **Project Interchange** to import the following projects into the workspace from the additional material supplied with this IBM Redbook:
   - \Wholesalers\WholesalerA\WholesalerA.zip
   - \Wholesalers\WholesalerB\WholesalerB.zip
   - \SerialProcess\ITSOLibrary.zip
   - \SerialProcess\OrderAck.zip
   - SerialProcess\ITSOElectronics.zip

   For details about accessing the additional materials, see Appendix E, “Additional material” on page 689.
3. Add these projects to the server, start the server, and test the Serial Process application.

Wholesaler F is based on the implementation of Wholesaler A. The only difference is in the naming of the item representations. To import Wholesaler F:

1. Select **File** → **Import** → **Project Interchange** to import the WholesalerF project into the workspace from the additional material supplied with this book. The file is in \Wholesalers\WholesalerF\WholesalerF.zip.
2. Add the WholesalerF project to the server.

Archived

Creating the ITSOItems relationship

We do not expect Wholesaler F to provide the cross-referencing. As far as Wholesaler F is concerned, ITSO Electronics uses the Wholesaler F representation for parts. Therefore we implement the Relationship in the ITSOElectronics module.

To create a relationship using the New Relationship wizard, follow these steps:

1. In the ITSOElectronics module, right-click **Mapping** and select **New** → **Relationship**.
2. In the New Relationship window (Figure C-4), name the relationship `ITSOItems` and click **Next**.

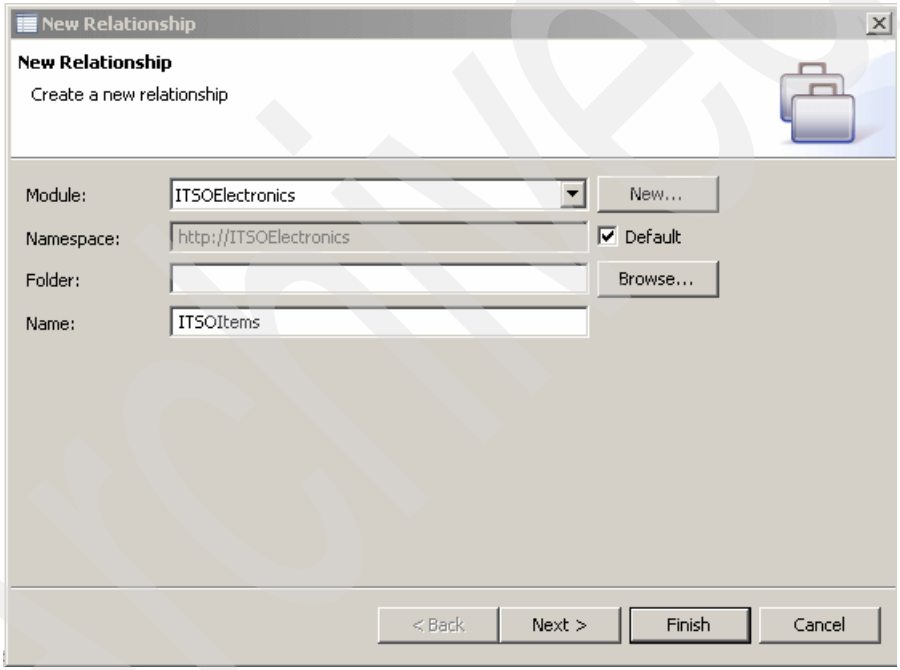


Figure C-4 Creating the new ITSOItems relationship

Archived

3. Choose the type of relationship. As shown in Figure C-5, select **A one-to-one, one-to-many or many-to-many relationship between business objects using any attribute** and click **Finish**.

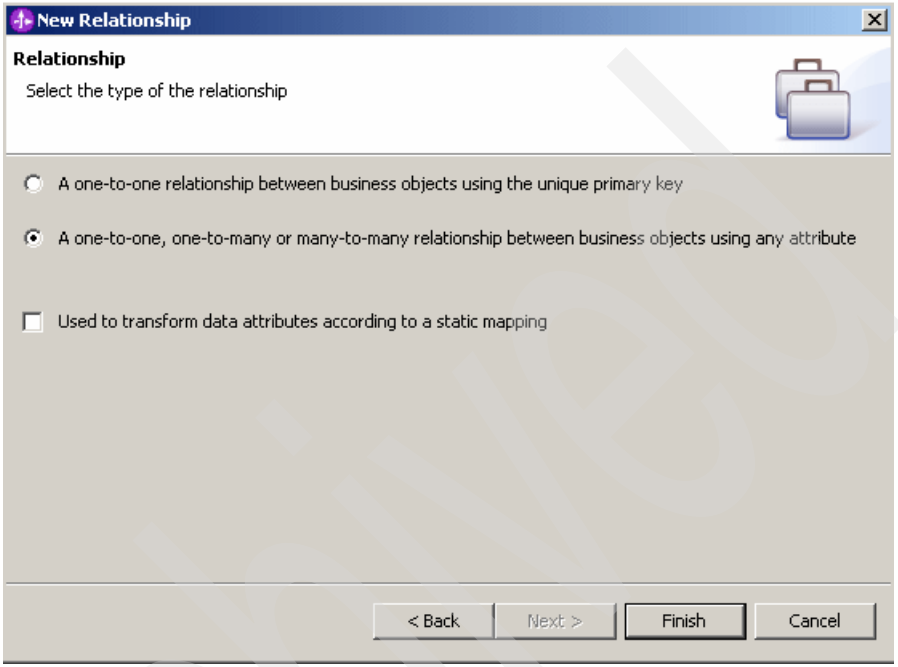


Figure C-5 Choosing the relationship type

4. In the Relationship Editor, click the **Add Role** icon to add a new role.
5. Add the role for the ITSOElectronics business process. Because the process uses the ITSOProcessOrder business object to communicate with Wholesaler F, we use this business object for the role type.

   In the Data Type Selection dialog, select **ITSOProcessOrder**, and click **OK**.
6. Highlight the new **ITSOItems_ITSOProcessOrder** role that is displayed, and click the **Add KeyAttribute** icon ( ).
7. Although we have used the ITSOProcessOrder business object as a basis for this role, we do not need to cross-reference all of the fields. In this example, we only want to cross-reference the part number, so we set that as the key.

   When the Select Key Attributes box appears, expand **OrderPart** and select **partNo**. Click **OK**.
8. Add another role for Wholesaler F. Select **WholesalerDeliveryDaysRequest** as the data type.

This is what Wholesaler F uses as the data type for the delivery days request. This is not ideal, because we also need to implement cross-referencing for the placeOrder interface. As we see later, we can reuse this role even with a different interface.

9. Highlight the new role and add a key of **partNo**.

   Figure C-6 shows how the relationship should look now, with two roles, each with its own business object and a key of part number. Do not worry at this stage if you see red error marks in the relationships.

   Note: The keys do not need exactly the same name in both roles. It is purely coincidence that they do here.

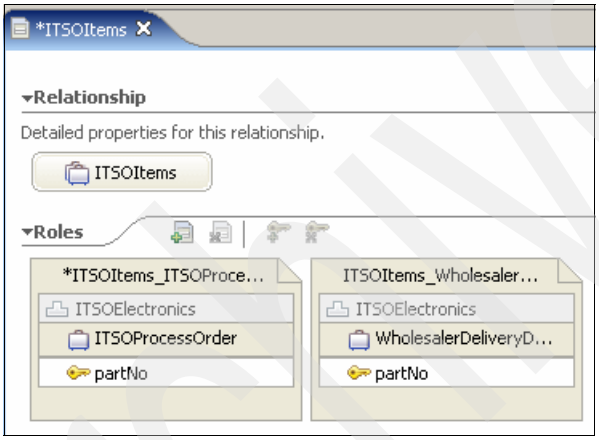


Figure C-6 The completed relationship

10. Press Ctrl+S to save the relationship. Any error marks should now be removed.

Deploying the relationship

Now that we designed the relationship, we must deploy it to the server before we can populate it with data. Since we already added ITSOElectronics to the server, we only need to restart the project:

1. Click the **Servers** view.
2. Right-click the ***server*** and select **Restart Project** → **ITSOElectronicsApp**.

Archived

Populating the relationship data

Now that we deployed the relationship to the server, we need to populate it with some data so that we can use the lookups in our maps. Typically, the data is deployed automatically by a data load process, or it is dynamically created by the interaction between the two systems. For our example, we enter it manually. Fortunately, we only enter two items of data.

1. Click the **Servers** tab and right-click the ***server***. Select **Launch** → **Relationship Manager**.
2. If you see a window open that explains that your session has become invalid, click **OK**.
3. At the Welcome screen, log into the administrative console.
4. From the left side, select **Integration Applications** → **Relationship Manager**.
5. In the Relationship Manager page (Figure C-7), in the Relationships section, click the **Relationships** hyperlink.

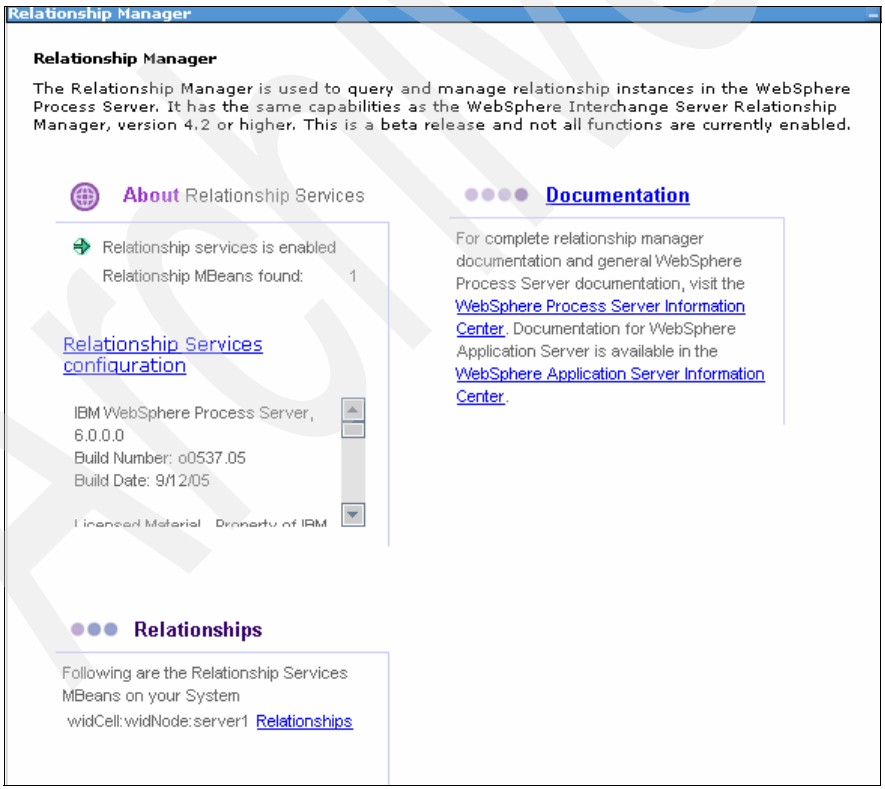


Figure C-7 Relationship Manager main page

Archived

6. In the Relationships page (Figure C-8), select the radio button next to **ITSOItems** and click **Query**.

> **Tip:** If you click the relationship name (ITSOItems), you see the details of the relationship and the types of participants.

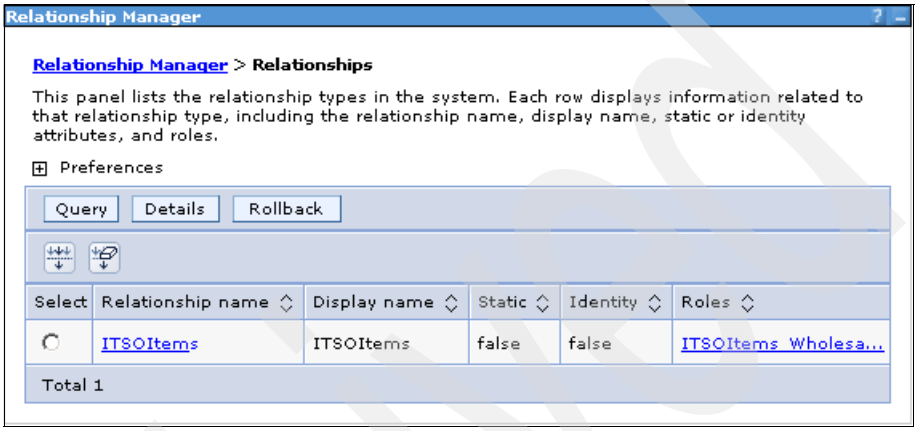


Figure C-8 Relationships page

7. In the Query ITSOItems page (Figure C-9), we can select the relationship instances that we want to work with. We look at all of them, so click **OK** for now.

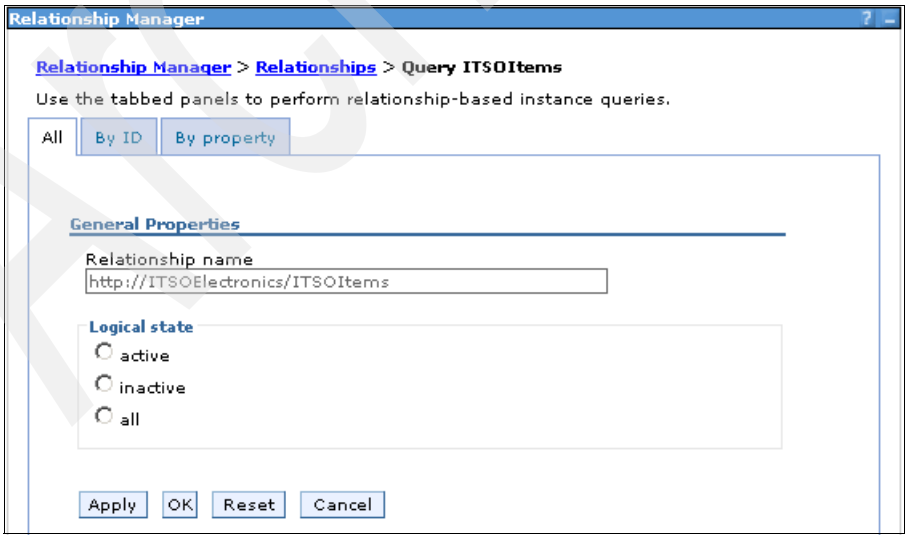


Figure C-9 The Query ITSOItems page

8. In the Relationship instances page (Figure C-10), at the moment, we have no relationship instances. Click **Create** to create a new one.

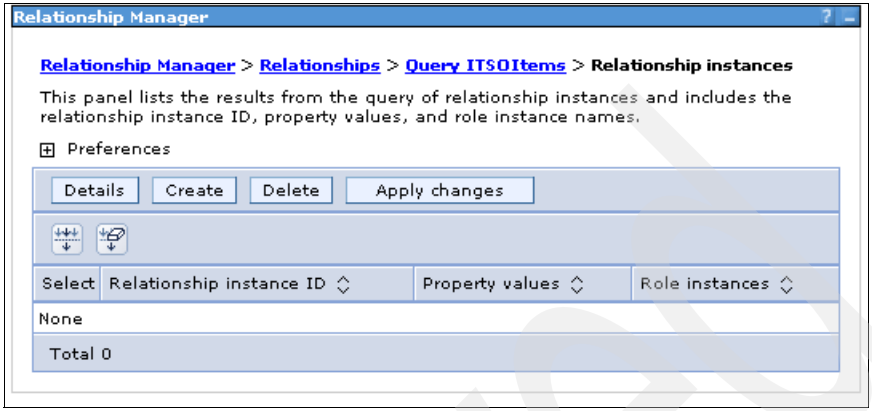


Figure C-10 The Relationship instances page

9. In the New relationships instance page (Figure C-11), notice that the Relationship instance ID is `unsaved-1`. This is because the instance that we are creating has not yet been saved to the database. When this happens, it is assigned a real number. Click **OK** to create the new instance.

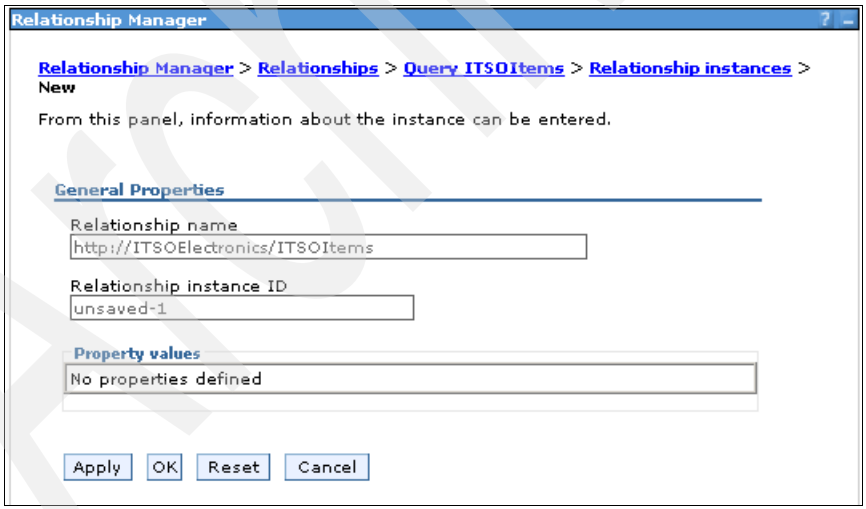


Figure C-11 The New relationship instance

10. We are now back at the Relationship instances page. Notice how we now have a new instance, although still named unsaved-1. Click **Select role** to create a new role instance.

11. In the Role instance summary page (Figure C-12), notice how the two roles that we defined before in WebSphere Integration Developer are shown. Also the Instance count for both of them is zero, indicating that we have no roles for this relationship instance.

    In this instance, we want to cross-reference part1 for ITSOElectronics to item1 for Wholesaler F. We must create one instance for each role. First, we create the ITSOElectronics instance. Select the radio button next to **ITSOItems_ITSOProcessOrder** and click **Create**.

Figure C-12 The role instance summary page

Archived

12. In the New page (Figure C-13), we see that we are still in instance `unsaved-1`. The key attribute, partNo that we defined for the role is shown. We add our cross-reference for ITSOElectronics here, so in the Key attributes section, in the Value box opposite partNo, type `part1` and click **OK**.

