Patterns: Serial and Parallel Processes for Process Choreography and Workflow

Use the Patterns for e-business to integrate business processes

WebSphere Application Server
Enterprise Process Choreographer

WebSphere MQ Workflow

Martin Keen
JinYoung Byun
Mark Grottoli
Li Hui
Ajay Mathur
Robert Skaer
Guru Vasudeva
Gerd Watmann
Peter Xu

ibm.com/redbooks
Patterns: Serial and Parallel Processes for Process Choreography and Workflow

April 2004
Note: Before using this information and the product it supports, read the information in “Notices” on page ix.
Contents

Notices ................................................................. ix
Trademarks .............................................................. x

Preface ................................................................... xi
The team that wrote this redbook........................................... xi
Become a published author ................................................. xiv
Comments welcome ......................................................... xv

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

Chapter 1. Introduction to Patterns for e-business ................. 3
  1.1 The Patterns for e-business layered asset model ............... 4
  1.2 How to use the Patterns for e-business ......................... 6
    1.2.1 Select Business, Integration, or Composite pattern or Custom design 6
    1.2.2 Selecting Application patterns ........................................ 11
    1.2.3 Review Runtime patterns ............................................. 12
    1.2.4 Review Product mappings ........................................... 15
    1.2.5 Review guidelines and related links .............................. 16
  1.3 Summary .................................................................. 16

Chapter 2. Fundamental concepts in Process Integration .......... 17
  2.1 The need for a unifying technique ................................. 18
  2.1.1 Similarities between intra- and inter-enterprise integration 18
  2.1.2 Summary ................................................................. 19
  2.2 Process Integration concepts and notations ..................... 19
    2.2.1 Collaboration and Interaction .................................... 19
    2.2.2 Connectors and Adapters ......................................... 21
    2.2.3 Classification of interaction between sub-systems .......... 24
  2.3 QoS capabilities framework ......................................... 26
    2.3.1 Operability .............................................................. 26
    2.3.2 Availability ............................................................. 26
    2.3.3 Federation ............................................................. 27
    2.3.4 Performance ........................................................... 27
    2.3.5 Security ................................................................. 27
    2.3.6 Standards compliance ............................................. 28
    2.3.7 Transactionality ....................................................... 28
  2.4 Application patterns for Application Integration ............... 28
  2.5 Application patterns for Extended Enterprise .................. 29
  2.6 Summary .................................................................. 30
### Chapter 11. Creating processes with human interaction  
11.1 Business scenario  268  
11.2 Business process model  268  
11.3 General design guidelines  270  
11.3.1 Design overview  270  
11.3.2 Design considerations  270  
11.3.3 An alternative solution  273  
11.4 WebSphere Process Choreographer guidelines  276  
11.4.1 Design guidelines  276  
11.4.2 Development guidelines  284  
11.4.3 Runtime guidelines  287  
11.5 WebSphere MQ Workflow guidelines  292  
11.5.1 Design guidelines  292  
11.5.2 Development guidelines  296  
11.5.3 Runtime guidelines  310

### Chapter 12. Creating processes with events and compensation  
12.1 Business scenario  316  
12.2 Business process model  316  
12.3 General design guidelines  318  
12.3.1 Design overview  318  
12.3.2 Design considerations  319  
12.4 WebSphere Process Choreographer guidelines  321  
12.4.1 Design guidelines  322  
12.4.2 Development guidelines  328  
12.4.3 Runtime guidelines  336  
12.5 WebSphere MQ Workflow guidelines  340  
12.5.1 Design guidelines  340  
12.5.2 Development guidelines  344  
12.5.3 Runtime guidelines  351

### Chapter 13. Process manager interoperability  
13.1 Business scenario  358  
13.2 Business process model  358  
13.3 General design guidelines  360  
13.3.1 Design overview  360  
13.3.2 Design considerations  360  
13.4 WebSphere MQ Workflow invoking Process Choreographer  362  
13.4.1 Design guidelines  362  
13.4.2 Development guidelines  367  
13.4.3 Runtime guidelines  378  
13.5 Process Choreographer invoking WebSphere MQ Workflow  383  
13.5.1 Design guidelines  383
Notices

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

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

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

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

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

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

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

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

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

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

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

- AIX®
- CICS®
- ClearCase®
- DB2®
- developerWorks®
- Domino®
- e(logo)server®
- e-business on demand™
- FlowMark®
- Holosofx®
- IBM®
- iSeries™
- Lotus®
- MQSeries®
- Notes®
- pSeries®
- Redbooks (logo) ™
- Redbooks™
- SupportPac™
- Tivoli®
- WebSphere®
- XDE™
- z/OS®
- zSeries®

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

- Rational®
- Rational Rose®

The following terms are trademarks of other companies:

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

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

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

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

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

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

The Patterns for e-business is a group of proven, reusable assets that can be used to increase the speed of developing and deploying Web applications. This IBM Redbook focuses on business process application integration using the Process-focused Application Integration::Serial and Parallel Process Application patterns for intra-enterprise.

Part 1 guides you through the process of selecting an Application and Runtime pattern. Next, the platform-specific Product mappings are identified based upon the selected Runtime pattern. The Runtime and Product mapping patterns in this book focus on the Serial and Parallel Process patterns.

Part 2 presents guidelines on applying the Patterns approach to a sample business scenario and on selecting application integration technologies. It also describes the capabilities of WebSphere® Process Choreographer and WebSphere MQ Workflow.

Part 3 provides detailed design, development, and runtime guidelines for five scenarios, each implemented using WebSphere Process Choreographer and WebSphere MQ Workflow. These implementations focus on automated Web service activities and human interaction activities.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.
Martin Keen is an Advisory IT Specialist at the International Technical Support Organization, Raleigh Center. He writes extensively and teaches IBM classes worldwide on WebSphere products. Before joining the ITSO, Martin worked as a technical consultant for Software Services for WebSphere in Hursley, UK. He holds a bachelor’s degree in Computer Studies from Southampton Institute of Higher Education.

JinYoung Byun works as a FTSS & AP-ATS of WebSphere Business Integration (WBI) within Software Group in IBM Korea, providing product technical support on WBI products. She mainly focuses on Business Process Management including WBI Modeler, WBI Monitor, and WebSphere MQ Workflow. She has ten years of experience in the IT field. Her expertise includes process analysis, the design of workflow systems, providing technical consultancy related to BPM, and Enterprise Application Integration. She holds a master degree in Computer Science from the University of New South Wales in Sydney, Australia.

Mark Grottoli is a Workflow Solution Architect in Canada. He has ten years of experience in the workflow field starting with FlowMark® version 1.0. He has worked at IBM for 20 years. His areas of expertise include WebSphere MQ Workflow, including WebSphere MQ and DB2®, and Enterprise Application Integration.

Li Hui is a software engineer working in IBM China Technical Sales Support. She has four years experience in providing customer technical support, mainly focused on WebSphere MQ and WBI. Li holds a master degree in physics from Beijing Normal University. She is a certified IBM MQ Solution Developer.
Ajay Mathur is a Software Engineer in IBM India Software Labs. He holds a bachelor of Engineering degree in Computer Science and Engineering from Engineering College Kota-India. He has been working with IBM for more than four years in the field of System software and middleware technologies. His areas of expertise include WebSphere Application Server, WebSphere Process Choreographer, and J2EE technologies. He provides product support for WebSphere Business Integration Connect. He has written extensively for IBM developerWorks® on system software, middleware technologies, and wireless technologies. Ajay is a Sun Certified Java™ Programmer.

Robert Skaer is a System Consultant in the USA. He has eight years of experience in the workflow field. He holds a degree in Business Management from LaVerne University in California. His areas of expertise include designing workflow systems and providing workflow education. He has written extensively on robotic systems for automation and banking software.

Guru Vasudeva is an Executive Architect with the IGS Architecture Center of Excellence. He has broad experience in various aspects of Solution Architecture and Enterprise Architecture disciplines in a number of industries including federal government, financial services, retail, and telecommunications. His areas of expertise include Web services, Service Oriented Architectures, Internet architectures, client/server solutions, component-based development, and object-oriented technologies. He is co-author of the book Patterns for e-business - A Strategy for Reuse. Mr. Vasudeva holds a BE in Computer Science from University of Mysore, India, and an MBA in International Finance from UC, USA.

Gerd Watmann works as an IT Specialist within the WebSphere lab-based Services department in the development laboratory at Boeblingen, Germany, providing expert product support focusing on the newest versions of WebSphere Portal Server and WebSphere Process Choreographer. In addition, he has three years of experience with different WebSphere products including WebSphere Application Server and WebSphere Studio Application Developer. He is a Sun Certified Java Programmer and an IBM Certified Solution Developer. Gerd holds a degree in computer science from the Academy of Information Technology, Boeblingen.

Peter Xu is a Consulting IT Specialist with the IBM Software Services for WebSphere group, helping customers deploy IBM products into their organizations. He provides consulting services, education, and mentoring on J2EE technologies, specifically focusing on WebSphere and WebSphere Studio products to Fortune 500 clients. Peter is a certified WebSphere Enterprise Developer and System Expert. He holds a Master's degree in Computer Science from the State University of New York.

The patterns framework presented in this redbook, and its application to the Patterns for e-business, is based on the work of:
Jonathan Adams and Paul Verschueren  
IBM UK

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

**Mark Endrei, Michele Galic, and Carla Sadtler**  
International Technical Support Organization, Raleigh Center

**Holger Kolata, Robert Junghuber, and Guenther Erhard Jornitz**  
IBM Workflow Systems - Architecture and Planning, Germany

**Andre Weiser, Friedemann Schwenkreis, Kurt Lind, and Rolf Baeurle**  
IBM Workflow Development, Germany

**Dominik Goetz**  
IBM Workflow Competence Center, Germany

**Jonas Grundler**  
IBM WebSphere Lab based Services, Germany

**Ed Greenaway**  
IBM Global Services, Australia

**George Galambos**  
IBM Global Services, Canada

**Laxminath Gopisetty**  
IBM Business Consulting Services, USA

---

**Become a published author**

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

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

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

ibm.com/redbooks/residencies.html
Comments welcome

Your comments are important to us!

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

- Use the online **Contact us** review redbook form found at:
  
  ibm.com/redbooks

- Send your comments in an Internet note to:
  
  redbook@us.ibm.com

- Mail your comments to:
  
  IBM® Corporation, International Technical Support Organization
  Dept. HZ8  Building 662
  P.O. Box 12195
  Research Triangle Park, NC 27709-2195
Part 1 provides an overview of IBM Patterns for e-business. It introduces the Business Interaction patterns that the Application Integration pattern is based on. It guides you through the process of selecting Application and Runtime patterns for intra-enterprise integration. The platform-specific Product mappings are identified based upon the selected Runtime pattern.

Included in Part 1 are the following chapters:

- Chapter 1, “Introduction to Patterns for e-business” on page 3
- Chapter 2, “Fundamental concepts in Process Integration” on page 17
- Chapter 3, “Application Integration pattern” on page 33
- Chapter 4, “Node types and Product descriptions” on page 69
- Chapter 5, “Runtime patterns and Product mappings” on page 83
Patterns: Serial and Parallel Processes
Chapter 1. Introduction to 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, an outline of the desired solution, and any special considerations or 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, as well as helping ensure delivery of a solid, properly-architected solution.

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

The Patterns for e-business approach enables architects to implement successful e-business solutions through the re-use of components and solution elements from proven successful experiences. The Patterns approach is based on a set of layered assets that can be exploited by any existing development methodology. These layered assets are structured in such a way that each level of detail builds on the last. These assets 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 describing how the application components and data within a Business pattern or Integration pattern interact
- Runtime patterns that define the logical middleware structure supporting an Application pattern (Runtime patterns depict the major middleware nodes, their roles, and the interfaces between these nodes.)
- Product mappings that identify proven and tested software implementations for each Runtime pattern
- Best-practice guidelines for design, development, deployment, and management of e-business applications

These assets and their relationships to each other are shown in Figure 1-1 on page 5.
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. Then, the architect can further define the additional functional pieces that the application will need to succeed by drilling down using the patterns process. Finally, the application can be built using coding techniques outlined in the associated guidelines.

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

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

1.2 How to use the Patterns for e-business

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

Composite patterns appear in the hierarchy shown in Figure 1-1 on page 5 above the Business patterns. However, Composite patterns are made up of a number of individual Business patterns and at least one Integration pattern. In this section, we discuss how to use the layered structure of Patterns for e-business assets.

1.2.1 Select Business, Integration, or Composite pattern or 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 you are trying to achieve. A proposed business scenario should be described, and each element should be matched to an appropriate IBM Pattern for e-business. You may find, for example, that the total solution requires multiple Business and Integration patterns or that it fits into a Composite pattern or Custom design.

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

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

**Business patterns**

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

**Table 1-1 The four primary Business patterns**

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

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

**Integration patterns**
Integration patterns allow us to tie together multiple Business patterns to solve a business problem. The Integration patterns are outlined in Table 1-2 on page 8.
These Business and Integration patterns can be combined to implement installation-specific business solutions. We call this a Custom design.