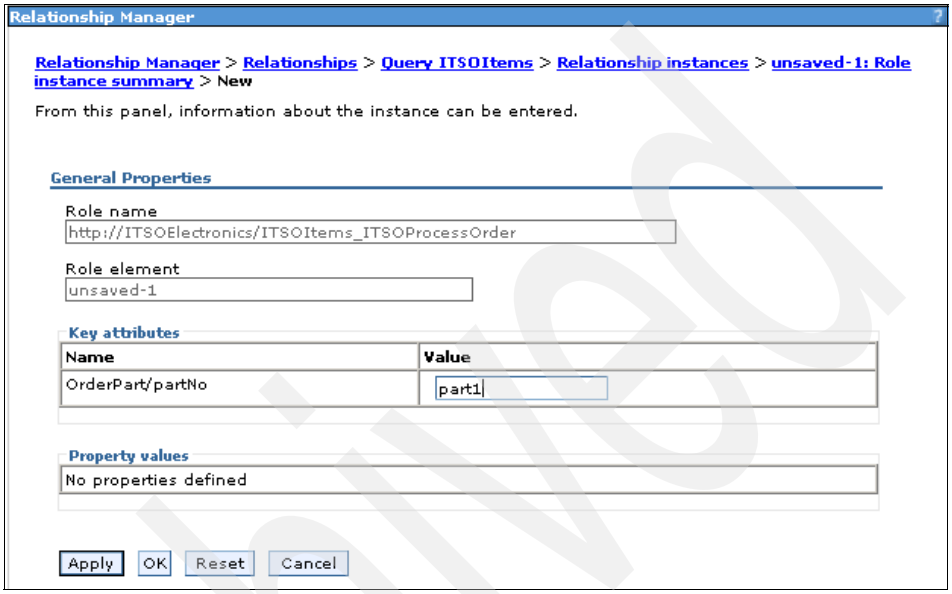


Figure C-13 Adding the part1 reference for ITSOElectronics to the relationship

13. In the panel in Figure C-14, although we added OrderPart/partNo=part1, the panel still shows `unsaved-1` as the ID. Click **Apply changes** to save the data.

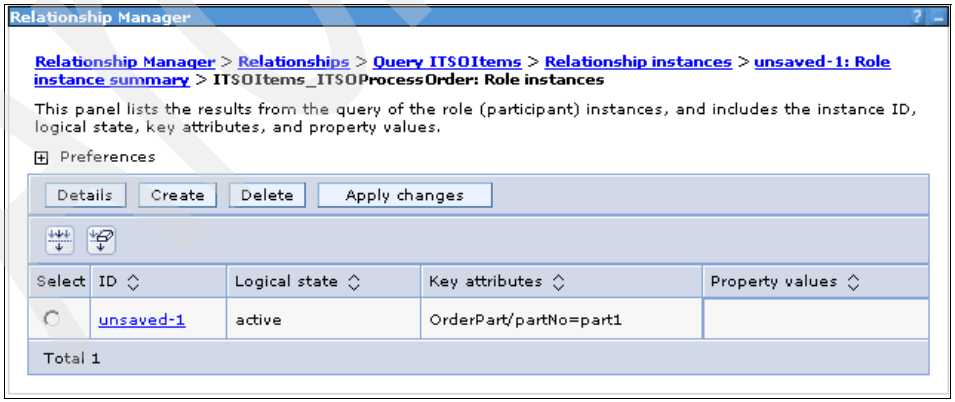


Figure C-14 The new role instance still unsaved

Archived

14. In the Relationship instances panel (Figure C-15), notice that we now have a value in the Relationship instance ID column, indicating that the instance has now been saved and assigned a number.

> **Note:** The relationship instance number you see on your panel may not match the one in ours. The relationship instances are generated from database stored procedures and sequences. It does not matter the instance number is, as long as the correct roles are linked to it.

Under Role instances value, select **ITSOItems_ITSOPro....** so that we can add a role for Warehouse F.

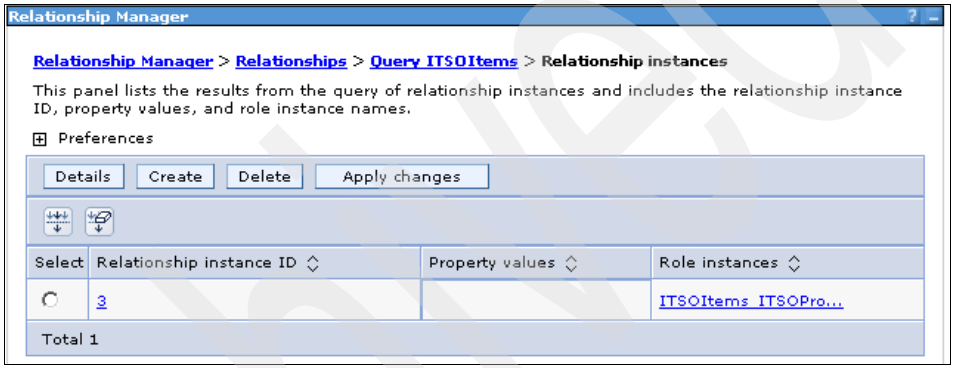


Figure C-15 The new instance created

15. Again we see the Role instance summary panel (Figure C-16), which now shows an instance count of 1 for the ITSOItems_ITSOProcessOrder role. Add an instance for the ITSOItems_WholesalerDeliveryDaysRequest role. Select **ITSOItems_WholesalerDeliveryDaysRequest** and click **Create**.

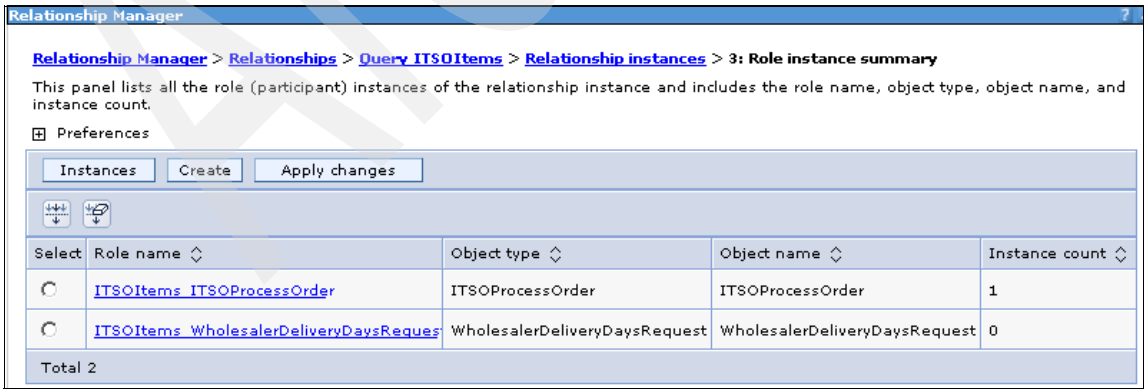


Figure C-16 Role instance summary with one role added

16. As shown in Figure C-17, in the Key Attributes, under Value, type `item1`, and click **OK**.

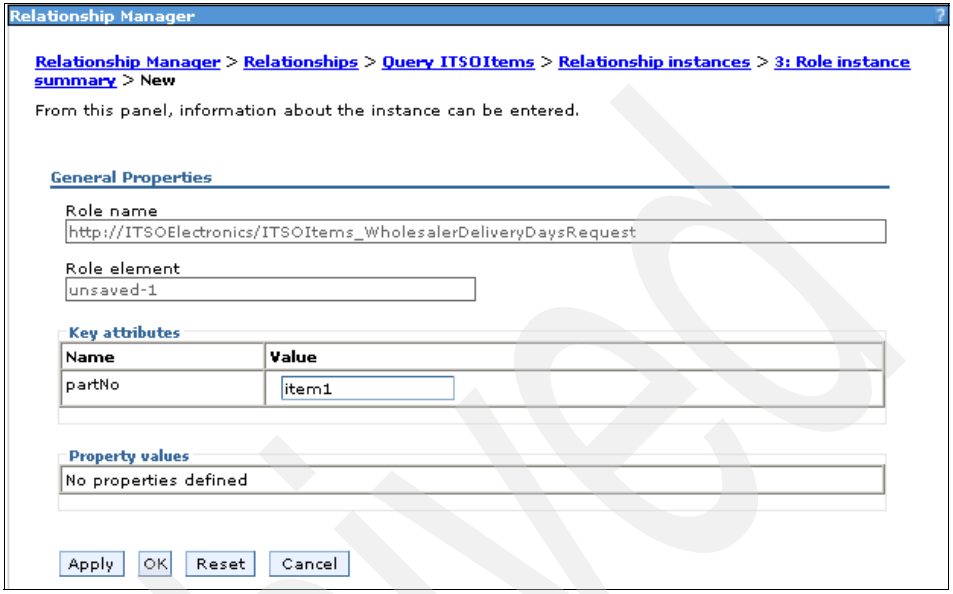


Figure C-17 Adding the Warehouse F role instance

17. At the Role instances page, as shown in Figure C-18, click **Apply changes** to save the instance.

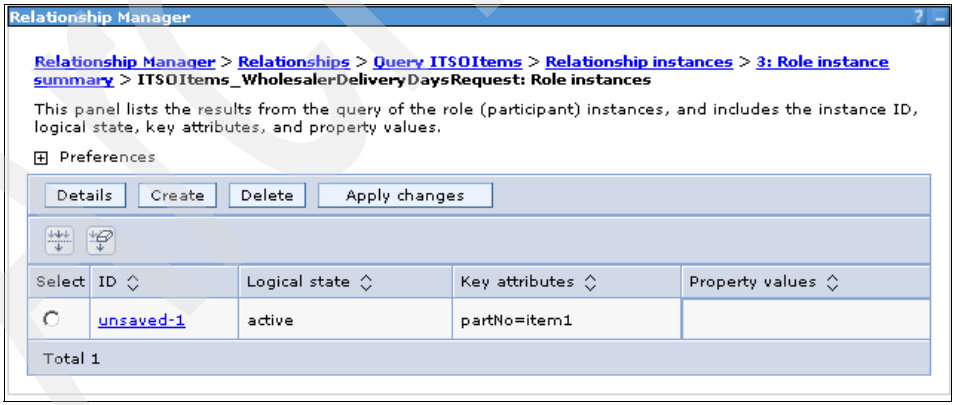


Figure C-18 Saving the wholesaler F role instance

Archived

18. We now have created one role for ITSOElectronics and one for Warehouse F. We are back at the relationship instances panel. If we now select the radio button next to the new instance and click **Details**, we see that the role instance summary panel shows that we only have one role, even though we created two. There's no need to panic, because your data is there. We prove this by requerying the relationship and refreshing the graphical user interface.

    a. Click the **Query ITSOItems** hyperlink near the top of the panel.
    b. In the Query ITSOItems panel, click **OK** to query the relationship.
    c. Back in the Relationship instances panel (Figure C-15), select the radio button next to the relationship instance and click **Details**.

    You should see that we have one instance of each part, as shown in Figure C-19, now that the data has been requeried.

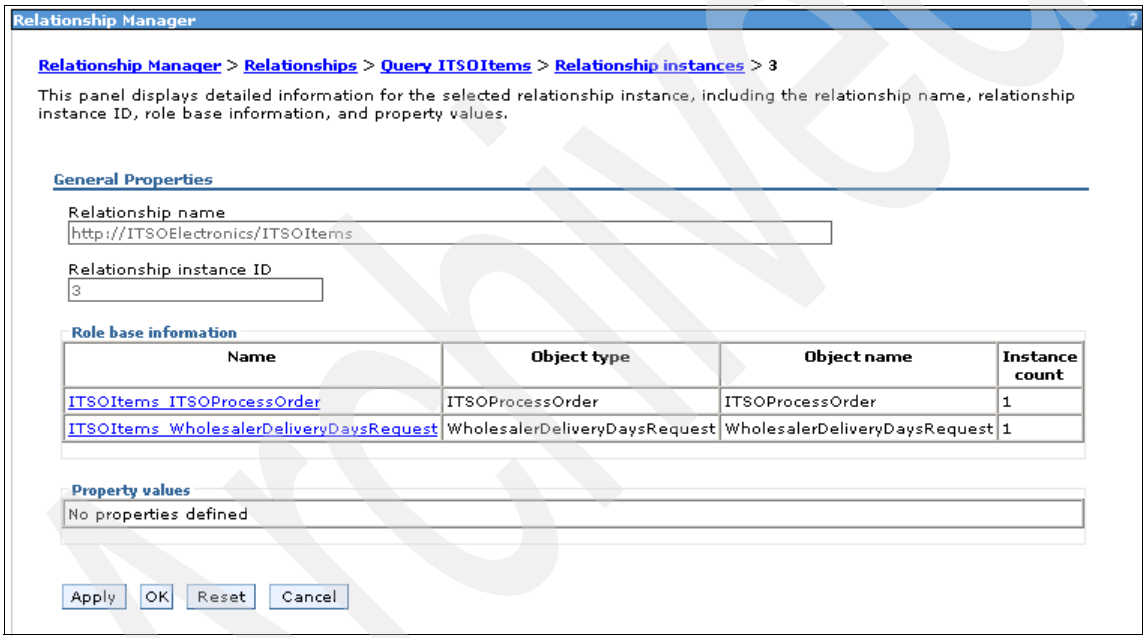


Figure C-19 One of each role in the relationship instance: Visible after requerying the data

19. Add another instance to cross reference part2 to item2 (so you can test with more than one item). You do this exactly the same way as for part1 to item1. Follow the sequence:

    a. Query the relationship.
    b. Create a new relationship instance.
    c. Create a role instance for ITSOElectronics, with key attribute value `part2`.
    d. Save the role instance (Apply changes).
    e. Create a role instance for Wholesaler F, with key attribute value `item2`.

f. Save the role instance.
g. Verify that the data is entered correctly. Go back to the Query ITSOItems page and requery the data. You should now see both your relationship instances, each with two role instances.

20. Close the Relationship Manager.

We have entered the cross-reference data for the two items and two roles into our relationship. They are now stored in database tables accessed by the WebSphere Process Server relationship service.

Creating the cross-reference data mapping: Delivery days

Now that we have created the relationship and added some data instances into it, we must use the relationship in our business process. We want to cross-reference the Wholesaler F part numbers to the ITSOElectronics part numbers only when we make service calls to Wholesaler F. We add the cross-referencing logic to maps which map the ITSOElectronics interface to the Wholesaler F interface.

Note: WebSphere Integration Developer does not allow an interface map to be constructed which maps an interface to itself. If Wholesaler F uses the same interface as ITSOElectronics, we have to place the cross-referencing logic elsewhere. The same relationship is still used.

We create an interface map between the ITSOElectronics interface ITSODelDays_I and the Wholesaler F interface WhDelReqWeb_I. This is used when querying the lead-times from Wholesaler F. We also create an interface map between ITSOPlaceOrd_I and WhPlaceOrdWeb_I. This is used to place the order with Wholesaler F.

Table C-2 summarizes the maps that we create.

Table C-2 Mapping artefact summary

Data map	From	To	Comments
ITSO_to_WHF_Del Interface Map			
ITSOOrder_To_WHFDelDays	ITSOElectronics	WholesalerF	Cross references item
ITSODelDays_to_Order	WholesalerF	ITSOElectronics	No cross references needed as item is not in this map. Reuse from Wholesaler A.

Archived

Data map	From	To	Comments
ITSO_To_WhF_Order Interface Map			
ITSOOrder_To_WhFOrder	ITSOElectronics	WholesalerF	Cross references item.
ITSOPlaceOrder_to_Order	WholesalerF	ITSOElectronics	No cross references needed as item is not in this map. Reuse from Wholesaler A.

First, we create the ITSOOrder_To_WHFDelDays map.

1. Create a new business object map in the ITSOElectronics module. Name it `ITSOOrder_To_WHFDelDays` and click **Next**.
2. For the input, select **ITSOProcessOrder**. For the output, select **WholesalerDeliveryDaysRequest**. Click **Finish**.
3. In the Business object map editor, map ITSOProcessOrder → OrderPart → qty to WholesalerDeliveryDaysRequest → quantity with a Move transformation.
4. Map the business object ITSOProcessOrder to the business object WholesalerDeliveryRequest with a Custom transformation. Figure C-20 shows how the map should look.

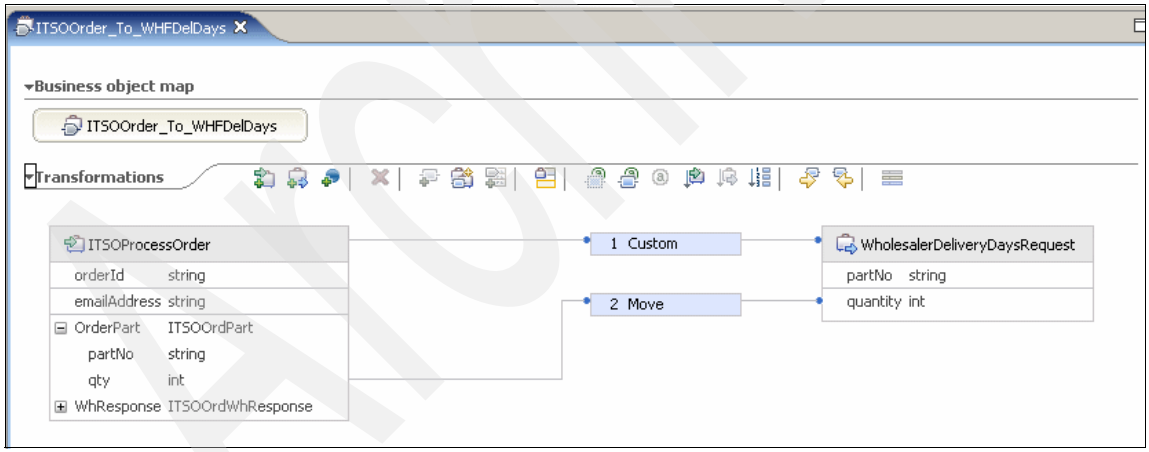


Figure C-20 The transformations for map ITSOOrder_To_WHFDelDays

5. Click the **Custom** transformation. Then click the **Properties** tab and the **Details** subtab.

6. When the Visual Snippet Editor opens, create the visual snippet shown in Figure C-21. Further details about how to build this snippet are provided in the sections that follow.

> **Note:** Your panel may not look exactly like the one shown in Figure C-21, but much wider. This example was retouched by shortening the length of the wires and moving some of the components. This is done purely to allow it to fit on the page. All the components are as shown, and all the wires are linked to the correct terminals.

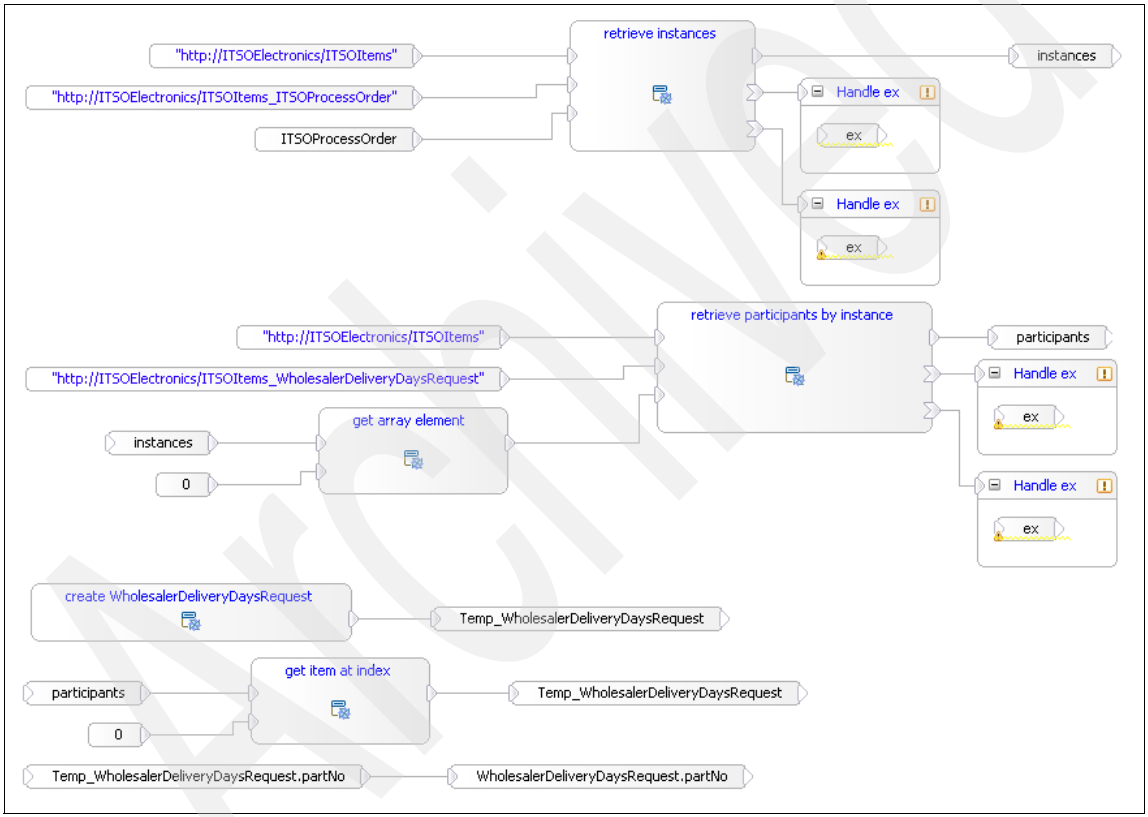


Figure C-21 The custom visual snippet incorporating the relationship API

Archived

Overview of the visual snippet logic

To cross-reference the ITSOElectronics part number to the Wholesaler F item number, follow these steps:

1. Look up the relationship instance which corresponds to the ITSOElectronics part number. We pass in the part number and receive an array of valid instances. Because we only expect one instance, we take the first (and only) array element.
2. Now that we have the instance, we look up the role (participant) for Wholesaler F which corresponds to that instance. This gives us the Wholesaler F item number. We pass in the instance number and receive back a list of business objects. Because we only expect one role for our instance, we take the first (and only) object in the list.
3. Populate the target business object with the item number obtained from the business object that we retrieved from the relationship.

The visual snippet logic consists of four lines of snippets:

- The retrieve instances snippet part
- The retrieve participants by instance snippet part
- The create WholsalerDeliveryDaysRequest snippet part
- The get item at index snippet part

In the following sections, we present each of these lines in turn.

The retrieve instances snippet part

The retrieve instances snippet part is where we take the ITSOElectronics part number and look up the relationship instance for it. We use the snippet *retrieve instances*, which is in the Standard → relationship service folder, accessed from the palette on the left of the Visual Snippet Editor.

The relationship instance is an integer. However, due to the many-many flexibility of relationships, there can be more than one instance for a given role. This means that the instances are returned as an integer array.

The snippet requires the following input:

- Relationship name: In our case, the string http://ITSOElectronics/ITSOItems
- Role name: In our case, the string http://ITSOElectronics/ITSOItems_ITSOProcessOrder
- Business object: In our case, ITSOProcessOrder

Important: The relationship name and role name require the fully qualified name, including the namespace. Without namespaces, the snippet does not work.

The business object ITSOProcess order is the object that corresponds to the role ITSOItems_ITSOProcessOrder. It is the object that we chose when we added the role to the relationship. Even though the whole object is passed into the snippet, only the attribute that we defined as the key (OrderPart → partNo) is used for the lookup.

The snippet returns an array of relationship instances which correspond to the role that we passed in. In our case, there is only one element in this array, so we take the first one.

In a production environment, it is important to check the length of the returned array. If it is empty, or has more than one element, then the result may not be what you expect and appropriate processing action should be taken.

The snippet also throws two exceptions if an error occurs:

- RelationshipServiceException
- RelationshipUserException

Important: If an instance is not found by the lookup, it does not throw an exception, but returns an empty instance array. In a production environment, always check the length of the returned instance array to make sure that it is what you expect.

In our example, we do not deal explicitly with the exceptions. Although we do not add processing for the exceptions, we must add handlers for them:

1. On the right side of the snippet, right-click the **exception terminal** and click **Add** → **Exception Handler**.
2. Repeat this for the second exception terminal.

Tip: The exception handlers catch the exception and do nothing, similar to an empty catch block in Java. This may be fine for our example if all works well, but is not best practice. To see the exceptions, right-click the exception handler and click **Add** → **Throw**. Click inside the exception handler and then wire the *ex* node to the throw Exception node. If an exception occurs in the snippet, it is now rethrown and shown in the log.

The retrieve participants by instance snippet part

We now have an array of instances. In our case, we only expect one instance, so we select the first array element by using the *get array element* snippet in the array folder and passing in the element number 0.

Each instance is represented by an integer. We now have the integer which represents the instance for our item.

To retrieve the Warehouse F role for our instance, we use the *retrieve participants by instance* snippet from the relationship services folder. A participant is another name for a role.

This gives us a list of participant objects which correspond to our instance. As before, due to the many-many capabilities of WebSphere Process Server, we can have many participants for a single relationship instance. In our case, we presume that we have one and take the first element from the list.

The retrieve participants by instance snippet requires the following inputs (make sure you include the namespaces where required):

- Relationship name: In our case, the string http://ITSOElectronics/ITSOItems

  This is the same as for the retrieve instances snippet because we are using the same relationship.

- Role name: In our case, the string http://ITSOElectronics/ITSOItems_WholesalerDeliveryDaysRequest

  We now use the role for Warehouse F.

- Instance ID: In our case, the integer which corresponds to the instance we retrieved earlier

  We get this as the first element of the instances array.

The snippet returns a list of business objects of type WholesalerDeliveryDaysRequest. Because we only expect one role (participant) for our relationship instance, we take the first (and only) business object in the list. Since the list is a Java list, we must cast the object into the correct type.

The create WholsalerDeliveryDaysRequest snippet part

We create a temporary instance of the WholesalerDeliveryDaysRequest business object to store the first item in the list of results and make sure that it is the correct business object type.

Archived

The get item at index snippet part

Here we access the first element in the results array. In a production situation, it is important to check the number of elements in the list. If you are expecting one element (as we are here), then if the list contains 0 or multiple elements, you must take the appropriate action. For our example, we presume that the data is correct.

We place the returned object into the temporary WholesalerDeliveryDaysRequest object and move the partNo attribute into the map target Wholesaler delivery days object. This partNo is the Wholesaler F part number that we wanted in the first place. The cross-reference is complete.

In the last snippet part, we cross referenced the ITSOElectronics representation of the part number to the Wholesaler F representation, using the ITSOItems relationship. We can reuse the relationship in any number of maps and only need to define the cross-reference data once.

Creating the interface map: Delivery days

Now that we have the cross-reference data map, we can create the interface map. We reuse the data map ITSODelDays_to_Order to map the data back from Wholesaler F. Because the part number is not part of the return interface, no cross-referencing is required. Refer to Table C-2 on page 645 for details about which data maps are used within which interface maps.

To create the interface map:

1. Create a new interface map named `ITSO_to_WhF_Del` in the ITSOElectronics module.
2. Map the getDeliveryDays operation of the ITSODelDays_I interface to the getDeliveryDays operation in the WhDelReqWeb_I interface.
3. Map the parameters using the maps listed in Table C-3.

Table C-3 Parameter mappings for the ITSO_to_WhFDel interface map

From parameter	To parameter	Data/business object map
getDeliveryDaysWhReq	getDeliveryDaysWholesaler Request	ITSOOrder_To_ WHFDelDays
getDeliveryDays WholesalerResponse	getDeliveryDaysWhRes	ITSODelDays_ to_Order

Archived

4. Figure C-22 shows how the interface map should look. Save the interface map and close the editor.

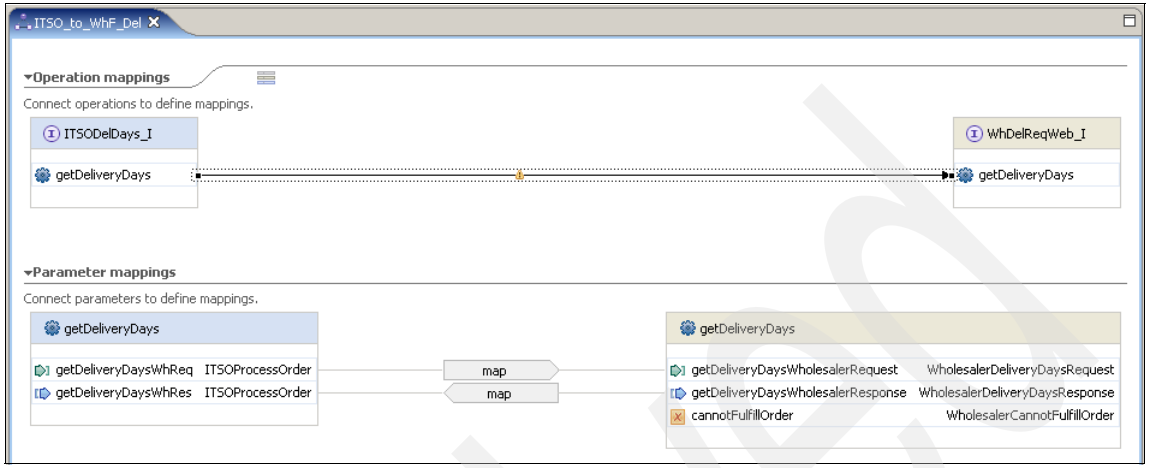


Figure C-22 The ITSO_to_WhFDel interface map

Test point: At this point, we recommend that you test your new interface map, to check that the relationship lookups are working. To do this, drag the interface map onto the ITSOElectronics assembly diagram, right-click the interface map and select Test. Use the test client to check that the map correctly cross-references part1 to item1.

In the test client, enter `part1` into the partNo attribute of the ITSOProcessOrder business object. The resulting WholesalerDeliveryDaysRequest object should have item1 in its partNo attribute for a successful test.

Creating the cross-reference data mapping: Place order

This map is similar to the one that we created for delivery days. As such, the instructions here are shorter. Refer to “Creating the cross-reference data mapping: Delivery days” on page 645 for any queries.

Now we create the interface map between the ITSOElectronics interface ITSOPlaceOrd_I and the Wholesaler F interface WhPlaceOrdWeb_I. This is used when querying the lead-times from Wholesaler F. Table C-2 on page 645 summarizes the maps that we create.

First, we create the ITSOOrder_To_WHFDelDays map.

1. Create a new data map in the ITSOElectronics module. Name it `ITSOOrder_To_WhFOrder`.
2. For input, select **ITSOProcessOrder**. For output, select **WholesalerPlaceOrderRequest**.
3. In the Business Object Map Editor, map ITSOProcessOrder → OrderPart → qty to WholesalerPlaceOrderRequest → quantity with a Move transformation.
4. Map the business object ITSOProcessOrder to the attribute WholesalerPlaceOrderRequest → partNo with a Custom transformation. Figure C-23 shows how the map should look.

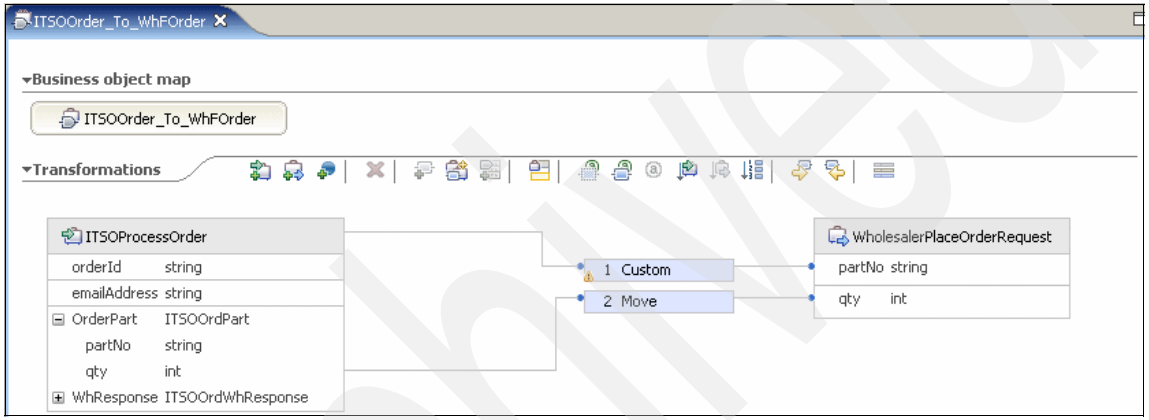


Figure C-23 The ITSOOrder_To_WhFOrder data map

Archived

5. Click the **Custom** transformation and the **Properties** → **Details** tab. When the Visual Snippet Editor opens, create the visual snippet (see Figure C-24).

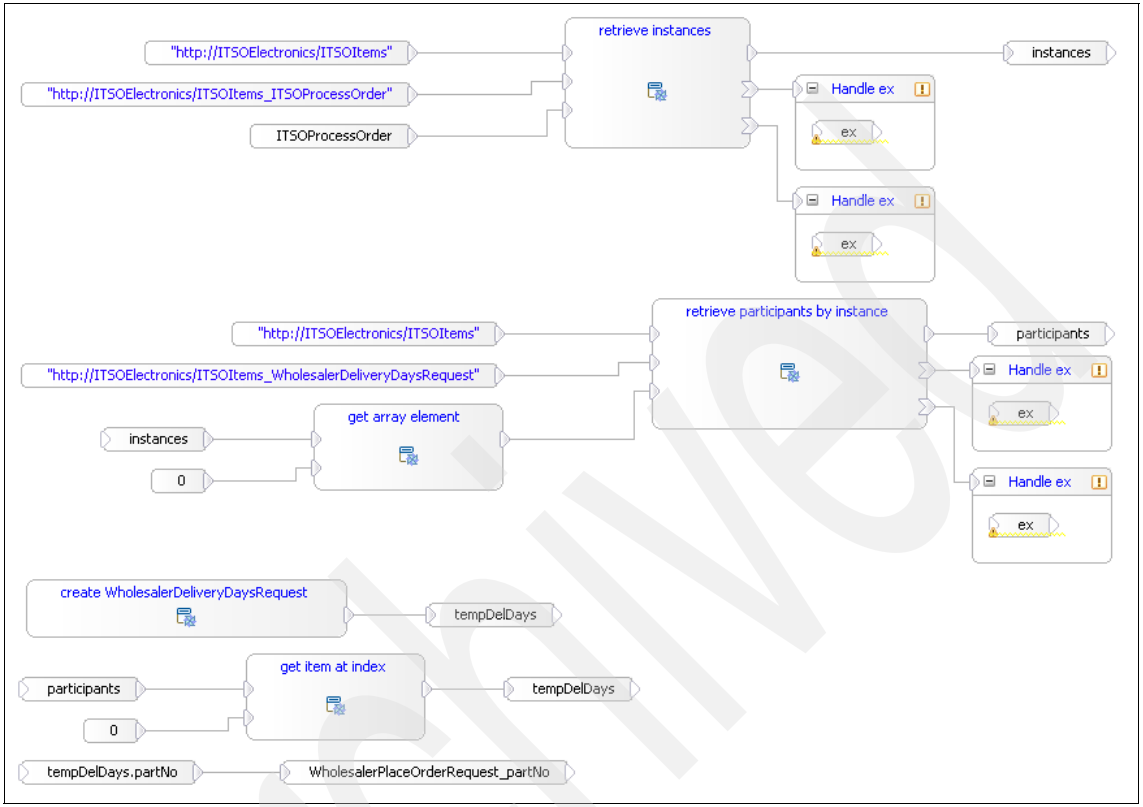


Figure C-24 The custom snippet

Creating the interface map: Place order

Now that we have the cross-reference data map, we can create the interface map. We reuse the data map ITSOPlaceOrder_to_Order to map the data back from Wholesaler F. Because the part number is not part of the return interface, no cross-referencing is required.

To create the interface map:

1. Create a new interface map named ITSO_to_WhF_Order in the ITSOElectronics module.
2. Map the placeOrder operation of the ITSOPlaceOrd_I interface to the placeOrder operation in the WhPlaceOrdWeb_I interface.

3. Map the parameters using the maps listed in Table C-4.

Table C-4 Parameter mappings for the ITSO_to_WhF_Order interface map

From parameter	To parameter	Data/business object map
orderReq	placeOrderRequest	ITSOOrder_To_WhFOrder
placeOrderResponse	orderConf	ITSOPlaceOrder_to_Order

4. Figure C-25 shows how the interface map should look. Save the interface map and close the editor.

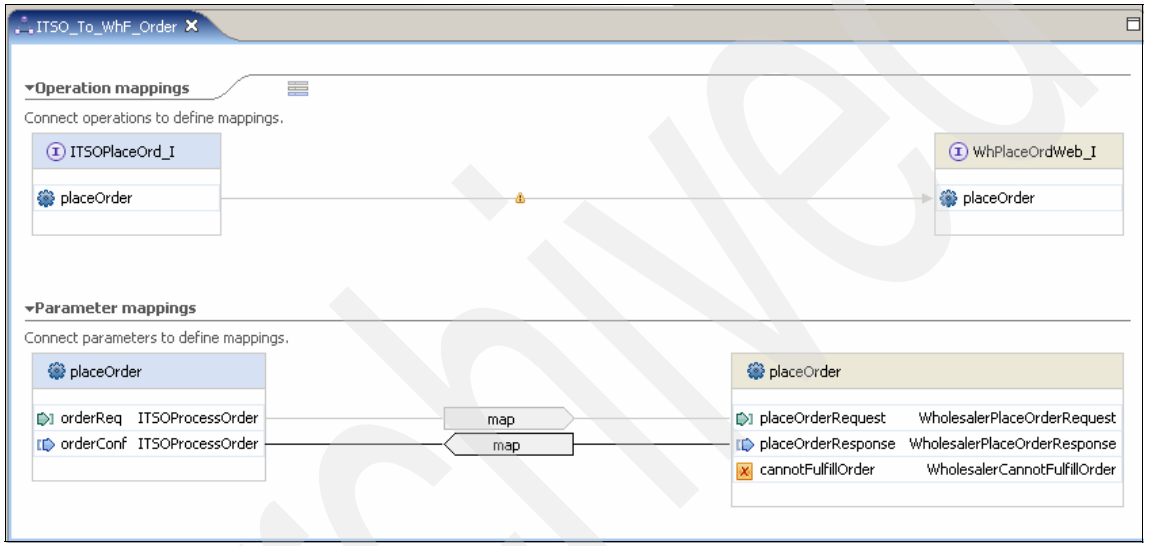


Figure C-25 The ITSO_to_WhF_Order interface map

Test point: At this point, we recommend that you test your new interface map, to check that the relationship lookups are working. To do this, drag the interface map onto the ITSOElectronics assembly diagram, right-click the ***interface map*** and select **Test**. Use the test client to check that the map correctly cross-references part1 to item1, in the same way as you tested the previous map.