**Custom design**

We can illustrate the use of a Custom design to address a business problem through an iconic representation, as shown in Figure 1-2.
If any of the Business or Integration patterns are not used in a Custom design, we can show the unused patterns as lighter blocks than those that are used. For example, Figure 1-3 shows a Custom design that does not have a Collaboration business pattern or an Extended Enterprise business pattern for a business problem.

A Custom design may also be a Composite pattern if it recurs many times across domains with similar business problems. For example, the iconic view of a Custom design in Figure 1-3 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. The identified Composite patterns are shown in Table 1-3 on page 10.
<table>
<thead>
<tr>
<th>Composite Patterns</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electronic Commerce</td>
<td>User-to-Online-Buying</td>
<td>• <a href="http://www.macys.com">www.macys.com</a>&lt;br&gt;• <a href="http://www.amazon.com">www.amazon.com</a></td>
</tr>
<tr>
<td>Portal</td>
<td>Typically designed to aggregate multiple information sources and applications to provide uniform, seamless, and personalized access for its users.</td>
<td>• Enterprise Intranet portal providing self-service functions such as payroll, benefits, and travel expenses.&lt;br&gt;• Collaboration providers who provide services such as e-mail or instant messaging.</td>
</tr>
<tr>
<td>Account Access</td>
<td>Provide customers with around-the-clock account access to their account information.</td>
<td>• Online brokerage trading apps.&lt;br&gt;• Telephone company account manager functions.&lt;br&gt;• Bank, credit card and insurance company online apps.</td>
</tr>
<tr>
<td>Trading Exchange</td>
<td>Allows buyers and sellers to trade goods and services on a public site.</td>
<td>• Buyer's side - interaction between buyer's procurement system and commerce functions of e-Marketplace.&lt;br&gt;• Seller's side - interaction between the procurement functions of the e-Marketplace and its suppliers.</td>
</tr>
<tr>
<td>Sell-Side Hub (Supplier)</td>
<td>The seller owns the e-Marketplace and uses it as a vehicle to sell goods and services on the Web.</td>
<td><a href="http://www.carmax.com">www.carmax.com</a> (car purchase)</td>
</tr>
<tr>
<td>Buy-Side Hub (Purchaser)</td>
<td>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.</td>
<td><a href="http://www.wre.org">www.wre.org</a> (WorldWide Retail Exchange)</td>
</tr>
</tbody>
</table>

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

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

Application patterns break the application down into the most basic conceptual components, identifying the goal of the application. In our example, the application falls into the Self-Service business pattern and the goal is to build a simple application that allows users to access back-end information. The Self-Service::Directly Integrated Single Channel application pattern shown in Figure 1-4 fulfills this requirement.

The Application pattern shown consists of a presentation tier that handles the request/response to the user. The application tier represents the component that handles access to the back-end applications and data. The multiple application boxes on the right represent the back-end applications that contain the business data. The type of communication is specified as synchronous (one request/one response, then next request/response) or asynchronous (multiple requests and responses intermixed).
Suppose that the situation is a little more complicated than that. Let's say that the automobile policies and the homeowner policies are kept in two separate and dissimilar databases. The user request would actually need data from multiple, disparate back-end systems. In this case, there is a need to (1) break the request down into multiple requests to be sent to the two different back-end databases (decompose the request), (2) gather the information sent back from the requests, and (3) put this information into the form of a response (recompose). In this case, the Self-Service::Decomposition application pattern shown in Figure 1-5 would be more appropriate.

![Figure 1-5 Self-Service::Decomposition](image)

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

### 1.2.3 Review Runtime patterns

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

As an example, let's assume that you, as our customer, have determined that your 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 your situation.
You know that you will have users on the Internet accessing your business data and will therefore require a measure of security. Security can be implemented at various layers of the application, but the first line of defense is almost always one or more firewalls that define who and what can cross the physical network boundaries into your company network.

He also needs to determine the functional nodes required to implement the application and security measures. The Runtime pattern shown in Figure 1-6 is one of the options.

![Diagram](image)

**Figure 1-6  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 will fulfill in the application. The presentation and application tiers will be implemented with a Web application server, which combines the functions of an HTTP Server and an application server. It handles both static and dynamic Web pages.
Application security is handled by the Web application server through the use of a common central directory and security services node.

A characteristic that makes this Runtime pattern different from others is the placement of the Web application server between the two firewalls. The Runtime pattern shown in Figure 1-7 is a variation on this. It splits the Web application server into two functional nodes by separating the HTTP Server function from the application server. The HTTP Server (Web server redirector) serves static Web pages and redirects other requests to the application server. It moves the application server function behind the second firewall, adding further security.

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

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

For example, the runtime variation in Figure 1-7 on page 14 could be implemented using the product set depicted in Figure 1-8.

![Figure 1-8 Directly Integrated Single Channel application pattern: Windows® 2000 Product mapping](image)
1.2.5 Review guidelines and related links

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

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

1.3 Summary

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

Process Integration enables companies to connect people, process, and applications across and beyond their enterprise. These solutions make it possible to leverage existing IT investments while providing the flexibility to adapt quickly to changing business conditions and emerging technologies.

This chapter introduces fundamental concepts in integrating people, process, and applications. It proposes a set of notations and a new technique for decomposing complex integration scenarios into simpler portions that can be solved by applying the fundamental concepts of integration. It also discusses Quality of Service (QoS) capabilities that must be considered in integration scenarios. As such, the concepts introduced in this chapter apply to those Architectural patterns documented by IBM Patterns for e-business that leverage various integration techniques. In particular, these concepts are directly applicable to the following areas of IBM Patterns for e-business:

- Application Integration pattern
- Extended Enterprise business pattern
2.1 The need for a unifying technique

There are many existing techniques and disciplines for integration. These are currently fragmented into stovepipes. Therefore, there are problems identifying the best techniques to use and problems in using different techniques together. In particular, terminology is a problem. Each of these disciplines often uses overloaded or ambiguous terminology that inhibits cross-discipline dialog. Use of similar terms in different domains may also mask incompatibilities that only become apparent at lower levels of design. EAI and B2B provide a good example: they have traditionally been seen as very different, whereas a simple diagram shows that they are solving some very similar problems.

Table 2-1 provides an example of terminology overload for “synchronous versus asynchronous.

Table 2-1  Synchronous versus asynchronous terminology overload

<table>
<thead>
<tr>
<th>Domain</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networking</td>
<td>Used to differentiate protocols that can detect transmission errors via acknowledgement messages</td>
</tr>
<tr>
<td>Application programming</td>
<td>Used to indicate whether the caller waits (blocks) until the operation completes</td>
</tr>
<tr>
<td>Messaging</td>
<td>Used to differentiate services that can store and forward messages (avoiding the need for all linked services to be available)</td>
</tr>
</tbody>
</table>

These inconsistencies significantly complicate the Process Integration efforts by impeding communication between different groups of skill sets needed to implement the end-to-end integration solution.

2.1.1 Similarities between intra- and inter-enterprise integration

As shown in Figure 2-1 and Figure 2-2 on page 19, intra-enterprise integration and inter-enterprise integration are both concerned with integrating source and target applications.

Figure 2-1  Intra-enterprise integration
Hence, the lessons learnt from traditional Enterprise Application Integration (EAI) solutions can be applied to business-to-business (also known as Extended Enterprise or inter-enterprise) integration. It is important to note, however, that there would be differences in Quality of Service (QoS) concerns and commercial considerations that are of particular significance to inter-enterprise integration.

For example, core concerns for inter-enterprise integration include security, interoperability, and governance (defining the responsibilities of each party). Nevertheless, we should expect that inter-enterprise solutions can leverage the majority of the intra-enterprise concepts.

### 2.1.2 Summary

As integration technologies have evolved, many similarities between the intra-enterprise and inter-enterprise integration approaches have become apparent. It should be possible to describe a set of underlying concepts that apply to both the areas.

### 2.2 Process Integration concepts and notations

In this section, we introduce basic concepts and notations for capturing the different types of interactions encountered in Process Integration.

#### 2.2.1 Collaboration and Interaction

Integration between people, process, and applications can be thought of as collaboration and interaction between participating entities.

**Collaboration**

In the most general sense, a *collaboration* denotes N-to-N activities between sub-systems within a distributed system. As shown in Figure 2-3 on page 20, complex collaborations between sub-systems can be broken down into more basic *interactions*. An interaction focuses on one-to-one or one-to-N activities originating from a single sub-system.
In this way, complex collaborations involving many sub-systems can be decomposed into simpler interactions that are easier to analyze. Data analysts use a similar approach when analyzing complex data with many-to-many relationships. Normalization is used to reduce many-to-many relationships between data to one-to-many relationships.

Note that we do not show a link from A-to-C on the right of Figure 2-3. This is because, in breaking the interaction down, we found that A only initiates interactions with B, D, and E. The C-to-A interaction will be modeled in another one-to-N interaction.

**Interaction**

As we just saw, an interaction is a collaboration originating from a single component. Figure 2-4 shows an interaction between a source application and a target application. The initiating operation is indicated by a small solid circle.

Complex interactions may be decomposed into several simpler interactions to enhance the level of detail. An example is shown in the Figure 2-5 on page 21, where a query for a quote is decomposed in a request step, an acknowledgement step, and a final reply step.
Figure 2-5   Decomposition of complex interactions

An ellipse spanning one or more basic interactions denotes a shared context involved in a complex interaction between two or more sub-systems. Examples of shared contexts are session, security, transaction, process control, and so forth.

Complex interactions with multiple target applications also can be decomposed into multiple one-to-one interactions, as long as there is one initiating operation within a source application. The interaction patterns approach can then be applied to these one-to-one interactions.

2.2.2 Connectors and Adapters

The terms connector and adapter are often used interchangeably. This section defines their use in a Patterns for e-business context.

Connectors

Connectors provide the connectivity between source and target applications. A connector is always present to facilitate interaction between two sub-systems.

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

- A primitive (or unmodelled) connector, represented by a simple line between sub-systems
- A component (or modelled) connector, represented by a rectangle on a line between sub-systems

For lower-level modelling, a primitive connector can always be decomposed into a modelled connector and two adjacent primitive connectors, as shown in Figure 2-6 on page 22. This way, connector models can be recursively decomposed until the correct level of detail is reached.
It is useful to distinguish two connector subtypes, as shown in Figure 2-7:

- An **adapter connector** is concerned with enabling logical connectivity by bridging the gap between the context schema and protocols used by the Source application (S type) and Target application (T type).
- A **path connector** is concerned with providing physical connectivity between Source and Target applications. It may be very complex (for example, the Internet) or very simple (an area of shared storage).

These connector subtypes are orthogonal, meaning a connector may be both an adapter connector and a path connector. The relationship between connectors and adapters is shown in Figure 2-7.
Adapters
Adapters provide the logical connectivity to an application. Without adapters, each application would need to implement the specific interface of the target application.

It is useful to distinguish three types of adapters:

► Control adapters are not concerned with content. They are only concerned with the activities involved in flow operations:
  – Transforming the protocol used between the segments
  – Segmenting, batching, and sorting data blocks
  – Correctly interacting with the path connector to execute the transport operation (This includes respecting the protocol rules.)

► View adapters are concerned with transforming content but only in terms of its technical representation. Examples include:
  – Element demarcation schemes, such as delimited, fixed-length, and XML
  – Element sequencing schemes, such as keys and collation sequences
  – Element encoding schemes, such as character set, number format, and date format

► Model adapters transform the semantic content and normally require business input to define correct operational rules. Some examples are:
  – Splitting out subsets of data
  – Joining external data (augmentation/enrichment)
  – Summarization
  – Translation of identifiers (key management)

Coupling adapter connectors
Coupling adapter connectors can be used to implement a common integration protocol such as messaging, RMI/IIOP, SOAP/HTTP, and so on. As shown in Figure 2-8 on page 24, the adapter functionality between the source application and the target application is decomposed into two halves. Each half adapts to and from a common intermediate protocol.
If there are multiple point-to-point connections between a group of sub-systems, this approach can significantly reduce the number of different adapters required. Each sub-system only needs one adapter (instead of needing a different adapter to connect to each sub-system).

**Connectors and synchronicity**

In order to describe the time dependencies between the initiating operation and the resulting collaborative activities, two cases may be distinguished:

- **Synchronous interaction**
  The initiating operation cannot complete until the interaction has been completed. In this case, the source application is synchronously coupled with the target application.

- **Asynchronous interaction**
  The initiating operation can complete before the interaction completes. The operation is then regarded as asynchronous, and the source application is synchronously decoupled from the target application.

### 2.2.3 Classification of interaction between sub-systems

The interactions involved in Process Integration can be broadly classified as *parallel* and/or *serial*.

**Parallel interaction**

An interaction is denoted as parallel if it includes a set of concurrent one-to-one interactions between a source application and multiple target applications, as shown in Figure 2-9 on page 25.
Serial interaction

An interaction is denoted as serial if it includes a series of one-to-one interactions between a source application and multiple target applications that are subject to time sequenced dependencies, as shown in Figure 2-10.

Classification of Interactions

Distributing parallel and serial interactions along two dimensions of a matrix provides the four combinations shown in Figure 2-11.