Archived

Modifying the ITSOElectronics assembly diagram

We now have a new Wholesaler F module and two new interface maps to map its interfaces from and to the ITSOElectronics interfaces. We also created the relationship and populated it with data. It's time to put the whole thing together. We modify the ITSOElectronics assembly diagram to replace Wholesaler A with Wholesaler F.

The ITSOElectronics assembly diagram currently looks something like the example in Figure C-26. The important components are:

- ITSO_to_WhA_Del: Interface Map
- ITSO_to_WhA_Order: Interface Map
- WhDelReqWeb_I: The import of the Wholesaler A delivery days service
- WhPLaceOrderWeb_I: The import of the Wholesaler A place order service

We replace these components with their Wholesaler F equivalents.

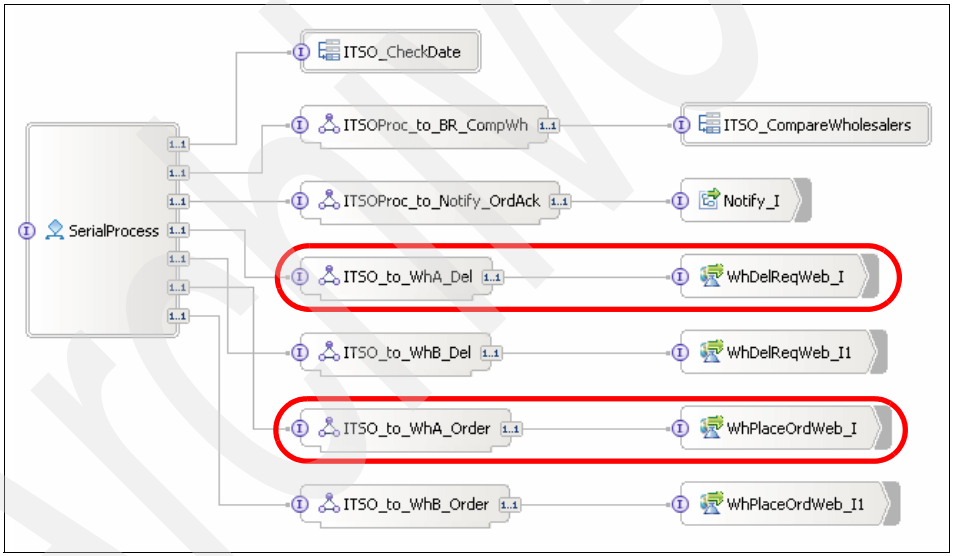


Figure C-26 ITSOElectronics assembly diagram

Archived

1. Remove the following components from the assembly diagram, by right-clicking each **component** and selecting **Delete**.
   - ITSO_to_WhA_Del
   - ITSO_to_WhA_Order
   - WhDelReqWeb_I
   - WhPLaceOrderWeb_I
2. All references to Wholesaler A are now removed. Add the Wholesaler F components. Drag the following interface maps onto the assembly diagram:
   - ITSO_to_WhF_Del
   - ITSO_to_WhF_Order
3. Now that we have added our maps, import the Wholesaler F services. Drag the following components from the WholesalerF assembly diagram to the ITSOElectronics assembly diagram and import with SCA bindings.
   - WholesalerF_GetDeliveryDateExport: Rename to `WhFDelDateImport`
   - WholesalerF_PlaceOrderExport: Rename to `WhFPlaceOrdImport`

   Note: If you see a Manage Project Dependencies window, click **Cancel** because we do not need to update the project dependencies at this point. WholesalerF and ITSOElectronics are independent modules.

4. Wire the following components together:
   - ITSO_to_WhF_Del to WhFDelDateImport
   - ITSO_To_WhF_Order to WhFPlaceOrdImport
   - ITSO_to_WhF_Del to the WhADeliveryPartner reference of the SerialProcess component
   - ISTO_To_WhF_Order to the WhAOrderPartner reference of the SerialProcess component

   The result (with some rearranging) should look like the example in Figure C-27. Don't worry if your layout is not similar. Make sure that the components are connected to each other in the way shown in the figure.

Archived

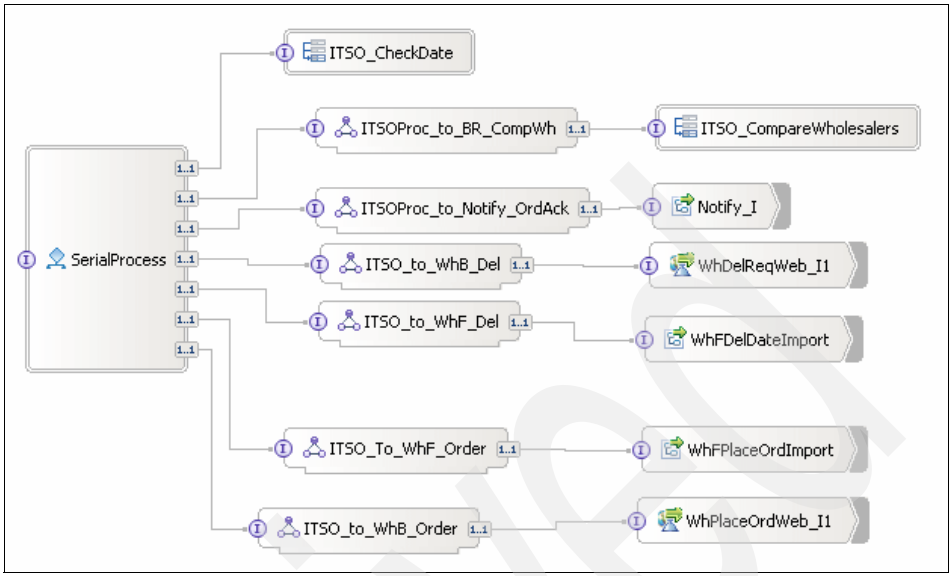


Figure C-27 Completed assembly diagram with Wholesaler F components

5. Save the assembly diagram.

Testing relationships

Using the Integration Test Client, we can test the relationships by using the Serial Process to query and place an order with Wholesaler F.

1. Right-click the **ITSOElectronics** module and select **Test** → **Test Module**.
2. In the Integration Test Client that opens, we place an order with Wholesaler F.
   a. Set the Component to SerialProcess to indicate that we want to test the Web Services Business Process Execution Language (WS-BPEL) business process. SerialProcess only has a single interface and operation, so these two items are selected automatically.
   b. Under the Initial request parameters, specify the values shown in Figure C-28. The important parameters are the partNo, which we set to `part1`, and qty, which we set to `1`. These parameters determine the behavior of the business process. This test should cause Wholesaler F to be selected as the preferred wholesaler.

Initial request parameters

Name	Type	Value
order	ITSOProcessOrder	
orderID	string	myOrderId
emailAddress	string	myEmail@itso.com
OrderPart	ITSOOrdPart	
partNo	string	part1
qty	int	1
WhResponse	ITSOOrdWhRespo...	
confNo	string	
bestWholes...	string	
bestWholes...	int	0
price	double	0

Figure C-28 Initial request parameters

c. Click the **Continue** button.

d. In the Deployment Location window that opens, click **Finish**.

After the test is complete, the editor should look like the example in Figure C-29. Although it indicates that Wholesaler A was chosen, you can see that the order was placed with Wholesaler F. Also notice that an order for item1 was placed (not part1) with Wholesaler F.

Events

- Invoke (SerialProcess:requestOrder)
  - Started
    - Invoke (SerialProcess:requestOrder)
    - Request (SerialProcess --> ITSO_to_Whf_Del:getDelivery
    - Request (ITSO_to_Whf_Del --> WhFDelDateImport:getD
    - Response (ITSO_to_Whf_Del <-- WhFDelDateImport:get
    - Response (SerialProcess <-- ITSO_to_Whf_Del:getDelive
    - Request (SerialProcess --> ITSO_CheckDate:checkDelive
    - Response (SerialProcess <-- ITSO_CheckDate:checkDeliv
    - Request (SerialProcess --> ITSO_to_WhF:placeOrder)
    - Request (ITSO_to_WhF --> WhFPlaceOrdImport:placeOr
    - Response (ITSO_to_WhF <-- WhFPlaceOrdImport:placeO
    - Response (SerialProcess <-- ITSO_to_WhF:placeOrder)
    - Request (SerialProcess --> ITSOProc_to_Notify_OrdAck:
    - Request (ITSOProc_to_Notify_OrdAck --> Notify_I:order
  - Stopped

General Properties

Detailed Properties

Module: ITSOElectronics

Source component: ITSO_to_WhF

Source reference: WhPlaceOrdWeb_I

Target component: WhFPlaceOrdImport

Target interface: WhPlaceOrdWeb_I

Target operation: placeOrder

Request parameters

Name	Type	Value
placeOrderRequest	WholesalerPlaceOr...	
partNo	String	item1
qty	Integer	1

Figure C-29 Successful response from Wholesaler F

Archived

3. Try another test, this time specifying a partNo of `part2`. Although Wholesaler B is selected to place the order, you should see that a request for delivery days for item2 is placed with Wholesaler F (Figure C-30).

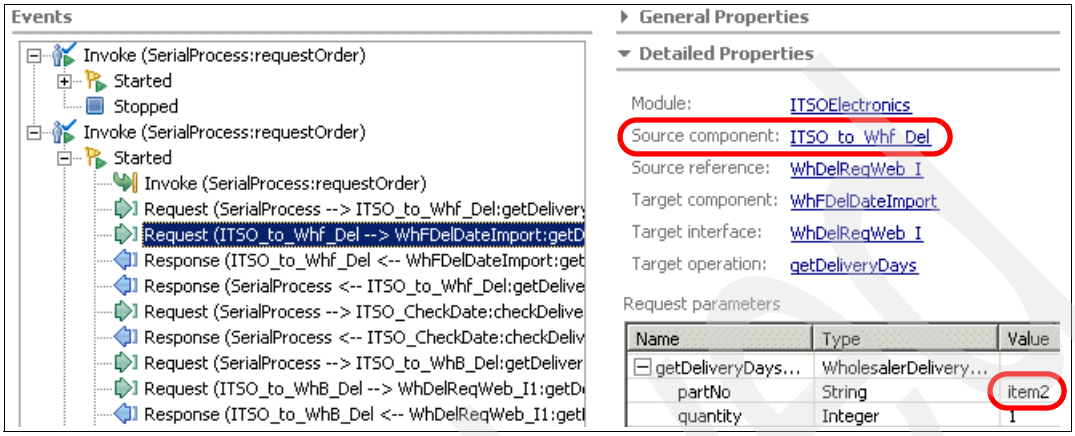


Figure C-30 Requesting delivery days from Wholesaler F

Archived

D

WebSphere Process Server dynamicity

In this appendix, we explain how to apply dynamicity to business processes. We apply dynamicity to the Parallel Workflow process, PlaceOrderProcess, built in Chapter 13, “Parallel Workflow scenario” on page 393. We also replace Wholesaler A with Wholesaler C without any changes to the Parallel Workflow process.

This appendix builds the service implementation for the Wholesaler C application and a mediator module, WholesalerCMediator, which implements the mediation between the Parallel Workflow module, ITSOPModule, and the Wholesaler C Web service.

If you do not want to build the Wholesaler C service implementation, or the mediation module WholesalerCMediator, you can import the project interchange file containing the modules into your workspace. You can find the project interchange file in the \Wholesaler\WholesalerC directory, in the additional materials supplied with this IBM Redbook. For details about accessing the additional materials, see Appendix E, “Additional material” on page 689.

© Copyright IBM Corp. 2006. All rights reserved.

Scenario implementation overview

Figure D-1 shows the completed assembly diagram of the Wholesaler C service implementation.

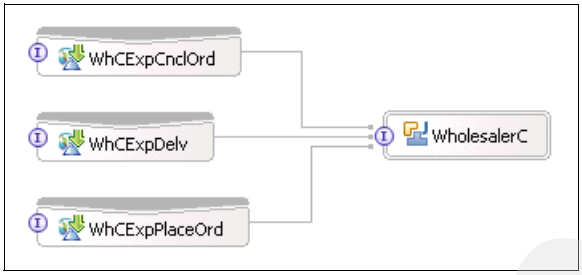


Figure D-1 The completed Wholesaler C assembly diagram

The WholesalerC service is implemented as a Java component with three Web service binding exports. Figure D-2 shows the completed Wholesaler C mediator implementation assembly diagram.

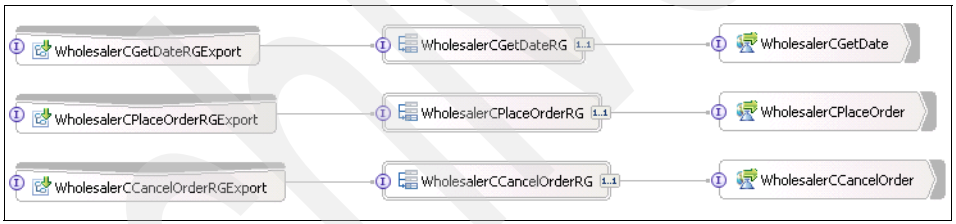


Figure D-2 The completed WholesalerCMediator assembly diagram

The WholesalerCMediator module consists of:

- Three Service Component Architecture (SCA) binding exports, which can be invoked by a selector residing in another module
- Three business rules that allow us to invoke WholesalerC even if Wholesaler has a different interface than Wholesaler A and Wholesaler B (In this scenario, the interfaces are the same.)
- Three Web service binding imports from WholesalerC

Archived

Preparing the workspace

To proceed with this appendix, you need a working Parallel Workflow workspace. If you do not have this, start a new workspace in WebSphere Integration Developer and import the following projects from the additional material:

- WholesalerA in \Wholesalers\WholesalerA\WholesalerA.zip
- WholesalerB in \Wholesalers\WholesalerB\WholesalerB.zip
- ITSOLibrary in \ParallelWorkflow\solution\ITSOLibrary.zip
- ITSOPModule in \ParallelWorkflow\solution\ITSOPModule.zip

Note: To import these modules, select **File** → **Import** → **Project Interchange**.

Creating the service implementation for Wholesaler C

In this section, we explain how to implement Wholesaler C. The business logic for this wholesaler is exposed as a Web service. Wholesaler C implements the same business operations as Wholesaler A and Wholesaler B. All three are implemented in Java:

- Cancel Order
- Get Delivery Days
- Place Order

Wholesaler C implements the same interfaces as Wholesalers A and B. See Appendix A, “Building Wholesaler A and B service implementations” on page 501:

- WhCnclOrdWeb_I
- WhDelReqWeb_I
- WhPlaceOrdWeb_I

Creating the Wholesaler C module

To create the Wholesaler C module:

1. Right-click the **Business Integration** view in the Business Integration perspective and click **New** → **Module**.
2. In the New Module wizard, complete these steps:
   a. In the Module name field, type `WholesalerC`.
   b. For Module Location Directory, select **Use default**.
   c. Click **Finish**.

The Wholesaler C module has no dependencies.

Archived

Copying the business objects and interfaces from WholesalerA

We need to copy the following business objects from the WholesalerA module to the WholesalerC module:

- WholesalerCancelOrderRequest
- WholesalerCancelOrderResponse
- WholesalerCannotFulfillOrder
- WholesalerDeliveryDaysRequest
- WholesalerDeliveryDaysResponse
- WholesalerPlaceOrderRequest
- WholesalerPlaceOrderResponse

We must also copy the following interfaces from the WholesalerA module to the WholesalerC module:

- WhDelReqWeb_I
- WhPlaceOrdWeb_I
- WhCnclOrdWeb_I

Note: Because WholesalerA and WholesalerC are separate entities, we copy the artifacts from the WholesalerA module to the WholesalerC module. If these wholesalers are part of the same company, it may be appropriate to use a shared library project to avoid duplication.

1. Highlight the business objects from the WholesalerA module, right-click the ***selected business objects***, and click **Copy**.
2. Right-click the **WholesalerC** module and click **Paste**. Notice that the namespaces are the same as for WholesalerA.
3. Highlight the interfaces from the WholesalerA module, right-click the ***selected interfaces***, and click **Copy**.
4. Right-click the **WholesalerC** module and click **Paste**.

Creating the business logic using the assembly diagram

In this section, we explain how to build the WholesalerC business logic using the WholesalerC module assembly diagram. The Wholesaler C services are implemented in Java. We also create Web service binding exports, which allows the Wholesaler C service implementation to be invoked as a Web service.

Note: You do not need to know Java to create this component. We provide step-by-step instructions and pre-built code for you to use.

Archived

Creating the WholesalerC Java component

Add the WholesalerC Java component to the WholesalerC module assembly diagram. Before you begin, open the Assembly Diagram Editor for the WholesalerC module.

1. On the Assembly Editor palette, click the **Java** icon ( ).
2. Click the **assembly diagram canvas** to add the new Java component.
3. Click the **Properties** page.
4. Click the **Description** tab. In the Display name field, type `WholesalerC` (Figure D-3).

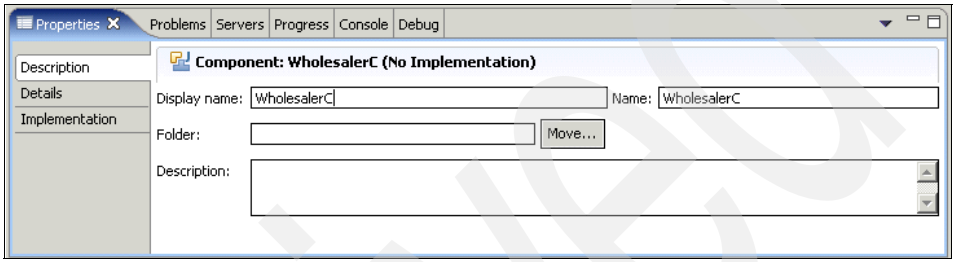


Figure D-3 WholesalerC Java component Properties page: Description tab

5. Click the **Details** tab.
6. Right-click **Interfaces** and click **Add Interface**.
7. In the Add Interface window, complete these steps:
   a. Click the **Show WSDL** radio button to display only the WSDL interfaces.
   b. Select the **WhCnclOrdWeb_I interface**.
   c. Click **OK**.
8. Repeat this process to add the interface for WhDelReqWeb_I and WhPlaceOrdWeb_I (Figure D-4).

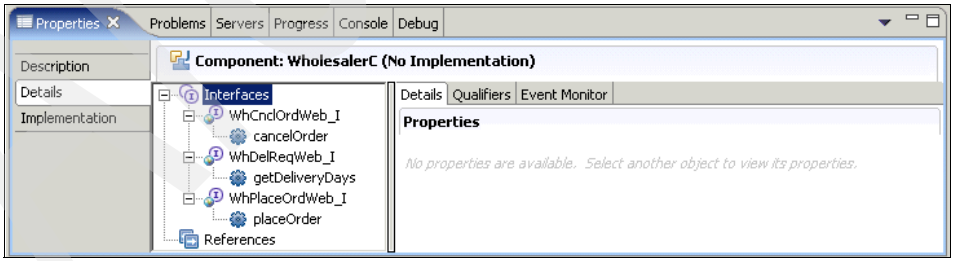


Figure D-4 WholesalerC Java component Properties page: Details tab

Archived

Figure D-5 shows the assembly diagram of WholesalerC.

Figure D-5 Assembly diagram of WholesalerC with the WholesalerC Java component

Generating the implementation for the Java component

Now we generate an implementation Java class for the WholesalerC Java component. When the skeleton Java class is generated, we provide implementations for the cancelOrder, getDeliveryDays, and placeOrder services.

1. Generate an implementation for the WholesalerC Java component:
   a. Right-click the **WholesalerC** Java component and click **Generate Implementation**.
   b. In the Generate Implementation window, complete these steps:
      i. Click **New Package**.
      ii. In Package name field, type `com.ibm.itso.wholesalerC`.
      iii. Click **OK**.
      iv. In the Generate Implementation window list box, select the **com.ibm.itso.wholesalerC** package. Click **OK**.
2. A skeleton Java class WholesalerCImpl.java is generated and opens in the Java Editor. This Java class contains one constructor and four methods. Leave the constructor and the private method getMyService() unchanged, but provide implementations for the following three public methods:
   – cancelOrder()
   – getDeliveryDays()
   – placeOrder()

   The complete code for this Java class is provided in the \Wholesalers\WholesalerC directory in the additional material supplied with this book. You can choose to paste the code into the WholesalerCImpl.java file or follow these instructions to add the code:

   a. Implement the cancelOrder() method (Example D-1). This method performs the following actions:
      - Prints to the server log that we entered the cancelOrder() method
      - Locates the BOFactory service, which we use to create the response business object
      - Creates the response business object using the BOFactory

        The response business object type is WholesalerCancelOrderResponse.

- Sets the orderCancelled attribute on the response object to true, to confirm that the order has been cancelled
- Prints to the server log that we are about to exit the cancelOrder() method
- Returns the response object

Example: D-1 The cancelOrder() method Java code

```
   public DataObject cancelOrder(DataObject cancelOrderRequest) {
      System.out.println("WholesalerCImpl.cancelOrder() entry");
      BOFactory boFactory = (BOFactory)
ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/BOFactory");
      DataObject response = boFactory.create("http://Wholesaler",
"WholesalerCancelOrderResponse");
      response.setBoolean("orderCancelled", true);
      System.out.println("WholesalerCImpl.cancelOrder() exit");

      return response;
   }
```

b. Implement the getDeliveryDays() method (Example D-2). The method performs the following actions:

- Prints to the server log that we entered the getDeliveryDays() method
- Locates the BOFactory service, which we use to create the response business object
- Creates the response business object using the BOFactory

  The response business object type is WholesalerDeliveryDaysResponse.
- Gets the part number from the method input object, getDeliveryDaysWholesalerRequest

  The part number is stored in the local variable partNo of type String.
- Defines a local variable, deliveryDays, of type int

  This variable is used to temporarily store the delivery days information based on the part number.
- Stores the delivery days information in the deliveryDays variable

  If the part number, partNo, is part1, then deliveryDays is 4; otherwise deliveryDays is 8.
- Sets the daysToDeliver attribute on the response object to the local variable deliveryDays
- Sets the price attribute of the response object to 5.9 multiplied by the quantity from the method input object

Archived

- Prints to the server log that we are about to exit the getDeliveryDays() method
- Returns the response object

Example: D-2 The getDeliveryDays() method Java code

```
   public DataObject getDeliveryDays(
         DataObject getDeliveryDaysWholesalerRequest)
         throws ServiceBusinessException {
      System.out.println("WholesalerCImpl.getDeliveryDays() entry");
      BOFactory boFactory = (BOFactory)
ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/BOFactory");
      DataObject response = boFactory.create("http://Wholesaler",
"WholesalerDeliveryDaysResponse");
      String partNo=getDeliveryDaysWholesalerRequest.getString("partNo");

      int deliveryDays;
      //If part1, 4 days, else 8 days
      if(partNo.equalsIgnoreCase("part1")) {
         deliveryDays=4;
      }
      else {
         deliveryDays=8;
      }

      System.out.println("response set daysToDeliver");
      response.setInt("daysToDeliver", deliveryDays);
      System.out.println("response set price");
      response.setDouble("price", 5.9*getDeliveryDaysWholesalerRequest.getInt("quantity"));
      System.out.println("WholesalerCImpl.getDeliveryDays() exit");

      return response;
   }
```

c. Implement the placeOrder() method (Example D-3). This method performs the following actions:

- Prints to the server log that we entered the placeOrder() method
- Locates the BOFactory service, which we use to create the response business object
- Creates the response business object using the BOFactory

  The response business object type is WholesalerPlaceOrderResponse.
- Sets the confirmCode attribute on the response object to order42 to confirm the order has been placed

- Prints to the server log that we are about to exit the placeOrder() method
- Returns the response object

Example: D-3 The placeOrder() method Java code

```
   public DataObject placeOrder(DataObject placeOrderRequest)
          throws ServiceBusinessException {
      System.out.println("WholesalerCImpl.placeOrder() entry");
      BOFactory boFactory = (BOFactory)
ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/BOFactory");
      DataObject response = boFactory.create("http://Wholesaler",
"WholesalerPlaceOrderResponse");
      response.setString("confirmCode", "order42");
      System.out.println("WholesalerCImpl.placeOrder() exit");

      return response;
   }
```

3. Press Ctrl+S to save WholesalerCImpl.java.
4. Close the WholesalerCImpl Java editor.
5. Press Ctrl+S to save the assembly diagram for WholesalerC.

Generating the Web service exports

To generate Web service exports from the WholesalerC Java component:

1. Right-click the **WholesalerC** Java component and click **Export** → **Web Service Binding**.
2. Complete the following steps:
   a. In the Select Interface window, select the **WhCnclOrdWeb_I** check box, and click **OK**.
   b. In the Binding File Generation window, click **No**. This creates an export in the Assembly editor.
   c. Click the **WholesalerCExport**.
   d. Click the **Properties** page.
   e. Click the **Description** tab. In the Display name field, type `WhCExpCnclOrd`.
   f. Right-click the **WhCExpCnclOrd** Web service export and click **Replace Binding**.
   g. In the Replace Binding window, click **OK**.

Archived

h. In the Binding File Generation window, click **Yes**.

i. In the Binding Information window, click **Yes**.

j. In the Select Transport window, select **soap/http** and click **OK**.

3. Press Ctrl+S to save the WholesalerC assembly diagram.

4. Right-click the **WholesalerC** Java component and click **Export** → **Web Service Binding**.

5. Complete the following steps:

   a. In the Select Interface window, click the **WhDelReqWeb_I** check box, and click **OK**.

   b. In the Binding File Generation window, click **No**. This creates an export in the Assembly editor.

   c. Click the **WholesalerCExport**.

   d. Click the **Properties** page.

   e. Click the **Description** tab. In the Display name field, type `WhCExpDelv`.

   f. Right-click the **WhCExpDelv** Web service export, and click **Replace Binding**.

   g. In the Replace Binding window, click **OK**.

   h. In the Binding File Generation window, click **Yes**.

   i. In the Binding Information window, click **Yes**.

   j. In the Select Transport window, select **soap/http** and click **OK**.

6. Press Ctrl+S to save the WholesalerC assembly diagram.

7. Right-click the **WholesalerC** Java component and click **Export** → **Web Service Binding**.

8. Complete the following steps:

   a. In the Select Interface window, click the **WhPlaceOrdWeb_I** check box, and click **OK**.

   b. In the Binding File Generation window, click **No**. This creates an export in the Assembly editor.

   c. Click the **WholesalerCExport**.

   d. Click the **Properties** page.

   e. Click the **Description** tab. In the Display name field, type `WhCExpPlaceOrd`.

   f. Right-click the **WhCExpPlaceOrd** Web service export and click **Replace Binding**.

   g. In the Replace Binding window, click **OK**.

Archived

h. In the Binding File Generation window, click **Yes**.

i. In the Binding Information window, click **Yes**.

j. In the Select Transport window, select **soap/http** and click **OK**.

9. Press Ctrl+S to save the WholesalerC assembly diagram.

> **Important:** Exporting all three interfaces at the same time creates three WSDL files but only one binding. Therefore, we recommend that you *do not* do this.

We have now generated the three Web service exports as three WSDL files with three different bindings in the WholesalerC module Web Service Ports folder:

- WhCExpCnclOrd_WhCnclOrdWeb_IHttpPort
- WhCExpDelv_WhDelReqWeb_IHttpPort
- WhCExpPlaceOrd_WhPlaceOrdWeb_IHttpPort

Figure D-6 shows the completed WholesalerC assembly diagram.

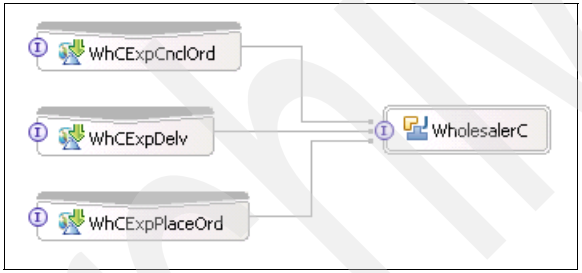


Figure D-6 The WholesalerC assembly diagram with WholesalerC and the Exports

Creating the implementation for Wholesaler C mediator

In this section, we describe how to implement the Wholesaler C mediator. The mediator is exposed as three SCA bindings that can be invoked from a Selector in another module.

The Wholesaler C business logic is exposed to the outside world as Web services. The reason we need the Wholesaler C mediator module is that the selectors in the Parallel Workflow module cannot directly call Web service exports in other modules, like WholesalerC. Therefore, we have to implement a mediator that enables the Parallel Workflow process to call the Web services. The Wholesaler C mediator module consists of:

Archived

- Three SCA binding exports, which can be invoked by a selector residing in another module
- Three business rules that allow us to invoke WholesalerC even if WholesalerC has a different interface than WholesalerA and WholesalerB (In this scenario, the interfaces are the same.)
- Three Web service binding imports from WholesalerC

Important: If you want to change the target of a selector at run time, it can only be changed to targets, with the same interface, inside its own module or SCA binding exports or Java Message Service (JMS) binding exports in other modules with the same interface. A selector *cannot* directly invoke Web service binding exports from other modules.

Creating the Wholesaler C mediator module

To create the Wholesaler C mediator module, WholesalerCMediator:

1. Right-click the **Business Integration** view in the Business Integration perspective and click **New** → **Module**.
2. In the New Module wizard, complete these steps:
   a. In the Module name field, type `WholesalerCMediator`.
   b. For Module Location Directory, select **Use default**.
   c. Click **Finish**.
3. To configure the WholesalerCMediator module required library ITSOLibrary:
   a. Right-click **WholesalerCMediator** and click **Open Dependency Editor**.
   b. In the WholesalerCMediator Dependencies window, click **Add**.
   c. In the Library Selection window, select the **ITSOLibrary** library, and click **OK**.
   d. Press Ctrl+S to save the module.

Note: Because the Parallel Workflow module and the WholesalerCMediator module are part of the same entity, we use a shared library project, ITSOLibrary, to avoid duplication.

Copying the Web service exports from WholesalerC

We need to copy the Web service exports from WholesalerC module to the WholesalerCMediator module, because these are the exports for which we are creating the mediator.

1. Copy the following Web service ports from the WholesalerC module to the WholesalerCMediator module:
   - WhCExpCnclOrd_WhCnclOrdWeb_IHttpPort
   - WhCExpDelv_WhDelReqWeb_IHttpPort
   - WhCExpPlaceOrd_WhPlaceOrdWeb_IHttpPort
2. Highlight the Web service ports from the WholesalerC module, right-click the selected ***Web service ports*** and click **Copy**.
3. Right-click the **WholesalerCMediator** module and click **Paste**.

Creating the mediator module using the assembly diagram

You can build the WholesalerCMediator module using the WholesalerCMediator module assembly diagram. The Wholesaler C mediator is implemented as:

- Three Web service binding imports from WholesaleC
- Three business rules that allow us to invoke WholesalerC even if WholesalerC has a different interface than WholesalerA and WholesalerB (In this scenario, the interfaces are the same.)
- Three SCA binding exports, which can be invoked by a selector residing in another module

Adding the WholesalerC Web service imports

To add the WholesalerC Web service imports to the WholesalerCMediator assembly diagram:

1. Open the WholesalerCMediator module assembly diagram.
2. Add the WholesalerCCancelOrder import to the assembly diagram:
   a. Drag the **WhCExpCnclOrd_WhCnclOrdWeb_IHttpPort Web Service Port** from the WholesalerCMediator module to the assembly diagram canvas.
   b. In the Component Creation window, select **Import with Web Service Binding**, and click **OK**.
   c. Click **Import1**.
   d. Click the **Properties** page.
   e. Click the **Description** tab. In the Display name field, type `WholesalerCCancelOrder`.

Archived

3. Add two additional imports to the WholesalerCMediator assembly diagram with the properties shown in Table D-1.

Table D-1 Additional imports to add to the WholesalerCMediator assembly diagram

Web service port	Display name
WhCExpDelv_WhDelReqWeb_IHttpPort	WholesalerCGetDate
WhCExpPlaceOrd_WhPlaceOrdWeb_IHttpPort	WholesalerCPlaceOrder

4. Press Ctrl+S to save the assembly diagram.

Figure D-7 shows how the assembly diagram looks after adding the imports.

Figure D-7 The WholesalerCMediator assembly diagram with the Imports

Adding the WholesalerCMediator business rule groups

To add the WholesalerC business rule groups to the WholesalerCMediator assembly diagram:

1. Add the WholesalerCCancelOrderRG business rule group to the assembly diagram.
   a. On the Assembly Editor palette, click the **Rule Group icon** ( ).
   b. Click the **assembly diagram canvas** to add the new rule group.
   c. Click the **Properties** page.
   d. Click the **Description** tab. In the Display name field, type `WholesalerCCancelOrderRG`.
   e. Click the **Details** tab. Right-click **Interfaces** and click **Add Interface**.
   f. In the Add Interface window, select the **WhCnclOrdWeb_I** interface, and click **OK**.

Archived

2. Add two additional rule groups to the WholesalerCMediator assembly diagram with the properties from Table D-2.

Table D-2 Additional Rule Groups for the WholesalerCMediator assembly diagram

Display name	Interface
WholesalerCGetDateRG	WhDelReqWeb_I
WholesalerCPlaceOrderRG	WhPlaceOrdWeb_I

3. Add a wire from the WholesalerCCancelOrderRG business rule group to the WholesalerCCancelOrder Import:

   a. On the Assembly Editor palette, click the **Wire** icon ( ).
   b. Click the **WholesalerCCancelOrderRG** component.
   c. Click the **WholesalerCCancelOrder** import.
   d. In the Add Wire window, click **OK**.

4. Add two additional wires to the WholesalerCMediator assembly diagram with the properties from Table D-3.

Table D-3 Additional wires to add to the WholesalerCMediator assembly diagram

Component	Import
WholesalerCGetDateRG	WholesalerCGetDate
WholesalerCPlaceOrderRG	WholesalerCPlaceOrder

5. Press Ctrl+S to save the assembly diagram.

Figure D-8 shows how the assembly diagram looks after adding the rule groups.

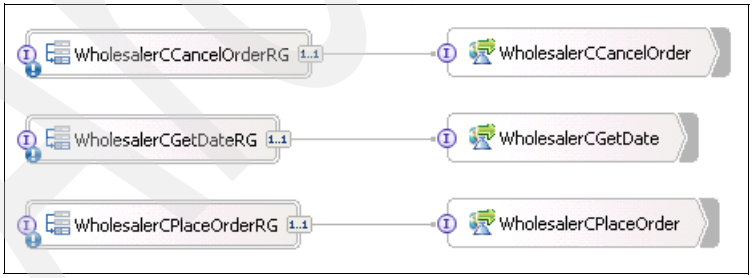


Figure D-8 The WholesalerCMediator assembly diagram with the rule groups

Archived

Generating the implementations for the business rules

To generate implementations for the three WholesalerC business rules.

1. Generate an implementation for the WholesalerCCancelOrderRG business rule group:
   a. Right-click the **WholesalerCCancelOrderRG** component and click **Generate Implementation**.
   b. In the Generate Implementation window, click **OK**.
   c. In the WholesalerCCancelOrderRG Rule Group window, click the **cancelOrder operation**. In the Active Destinations section, click **Enter Destination**, and from the helper menu, click **New Ruleset**.
   d. In the New Rule Set window, in the Name field, type `WholesalerCCancelOrderRS`. Then click **Finish**.
   e. In the WholesalerCCancelOrderRS Rule Set window, complete these steps:
      i. Click the **Add Action Rule** icon ( ).
      ii. Click **Action**.
      iii. Click **Invoke**.
      iv. Click **Select Partner Link**, and from the helper menu, click **WhCnclOrdWeb_IPartner**.
      v. Click **Select Operation**, and from the helper menu, click **cancelOrder**.
      vi. On the Input row, click **Enter Value**, and from the helper menu, click **cancelOrderRequest**.
      vii. On the Output row, click **Enter Value**, and from the helper menu, click **cancelOrderResponse**.
   f. Press Ctrl+S to save the WholesalerCCancelOrderRS business rule set.

      Figure D-9 shows the completed WholesalerCCancelOrderRS business rule set.

Archived

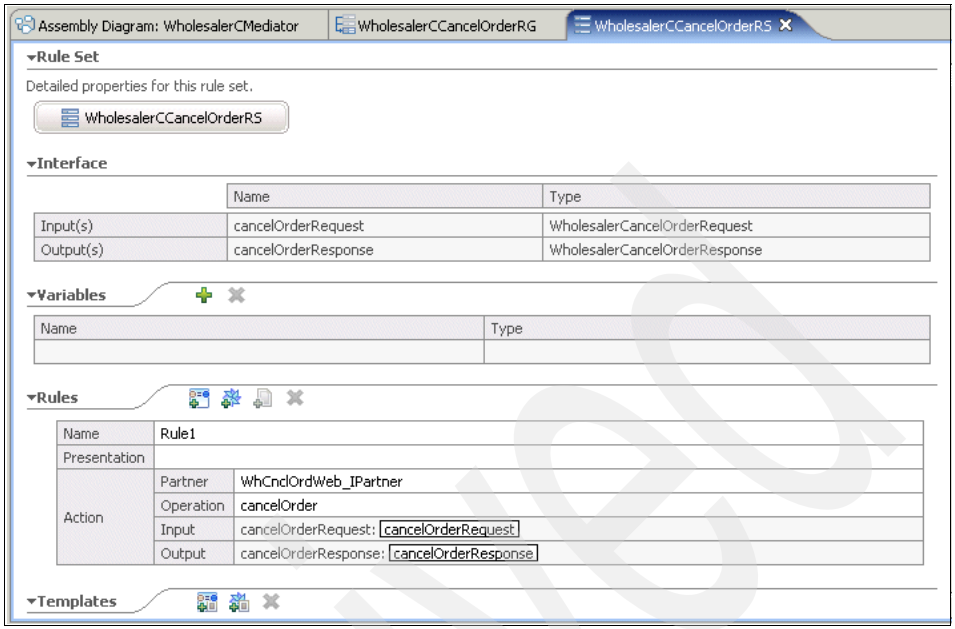


Figure D-9 The WholesalerCCancelOrderRS business rule set

g. Close the WholesalerCCancelOrderRS business rule set editor.

h. Switch to the WholesalerCCancelOrderRG business rule group editor.