---

**Figure 2-9** Parallel interaction

**Figure 2-10** Serial interaction

**Figure 2-11** Classification of interactions
This classification framework is used later in this chapter to classify Application patterns for both Application Integration patterns (intra-enterprise) and Extended Enterprise business patterns (inter-enterprise).

### 2.3 QoS capabilities framework

This section documents the most frequently observed Quality of Service (QoS) concerns that must be considered in implementing integration solutions. These QoS manifest themselves with differing degrees of importance and specificity in different integration scenarios.

The following QoS concerns are defined in this section:

- Operability
- Availability
- Federation
- Performance
- Security
- Standards compliance
- Transactionality

#### 2.3.1 Operability

This QoS concern focuses on the systems-management requirements of the deployed solution. It focuses on issues such as monitoring, logging, traces, recovery, and manageability of the solution during operations in a production environment.

#### 2.3.2 Availability

Availability is a measure of the time that a service is functioning normally. It also serves as a measure of the time the recovery process requires after the service fails. In other words, it is the downtime that defines service availability. This downtime includes both planned and unplanned downtime.

High availability generally requires that a topology provide some degree of redundancy in order to eliminate single points of failure. This allows the downtime caused by a component failure to be minimized (ideally zero). It can also allow a service to continue functioning normally during the downtime of a component for planned maintenance or backup procedures, for example.
2.3.3 Federation

Federation is fundamentally about enabling services to interoperate across trust boundaries. It lets access control functions span across multiple domains, crossing application, product, platform, site, business unit, and organization boundaries.

Federation requires that each partner domain is trusted to authenticate the identity of its own users. Mechanisms are needed for passing resource and user authentication and authorization information between domains.

2.3.4 Performance

The performance of a service is measured in terms of throughput and latency. Throughput represents the number of requests served in a given time period. Latency is the round-trip time between sending a request and receiving the response. Higher throughput and lower latency values represent good performance of a service.

Scalable topologies are able to service higher loads by adding the appropriate processing power. This can be achieved by using techniques such as using a faster machine, using a special purpose machine, or creating a cluster of machines.

Other performance improvement techniques include caching, batching, and connection pooling.

2.3.5 Security

Permission to access the participating applications may be associated with the requesting application itself, or this application may carry the credentials of a user initiating the actions. Consequently, access control can be applied as far as the requesting application (a transit of trust) or only from an integration hub (a trusted source) that authenticated the original request.

Securing messages transported and ensuring that integration is achieved only with authorized applications under the correct user credentials is a must. The integration solution needs to provide:

- Data protection through encryption
- Authentication of users and subscribing applications (in cases where non-repudiation of the end-user is required, authentication of the end-user)
- Authorization of the user for participation in an integration activity
2.3.6 Standards compliance

Standards compliance is concerned with identifying and applying the appropriate standards to a scenario. Standards compliance is an important factor for controlling development and integration costs. Even private standards are beneficial, but widely-accepted public standards have the added advantage of enabling interoperability in the broadest contexts.

2.3.7 Transactionality

A transaction can be viewed as an activity between two or more parties that must be completed in its entirety with the mutually agreed outcome. Transactionality enables multiple application operations to be coordinated to provide an atomic deterministic end result.

Resource managers are used to control access to the resources involved in a transaction. A transaction manager is responsible for coordination and transaction control. Transactional considerations include:

- ACID versus compensating transactions
- Flat versus nested transactions
- System versus client commit control
- Local versus distributed transactions

2.4 Application patterns for Application Integration

Using the interaction classification framework introduced in this chapter in Figure 2-11 on page 25 as a guide, we observe the following four Application patterns and their variations for Process-focused Application Integration (also known as intra-enterprise integration):

- Direct Connection Application pattern and its Message/Call Connection variations
  Allows a single interaction from the source application to be adapted and transported to one target application

- Broker Application pattern and its Router variation
  Allows a single interaction from the source application to be switched, split, and joined to multiple target applications concurrently

- Serial Process Application pattern and its Serial Workflow variation
  Allows a single interaction from the source application to execute a series of interactions with multiple target applications
Parallel Process Application pattern and its Parallel Workflow variation

Allows a single interaction from the source application to concurrently execute multiple series of interactions with multiple target applications.

These four Application patterns for Application Integration are summarized in Figure 2-12. One dimension shows support for concurrent interactions to multiple target applications in parallel. The other dimension shows support for non-concurrent interactions to multiple targets in series.

Figure 2-12 Classification of Process-focused Application Integration patterns

2.5 Application patterns for Extended Enterprise

Using the interaction classification framework introduced in this chapter in Figure 2-11 on page 25 as a guide, we observe the following three Application patterns and their variations for the Extended Enterprise business pattern (also known as inter-enterprise integration):

- Exposed Direct Connection application pattern and its Exposed Message/Call Connection variations
  
  Allows a single interaction from the source application to be adapted and transported to one partner target application.
- Exposed Broker application pattern and its Exposed Router variation
  Allows a single interaction from the source application to be switched, split, and joined to multiple partner target applications concurrently

- Exposed Serial Process application pattern and its Exposed Serial Workflow variation
  Allows a single interaction from the source application to execute a series of interactions with multiple partner target applications

Each of the Extended Enterprise pattern names are prefixed with *Exposed* to highlight that these patterns are concerned with exposing applications outside of the enterprise boundaries.

These three Application patterns for Extended Enterprise are summarized in Figure 2-13. One dimension shows support for concurrent interactions to multiple target applications in parallel. The other dimension shows support for non-concurrent interactions to multiple targets in series. Here the top right-hand corner has been left blank to indicate that Parallel Process implementations are currently not widely implemented in Extended Enterprise scenarios. As the process composition technologies mature, we expect to see more widespread use of the Exposed Parallel Process application pattern and its Exposed Parallel Workflow variation.

![Figure 2-13 Classification of Extended Enterprise patterns](image)

### 2.6 Summary

This chapter introduces fundamental concepts in Process Integration that bridge the gap between various disciplines. It presents a set of notations and
techniques that can be iteratively applied to a complex Process Integration scenario, where each iteration refines and further details the integration solution. It also introduces the key set of QoS concerns that must be addressed in Process Integration efforts. Finally, this chapter presents an interaction classification framework that is used to capture the commonly occurring Application patterns in the field of Application Integration and Extended Enterprise.

Please note, this redbook only focuses on the following Application patterns:

- The Process-focused Application Integration: Serial Process and Parallel Process Application patterns, which are introduced in Chapter 3, “Application Integration pattern” on page 33.
Application Integration pattern

The Application Integration pattern captures commonly observed solution alternatives in the domain of Enterprise Application Integration (EAI). They capture best practices around back-end integration of applications and data, process automation, and workflow implementations involving human interactions. It is important to note that front-end integration such as the composition of a portal or single sign-on across multiple applications is captured by the Access Integration pattern.

The Application Integration pattern can be implemented using any one of the four Process-focused application patterns and the Data-focused application patterns. These various designs provide solution flexibility to address the specific needs of the business process being automated.
3.1 Using the pattern

Within the context of Patterns for e-business, Application Integration patterns are used at two different levels:

- To design complex solutions by combining more than one Business pattern. An example might be creating an e-commerce site by combining Self-Service and Information Aggregation patterns.
- To support the implementation of a given Business pattern. For example, it is hard to imagine the Self-Service::Decomposition application pattern without leveraging the Application Integration best practices. Similarly, all the Application patterns for Extended Enterprise are in one sense the implementation of Application Integration across organizational boundaries.

Typically, the requirements that drive application integration call for the seamless execution of multiple applications and access to their respective data in order to automate a complex and new business function. Reliable integration of applications (be they legacy stovepipe applications, packaged software applications, or custom applications) requires the use of proven, repeatable patterns. At its highest level, application integration can be divided into two essentially different 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

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

**Note**: Certain types of integration between applications can be accomplished at the user interface level as well, as covered in the Access Integration pattern.

What's next

Enterprise Application Integration is a complicated undertaking. It requires, first, a thorough understanding of both the individual applications being integrated and the possible methods that can be used to interconnect them.
3.2 Defining the Application Integration patterns

This section defines the Application Integration patterns by documenting the Business and IT drivers that lead to the selection of this pattern, the context within which it can be used, the proposed solution, and examples of its usage.

It also discusses typically observed application integration requirements that can help determine which of the two Application Integration categories (Process-focused or Data-focused) you should use in designing your e-business solution.

3.2.1 Business and IT drivers

Typical 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.

3.2.2 Context

Application Integration patterns can be observed in solutions that call for close integration with systems and databases that exist within the organization. They serve as back-end integration patterns and are critical for the successful implementation of certain Business patterns. For example, solutions that use the Self-Service business pattern or Extended Enterprise business pattern often rely on these same application integration techniques. Similarly, many Custom designs and Composite patterns use Application Integration application patterns.

For example, take the case of a company that wants to integrate its retail and wholesale departments. Currently, both departments have proven IT infrastructures but have no inter-connectivity. The Process-focused Application Integration patterns address this problem. These patterns can be applied in a case where the business process needs to be integrated between existing business systems within the organization. The Process-focused Application Integration patterns can be used to integrate the retail ordering and wholesale inventory systems, thus eliminating ordering lag and providing an up-to-date inventory.
3.2.3 Solution

The Application Integration pattern typically consists of the following elements:

- Business applications and data that need to communicate, interact, and integrate with other business applications and data within the organization
- A network that:
  - Is based either (1) on TCP/IP and other Internet technologies or (2) on proprietary protocols
  - Can be a dedicated LAN or WAN connection
- Other business applications and data that can be:
  - Custom developed systems (old and new)
  - ERP systems and other packaged applications, such as SAP, BAAN, and PeopleSoft
  - Databases
- Application integration services that include:
  - Protocol Adapters
  - Message handlers
  - Data transformation
  - Decomposition/recomposition
  - Routing/navigation
  - State management
  - Security

3.2.4 Putting the pattern to use

This is probably one of the most commonly used patterns and it can be observed in any solution where an application needs to integrate with other applications, legacy systems, and databases. Examples include the following:

- An electronics retailer/wholesaler, such as ITSO Electronics from our sample scenario, needs to integrate its retail ordering process with its inventory management system.
- A telecommunications company needs to integrate its online sales systems and its core provisioning systems to improve efficiency and customer service.

3.2.5 Application Integration considerations

Choosing the right Application Integration pattern can only be done in the context of specific solution requirements. These requirements encompass not only the specific needs of application integration to be deployed, but also the constraints
posed by the enterprise's existing IT infrastructure and technology investments. This section details considerations to be made and questions to ask in determining the best integration technique for a solution under consideration.

**Request for information versus request for processing**
Is the integrated solution for informational access only or is it intended to integrate requests for processing? The Process-focused Application Integration patterns are concerned with integration of the functional flow of processing between applications. The Data-focused Application Integration patterns are concerned with integration of the information used by applications.

**Foreground versus background integration**
Is there a user awaiting the outcome of the operation or is this operation running behind the scenes? An example of a foreground (or real-time) process may be a user retrieving a price quote for the purchase of product. Conversely, a background (or batch) process would be the synchronization of pricing information from the central office out to all of the local stores.

**Scope of integration**
Does the integration project involve only a single Business pattern, multiple Business patterns, or the creation of an entire e-infrastructure for multiple e-business solutions?

**Operation latency (applications and/or data queries)**
How long will it take the operation to complete in the application? Operations that can not complete in less than a few seconds typically dictate the need for asynchronous methods of integration. A query on product inventory may be a quick operation, while the computation of the production plan for the manufacturing of that inventory could take minutes to hours to complete.

**Geographic proximity**
How close do the applications being integrated reside to one another? Similar to the idea of operation latency, an often overlooked element of the EAI design is the proximity of the participating applications in relation to each other. Integration of applications residing in the same data center has a much smaller integration latency than integration of applications spread around the world.

**Process re-engineering**
Is there a need to re-engineer business processes or extend an existing business process? Most legacy business processes are embedded within the applications themselves. Business Process Management (BPM) is performed by the existing application(s). Sometimes the EAI effort is merely trying to better
integrate functional operations of a disconnected, narrow (or “stovepipe”) business process. Other endeavors are more ambitious, with the desire to improve business processes through integration.

There are varying degrees of process extensions for application-based BPM:

- Extending the reach of the business process by integration with other applications
- Joining two separate application-based business processes together into one unified process
- Separating BPM from application logic by implementing the process in a Process manager (This option extends the domain of the process by allowing it to encompass any participating application under any specific sequencing and process flow control.)

**Application portfolio**

What is contained in the mix of applications? This mixture might include pre-packaged software, legacy applications, or newly developed applications. One of the most important elements of an EAI project is a survey of the application landscape. Some environments are heavily based on pre-packaged software. Other environments are completely homegrown custom applications. Still, other environments may be a mixture of pre-packaged applications working with legacy homegrown applications.

This survey will detail several key points about the enterprise environment:

- Can the application interfaces be extended as part of the integration activity? Homegrown applications may have standardized interfaces or can be extended to implement standards. Interfaces into pre-packaged applications typically can only be standardized through implementation of sophisticated adapters.

- Is there a central cornerstone application in the enterprise environment or a portfolio of peer applications? Is the business processing focused around one key application (perhaps an ERP system) with all other applications being subservient to that application?

- How many applications are being integrated? For instance, a typical Self-Service application may be integrating the Web application server with one back-end system. At the other end of the spectrum would be a project creating a centralized customer information system that may require feeds from 100+ different applications. There is a significant difference in the selected solution for integrating two applications versus integrating 100+s of applications.
Key characteristics of the application portfolio and enterprise architecture that affect the EAI approach include these:

- Number of applications
- Degree of centralization of the data repositories
- Completeness of the application interfaces
- Conformity of the participating applications to the EAI data and interface model

**Tight coupling versus loose coupling**
What is the level of independence between the application implementation and the EAI interface? How likely is it that changes to the application will require changes to the interface or changes to the integration approach? The degree of invasiveness not only affects the application adapter. It can also affect the integration hub processing and even require changes to the partner application. The farther across the application integration topology a change ripples, the more expensive this change will be. The degree of invasiveness is often described in terms of coupling (loose coupling versus tight coupling) or a black box versus a white box approach.

Ideally, the less invasive the integration, the more successful the integration will be in the long-term. This is the primary reason for the use of messaging-based integration to isolate as much of the integration processing from any application-specific dependencies. EAI best practices should be employed to ensure that the integration is as non-invasive as possible.

However, EAI projects will vary in the level of independence available based on the completeness of the functionality and interfaces of the participating applications. It is best to implement environments with heavy application-specific processing required by using a sophisticated integration broker component supported by easy-to-use application development tools. This ensures that future extensions to the integration can be implemented quickly and easily.

### 3.2.6 What's next

If you have determined that the Application Integration pattern is appropriate for use in your solution, the next step is to select an associated Application pattern based on your specific business and IT drivers.

If the Application Integration pattern is not appropriate for your development efforts, review the Business patterns to determine which pattern best addresses your e-business needs.
3.3 Application patterns

Application patterns for Application Integration can be broadly categorized as either Process-focused or Data-focused. These two categories enable different types of integration functionality.

The focus of this redbook is Application patterns for Process-focused Application Integration. In particular the Serial Process and Parallel Process application patterns and their Workflow variants are explored. A brief overview of the other Application Integration patterns is provided. For full details on the other Application Integration patterns, please see the IBM Patterns for e-business Web site:


The diagram conventions shown in Figure 3-1 are used in the Application patterns that follow.

3.4 Process-focused Application patterns

Process-focused Application Integration patterns are observed where multiple automated business processes are combined to yield a new business offering or to provide a consolidated view of some business entity by integrating multiple
corporate business systems. An often quoted example is the consolidated view of the state of all relationships of the business with a particular customer.

This mode of integration is highly flexible. In its more sophisticated form, it enables late binding of the targets of integration and is particularly useful in tying together different platforms and technologies. However, it represents a more difficult design and development task compared to data-focused integration. Moreover, it often requires complex middleware.

The Process-focused Application Integration patterns are presented here 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. From the following Application patterns, select the one that best fits your requirements:

- **Direct Connection application pattern**: Message/Call Connection variations
- **Broker application pattern**: Router variation
- **Serial Process application pattern**: Serial Workflow variation
- **Parallel Process application pattern**: Parallel Workflow variation

**Business and IT drivers**

Table 3-1 and Table 3-2 on page 42 summarize the business and IT drivers for the Process-focused Application Integration patterns and their variations.

<table>
<thead>
<tr>
<th>Business Drivers</th>
<th>Direct Connection Message variation</th>
<th>Direct Connection Call variation</th>
<th>Broker Router variation</th>
<th>Broker Serial Process</th>
<th>Serial Workflow variation</th>
<th>Parallel Process</th>
<th>Parallel Workflow variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improve the organizational efficiency</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reduce the latency of business events</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Support a structured exchange within the organization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Business Drivers

<table>
<thead>
<tr>
<th></th>
<th>Direct Connection Message variation</th>
<th>Direct Connection Call variation</th>
<th>Broker</th>
<th>Router variation</th>
<th>Serial Process</th>
<th>Serial Workflow variation</th>
<th>Parallel Process</th>
<th>Parallel Workflow variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support real-time one-way 'message' flows</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Support real-time request/reply 'message' flows</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Support dynamic routing of 'messages' to one of many target applications</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Support dynamic distribution of 'messages' to multiple target applications</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Support automated coordination of business process flow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reduce cycle time through parallel execution of portions of a process flow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Support human interaction and intervention within the process flow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

**Table 3-2  IT drivers**

## IT Drivers

<table>
<thead>
<tr>
<th></th>
<th>Direct Connection Message variation</th>
<th>Direct Connection Call variation</th>
<th>Broker</th>
<th>Router variation</th>
<th>Serial Process</th>
<th>Serial Workflow variation</th>
<th>Parallel Process</th>
<th>Parallel Workflow variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimize total cost of ownership (TCO)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Leverage existing skills</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Quality of Service concerns

This section highlights Quality of Service (QoS) capabilities that are of particular concern in the Process-focused Application Integration domain. A QoS capabilities framework for business integration is based on the following general concerns:

- Operability
- Availability
- Federation
- Performance
- Security
- Standards compliance
- Transactionality

Important: This profile is intended as a rough first guide to QoS concerns that differentiate this domain, suitable for high level architectural design. They are not a substitute for thorough analysis at a later design stage.
The following QoS concerns are of particular importance when working in the Process-focused Application Integration domain.

**Operability**
The complexity of IT infrastructure is increasing so that systems management capabilities are needed to ensure that Application Integration solutions can be managed effectively. For example, clustering solutions may be a consideration for availability management and reducing operational costs.

**Performance**
High volume workloads are often experienced in the intra-enterprise integration domain. Therefore, there is a generally a need to carefully assess the expected workload and to plan for future growth in workload.

**Standards compliance**
Rather than using different approaches for each application integration exercise that an organization performs, standards need to be identified and applied in order to control development and integration costs.

Private standards are also acceptable when beneficial. Adopting WebSphere MQ, for example, as intra-enterprise message-oriented middleware provides assured, once-only delivery messaging that can be widely used across the organization.

**Transactionality**
Transaction services are often important in intra-enterprise application integration scenarios in order to preserve data integrity and to avoid data loss. Consider using transaction management products that work with XA-compliant resource managers to provide a commit and rollback facility, ensuring that either all resource updates are completed or all updates are rolled back.

### 3.4.1 Direct Connection Application pattern

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

More complex point-to-point connections will have modeled connection rules such as business rules associated with them, as shown in Figure 3-2 on
Connection rules are generally used to control the mode of operation of a connector depending on external factors. Examples of connection rules are:

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

The Direct Connection application pattern has two variations:

- Message Connection variation
- Call Connection variation

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

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

We examine these two variations in more detail later in this section.
**Business and IT drivers**
The business and IT drivers for choosing the Direct Connection application pattern are to:

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

The primary goal is to allow one application to gain direct and real-time access to another in order to reduce the latency of business events.

**Solution**
This Application pattern, as shown in Figure 3-2 on page 45, is divided into a number of logical components:

- The Source Application tier represents one or more applications that are interested in initiating an interaction with the target application.

- The Connection is the line between the source application and the target application, representing a point-to-point connection between the two applications.

- The Connection Rules tier represents any business rules associated with the connection, such as data mapping rules and security rules.

- The Target Application tier represents a new application, a modified existing application, or an unmodified existing application. This application is responsible for implementing the necessary business services.

**Guidelines for use**
Direct integration between applications can be inflexible, in that any changes to one application may have knock-on effects on other applications. Changes to the target application may also require changes to the source application. Such changes can become both expensive and time consuming, especially when the target application is being accessed by a number of different source applications.

Different IT departments also may be responsible for developing and maintaining the source and target applications. Under such a scenario, development might be difficult to coordinate, especially if the interfaces between the applications being integrated are not properly defined and documented. Because of this, it is important to clearly define such interfaces in advance.
**Benefits**
The Direct Connection application pattern offers the following benefits:

- It works with applications that have simple integration requirements with only a few back-end applications.
- It increases the organizational efficiency and reduces the latency of business events by (1) providing real-time access to business data and business logic and (2) avoiding manual synchronization of data between applications.
- Direct access to back-end applications reduces the duplication of business logic across multiple tiers. As a result, changes to business logic can be made in one tier rather than in multiple applications.
- It can enable re-use of investments already made with the organization.

**Limitations**
Although this is a reasonable starting Application pattern for integrating applications in a one-to-one relationship with one another, this pattern will result in a many-to-many spaghetti configuration with point-to-point integration mappings for each application pair. Also, the expansion of this implementation into a multi-point configuration will require additional application logic to handle the coordination.

This pattern cannot be used for intelligent routing of requests, for decomposition and re-composition of requests, and for invoking complex business process workflow as a result of a request from another application. Under such circumstances, you should consider a more advanced Application pattern, such as Broker or Serial/Parallel Process.

**Putting the Application pattern to use**
ITSO Electronics, an electronics retailer/wholesaler, wants to integrate its retail and wholesale departments. Currently, both organizations have proven IT infrastructures but have no interconnectivity. The first process that ITSO Electronics wants to focus on is the inventory and order replenishment process. Currently, the items sold are tallied at the end of the month by the retail ordering process and delivered to the wholesale organization by internal mail. This method creates a lag in the inventory replenishment process and causes many out-of-stock situations. A primary business goal is to minimize the loss of sales due to out-of-stock situations. To meet these requirements, ITSO Electronics selects the Direct Connection application pattern.
3.4.2 Direct Connection: Message Connection variation

The Message Connection variation, shown in Figure 3-3, applies to solutions in which the business process does not require a response from the target application within the scope of the interaction.

![Secure Zone Diagram]

Figure 3-3  Message Connection variation

**Note:** We have not shown the connection rules box in Figure 3-3 because we want to focus on the connection itself.

**Business and IT drivers**

The business and IT driver for choosing the Message Connection variation of the Direct Connection application pattern is to:

- Support real-time one-way message flows

The main driver for selecting this variation is when the business process has no interest in the result of the operation. This variation also has the most natural fit when message-oriented middleware is used, such as IBM WebSphere MQ.

**Putting the Application pattern to use**

In our scenario, the retail department of the ITSO Electronics organization needs to notify the wholesale department to update its inventory records when a part needs to be ordered. The retail department does not require any acknowledgement of this request. To meet these requirements, ITSO Electronics chooses the Message Connection variation of the Direct Connection application pattern.
3.4.3 Direct Connection: Call Connection variation

The Call Connection variation, shown in Figure 3-4, applies to solutions in which the business process depends on the target application to process a request and return a response within the scope of the interaction.

![Secure Zone diagram](image)

Figure 3-4 Call Connection variation

**Note:** We have not shown the connection rules box in Figure 3-4 because we want to focus on the connection itself.

**Business and IT drivers**

The business and IT driver for choosing the Call Connection variation of the Direct Connection application pattern is to:

- Support real-time request/reply message flows

The main driver for selecting this variation is when the business process does require a result message from the interaction.

**Putting the Application pattern to use**

In our scenario, the retail department of the ITSO Electronics organization needs to be advised by the wholesale department of the expected delivery date of a part on order that is out of stock with the retail department. To meet these requirements, ITSO Electronics chooses the Call Connection variation of the Direct Connection application pattern.

3.4.4 Broker Application pattern

The Broker application pattern, shown in Figure 3-5 on page 50, is based on a one-to-N topology that separates distribution rules from the applications. It allows a single interaction from the source application to be distributed to multiple target
applications concurrently. This application pattern reduces the proliferation of point-to-point connections.

![Broker application pattern](image)

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

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

The Broker application pattern was previously known as the Aggregator application pattern for read intent calls and the Broker application pattern for Messages and update intent calls. However, this distinction was found to be of insufficient value to warrant a separate pattern. Therefore, it has been dropped from the revised PI patterns.

**Business and IT drivers**

The primary business driver for selecting this Application pattern is to allow one application to interact with one or more of multiple target applications. Using a hub-and-spoke architecture instead of a point-to-point architecture allows for the seamless integration of applications while minimizing the complexity. A request for information can be routed to one of many targets or simultaneously to multiple targets. The resulting request message can be decomposed into multiple request messages. Then, the reply messages are recomposed into a single reply message using appropriate recomposition rules.
This externalization of routing, decomposition, and recomposition rules from individual source and target applications increases the maintainability and flexibility and reduces the enterprise-wide integration complexity.

This Application pattern is particularly important when a processing request requires execution of multiple interactions concurrently or when the source application should be relieved of the need to know anything about its targets.

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

**Solution**

This Application pattern, as shown in Figure 3-6 on page 53, is divided into a number of logical components:

- The Source Application tier represents one or more applications that are interested in interacting with the target applications.

- The Broker Rules tier reduces the proliferation of direct connections. In addition, it supports message routing, decomposition and recomposition, message enhancement, and transformation. These rules are often captured as business rules that govern the behavior of the broker tier. This tier also uses a work-in-progress data store to retain the intermediate results from the responses coming back from target applications until all the necessary responses are received.

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

**Guidelines for use**

To increase the flexibility of the solution and responsiveness to changing business requirements, it is recommended that particular attention is paid to definition of reusable messages/services that pass through the Broker tier.

Robust transaction processing systems should be used to implement the back-end applications to ensure availability, scalability, and performance.