   The completed WholesalerCCancelOrderRG business rule group should contain the Available Destination WholesalerCCancelOrderRS (Figure D-10).

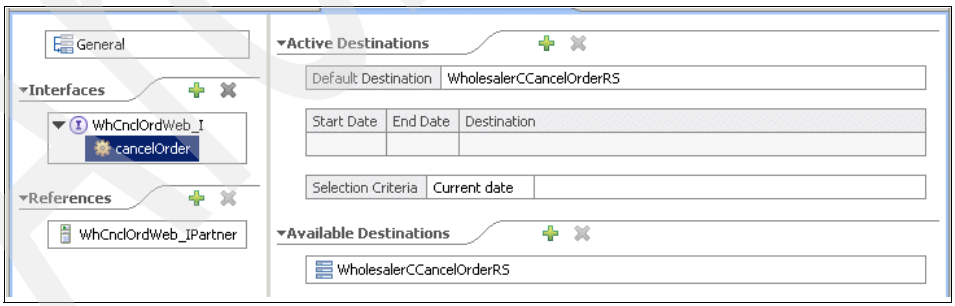


Figure D-10 The WholesalerCancelOrderRG business rule group

i. Press Ctrl+S to save the WholesalerCCancelOrderRG business rule group.

j. Close the WholesalerCCancelOrderRG business rule group editor.

2. Generate implementation for the WholesalerCGetDateRG business rule group with the properties from Table D-4.

Table D-4 WholesalerCGetDateRG Generate Implementation properties

Rule Set name	Partner	Operation	Input	Output
WholesalerCGet DateRS	WhDelReq Web_IPartner	getDeliveryDays	getDeliveryDays WholesalerRequest	getDeliveryDays WholesalerResponse

3. Generate implementation for the WholesalerCPlaceOrderRG business rule group with the properties from Table D-5.

Table D-5 WholesalerCPlaceOrderRG Generate Implementation properties

Rule Set name	Partner	Operation	Input	Output
WholesalerC PlaceOrderRS	WhPlaceOrdWeb_I Partner	placeOrder	placeOrderRequest	placeOrderResponse

4. Press Ctrl+S to save the assembly diagram.

Generating the SCA binding exports

To generate SCA binding exports from the WholesalerCMediator business rules.

1. Right-click the **WholesalerCCancelOrderRG** rule group and click **Export** → **SCA Binding**.
2. Right-click the **WholesalerCGetDateRG** rule group and click **Export** → **SCA Binding**.
3. Right-click the **WholesalerCPlaceOrderRG** rule group and click **Export** → **SCA Binding**.
4. Press Ctrl+S to save the WholesalerCMediator assembly diagram.

Figure D-11 shows the completed WholesalerCMediator assembly diagram.

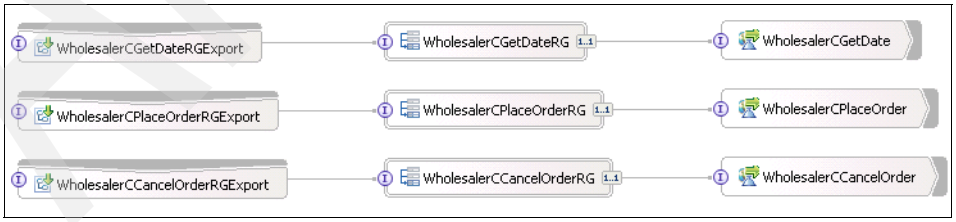


Figure D-11 The completed WholesalerCMediator assembly diagram

Replacing WholesalerA with WholesalerC at run time

You can replace Wholesaler A with Wholesaler C at run time without any changes to the Parallel Workflow process.

Adding the required modules to the test server

To add the required modules to the WebSphere Integration Developer test server:

1. In the Servers view, right-click the **WebSphere Process Server v6.0** server and click **Add and remove projects**.
2. Add the following modules to the WebSphere Integration Developer test server:
   - WholesalerAApp
   - WholesalerBApp
   - ITSOPModuleApp
   - WholesalerCMediatorApp
   - WholesalerCApp

   Click **Finish**.
3. Wait for the server to start. Look for the `Server server1 open for e-business` message in the console.

Changing the selector targets in the test server admin console

Now we replace WholesalerA with WholesalerC. We provide the steps needed in the WebSphere Process Server administrative console to change the targets of the selectors in the Parallel Workflow process.

1. After the server starts and the applications are installed, right-click the **WebSphere Process Server v6.0** server and click **Run administrative console**.
2. In the Admin Console logon window, enter any valid user ID and password click **Log in**.
3. Change the WholesalerGetDateSelector target from WholesalerA to WholesalerC:
   a. In the navigation pane, click **Servers** and click **Application servers**.
   b. Click **server1**.
   c. On the Configuration tab in the Business Integration section, click **Selectors**.

Archived

d. In the Selectors window (Figure D-12), click **WholesalerGetDateSelector**.

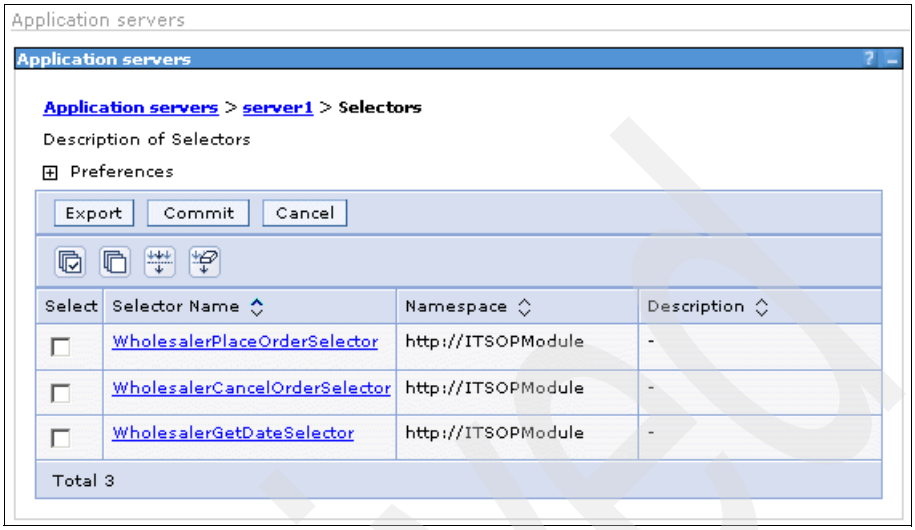


Figure D-12 The server1 Selectors window

e. Click **getDeliveryDays**.

f. In the Selector targets for operation window (Figure D-13), click **New**.

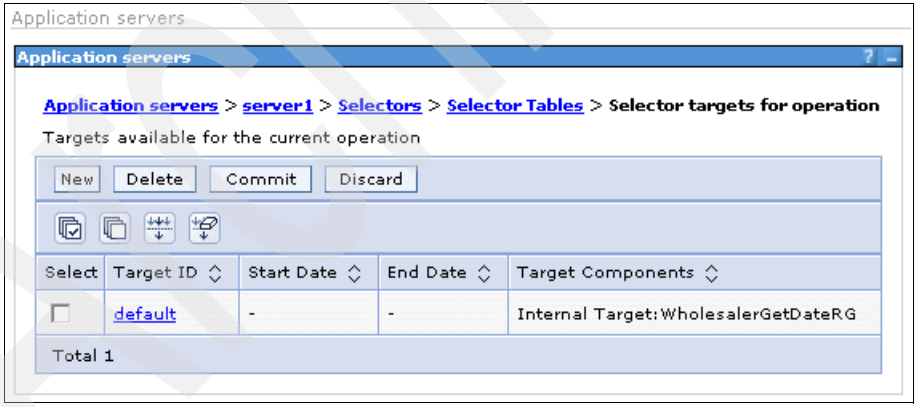


Figure D-13 The selector targets for the getDeliveryDays operation

Archived

g. In the Selector targets for operation → New window (Figure D-14), complete these steps:

   i. For Target Components, select **Exported Target: WholesalerCMediator WholesalerCGetDateRGExport**.

   ii. In the Start date field, enter the start timestamp, or use the default start timestamp which is the current timestamp. This tells the selector when to start invoking the selected target.

   iii. In the End date field, enter the end timestamp, or use the default end timestamp which is the current timestamp plus one year. This tells the selector when to stop invoking the selected target.

   iv. Click **OK**

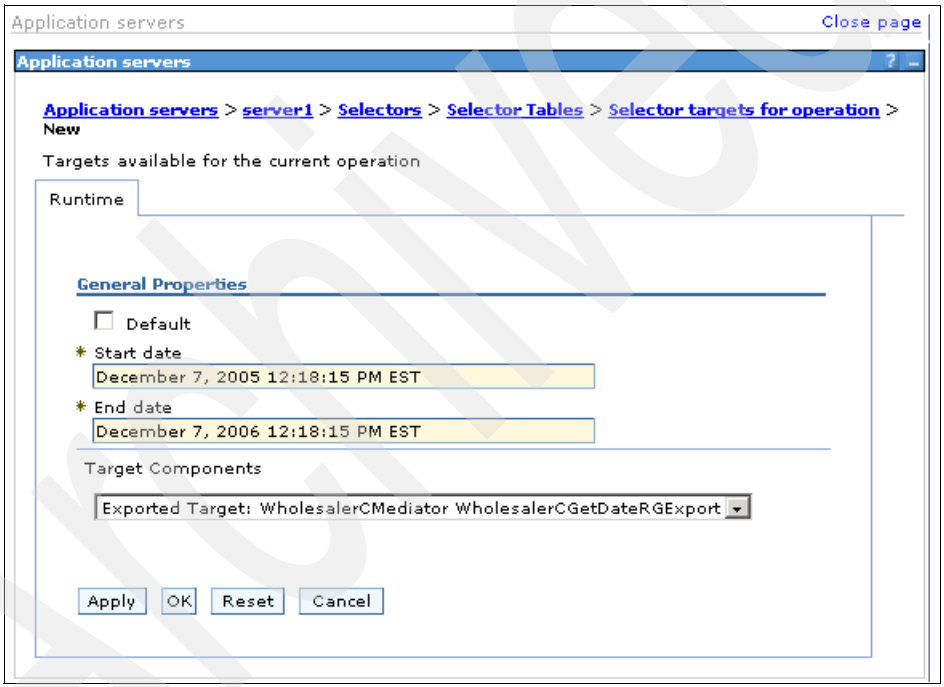


Figure D-14 The selector targets for the getDeliveryDays operation: New window

Archived

h. In the selector targets window (Figure D-15), click **Commit** to commit the changes.

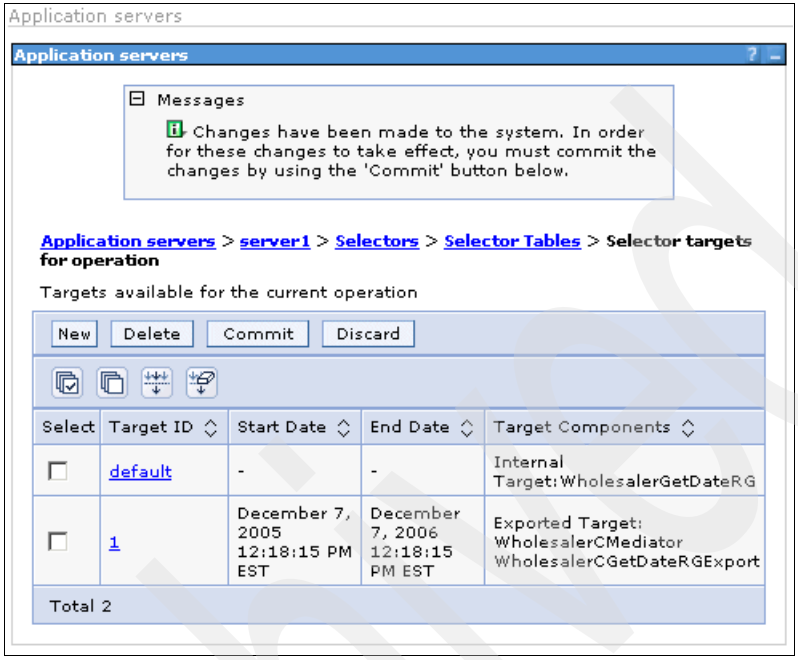


Figure D-15 The selector targets for the getDeliveryDays operation with the new target

4. Repeat the previous step for the WholesalerPlaceOrderSelector and WholesalerCancelOrderSelector selectors, to change the selector targets from WholesalerA to WholesalerC using the properties in Table D-6.

Table D-6 The selector targets for WholesalerC

Selector name	Operation	New selector target
WholesalerPlace OrderSelector	placeOrder	Exported Target: WholesalerCMediator WholesalerCPlaceOrderRGExport
WholesalerCancel OrderSelector	cancelOrder	Exported Target: WholesalerCMediator WholesalerCCancelOrderRGExport

Archived

Testing that WholesalerA has been replaced by WholesalerC

In this section, we verify that WholesalerA has been replaced by WholesalerC at run time without any changes to the Parallel Workflow process in the ITSOPModule. Start a process instance using the Integration Test Client as explained in 13.4.2, “Testing the Parallel Workflow process” on page 485.

> **Important:** Running process instances of the Parallel Workflow process also invokes WholesalerC instead of WholesalerA when the selector target changes are committed.

Figure D-16 shows how the Test Client Event page Event section looks after the process instance has finished.

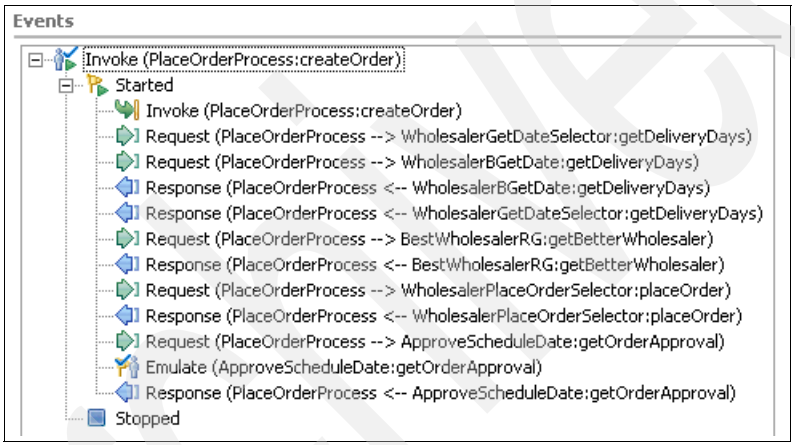


Figure D-16 Test Client Event page Event section

When the Wholesaler_A invoke activity is executed in the Parallel Workflow PlaceOrderProcess, we see the server log (Example D-4).

Example: D-4 The server log when the Wholesaler_A getDeliveryDays() invoke activity is executed

```
[12/7/05 14:53:27:785 EST] 0000006d SystemOut     O WholesalerCImpl.getDeliveryDays() entry
[12/7/05 14:53:28:045 EST] 0000006d SystemOut     O response set daysToDeliver
[12/7/05 14:53:28:045 EST] 0000006d SystemOut     O response set price
[12/7/05 14:53:28:045 EST] 0000006d SystemOut     O WholesalerCImpl.getDeliveryDays() exit
```

Exporting the selector changes to WebSphere Integration Developer

We must export the runtime selector changes to the WebSphere Integration Developer tooling to reflect the changes. This ensures that the selectors changes made at run time are not lost in future redeployments or upgrades.

Exporting the selectors from the admin console

To export the selectors from the runtime environment using the WebSphere Process Server administrative console:

1. In the navigation pane, click **Servers** and click **Application servers**.
2. Click **server1**.
3. On the **Configuration** tab in the Business Integration section, click **Selectors**.
4. In the Selectors window (Figure D-17), under the Select column, choose the selector names that changed:
   a. Click the **WholesalerCancelOrderSelector** check box.
   b. Click the **WholesalerPlaceOrderSelector** check box.
   c. Click the **WholesalerGetDateSelector** check box.
   d. Click **Export**.

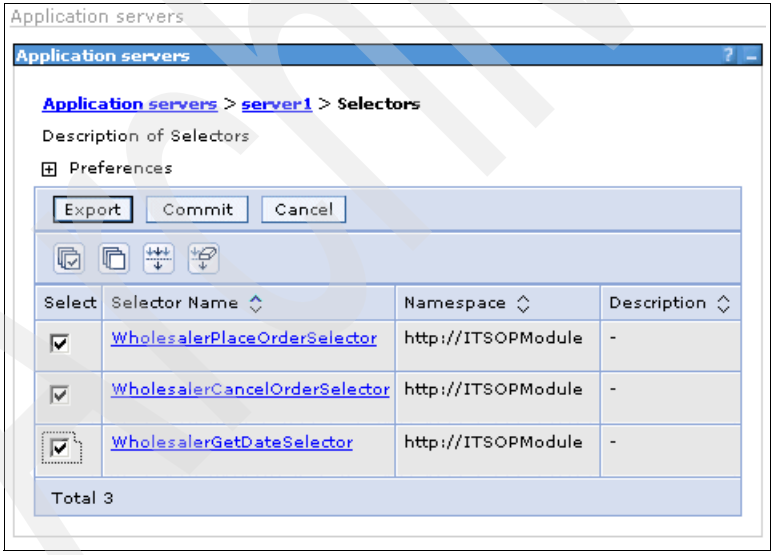


Figure D-17 The server1 Selectors window

5. In the Selectors to Export window (Figure D-18), click the **ITSOPModule_WholesalerCancelOrderSelector.zip** link to export the WholesalerCancelOrderSelector.