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

Standards should be used where possible to minimize future changes required to the source and target applications.
Benefits
The benefits of this Application pattern are:

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

Limitations
Logic must be implemented at the broker for routing and decomposition/recomposition tasks.

Putting the Application pattern to use
ITSO Electronics consists of multiple retail stores and wholesale departments. The retail stores get their supplies from the wholesale departments and have a need to request the delivery dates of those supplies before ordering. Currently there is no integration of the retail and wholesale applications. All interaction between the two are done over the phone or by mail. A solution must be found to allow retail stores to request delivery dates from the wholesale departments. To eliminate the need for the retail departments to know which wholesale department carries which supplies, a Broker is needed to take incoming requests and direct them based on part numbers to the wholesale department that carries them. In the event that a part is carried by multiple wholesale departments, the broker must get delivery dates from each and return the best date and the wholesale department that can supply it to the retail department.

3.4.5 Broker: Router variation
The Router variation of the Broker application pattern, shown in Figure 3-6 on page 53, applies to solutions in which the source application initiates an interaction that is forwarded to at most one of multiple target applications.
Where the Broker application pattern enables 1:N connectivity, the Router application pattern enables 1:1 connectivity where the Router Rules tier selects the target.

The Router variation of the Broker application pattern was previously known as the Router variation of the *Aggregator* application pattern.

**Business and IT drivers**

The primary business driver for selecting this Application pattern is similar to that of the Broker application pattern. The difference lies in the fact that the Router tier routes the request to only one of multiple target applications. The requirement for transformation of message and interface format still applies. Externallizing the routing from individual source and target applications increases the maintainability and flexibility and reduces the enterprise-wide integration complexity.

This Application pattern is particularly important when a processing request requires the source application to be relieved of the need to know anything about its targets.

The primary IT driver for selecting this Application pattern is to allow loose coupling of clients and services with minimum modification to each. The solution should allow for multiple transmission protocols to be used and for transformation of protocols between client and service.
Solution
This Application pattern provides a routing function to allow any attached (initiating) application using a single router link to connect to one of multiple target applications. While access to multiple applications is supported, at any given time an application is connected to only one other application.

This Application pattern, as shown in Figure 3-6 on page 53, is divided into a number of logical components:

- The Source Application tier represents one or more applications that are interested in interacting with the target applications, one target at a time.

- The Router Rules tier represents any business rules associated with the message handling, such as routing and transformation. It receives requests from multiple source applications and routes them intelligently to the appropriate target applications. The resulting integration is essentially a point-to-point connection between source and target. This tier implements minimal business logic.

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

Guidelines for use
The guidelines for this application pattern are the same as those for the Broker application pattern.

Benefits
The benefits of this Application pattern are:

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

Limitations
With the Router variation, there is limited ability in the router to manipulate the requests. It performs intelligent routing and protocol transformation, but it does not have the ability to send simultaneous requests to the target applications.
Putting the application pattern to use
ITSO Electronics consists of multiple retail stores and wholesale departments. The retail stores get their supplies from the wholesale departments and have a need to request the delivery dates of those supplies before ordering. Currently there is no integration of the retail and wholesale applications. All interaction between the two are done over the phone or by mail. A solution must be found to allow the retail stores to request delivery dates from the wholesale departments. To eliminate the need for the retail departments to know which wholesale department carries which supplies, a Router is needed to take incoming requests and direct them based on part numbers to the wholesale department that carries them. This differs from the example outlined in the Broker pattern in that only one wholesale department will carry a part. There is no need to distribute one request to multiple wholesale departments simultaneously to see who can supply the part at the earliest date.

3.4.6 Serial Process Application pattern
The Serial Process application pattern, shown in Figure 3-7, extends the one-to-N topology provided by the Broker application pattern by facilitating the sequential execution of business services hosted by a number of target applications. Thus it enables the orchestration of a serial business process in response to an interaction initiated by the source application.
Business and IT drivers
The primary business driver for selecting this Application pattern is to support the composition of end-to-end business process flows by leveraging business services implemented by a number of target applications.

From an IT perspective, the key driver for selecting this Application pattern 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 is the same as for the Broker application pattern.

- 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. 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 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 utilizes a work-in-progress (WIP) database to store the intermediate results from the execution of different process steps.

- The Target Application tier is the same as for the Broker application pattern.

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

Typically, legacy applications are not designed with this thinking in mind. Similarly, many of the legacy applications have significant amounts of process logic embedded within them. These constraints in existing environments may pose challenges to fully implementing the vision promised by this Application pattern.
pattern. Careful refactoring of legacy and packaged applications by wrapping them into business services is a good starting point for the eventual widespread implementation of this Application pattern within an enterprise.

Composition of process flows by tying together different applications may introduce the need for compensating transaction support. This is especially the case when certain participating target applications do not leverage XA-compliant transaction processing engines. In such cases, it may be necessary to design compensating transaction pairs for every affected transaction and execute them if there is a need to reverse a particular portion of the process flow. Participating legacy and packaged target applications may need to be modified to introduce compensating transactions if they do not already implement such mechanisms.

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 that facilitate automation of business processes. The eventual 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 the process design to measure the effectiveness of process implementation 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.

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

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

Similarly it does not support the parallel execution of multiple tasks. Under such circumstances, consider the more advanced Parallel Process application pattern discussed later in this chapter.
Putting the Application pattern to use

ITSO Electronics, an electronics retailer/wholesaler, wants to integrate its retail department with its two inventory wholesale departments, namely Wholesale A and Wholesale B. Currently, these three departments have proven IT infrastructures but have no interconnectivity. ITSO Electronics wants to focus on automating the inventory replenishment process. Typically, the retail department places orders with Wholesale A. However, when the Wholesale A is unable to guarantee delivery within seven days, Wholesale B is contacted to check the anticipated delivery date. Then, the order is placed with departments that guarantee the shortest delivery date.

To meet these business process automation requirements, ITSO Electronics chooses the Serial Process application pattern. The primary driver for this selection is the need for externalization of process logic from individual applications, thus promoting flexibility and responsiveness to changing business needs.

### 3.4.7 Serial Process: Workflow variation

The Serial Process Workflow variation of the Serial Process application pattern, shown in Figure 3-8, extends the basic serial process orchestration capability by supporting human interaction for completing certain process steps.
Business and IT drivers
All the business and IT drivers listed under the Serial Process application pattern apply to this variation as well. The additional business driver for selecting this variation is the need to support human interaction and intervention within the process flow. Support for long-running transactions is another IT driver, which is often a pre-requisite for the automation of complex process flows involving human interaction.

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

- The Source Application tier is the same as for the Serial Process application pattern.
- The Serial Workflow Rules tier supports all the services provided by the serial process rules tier within the Serial Process application pattern. In addition, it supports certain tasks within the process to be routed to human actors for completion. To accomplish this, the process execution rules are augmented with task-resource relationships that define which resources are capable of performing which tasks. In this context:
  - A task is a portion of the end-to-end process.
  - Resources are capable of executing these tasks.
  - People, departments, and target applications can all be resources capable of executing a particular task.

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

Finally, this tier provides support for long-running transactions and utilizes a work-in-progress (WIP) database to store the intermediate results from the execution of different process steps until the complete execution of the end-to-end process.

- The Target Application tier is the same as for the 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.

It is recommended that people-based exception handling be implemented for the majority of the automated tasks within the process. In other words, if an
automated task reaches certain error conditions, human actors must be able to intervene and handle the exceptions.

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

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

**Limitations**
It does not support the parallel execution of multiple tasks. Under such circumstances, consider the more advanced Parallel Process application pattern and Parallel Workflow variation discussed later in this chapter.

**Putting the Application pattern to use**
ITSO Electronics, an electronics retailer/wholesaler, wants to integrate its retail department with its two wholesale departments, namely Wholesale A and Wholesale B. Currently, these three departments have proven IT infrastructures but have no interconnectivity. ITSO Electronics wants to focus on automating the inventory replenishment process. Typically, the retail department places orders with Wholesale A. However, when the Wholesale A is unable to guarantee delivery within seven days, Wholesale B is contacted to check the anticipated delivery date.

The main change from the scenario used in the Serial Process application pattern section is documented below. If both Wholesale A and Wholesale B cannot offer delivery within seven days, a retail department manager must review the shortest anticipated delivery date proposed by the wholesale department systems and approve the order before placing the same. The intent of this review is to determine whether other sourcing options must be considered.

To meet these business process automation requirements, ITSO Electronics chooses the Serial Workflow variation of Serial Process application pattern. The primary drivers for this selection include the need for externalization of process logic from the individual application, thus promoting flexibility and responsiveness to changing business needs and the need to support human interaction.
3.4.8 Parallel Process Application pattern

The Parallel Process application pattern, shown in Figure 3-9, extends the basic serial process orchestration capability provided by the Serial Process application pattern by supporting parallel (concurrent) execution of the sub-processes.

![Parallel Process application pattern](image)

**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 business driver for selecting this pattern is the need to reduce cycle time through the parallel execution of certain portions of the process flow.

**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 control parallel (concurrent) sub-processes on multiple target applications. Each sub-process may consist of a sequence of operations executed in succession on a target application. This parallelism requires that additional start and join conditions be defined for sub-processes executing in parallel. This requires sophisticated
runtime engines that can 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 implementation of parallel processes without sufficient support from the selected runtime engine would require the development of excessive custom code. The need for parallel process execution must be analyzed before middleware selection decisions are finalized.

Judicious use of parallelism is a powerful tool for reducing the cycle time of a process in the right circumstances. However, in practice, it is critical to 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. Hence, 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 ideally suited for straight-through processing where human interactions are not necessary to complete an end-to-end process. If support for human interactions are needed to complete certain process steps, consider the Workflow variation of this Application pattern.

**Putting the Application pattern to use**
ITSO Electronics, an electronics retailer/wholesaler, wants to integrate its retail department with its two wholesale departments, namely Wholesale A and Wholesale B. Currently, these three departments have proven IT infrastructures
but have no interconnectivity. ITSO Electronics wants to focus on automating the inventory replenishment process.

The main difference from the scenario used in the Serial Process and Serial Workflow application patterns sections is that here both wholesalers are queried in parallel to find who offers the shortest delivery time. In other words, Wholesale Dept. A is not considered as the defacto supplier of parts in this scenario. The order is then automatically placed with the wholesale department that offers the shortest delivery date.

To meet these business process automation requirements, ITSO Electronics chooses the Parallel Process application pattern. The primary drivers for this selection include the need for externalization of process logic from the individual application, thus promoting flexibility and responsiveness to changing business needs and addressing the need for reducing cycle time of queries by simultaneously sending enquiries to the two departments for the best delivery date.

3.4.9 Parallel Process: Workflow variation

The Parallel Process Workflow variation of the Parallel Process application pattern, shown in Figure 3-10 on page 64, extends the basic parallel process orchestration capability by supporting human interaction for completing certain process steps. This is the most sophisticated Process-focused Application pattern in the domain of Application Integration patterns.
**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 for selecting this variation is the need to support human interaction and intervention within the process flow. Support for long running transactions is another IT driver, which is often a prerequisite for the automation of complex process flows that involve human interaction.

**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 certain tasks within the process to be routed to human actors for completion. To accomplish this, the process execution rules are
augmented with task-resource relationships that define which resources are capable of performing which tasks. In this context:

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

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

Finally, this tier provides support for long-running transactions and utilizes a work-in-progress (WIP) database to store the intermediate results from the execution of different process steps until the complete execution of the end-to-end process.

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

**Guidelines for use**

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

It is recommended that people-based exception handling be implemented for all automated tasks within the process. In other words, if an automated task reaches certain error conditions, human actors must be able to intervene and handle the exceptions.

**Benefits**

The Parallel Workflow application pattern improves the flexibility and responsiveness of an organization by implementing end-to-end process flows that externalize process logic from individual applications. Further flexibility is introduced by the externalization of task-resource resolution rules.

It supports the reduction of cycle time by supporting parallel execution of portions of a process flow.

In addition, it provides a foundation for automated support for Business Process Management that enables monitoring and measurement of the effectiveness of business processes.
Limitations
Only a few middleware products are capable of supporting all the capabilities needed to realize this Application pattern. If this Application pattern is implemented using middleware products that do not support the necessary capabilities, the implementation could be very complex.

Putting the Application pattern to use
ITSO Electronics, an electronics retailer/wholesaler, wants to integrate its retail department with its two wholesale departments, namely Wholesale A and Wholesale B. Currently, these three departments have proven IT infrastructures but have no interconnectivity. ITSO Electronics wants to focus on automating the inventory replenishment process.

The main difference from the scenario used in Parallel Process application patterns sections is documented here. In this scenario, both wholesalers are queried in parallel to find who offers the shortest delivery time. The order is then automatically placed with the wholesale department that offers the shortest delivery date, unless the shortest delivery time received from the wholesale departments exceeds 10 business days. In that case, a human intervention is required by the Retail Department Manager to review the anticipated delivery date to determine other sourcing options that must be considered.

To meet these business process automation requirements, ITSO Electronics chooses the Parallel Workflow variation of the Parallel Process application pattern. The primary drivers for this selection include the need for the externalization of process logic from the individual application, thus promoting flexibility and responsiveness to changing business requirements, the need for reducing cycle time of queries by simultaneously sending enquiries to the two departments for the best delivery date, and the need for supporting human interaction during the execution of the process flow.

3.5 Data-focused Application patterns

When applications need to share information rather than coordinate processing, data-focused application integration is more appropriate than a process-focused approach. Note, however, that when the frequency of data update is extremely high (for example, when integrating an order entry system with a back-end ERP system), process integration is the best solution. When this is not the case, however, integration of (application) data repositories is handled outside of any specific application request.

In delineating Data-focused Application Integration patterns, two key environmental questions should be asked:
Is the enterprise data topology centralized or decentralized?

- **Centralized**: This integration effort will bring about centralized access to all or a subset of the enterprise data model.
- **Decentralized**: Applications will retain their isolated repositories, but now will have cohesion based on data integration.

What is the database affinity type?

- **Homogeneous**: All repositories are of the same type.
- **Multi-vender Relational**: All repositories are relational with ODBC/JDBC support for interoperability but are from different vendors.
- **Heterogeneous Structured**: Repositories are not all relational, but all have a structured layout.
- **Structured/Non-Structured**: This addresses the need to integrate non-structured (for example, free-form text) with structured data sources.

Refer to the IBM Patterns for e-business Web site for further details:


### 3.6 Previous Application Integration patterns

Table 3-3 on page 68 provides an overview of the relationship between the previous Process-focused Application Integration patterns and the revised Process-focused Application Integration patterns presented in this chapter:

- Direct Connection is retained for application coordinated requests.
- Transactional is now a quality of service. Transactionality may apply to all of these patterns, so it is applied as a quality of service rather than being a separate pattern.
- Aggregator/Broker are combined into Broker for broker-coordinated requests.
- Managed Process is split into Serial Process and Parallel Process for process-managed coordinated requests.
- The read-only versus read/write classification used with old patterns is not used with the new patterns, since:
  - For Transactional and Managed Process, read-only is not applicable.
  - For Direct Connection, the same pattern applies in both cases.
  - For Aggregator/Broker, the observed patterns are identical.
Table 3-3  Relationship to old Process-focused Application Integration patterns

<table>
<thead>
<tr>
<th></th>
<th>Old Pattern</th>
<th>New Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Information</td>
<td>Processing</td>
</tr>
<tr>
<td></td>
<td>Request (R/O)</td>
<td>Request (R/W)</td>
</tr>
<tr>
<td>Application</td>
<td>Direct Connection</td>
<td>Direct Connection</td>
</tr>
<tr>
<td>Coordinated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transactional</td>
<td>Not applicable</td>
<td>Transactional</td>
</tr>
<tr>
<td>Coordinated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Broker</td>
<td>Aggregator</td>
<td>Broker</td>
</tr>
<tr>
<td>Coordinated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>Not applicable</td>
<td>Managed Process</td>
</tr>
<tr>
<td>Managed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coordinated</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Node types and Product descriptions

This chapter provides definitions of the nodes used in the intra-enterprise Runtime patterns that are described in 5.1, “Runtime patterns” on page 84.

It also provides Product definitions for the products used in the intra-enterprise Product mappings that are described in 5.2, “Product mappings” on page 92.
4.1 Node types

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

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

**Application Server/Services**
The applications rely on services provided by their hosting server to interact with other applications. These are modeled using the Application Server/Services node. Examples of services provided by Application Server/Services could include:

- A TCP/IP pipe established using the hosting operating system
- A servlet or EJB invoked by WebSphere Application Server
- The JMS or J2EE Connector APIs provided by WebSphere
- and so on

**Connector**
Connectors provide the connectivity between two components. A connector is always present to facilitate interaction between two components.

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

- A primitive (or unmodelled) connector, represented by a simple line between components.
- A component (or modelled) connector, represented by a rectangle on a line between components.

A connector may be an adapter connector, a path connector, or both.

See also:
- “Adapter connector” on page 71
- “Path connector” on page 71
**Adapter connector**
Adapter connectors are concerned with enabling logical connectivity by bridging the gap between the context schema and protocols used by the source and target components. It is a connector that supports the transformation of data and protocols.

See also:
- “Connector” on page 70

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

See also:
- “Connector” on page 70

**Rules directory**
The rules directory contains the rules 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)
- Autonomic rules (such as priority in a shared environment)
- Security rules
- Capacity and availability rules

The rules directory may or may not exist. If it does exist, it could still be left off the Runtime pattern when analysis determines that interaction rules are not an important part of the solution, for example.

**Domain QoS providers**
The integration pattern for a domain is composed of a topology pattern and domain QoS providers. Intra-enterprise integration and inter-enterprise integration are both examples of domains. This combination of topology pattern and QoS providers is used to describe observed patterns in the domain, as follows:

\[
\text{Integration pattern} = \text{topology pattern} + \text{QoS providers}
\]

The QoS capabilities framework can be used to address the particular QoS concerns for the domain:
- Autonomic
- Availability
The domain QoS providers may or may not exist. If one does exist, it could still be left off the Runtime pattern when analysis determines that domain QoS providers are not an important part of the solution, for example.

**Local Area Network**
The Local Area Network (LAN) node is a communications network that serves users within a confined geographical area. It is made up of servers, workstations, a network operating system, and a communications link.

**Wide Area Network**
The Wide Area Network (WAN) node is a communications network that covers a wide geographic area, such as state or country.

### 4.2 Runtime product descriptions

The next step after choosing a Runtime pattern is to determine the actual products and platforms to be used. It is suggested that you make the final platform recommendation based on the following considerations:

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

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

Our sample business processes have been implemented using two Process manager implementations:

- WebSphere Process Choreographer, which is a component of WebSphere Application Server Enterprise V5.0.2
- WebSphere MQ Workflow V3.4

This chapter introduces the development tools and runtime environment used by these Process managers as they apply to this Runtime pattern.
4.2.1 WebSphere Application Server

IBM WebSphere Application Server V5.0 represents a continuation of the evolution to a single, integrated, cost-effective, Web services-enabled, J2EE server foundation for applications that offer customers the following:

- One deployment model
- One administration point
- One programming model
- One integrated application development environment

With IBM WebSphere Application Server V5.0, IBM enables customers to expand their business opportunities and productivity through a world-class infrastructure ready for e-business on demand™.

IBM WebSphere Application Server V5.0 comes in a number of editions, each offering a unique combination of features geared toward different customer needs.

**WebSphere Application Server Express V5.0**

IBM WebSphere Application Server Express V5.0 provides a combination of development tools and application servers in a single integrated package geared toward developing Web page-centric applications. It provides a simplified programming model that allows you to create new Web applications and to convert existing static applications to dynamic applications.

It provides a cost-effective starting point for businesses that want to have a presence on the Web. As your business needs grow, the WebSphere family provides a smooth migration path to higher-end WebSphere Application Server and WebSphere Studio configurations.

More information about IBM WebSphere Application Server Express V5.0 can be found at:

http://www.ibm.com/software/webservers/appserv/express

**WebSphere Application Server base V5.0**

IBM WebSphere Application Server base V5.0 provides a robust application deployment environment for single-server light production situations.
It contains a base application server that supports the full J2EE 1.3 environment. It allows a full range of enterprise integration and offers enhanced security, performance, availability, connectivity, and scalability options. Administration is done through a Web-based interface or through a scripting tool.

More information about IBM WebSphere Application Server base V5.0 can be found at:

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

**WebSphere Application Server Network Deployment V5.0**

IBM WebSphere Application Server Network Deployment V5.0 is designed to add non-programming enhancements to the features provided in the base edition. These enhancements add scalability features, allowing you to run applications on multiple servers and on multiple physical nodes.

In addition to the features included with the base edition of WebSphere Application Server you get:

- The Deployment Manager, which allows you to centrally manage a number of different application server instances and clustering for workload management and failover.
- The Network Dispatcher and Caching Proxy Server. These features provide the edge of network functions required to set up a classic DMZ in front of the application server.
- A private UDDI registry that facilitates easier deployment of internal Web services applications and a Web Services Gateway.

More information about IBM WebSphere Application Server Network Deployment V5.0 can be found at:

http://www.ibm.com/software/webservers/appserv/was/network/

**WebSphere Application Server Enterprise V5.0**

IBM WebSphere Application Server Enterprise V5.0 provides the features in IBM WebSphere Application Server Network Deployment V5.0 plus programming model extensions for sophisticated application designs.

It offers advanced capabilities such as advanced application adapters, application workflow composition and choreography, extended messaging, dynamic rules-based application adaptability, internationalization, and asynchronous processing.

**Note:** The WebSphere Application Server Enterprise Process Choreographer runtime is part of WebSphere Application Server Enterprise V5.0.
WebSphere MQ is bundled with the package (except on z/OS®).

More information about IBM WebSphere Application Server Enterprise V5.0 can be found at:

http://www.ibm.com/software/webservers/appserv/enterprise/

### 4.2.2 WebSphere Application Server Enterprise Process Choreographer

IBM WebSphere Application Server Enterprise Process Choreographer provides business process support within the WebSphere Application Server. WebSphere Application Server Enterprise V5.0 includes the runtime environment for IBM WebSphere Process Choreographer V5.0. WebSphere Process Choreographer is not in itself a separate product. The next release of WebSphere Process Choreographer is part of WebSphere Business Integration Server Foundation V5.1. We do not focus on this release of WebSphere Process Choreographer in this redbook.

**Note:** Throughout this book we refer to this component simply as WebSphere Process Choreographer.

WebSphere Process Choreographer can be used for the choreography of all kinds of business processes or flows. The business processes that are implemented in an enterprise typically require a mixture of human and IT resources. The types of business processes can vary greatly, ranging from Web services or Web page navigation to business transaction support. Business processes can be automatic, recoverable processes, or processes that require human interaction. With WebSphere Process Choreographer, you can combine business process technology with any other service offered by the open J2EE architecture.

WebSphere Process Choreographer includes the following features:

- **Human interaction**
  
  This offers support for including activities that require a person to perform a task as a step in an automated business process. Specialized staff support allows the dynamic assignment of responsibilities based on existing organizational definitions. Worklists can be created to let the designated recipient know that action is required.

- **Event triggering**
  
  This offers support for allowing asynchronous events such as Web services or human interactions to be included as part of business processes. Events can be used to trigger the start of a business process, or a business process...
can be configured to stop and wait for an external event to occur before resuming the process.

► **Compensation pairs**

This provides transaction rollback like support for long-running, loosely coupled business processes that cannot be undone automatically by the application server. For example, the compensating transaction for an order that has already started manufacturing might be to put the complete item into inventory (rather than disassembling the item). Compensation pairs allow you to visually define the associated undo service for each step in your business process.

► **Flexible workflow design**

This provides developers with the ability to design workflows using a top-down, bottom-up, or meet-in-the-middle approach. Using top-down, developers can create skeleton processes that choreograph the sequence of events in a workflow without worrying about the underlying implementation. Building from the bottom-up, developers first create the individual components and then use them as building blocks to define a workflow. Meet-in-the-middle offers the flexibility of using both approaches at the same time.

The result is faster development of new applications, improved consistency, and lower costs through the reuse of existing IT investments.

More information about IBM WebSphere Process Choreographer V5.0 can be found at:

http://www.ibm.com/developerworks/websphere/zones/was/wpc.html

### 4.2.3 WebSphere MQ

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 will enable it to (1) receive (get) messages from local queues or (2) 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 (1) automatically balance the workload across available resources and (2) 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 (such as MQI, AMI, and JMS) that provide support for several programming languages as well as the point-to-point and publish/subscribe communication models. Beside 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 name a few.

More information can be found at the IBM WebSphere MQ Web site:
http://www.ibm.com/software/ts/mqseries

4.2.4 WebSphere MQ Workflow

IBM WebSphere MQ Workflow is an IBM product aimed at helping organizations to automate their business processes. By introducing a process manager such as WebSphere MQ Workflow, the productivity of an organization can be improved. WebSphere MQ Workflow electronically enables an organization to deliver the right task, to the right person, at the right time, with the right data and the right tools. By automating the business processes, not only does WebSphere MQ Workflow help to optimize the organization’s resources, but it can reduce cycle time and associated costs. Using a process manager also decouples the process logic from the business logic and enables the possibility that a change in either the process logic or the business does not require a change in the other.

WebSphere MQ Workflow is best suited for automating process-centric business processes, as its strength lies in people-based workflows. WebSphere MQ Workflow supports many different staff delegation algorithms and it can also drive system integration via the implementation of one or more User Program Execution Servers (UPES). A UPES activity within a process sends a WebSphere MQ XML-formatted message to a user-defined WebSphere MQ queue. The UPES is a custom program that receives this message and performs the request, constructs an XML response that the WebSphere MQ Workflow
server will understand, and sends the message back to the server. Since a
UPES request is delivered by WebSphere MQ, this means that the UPES can
run on any of the many operating system platforms that WebSphere MQ runs on.
Note that all of the UPES activities in our scenarios invoke Web services, so a
broker such as WBI Message Broker is not required.