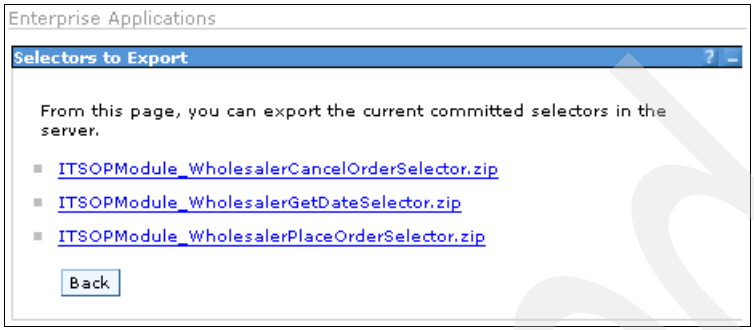


Figure D-18 The Selectors to Export window

6. In the File Download window (Figure D-19), click **Save**.

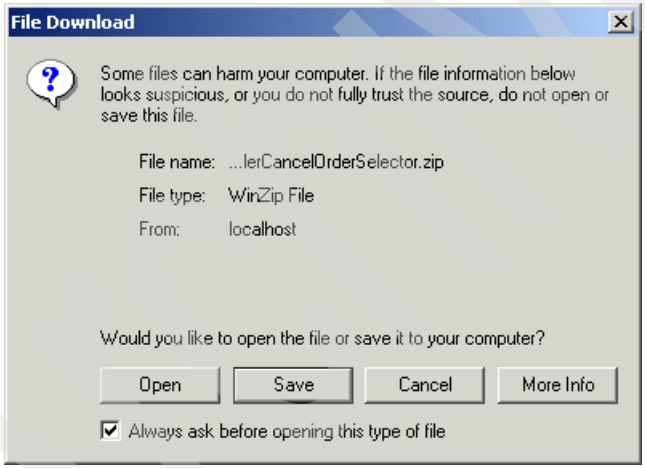


Figure D-19 The File Download window

a. In the Save As window, choose the directory to save the zip file to and click **Save**.

b. In the Download Complete window, click **Close**. The ITSOPModule_WholesalerCancelOrderSelector.zip file contains a new .sel and .selt file for the WholesalerCancelOrderSelector.

7. Repeat this process to download the ITSOPModule_WholesalerGetDateSelector.zip and ITSOPModule_WholesalerPlaceOrderSelector.zip files.
8. Right-click the **WebSphere Process Server v6.0** server and click **Stop**.
9. Wait for the server to stop.

> **Tip:** Make a backup of ITSOPModule before you import the changed selectors into to the module. For instance, you can export a Project Interchange file with the ITSOPModule.

Importing the selectors to the tooling

Import the WholesalerCancelOrderSelector selector to WebSphere Integration Developer tooling:

1. Right-click the **ITSOPModule** module and click **Import**.
2. In the Import wizard Select window, in the Select an import source list box, select **Zip file**. Then click **Next**.

Archived

3. In the Import wizard Zip file window (Figure D-20), complete these steps:
   a. Click **Browse**.
   b. In the Import from Zip file window, navigate to and select the **ITSOPModule_WholesalerCancelOrderSelector.zip** file. Click **Open**.
   c. In the Into folder field, type `ITSOPModule`.
   d. Click **Finish**.

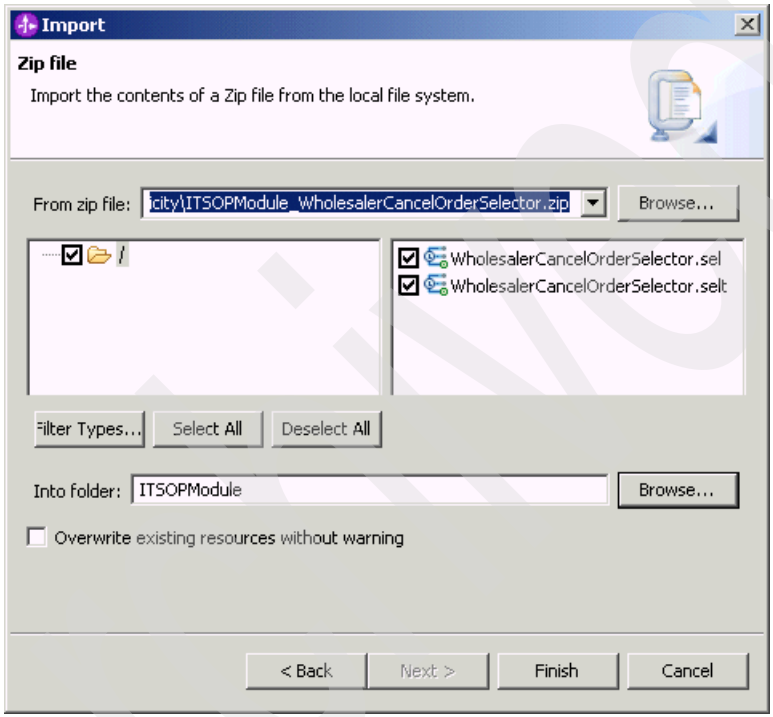


Figure D-20 The Import wizard: Zip file page

   e. In the Question window (Figure D-21), click **Yes To All**.

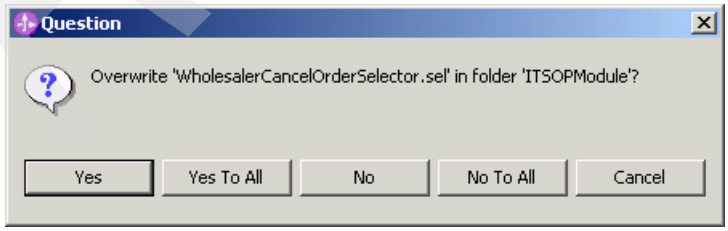


Figure D-21 The Import wizard: Zip file page Question window

4. To verify that the selector was imported successfully:
   a. Right-click the **WholesalerCancelOrderSelector** and click **Open**.
   b. When the WholesalerCancelOrderSelector editor (Figure D-22) opens, click **cancelOrder**.

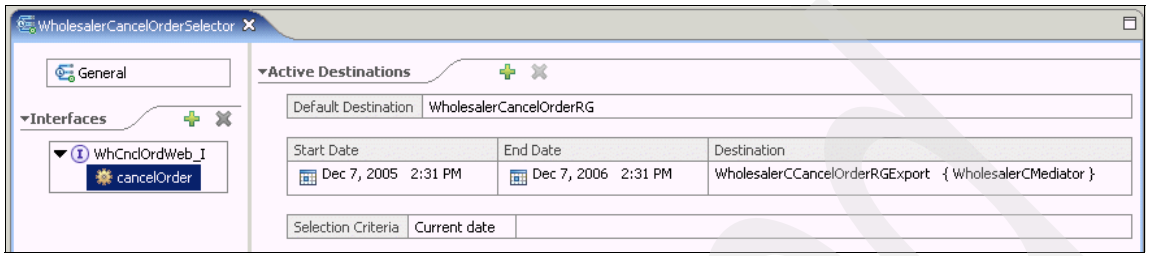


Figure D-22 The imported WholesalerCancelOrderSelector

5. Repeat these steps to import ITSOPModule_WholesalerGetDateSelector.zip and ITSOPModule_WholesalerPlaceOrderSelector.zip files.

Archived

E

Additional material

This IBM 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 IBM 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/SG247205

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 IBM Redbook form number, SG247205.

Using the Web material

The additional Web material that accompanies this book includes the following files:

File name	*Description*
SG247205.zip	Zipped code samples

© Copyright IBM Corp. 2006. All rights reserved.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 10 GB
Operating System: Windows 2000 or 2003
Memory: 2 GB

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

Abbreviations and acronyms

API	application programming interface
ASBO	Application Specific Business Object
BPC	Business Process Container
BR	Business Rules
BSC	Business Service Choreography
CBE	Common Base Event
CCI	Common Client Interface
DMZ	demilitarized zone
DTD	document type definition
EAI	enterprise application integration
EIS	enterprise information systems
EJB	Enterprise JavaBeans
EMD	Enterprise Metadata Discovery
ESB	enterprise service bus
ESD	Enterprise Service Discovery
GBO	Generic Business Object
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
J2C	J2EE Connector Architecture
J2EE	Java 2 Platform, Enterprise Edition
JCP	Java Community Process
JMS	Java Message Service
JSP™	JavaServer Pages
JVM	Java Virtual Machine
LPTA	Lightweight Third Party Authentication
QOS	Quality of Service
RDBMS	relational database management system
SCA	Service Component Architecture
SCDL	Service Component Definition Language
SDO	Service Data Objects
SOA	service-oriented architecture
SSL	Secure Sockets Layer
TCO	total cost of ownership
TLS	Transport Layer Security
UDDI	Universal Description, Discovery, Integration
VANs	Value-Added Networks
W3C	World Wide Web Consortium
WIP	Work In Progress
WSDL	Web Services Description Language
XSD	XML Schema Definition
XSLT	XML Transformations

Archived

© Copyright IBM Corp. 2006. All rights reserved.

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this IBM Redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 695. Note that some of the documents referenced here may be available in softcopy only.

- *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303
- *Patterns: Implementing an SOA using an Enterprise Service Bus*, SG24-6346
- *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6*, SG24-6494
- *Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture*, SG24-6773
- *Patterns: Extended Enterprise SOA and Web Services*, SG24-7135

Other publications

The following publication is also relevant as a further information source:

- Adams, Jonathan; Galambos, George; Koushik, Srinivas; Vasudeva, Guru, *Patterns for e-business: A Strategy for Reuse*, Mc Press, 2001, ISBN 1931182027

Online resources

These Web sites and URLs are also relevant as further information sources:

- Patterns for e-business Web site

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

- IBM WebSphere Process Server

  http://www.ibm.com/software/integration/wps

Archived

© Copyright IBM Corp. 2006. All rights reserved.

- IBM WebSphere Integration Developer

  http://www.ibm.com/software/integration/wid
- DB2 Universal Database for Linux, UNIX®, and Windows

  http://www.ibm.com/software/data/db2/udb
- IBM Cloudscape

  http://www.ibm.com/software/data/cloudscape
- IBM WebSphere Adapters

  http://www.ibm.com/software/integration/wbiadapters
- IBM WebSphere MQ

  http://www.ibm.com/software/ts/mqseries
- Web Services Business Process Execution Language (WS-BPEL) specification and resources

  http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
- Service Component Architecture specification

  http://www.ibm.com/developerworks/library/specification/ws-sca/
- Service Data Objects specification

  http://www.ibm.com/developerworks/webservices/library/specification/ws-sdo/
- SOAP specification

  http://www.w3.org/TR/soap12-part1/
- The role of private UDDI nodes in Web services, Part 1: Six species of UDDI

  http://www.ibm.com/developerworks/webservices/library/ws-rpu1.html
- The role of private UDDI nodes, Part 2: Private nodes and operator nodes

  http://www.ibm.com/developerworks/webservices/library/ws-rpu2.html
- Security in a Web Services World: A Proposed Architecture and Roadmap

  http://www-128.ibm.com/developerworks/library/specification/ws-secmap/
- Web Services Security: Moving up the stack

  http://www.ibm.com/developerworks/webservices/library/ws-secroad/
- WebSphere Application Server Information Center

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp

Archived

- WebSphere Process Server Information Center

  http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp

- IBM WebSphere Business Process Management Version 6.0 Information Center

  http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?

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

Archived

Index

Numerics
80/20 situation 9

A
accounting system 120, 338, 394
  transactional update 394
action 265, 270
activity name 418
actor 117
adapters 340
advanced and future Web services standards for Web services security
  WS-Policy 139
  WS-Privacy 139
  WS-Security 139
  WS-Trust 139
alias 278
API (application programming interface) 98, 132
App Server/Services node 78
Application Integration pattern 47–48
  business and IT drivers, process-focused 51
  data-focused integration 48
  process-focused 48
Application pattern 6, 10, 16, 48–49, 74, 119, 150, 240, 335, 396
  Exposed Serial Process 67
  Extended Enterprise 62
  generic and SOA Runtime patterns 75
  process-focused 49
  selection 16
application programming interface (API) 98, 132
application server 19–20, 102, 133, 153, 195, 314, 542
  HTTP server function 20
  system contracts 133
  work tasks 133
application specific business object 159
ApproveScheduleDate human task 411, 471, 498, 603
ASBO/GBO 157, 159, 631
assembly diagram 43–44, 164, 213, 251, 284, 302, 352, 354, 356, 418–419, 467, 546–547, 583, 623, 652, 655–656, 662
  BSMChooseWh rule group 302
  business process 308
  canvas 214, 299–300, 467, 559
  components 657
  current state 374
  interface maps 657
  resource adapter 623
  WholesalerB 501
Assembly Diagram Editor 665
Assembly Editor 42, 158–159, 248, 348, 547, 572
  assembly diagram 547
  unimplemented components 579
assign activity 35
assignment activity 37
asynchronous invocation 154
attach (to a module) 585
auditing events 339

B
back-end application
  integration 52, 119
  tier 16
best practice guidelines 10
best practices 22
BestWholesaler business rule 350, 405, 407, 411, 419
binding/service window 370
  WSDL file 370
boolean-valued expression 36
bottom-up design 158
BPC Explorer 494, 586
BPEL4WS (Business Process Execution Language for Web Services) 33
business and IT drivers
  Application Integration pattern, Process-focused 51
  Exposed Serial Process application pattern 67
  Extended Enterprise application pattern 65
  Parallel Process application pattern 56
  Parallel Workflow variation 60
  Serial Process application pattern 53
  Serial Workflow variation 58, 70
business graph 158, 384

Archived

© Copyright IBM Corp. 2006. All rights reserved.

business objects 158
business logic 26, 74, 158–159, 248, 350, 502, 663
Java class 562
WS-BPEL part 182
business object 7, 94, 145, 156, 173, 193, 256, 287, 341, 406, 501, 506, 580, 628, 664
given property 580
one-to-one, one-to-many or many-to-many relationship 635
special type 158
temporary instance 650
Business Object Editor 174, 176, 509
Business pattern 10, 13, 47, 116, 149, 334, 395
types for e-business 47
business problem 4, 9, 12, 39, 145, 235, 331, 393
business process 5, 13, 25, 38, 47, 76–77, 94–95, 104, 114, 139, 145, 148, 239, 243, 251, 288, 332–333, 396, 504, 571, 597, 627, 661
different kinds 147
different steps 26
first stage 358
formal specification 139
inbound interfaces 182
interface partner 183
ITSO_CheckDate business rule 251
JDBC adapter 375
supporting services 215
true parallelism 359
type 148
wholesalers directory 159
Business Process Execution Language for Web Services (BPEL4WS) 33
business process management 26
business process reengineering 26
business process template 243
business requirement 117, 147, 238, 334, 395
business rule 7–8, 26, 95, 117, 145, 194, 251, 353, 400, 501–502, 542, 571, 662
Business Rules Manager 590
business scenario 6, 115, 235, 331, 393, 590
intra-enterprise integration 115
business service 39, 53, 78, 159–160, 340
packaged applications 54
sequential execution 53
universal model 39
Business Service Choreography node 77
Business Service Directory node 78
business state machine 7, 95, 127, 235, 324, 326, 328, 403
business-level event 339
Business-to-Business (B2B) pattern 239

C

Call Connection variation 50, 63
product mapping 103–105, 152, 337, 398
cancelOrder operation 403, 543, 676
catch activity 38
CBE (Common Base Event) 95, 571
Event Browser 596
CBE Event Browser 572, 596
CEI (Common Event Infrastructure) 340
CEI event 362, 597
choice activity 191, 438, 537
choice case 187, 290
Common Base Event (CBE) 95, 571
Event Browser 596
Common Event Infrastructure (CEI) 340
compensation 344, 401
actions 402
adverse effects 402
for multiple paths and conditional processes 403
compensation handler 37–38, 445
other activities 446
complexity 346
Composite patterns 10, 15
confirmation code 362, 419, 521
connector node 76
correlation 37, 278
correlation set 37
cost reduction 399
createOrder operation 349, 411
cross-reference data 627
cryptographic protocols 246
custom design 14
customer record 630–631
new identifier 631
cycle time 51, 123, 332, 345

D

data graph 132
data handling 36
data integration 14
data mapping cross-reference 652
data mediator service 132
data type 76, 156, 172, 287, 358, 413, 504, 635
data-focused application integration 48

Archived

data-focused integration 48
datapool 584
DB2 Universal Database 96, 104, 153, 336, 388, 619
  multi-user version 96
  V8.2 96
deadline-valued expression 36
deadlocking 345
default namespace 173, 407
delivery date 118, 146, 237, 333
delivery day 171, 636
dependencies 345
design guideline 7, 22, 145, 235, 331, 393
design option 153, 243, 337, 399
development guidelines 348, 404
Direct Connection application pattern
  Call Connection variation 50, 63
  Message Connection variation 50, 63
Directory and Security services node 76
Display name field 191, 261, 351, 420, 422, 538, 665, 669
document type definition (DTD) 136
Domain Firewall node 79
DTD (document type definition) 136
dynamicity
  business rule groups 468
  WebSphere Process Server 8, 340, 661

E

EAI (enterprise application integration) 47
EIS (Enterprise Information System) 39, 95, 132
EMD (Enterprise Metadata Discovery) 97
empty activity 35
emulator 581
endpoint reference 34
enterprise application integration (EAI) 47
  technology 28
Enterprise Demilitarized Zone 88
Enterprise Information System (EIS) 39, 95, 132
Enterprise Metadata Discovery (EMD) 97
Enterprise Secure Zone 88
enterprise service bus (ESB) 4, 78, 151, 241, 336, 397
  Exposed ESB Gateway 77
  SOA concept 4
Enterprise Service Discovery (ESD) 377
ESB (enterprise service bus) 4, 78, 151, 241, 336, 397
  Exposed ESB Gateway 77
  SOA concept 4
ESB node 77
escalation 400
ESD (Enterprise Service Discovery) 377
event auditing 339
event handling 403
event monitoring 339
Exposed ESB Gateway 77
Exposed Parallel Workflow variation 71
Exposed Process Manager node 80
Exposed Serial Process
  application pattern 67, 241
    business and IT drivers 67
    Workflow variation 69
  pattern 235
  pattern, key nodes 106
  product mapping, WebSphere Application Server Network Deployment 107
Extended Enterprise
  application pattern 62, 64
    business and IT drivers 65
  business pattern 6, 15, 47–48, 239
  domain 64
  pattern 5, 47, 62
  product mappings 106
  runtime pattern 87
  scenario 69, 240
Extensible Markup Language (XML) 33
external wholesaler 112, 124, 236

F

Failed Event Manager 599
fault handler 35, 37–38, 130, 400
fault handling 35, 393, 400, 488
file transfer application 618
finite state machine 247
firewall 76
flow activity 36

G

generated Java class, Web services interfaces 565
generating implementation 474, 477
generic business object 157, 631
generic profile 75, 80, 82, 84, 86, 89–90, 151, 241, 336, 397
get delivery date use case 121, 128
getDeliveryDays operation 172, 371, 428, 535,

651, 680
selector targets 680
getDeliveryDays Web services 418
guidelines
Application patterns, Runtime patterns, product mappings 22
development 348, 404