WebSphere MQ Workflow can help to improve customer satisfaction when the
audit trail is used to provide status updates. The WebSphere MQ Workflow audit
trail produces either an entry in the audit trail table in DB2 or an WebSphere MQ
message in XML format for each workflow-relevant event that occurs. This audit
trail can provide enormous value to the business since process tracking can be
made available, thereby allowing any interested parties in the process to
determine where the process is. WebSphere MQ Workflow can be used with
WebSphere Business Integration Workbench to provide real-time process
tracking. Also, real production metrics from the audit trail can be used within
WebSphere Business Integration Monitor to analyze an organization’s
processes. Using these products together provides any customer with the
information necessary to achieve continuous process improvement.

WebSphere MQ Workflow is built upon proven IBM technology. The
communication layer is built upon WebSphere MQ, while the database can use
DB2. The servers can run on many OS platforms, including z/OS, providing
flexibility for an organization’s environment. The operational model that can be
developed for a WebSphere MQ Workflow implementation can be designed for
highly-available environments. The staff definitions kept within WebSphere MQ
Workflow can be managed via the LDAP Bridge tool provided in the product with
version 3.4. Using the LDAP bridge eliminates the need for managing the staff
manually.

More information can be found at the IBM WebSphere MQ Workflow Web site:
http://www.ibm.com/software/integration/wmqwf/

4.3 Development product descriptions

In order to build business process applications to run in a Process manager, a
development tool must be used. This section describes these development tools.

4.3.1 WebSphere Studio Application Developer Integration Edition

IBM WebSphere Studio Application Developer Integration Edition builds on the
complete set of functionality offered by WebSphere Studio Application Developer
to deliver a next-generation development environment designed to deliver
on-demand e-business applications by simplifying build-to-integrate tasks,
accelerating large-scale application development, and enabling real-time application flexibility for applications that deploy to WebSphere Application Server Enterprise, V5.

**Note:** WebSphere Studio Application Developer Integration Edition V5 is used to develop processes for WebSphere Process Choreographer.

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

More information can be found at the IBM web site:

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

### 4.3.2 IBM Rational® Rose XDE™ Modeler

IBM Rational Rose® XDE Modeler enables architects and designers to practice model-driven development with the Unified Modeling Language (UML). Such users can produce platform independent models of software architecture, business needs, reusable assets and management-level communication. Industry standard UML support and a powerful pattern engine allow users to create a semantically rich application architecture that meets business needs and is readily understood by the development team.

Architects and designers can use Rational Rose XDE Modeler’s multi-model support to separate concerns of analysis, architecture, design and implementation. Developers can use architectural models and patterns as the basis for implementation, thereby accelerating the development of applications to conform to their architecture. Further, features such as free-form modeling, Web publishing and reporting allow users to share architecture and designs with all stakeholders, whether or not the stakeholders use Rational Rose XDE Modeler.

More information can be found here:

http://www.ibm.com/software/awdtools/developer/modeler/features/
4.3.3 WebSphere Business Integration Workbench

The WebSphere Business Integration Workbench is a modeling tool that is used to design and test business processes. The Workbench can be used by IT or business individuals with its shared workspace environment.

The Workbench allows a business individual can produce a Line of Visibility (LOV) chart for a process that an IT person can refine into process model. The Workbench can also define organizations, business rules, and goals.

A significant feature of the Workbench is that the user can use process simulation to analyze where potential bottlenecks may occur within a process. The tool also features what-if analysis, the ability to analyze cost and time performance of business processes, and reports that measure and compare process performance.

WebSphere Business Integration Workbench is integrated with other WebSphere Business Integration tools, including WebSphere MQ Workflow. Thus, processes that are created in the Workbench can be exported in FDL and imported into the WebSphere MQ Workflow runtime database.

More information can be found at the IBM Web site:


4.3.4 WebSphere MQ Workflow Buildtime

The Buildtime client is the component used by the process modeler to develop the process models using a graphical user interface to depict the business processes. Either WebSphere Business Integration Workbench or WebSphere MQ Workflow Buildtime can be used to model business processes. The WebSphere MQ Workflow Buildtime is a client contained within WebSphere MQ Workflow. The modeler documents not only the control flow within the process, but also the data flow between the activities.

Before the process modeling can be completed, the programs that are invoked at each activity in the process need to be registered. Additionally, data structures are defined that contain all of the workflow-relevant process data. The roles, organizations, and staff are also defined by the modeler in Buildtime.

Buildtime can use a DB2 database to store all of the definitions that are defined by the modeler. This database is separate from the WebSphere MQ Workflow runtime database and may or may not be present on the same database server. In a single modeler environment, the Buildtime database typically resides on the modeler's workstation. In multi-modeler environments or in larger organizations,
the Buildtime database is located centrally and the interested parties have a Buildtime client that connects to this remote database.

Once the process definitions are completed, the objects are exported to a file as Flow Definition Language (FDL). This file is then moved to a database client of the WebSphere MQ Workflow runtime server (or even the server itself) and imported. Once the file is successfully imported, the new objects are ready for use in the WebSphere MQ Workflow runtime database.

More information can be found at the IBM Web site:

Runtime patterns and Product mappings

This section describes the Runtime patterns and Product mappings for the Serial and Parallel Process patterns.

To understand the Runtime patterns and Product mappings in this chapter, you will need to review the node and product definitions in Chapter 4, “Node types and Product descriptions” on page 69.
5.1 Runtime patterns

After determining the Application pattern to use, the next step is to choose the Runtime pattern that most closely matches the requirements of the application. Runtime patterns are used to define the logical middleware structure supporting the Application pattern. In other words, Runtime patterns describe the logical architecture required to implement an Application pattern. Runtime patterns depict the major middleware nodes, their roles, and the interfaces between these nodes.

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

Note: This redbook covers Runtime patterns for Serial and Parallel Process application patterns and their variations in this section. The Direct Connection runtime patterns and Broker runtime patterns are covered in other Redbooks.

5.1.1 Serial Process Runtime pattern

The Runtime pattern shown in Figure 5-1 on page 85 represents a basic topology for the implementation of the Serial Process Application pattern. It can be further enhanced by clustering the key nodes in order to improve the availability characteristics.
This basic topology leverages the following nodes with their associated responsibilities:

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

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

- The App Server/Services node and the Process Manager
  This allows a source application to communicate with a Process Manager. Process Managers usually 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 usually offer multiple connector types.

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

Repository node
This 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 also 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 DBMS and flat file) for the different data types. In some cases, non-persistent storage is used to store the Immediate Results.

App Server/Service nodes
These nodes run either the source or target application.

5.1.2 Serial Process: Workflow variation Runtime pattern
The Runtime pattern shown in Figure 5-2 on page 87 represents a basic topology for the implementation of the Serial Process: Workflow variation. It can be further enhanced by clustering the key nodes in order to improve the availability characteristics.
Figure 5-2  Runtime pattern for Serial Process: Workflow variation

This basic topology leverages the following nodes with the associated responsibilities:

- **Workflow Manager node**
  This 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. The node maintains 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 the 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 the humans interact within the end-to-end workflow.

► **Connector nodes**
A connector node is used to provide connectivity between the following:
- The App Server/Services node and the Process Manager
  This allows a source application to communicate with a Process Manager. Process Managers usually 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 usually offer multiple connector types.

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

► **Repository node**
This 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 also 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 DBMS and flat file) for the different data types.

► **App Server/Service nodes**
These nodes run either the source or target application.

### 5.1.3 Parallel Process Runtime pattern

The Runtime pattern shown in Figure 5-3 on page 89 represents a basic topology for the implementation of the Parallel Process Application pattern. It can be further enhanced by clustering the key nodes in order to improve the availability characteristics.
Figure 5-3  Runtime pattern for Parallel Process application pattern

This basic topology leverages the following nodes with the associated responsibilities:

- **Process Manager node**

  This node contains the process flow execution engine. 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 organizational boundaries within an enterprise. The node maintains 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 their associated connectors.
▶ **Connector nodes**
A connector node is used to provide connectivity between the following:
- The App Server/Services node and the Process Manager
  This allows a source application to communicate with a Process Manager. Process Managers usually 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 usually offer multiple connector types.

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

▶ **Repository node**
This 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 also 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 DBMS and flat file) for the different data types.

▶ **App Server/Service nodes**
These nodes run either the source or target application.

### 5.1.4 Parallel Process: Workflow variation Runtime pattern

The Runtime pattern shown in Figure 5-4 on page 91 represents a basic topology for the implementation of the Parallel Process: Workflow variation. It can be further enhanced by clustering the key nodes in order to improve the availability characteristics.
This basic topology leverages the following nodes with the associated responsibilities:

- **Workflow Manager node**
  This 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. The node maintains 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 the target application as necessary through the associated connectors. Finally, when a particular task requires human interaction, this node creates a work item and identifies a particular person or a department is 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 the humans interact within the end-to-end workflow.

- **Connector nodes**
  A connector node is used to provide connectivity between the following:
  - The App Server/Services node and the Process Manager
    This allows a source application to communicate with a Process Manager. Process Managers usually 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 usually offer multiple connector types.

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

- **Repository node**
  This 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 also 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 DBMS and flat file) for the different data types.

- **App Server/Service nodes**
  These nodes run either the source or target application.

## 5.2 Product mappings

The next step after choosing a Runtime pattern is to determine the actual products and platforms to be used. It is suggested that you make the final platform recommendation based on the QoS requirements that have been gathered during the analysis process, including due consideration of Quality of Product/Architecture/Provider type considerations such as:
Existing systems and platform investments
Customer and developer skills available

The platform selected should fit into the customer’s environment and ensure qualities of service such as availability and performance so that the solution can grow along with the e-business.

This chapter provides an overview of the products used in the scenarios as they apply to the Serial and Parallel Process Runtime patterns.

**Note:** The Product mappings documented in this section for Serial Process and Parallel Process Application patterns and their workflow variations primarily focus on the key nodes (the Process Manager and Workflow Manager nodes) in the appropriate Runtime patterns. The Product mappings shown for the nodes associated with Source and Target application tiers (the App Server/Services and Adapter nodes) are only indicative set of Product mappings that represent the products we used in the scenarios of this redbook.

A number of Product mapping variations can be used for the connectivity between the Process Manager and/or Workflow Manager nodes and the Source and Target applications. The specific Product mapping for the Connector nodes depends on the technology used by the Source and Target applications and the QoS required by the connection.

Please refer to 4.2, “Runtime product descriptions” on page 72 for descriptions of the products used in these Product mappings.

**Note:** We developed these Product mappings from our sample scenario on the Windows 2000 operating system. There are a number of other platform options available as IBM WebSphere products that run on a wide range of platforms (for example Windows 2000, Linux, pSeries®, iSeries™ and so on).

### 5.2.1 Serial Process Product mappings

This section presents Product mappings for the Serial Process pattern using the following:

- WebSphere Process Choreographer
- WebSphere MQ Workflow

The Serial Process pattern is implemented in a scenario using both of these Product mappings in Chapter 9, “Creating serial processes” on page 149.
WebSphere Process Choreographer

Figure 5-5 shows the Product mapping using the WebSphere Process Choreographer for the Serial Process pattern.

In this Product mapping, the Source Application running on WebSphere Application Server Enterprise invokes the automated process instance implemented by the Process Manager node using the Web Services Invocation Framework (WSIF). Directory and Security services and the Repository run on products that are external to the Process Manager.

The Process Manager invokes the services provided by the Target Applications running on WebSphere Application Server and WebSphere Application Server Enterprise through WSIF providers that create RMI over IIOP and SOAP over HTTP messages. Both Target Applications are implemented as Enterprise JavaBeans.

One of the target applications shares the same WebSphere Application Server Enterprise server as the Process Manager. Alternatively, this Enterprise bean could run on an external non-Enterprise WebSphere Application Server.
WebSphere MQ Workflow

Figure 5-6, shows the Product mapping using WebSphere MQ Workflow for the Serial Process pattern.

In this Product mapping, the Source Application is the WebSphere MQ Workflow Web client running on WebSphere Application Server. It sends an MQ message to the Process Manager node.

The Process Manager invokes the services provided by the Target Applications running on WebSphere Application Server using the Web Services Toolkit connector. Both Target Applications are implemented as Enterprise JavaBeans.

The connectivity between the Directory & Security Services and the Process Manager has been illustrated to be batch connectivity using dotted lines, since the WebSphere MQ Workflow best practice is leverage the internal directory that ships with the product rather than the external LDAP based user directory.
5.2.2 Serial Process: Workflow variation Product mappings

This section presents Product mappings for the Serial Process: Workflow variation using the following:

- WebSphere Process Choreographer
- WebSphere MQ Workflow

**WebSphere Process Choreographer**

Figure 5-7, shows the Product mapping using the WebSphere Process Choreographer for the Serial Process: Workflow variation.

![Diagram of Serial Process: Workflow variation: WebSphere Process Choreographer Product mapping]

This Product mapping shares many of the same characteristics as the Serial Process WebSphere Process Choreographer Product mapping. The Process Manager node is replaced with the Workflow Manager node, although this node is still implemented by WebSphere Process Choreographer.
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. WebSphere Process Choreographer offers several such adapters, including the LDAP staff plugin provider.

**WebSphere MQ Workflow**

Figure 5-8, shows the Product mapping using WebSphere MQ Workflow for the Serial Process: Workflow variation.