H

HTTP 142
HTTP authentication 139
HTTP server 20, 78, 142, 243, 313–314, 316
secured connection 314
HTTPS 142
human interaction 8, 47, 78, 105, 119, 148, 240, 335, 395, 399
straight-through processing 55
human task 95, 130, 148, 393, 524, 586
inline 160
potential future replacement 161
reader role 481
SCA 160

I

IBM Cloudscape 96, 153
IBM HTTP Server 313
keystore creation 314
SSL pass-through 312
implicit compensation 38
industry support 33
inline human task 160
Integration Developer
V6.0 99
Integration pattern 10, 14, 48, 62, 116
Integration Test Client 158, 319, 321, 366, 484–485, 552, 571, 658, 683
detailed description 484
input validation 579
multiple instances 584
Inter-enterprise Zone 78, 88, 242
network infrastructure 78
interface 512
interface creation 179
interface map 100, 145, 284, 354, 512, 645
assembly diagram 226
intermediate result 54, 76
intra-enterprise integration 115
invoke action 270
invoke activity 34
ITSO Electronics 6, 48, 111, 145, 235, 331, 393, 502, 632
background information 112
direct calls 239
existing environment 113
intra-enterprise integration 115
non-functional requirements 114
order clerk 333
preferred supplier 146
response message 507
retail crew 120
ITSOElectronics assembly diagram 219, 284, 655
interface map 655
Web services 301
wholesaler Web services 298
ITSOElectronics interface
ITSODelDays_I 645
ITSOPlaceOrd_I 652
ITSOElectronics module 162, 253, 634
assembly diagram 225
business object map
ITSODelDays_to_Order 221
ITSOOrder_to_DelDays 220
ITSOOrder_to_PlaceOrder 221
ITSOPlaceOrder_to_Order 222
interface map 219
interface map ITSO_to_WhA_Del 222
interface map ITSO_to_WhA_Order 223
interface map ITSO_to_WhB_Del 224
interface map ITSO_to_WhB_Order 225
interface map ITSOProc_to_Notify_OrdAck 284
interfaces 179
ITSOElectronics Assembly Diagram 284
new business object 177
new business object map 646
rule group ITSO_CheckDate 310
state machine artefact 255
Web service reference 298
WholesalerA Web service 308
WSDL file 301
ITSOLibrary 162, 406, 663
ITSOPModule 348, 406, 661
ITSOPModule assembly diagram 370–371, 419, 426
business rule 431
human task 419

Archived

J
J2EE 140
  Web services for 141
J2EE Connector Architecture (JCA) 133
Java API for XML-based RPC (JAX-RPC) 141
Java class 40, 502, 666
  preferred form 40
Java Community Process (JCP) 140
Java Message Service (JMS) 138, 141
Java snippet 145, 418, 524, 582
Java Virtual Machine (JVM) 97, 140
  consumer 340
JAX-RPC (Java API for XML-based RPC) 141
JCA (J2EE Connector Architecture) 133
JCP (Java Community Process) 140
JMS (Java Message Service) 138, 141
JVM (Java Virtual Machine) 97, 140

L
Lightweight Third Party Authentication (LTPA) 609
link activity 36
long-running process 154, 331, 333, 342
long-running transaction 58
LTPA (Lightweight Third Party Authentication) 609

M
manual emulation 582
Message Connection variation 50, 63
modeling a process 347
module 160
monitor 583
monitoring events 339
multiple application 14, 47, 76, 149, 245, 334, 395
  seamless execution 47

N
namespace 508, 511, 649, 664
Network Infrastructure node 79
node types 75
non-functional requirements 114

O
operation 268
Operation field
  findAcceptableWholesaler 268
  select requestOrder 271
  wholesalerCanFulfill 268
order request 172, 253, 332, 399
Order.OrderPart.part No 292
OrderAck module 162, 217, 251
  OrderAck assembly diagram 217
OrderEntry 351, 359
OrderFromA 419
orderId 278, 349, 414
outbound communication 134

P
Parallel Activities icon 427
Parallel Process
  exposed version 240
  human interaction 337
  module 348
  non-interruptible process 343
  product mapping 105
  product mappings, WebSphere MQ Workflow 105
  rule 61
  rules tier 56
  runtime pattern 84
  scenario 8, 119, 335, 394, 401
  valid step 401
Parallel Process application pattern 4, 50, 56, 59, 331
  as an alternative 55
  business and IT drivers 56
  drivers 116
  Parallel Workflow variation 60
  pattern selection 123
  product mapping 104
  Runtime pattern 73–74
  SOA profile 85
  Workflow variation, business and IT drivers 60
Parallel Workflow
  rules tier 61
  variation 50, 59–60
parallelism 343
partner link types 34
Partner Zone 88
partners 34
Password field 487, 607
pattern selection 395
Patterns for e-business
  Application patterns 10, 16
  best practices 10, 22
  Business patterns 10, 13

Archived

Composite patterns 10, 15
guidelines 10, 22
Integration patterns 10, 14
product mapping 10, 21
Runtime patterns 10, 18
Web site 11
performance 344
pick activity 36
Place Order activity 306
place order cross-reference data mapping 652
Place Order Process Scope
activity 421
Event Handler 421
fault handler 455
Properties page 420
place order service 171
Place Order use case 117
PlaceOrderProcess component 351–352, 482, 488
additional references 351
BestWholesalerPartner reference 356
generated business process 357
PlaceOrderResponse business object, new instance 529
process 148
Process Editor palette 191, 420
Choice icon 423
Parallel Activities icon 423
Rethrow icon 420
Scope icon 427
Sequence icon 422
Snippet icon 424
process flow 51, 76, 105, 113, 394
execution engine 80
human interactions 398
logic 53
process integration 14
process logic 52, 77, 119, 298, 340
significant amounts 54
Process Manager
asynchronous invocation 154
synchronous invocation 154
process manager database 148
Process Manager node 76, 102
process modeling and simulation 347
process-focused Application pattern 49
process-focused integration 48
product mapping 10, 21, 102, 149, 336, 398
Call Connection variation 103–105, 152, 337, 398
Protocol Firewall node 79

Q

QoS (Quality of Service) 472
Quality of Service (QOS) 472

R

receive activity 34
Redbooks Web site 695
Contact us xvi
reference partner 182, 288, 386, 405
relationship 627
relationship instance 631
Relationship Manager 571, 602, 631
relationship service 627
reply activity 34
Repository node 76
request/one response 17
request-response operation 350
resource adapter 133, 377, 619
transactional functionality 133
resource loading 346
response object 349, 589, 667–668
orderCancelled attribute 667
price attribute 667
retail department 112, 127, 146
business processes 116
retail system 117, 148, 238
Rethrow icon 461
rule 194
Rule Group
Editor 196, 200, 282–283, 372, 543
icon 351, 468, 674
window 474, 676
rule group 195, 280, 282, 353, 435, 468, 542, 591, 593, 674
Rule Set 283, 593
rules directory 79
runtime guideline 4, 7, 384, 388, 484
Runtime pattern 6, 10, 18, 73, 101
Application pattern 19, 74
Exposed Serial Process application pattern 241
Parallel Process application pattern 336
Parallel Workflow runtime pattern 397
product mapping 6
proven and tested software implementations 10
Serial Process application pattern 151
SOA profile category 75

Archived

typical and proven implementations 6

S

SAML (Security Assertion Markup Language) 245
SCA (Service Component Architecture) 6, 25, 39, 132, 244
component 40, 165, 171, 246, 506, 579
component main terms 40
destination 371, 477
human task 160
Process Manager 102
WebSphere Process Server V6 94
SCDL (Service Component Definition Language) 45
schedule date 408, 419
schema 136
scope element 37
Scope icon 420
scripted emulation 582
SDO (Service Data Objects) 132
Secure Socket Layer (SSL) 142, 246
Security Assertion Markup Language (SAML) 245
security triad of WS-Security 244
selector-business rule pattern 412
sequence activity 35
Sequence icon 446
Serial Process 83, 145, 332, 633
application 633
application pattern 50, 56, 64
generic profile Runtime pattern 82
implementation 248
multiple instances 337
pattern 4, 55, 73, 85, 89, 106, 145, 343
product mapping 102, 106, 242, 336
rule 53
rules tier 53, 68
runtime pattern 82, 152
connector node 76
Directory and Security Services node 76
Repository node 76
scenario 7, 162, 235
scenario implementation 127
SOA profile runtime pattern 83
workflow variation 58
Serial Process application pattern 50, 53, 56, 80, 151
business and IT drivers 53, 56
Workflow variation 55
business and IT drivers 58, 70
Serial Process scenario 239
Serial Process Workflow variation Runtime pattern
Staff Worklist Adapter node 78
Workflow Manager node 78
Serial Workflow pattern 69
Serial Workflow rules tier 59, 70
Serial Workflow variation 50, 73, 82
SerialProcess component 311, 657
WhADeliveryPartner reference 657
WhAOrderPartner reference 657
server log 488, 490, 666
Server tab 359–360, 429
service 28–29
Service Component Architecture (SCA) 6, 25, 39, 132, 244
component 40, 165, 171, 246, 506, 579
destination 371, 477
human task 160
Process Manager 102
WebSphere Process Server V6 94
Service Component Definition Language (SCDL) 45
service consumer 30, 77, 90, 135, 346, 549, 551
Service Data Objects (SDO) 132
service implementation 8, 30, 501, 661
service provider 30, 77, 135, 172, 246, 343, 632
many complex interactions 343
new location 135
service reference 41
service-oriented architecture (SOA) 3, 25, 28, 54, 75, 94, 132
brief overview 25
component-based design 28
customized 31
drivers 28
flexible pricing 28
increasing speed 28
Object Oriented development 28
reducing costs 28
return on investment 28
simplifying integration 28
proprietary 31
service
deployment time 30
implementation-independent 29
loosely bound 29
reusable 29
runtime 29

substitution 30
ServletWrapper 488
short-running process 148, 342
Simple Object XML 136
simulation, process 347
Snippet icon 422, 527
SOA (service-oriented architecture) 3, 25, 28, 54, 75, 94, 132
brief overview 25
component-based design 28
customized 31
drivers 28
flexible pricing 28
increasing speed 28
Object Oriented development 28
reducing costs 28
return on investment 28
simplifying integration 28
proprietary 31
service
deployment time 30
implementation-independent 29
loosely bound 29
reusable 29
runtime 29
substitution 30
SOA profile 75, 81, 83, 85, 87, 90–91, 151, 241, 336, 397
SOAP 137
envelope 137
faults 137
over HTTP (SOAP/HTTP) 138
SOAP message 137–138, 245, 551
Encrypts specific parts 318
headers section 137
sensitive parts 245
source application tier 53, 56, 59, 61, 68, 70
SSL (Secure Socket Layer) 142, 246
SSL authentication 139
SSL configuration 316
SSL encryption 139
Staff Worklist Adapter node 78
state machine 235, 238, 251
correct instance 278
submap 205
switch activity 36
SYNC_RECEIVE 134
SYNC_SEND 134
SYNC_SEND_RECEIVE 134
synchronous invocation 154

T

target application 50, 53, 76, 102, 119, 152, 240, 630
execution rules 53
target application tier 54, 57, 59, 61, 68, 70
TCO (total cost of ownership) 52, 119
terminate activity 35
test client 158, 311, 319, 366, 484, 571, 652
actual behavior 575
various events 323
test server 227, 478, 484, 603, 679
admin 679
testing module 8, 571
throughput 345
throw activity 35
TLS (Transport Layer Security) 142, 246
toolbar 527
top-down design 158
total cost of ownership (TCO) 52, 119
transactions 338
transition 247
Transport Layer Security (TLS) 142, 246
transport protocol 142

U

U.S. government taxonomy 138
UDDI (Universal Description, Discovery, Integration) 138
Universal Description, Discovery, Integration (UDDI) 138
update inventory use case 118
use case
get delivery date 121, 128
update inventory 118
user ID 76, 479, 603
Verb field select users 479

V

variables 37
visual snippet 7, 145, 161, 270, 361, 364, 418, 528, 582, 647
action 270
box select Vector 207
generated code 538
Visual Snippet Editor 191, 273, 363, 440, 529, 626,

Archived

648
canvas 530–533

W

wait activity 35
WC_defaulthost_secure 315
Web service 23, 31, 78, 95, 102, 131, 134, 152, 236, 251, 341, 346, 405, 412, 501, 599, 661
deployment model 141
different implementations 33
existing uses 31
logical links 31
main aims 135
popular protocol 246
private UDDI nodes 138
runtime models 141
select import 300, 673
transport level security 246
Web Service Business Process Execution Language (WS-BPEL) 95, 139, 145
Web service client
binding 317
ITSOElectronics 318
Web service port 673
Web Service Ports folder 299
Web services
interoperability 141
loose coupling 135
SOAP
body 137
header 137
UDDI 138
usage models 135
basic callback 135
one-way 135
synchronous request/response 135
world 140
Web Services Description Language (WSDL) 31, 99, 134, 137
Web Services Explorer 571
Web Services Interoperability Organization (WS-I) 134
Web site 11, 21–22, 95–100, 136, 689
WebSphere Adapter 8, 97, 160, 331, 348, 603
WebSphere Integration Developer 4, 99, 152, 242, 339, 406, 501, 571, 631, 679
application development steps 7
appropriate editor 319
Business Integration perspective 558
Business Rules Manager 591
installation 379
Integration Test Client 158
JDBC resource adapter 377
new workspace 663
other features 8
previous experience 7
relationship editor 631
simple business process solution 7
test client 179
test environment 389
test server 484–485, 604
tooling 684
V6.0.1 162, 249, 348, 506
various components 624
Web browser 586
workspace 161, 251, 348, 406, 577
WebSphere MQ
queue manager 98
transport layer 98
WebSphere Process Server 4, 8, 25, 42, 61, 94, 102, 120, 145, 235, 331, 393, 504, 571, 628, 679
Architectural model 94
bi-directional communication 340
Business Integration view 43
business objects 341
business process execution 347
business processes 7
capabilities 650
compensation facilities 401
dynamicity 8, 340, 661
feature 8
functionality 130
fundamental technology 39
implementation 247
Information Center 243, 388
module 299
parallel process 337
product 102
product mapping 103, 152, 242, 336, 398
real environment 165
relationship 627
relationship service 628
run time 160, 389
secure services 246
security 243
selector functionality 340
service components 94

Archived

service integration bus 243
SSL port 316
support site 579
Test Environment 229
V6 6
V6.0.1 242
Web service capabilities 504
WhCnclOrdWeb_I interface 469, 665
while activity 36
wholesale system 115, 148
Wholesaler B 120, 146, 236, 332, 396, 501, 660, 663
service implementation 556
Wholesaler C 661
application 661
business logic 671
mediator 662
mediator module 671
module 663
service 664
service implementation 661
Web service 663
Wholesaler D 280, 285, 324
artifact 287
business logic 287
module 293
Wholesaler E 280, 285, 326
business logic 287
re-using 287
Wholesaler D artifacts 287
Wholesaler F 632–633, 648
component 658
delivery days 660
equivalent 656
interface 645
interface WhDelReqWeb_I 645
interface WhPlaceOrdWeb_I 652
item number 648
module 656
part number 645
representation 632
role instance 644
service 657
Wholesaler F components 657
Wholesaler F interface
WhDelReqWeb_I 645
WhPlaceOrdWeb_I 652
WholesalerA 163, 252, 406, 501, 662
WholesalerA module 163, 252, 348, 466, 506, 664
business object 664
interface 664
Web service port 466
WholesalerB module 163, 348, 369, 406, 467, 558
Business Logic folder 564
Interfaces folder 558
Web service port 467
Web service port definitions 369
WholesalerC Java component 665
Web service exports 669
WholesalerC assembly diagram 666
WholesalerC module 664
Assembly Diagram Editor 665
Web service export 672
Web service port 673
WholesalerCMediator assembly diagram 662, 667, 673
WholesalerCMediator module 662
WholesalerD 253
WholesalerD module 287
data types 294
other modules 287
Web service port 300
Web Service Port folder 301
Wholesalers D 251
service port files 301
wired business process assembly diagram 308
workflow 148
Workflow Manager 14, 82, 102, 398
Workflow Manager node 78
Workflow variation 55, 57, 69, 73, 105, 119
product mappings 105
work-in-progress (WIP) 54
workspace setup 348
WS-BPEL
activities 34
assignment 37
basic activities 34
assign 35
empty 35
invoke 34
receive 34
reply 34
terminate 35
throw 35
wait 35
business process 8, 162–163, 305, 521, 658
correlation 37
data handling 36

Archived

expression 36
boolean-valued expression 36
deadline-valued expression 36
endpoint references 34
partner link types 34
partners 34
scope 37
compensation handlers 38
data handling 38
event handlers 39
fault handlers 38
structured activities 35
flow 36
links 36
pick activity 36
sequence activity 35
switch activity 36
while activity 36
variables 37
WS-BPEL (Web Service Business Process Execution Language) 95, 139, 145
WS-BPEL process 7, 33, 164, 182, 248–249, 357, 501
reference partners 182
step-by-step development 7
WSDL (Web Services Description Language) 31, 99, 134, 137
WSDL file 102, 172, 298, 370, 504, 569, 671
import 551
state 551
WholesalerA_PlaceOrderExport_WhPlaceOrdWeb_IHttp_Service 551
WS-I (Web Services Interoperability Organization) 134
WS-Policy 139
WS-Privacy 139
WS-Security 139
security triad 244
WS-Trust 139

X

XML (Extensible Markup Language) 33
XML data
source 132
type 36
XML Schema
definition 99
type DateTime 36
type duration 36
XML Schema Definition (XSD) 136
XML Signature 245
XSD (XML Schema Definition) 136

Archived

Archived

Redbooks

Patterns: Building Serial and Parallel Processes for IBM WebSphere Process Server V6

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

Archived

Archived

Archived

Patterns: Building Serial and Parallel Processes for IBM WebSphere Process Server V6

Develop business process solutions using IBM WebSphere Integration Developer V6

Follow design, development, and runtime guidelines

Learn by going through practical scenarios

IBM WebSphere Process Server, and the accompanying IBM WebSphere Integration Developer development tool, provide a vast array of powerful features for building business process solutions. But how can you use those features for your organization? This IBM Redbook takes a scenario-based approach to demonstrate these product features and employs the IBM Patterns for e-business to illustrate proven business process patterns.

Part 1 of this redbook introduces the Patterns for e-business. The Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying On Demand Business applications. This redbook focuses on the Serial and Parallel Process patterns from the asset catalog.

Part 2 describes the business scenario used in this book. It also explains the key technologies that are relevant to the scenarios.

Part 3 guides you through the building of business process solutions in WebSphere Integration Developer for deployment to WebSphere Process Server. It includes the use of Web Services Business Process Execution Language (WS-BPEL) business processes, business state machines, and other product features. Four solutions are described, each adhering to a separate pattern. Each solution includes architectural considerations for designing the solution and step-by-step instructions for building the scenario.

SG24-7205-00 ISBN 0738497320

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