This Product mapping shares many of the same characteristics as the Serial Process WebSphere MQ Workflow Product mapping. The Process Manager node is replaced with the Workflow Manager node, although this node is still implemented by WebSphere MQ Workflow.

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. This is an adapter internal to WebSphere MQ Workflow, although it could be implemented on an LDAP server using the LDAP Bridge.
5.2.3 Parallel Process Product mappings

This section presents Product mappings for the Parallel Process pattern using the following:

- WebSphere Process Choreographer
- WebSphere MQ Workflow

The Parallel Process pattern is implemented in a scenario using both of these product mappings in Chapter 10, “Creating parallel processes” on page 245.

The Parallel Process pattern is implemented with a Process Manager acting as the Target Application in Chapter 13, “Process manager interoperability” on page 357.

WebSphere Process Choreographer

Figure 5-9, shows the Product mapping using the WebSphere Process Choreographer for the Serial Process pattern.

Figure 5-9  Parallel Process: WebSphere Process Choreographer Product mapping
In this Product mapping, the Source Application running on WebSphere Application Server Enterprise invokes the automated process instance implemented by the Process Manager node using the Web Services Invocation Framework (WSIF). Directory and Security services and the Repository run on products that are external to the Process Manager.

The Process Manager invokes the services provided by the Target Applications running on WebSphere Application Server and WebSphere Application Server Enterprise through WSIF providers that create RMI over IIOP and SOAP over HTTP messages. Both Target Applications are implemented as Enterprise JavaBeans. The Target Applications are invoked in parallel.

**WebSphere MQ Workflow**

Figure 5-10, shows the Product mapping using WebSphere MQ Workflow for the Parallel Process pattern.

In this Product mapping, the Source Application is the WebSphere MQ Workflow Web client running on WebSphere Application Server. It sends an MQ message to the Process Manager node.
The Process Manager invokes the services provided by the Target Applications running on WebSphere Application Server using the Web Services Toolkit connector. Both Target Applications are implemented as Enterprise JavaBeans. The Target Applications are invoked in parallel.

The connectivity between the Directory and Security Services and the Process Manager has been illustrated to be batch connectivity using dotted lines, since the WebSphere MQ Workflow best practice is to leverage the internal directory that ships with the product rather than the external LDAP based user directory.

5.2.4 Parallel Process: Workflow variation Product mappings

This section presents Product mappings for the Parallel Process: Workflow variation using the following:

- WebSphere Process Choreographer
- WebSphere MQ Workflow

The Parallel Process: Workflow variation pattern is implemented in scenarios using both of these Product mappings: Chapter 11, “Creating processes with human interaction” on page 267 and Chapter 12, “Creating processes with events and compensation” on page 315.

WebSphere Process Choreographer

Figure 5-11 on page 101, shows the Product mapping using the WebSphere Process Choreographer for the Parallel Process: Workflow variation.
This Product mapping shares many of the same characteristics as the Serial Process WebSphere Process Choreographer Product mapping. The Process Manager node is replaced with the Workflow Manager node, although this node is still implemented by WebSphere Process Choreographer.

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. WebSphere Process Choreographer offers several such adapters, including the LDAP staff plugin provider.

**WebSphere MQ Workflow**

Figure 5-12 on page 102, shows the Product mapping using WebSphere MQ Workflow for the Parallel Process: Workflow variation.
This Product mapping shares many of the same characteristics as the Serial Process WebSphere MQ Workflow Product mapping. The Process Manager node is replaced with the Workflow Manager node, although this node is still implemented by WebSphere MQ Workflow.

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. This is an adapter internal to WebSphere MQ Workflow, although it could be implemented on an LDAP server using the LDAP Bridge.
Part 2 presents guidelines on (1) applying the Patterns approach to a sample business scenario and (2) selecting application integration technologies. It also describes the capabilities of two process managers used in the Serial and Parallel Process Product mappings.

Included in Part 2 are the following chapters:

- Chapter 6, “Business scenarios used in this book” on page 105
- Chapter 7, “Technology options” on page 119
- Chapter 8, “Process manager capabilities” on page 131
Business scenarios used in this book

To demonstrate the use of the Patterns for e-business presented in this redbook, we created the business scenarios outlined in this chapter for our imaginary customer, ITSO Electronics. Of course, simple business scenarios like these do not exist in reality, but they help us to explain how the Patterns for e-business approach can be applied.
6.1 Customer overview

This section provides some background information about our imaginary client, ITSO Electronics, including the following:

- Business profile and goals
- Existing environment
- Non-functional requirements or QoS requirements

6.1.1 Business profile

ITSO Electronics is a 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.

6.1.2 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. This should be achieved by automating the retail stock replenishment order process.

- Increase customer satisfaction by reducing latency between the retail ordering process and the wholesale order fulfillment process, thereby decreasing the likelihood of an item being out-of-stock.

After the internal processes have been integrated, ITSO Electronics plans to enable their external resellers to place stock replenishment orders with the wholesale department. This requirement represents an inter-enterprise scenario, and it is not addressed in this redbook.

6.1.3 Existing environment

This section describes the existing environment at ITSO Electronics.

Business perspective

From the business perspective, the existing environment includes:

- The wholesale ordering process shown in Figure 6-1 on page 107. This process uses the wholesale department's inventory business process, which
is accessed through manual channels in the form of paper-based forms and manual processes.

- The internal retail department and a number of external reseller business partners, who use the wholesale department to place their stock replenishment orders.

![Figure 6-1  ITSO Electronics wholesale ordering process flow](image)

**IT perspective**

The existing IT environment, shown in Figure 6-2 on page 108, includes:

- An existing wholesale inventory system: An important legacy system that implements the core business processes of the wholesale department
- An existing retail ordering system: The system used by retail staff, recently upgraded to a Self-Service browser-based J2EE application
- External resellers that own heterogeneous IT infrastructures
- A limited existing application integration infrastructure
6.1.4 Non-functional requirements

ITSO Electronics requires that all solutions provide a standard QoS set. 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.

► 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.
  – Non-repudiation 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. Of course, you would do so in a real-world implementation to ensure that the delivered solution meets the demands of the organization.

6.2 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 items sold 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. A primary business goal is to minimize the loss of sales due to items being out of stock.

Selecting a Business/Integration pattern
In 3.2.1, “Business and IT drivers” on page 35, 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 integrating the existing retail business system with the existing wholesale business system. The pattern would integrate the retail order information with the wholesale inventory information, eliminating the lag and providing an up-to-date inventory.

The Application Integration pattern can be applied in our scenario, as shown in Figure 6-3 on page 110.
6.2.1 Stage One: Internal ordering on demand

In the first stage of the internal implementation, ITSO Electronics wants to integrate the retail system and the wholesale system. The primary goal is to integrate the internal retail ordering system with the internal wholesale system to automate the process of retail stock replenishment orders. The retail group will be able to place orders without the manual effort of determining from which wholesale location to order. The integrated system will meet the business requirement for order confirmation and the requirement for an audit trail.

For Stage One of our business scenario, we can identify two actors:

- The retail system
- The wholesale system

We can also identify a use case:

- Place Order

**Actors**

Table 6-1 on page 111 provides details on the retail system actor.
Table 6-1  Retail system actor details

<table>
<thead>
<tr>
<th>Actor name</th>
<th>Retail system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief description</td>
<td>The retail system implements the retail ordering business process.</td>
</tr>
<tr>
<td>Status</td>
<td>Primary</td>
</tr>
<tr>
<td>Relationships</td>
<td></td>
</tr>
<tr>
<td>Associations to use cases</td>
<td>001 Place Order</td>
</tr>
</tbody>
</table>

Table 6-2 provides details on the wholesale system actor.

Table 6-2  Wholesale system actor details

<table>
<thead>
<tr>
<th>Actor name</th>
<th>Wholesale system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief description</td>
<td>The wholesale system implements the order fulfillment business process.</td>
</tr>
<tr>
<td>Status</td>
<td>Primary</td>
</tr>
<tr>
<td>Relationships</td>
<td></td>
</tr>
<tr>
<td>Associations to use cases</td>
<td>001 Place Order</td>
</tr>
</tbody>
</table>

Use case 001: Place Order
Table 6-3 provides details on the Place Order use case.

Table 6-3  Use case 001: Place order

<table>
<thead>
<tr>
<th>Use case name</th>
<th>001 Place Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject area</td>
<td>Wholesale ordering</td>
</tr>
<tr>
<td>Business event</td>
<td>An item sold by the retail division needs to be replaced from the wholesale inventory.</td>
</tr>
<tr>
<td>Actors</td>
<td>Retail system, wholesale systems</td>
</tr>
<tr>
<td>Use case overview</td>
<td>The retail system places a replenishment order for a sold part with a wholesale system.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The retail system supplies a part number and the quantity to be ordered.</td>
</tr>
<tr>
<td>Termination outcome 1</td>
<td>The retail inventory system records the confirmation number of the order, the wholesale system where the order was placed, and the expected delivery date.</td>
</tr>
<tr>
<td>Notes</td>
<td></td>
</tr>
</tbody>
</table>
The Stage One use case model is shown in Figure 6-4.

![Figure 6-4 Stage One use case model](image)

**Selecting an Application pattern**

In Figure 3-1 on page 41 and Figure 3-2 on page 42, the following drivers are listed for selecting the Process-focused 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 legacy investment
- Enable back-end application integration
- Minimize application complexity
- Minimize enterprise complexity
- Improve Maintainability
- Improve flexibility by externalizing process logic from application logic

In addition, the following driver is listed for selecting the Process-focused Application Integration::Parallel Process Application pattern:

- Reduce cycle time through parallel execution of portions of a process flow.
These drivers are a good match for the Stage One scenario. We can implement this scenario using either the Serial or Parallel Process Application pattern. If we were integrating a process that would benefit from human interaction, the Workflow variation for the Serial or Parallel Process Application pattern would be required.

The Serial Process Application pattern is applied to the Stage One scenario. Two Serial Process implementations are described; one for WebSphere Process Choreographer and one for WebSphere MQ Workflow. This is described in following chapter:

Chapter 9, “Creating serial processes” on page 149

The Parallel Process Application pattern is also applied to the Stage One scenario, again using two implementations. This is described in the following chapter:

Chapter 10, “Creating parallel processes” on page 245

The Parallel Process Application pattern is additionally applied to the Stage One scenario, using two Process Managers in the same implementation. The primary reason for using two Process Managers is as a first step in migrating from one Process Manager to another. This is described in the following chapter:

Chapter 13, “Process manager interoperability” on page 357

6.2.2 Stage Two: Internal ordering on demand with approval workflow

In the second stage of the internal implementation, ITSO Electronics wants to further integrate its internal retail and wholesale ordering processes. They now require the inclusion of a human interaction activity for order approval in exception situations. To improve customer satisfaction, the retail department wants to be able to quickly authorize the fulfillment of orders in situations when neither wholesaler can provide a delivery span of less than or equal to seven days but the item should still be ordered to meet the customer commitments of the retail department.

For Stage Two of our business scenario we identify an additional actor:

- Workflow Manager

We can also identify an additional use case:

- Place Order with Approval Workflow

Actors

Table 6-4 on page 114 provides details on the Workflow Manager actor.
Table 6-4  Workflow manager actor details

<table>
<thead>
<tr>
<th>Actor name</th>
<th>Workflow manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief description</td>
<td>A manager from the Retail department who is authorized to approve or reject orders.</td>
</tr>
<tr>
<td>Status</td>
<td>Primary</td>
</tr>
<tr>
<td>Relationships</td>
<td></td>
</tr>
<tr>
<td>Associations to use cases</td>
<td>002 Place Order with Approval Workflow</td>
</tr>
</tbody>
</table>

Use case 002: Place Order with Approval Workflow

Table 6-5 provides details on the Place Order with Approval Workflow use case.

Table 6-5  Use case 002: Place order with approval workflow

<table>
<thead>
<tr>
<th>Use case name</th>
<th>002 Place Order with Approval Workflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject area</td>
<td>Wholesale ordering</td>
</tr>
<tr>
<td>Business event</td>
<td>An item sold by the retail division needs to be replaced from the wholesale inventory.</td>
</tr>
<tr>
<td>Actors</td>
<td>Retail system, wholesale systems, workflow manager</td>
</tr>
<tr>
<td>Use case overview</td>
<td>The retail system places a replenishment order for a sold part with a wholesale system. If an order can not be placed with either wholesaler within the seven day business requirement, an order approval process is triggered.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The retail system supplies a part number and the quantity to be ordered.</td>
</tr>
<tr>
<td>Termination outcome 1</td>
<td>The retail inventory system records the confirmation number of the order, the wholesale system where the order was placed, and the expected delivery date.</td>
</tr>
<tr>
<td>Notes</td>
<td></td>
</tr>
</tbody>
</table>

The Stage Two use case model is shown in Figure 6-5 on page 115.
Selecting an Application pattern

Figure 3-1 on page 41 and Figure 3-2 on page 42 lists the drivers for selecting the Workflow variation of the Process-focused Application Integration::Parallel Process Application pattern. In addition to the drivers listed in Stage One, the following drivers are defined:

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

All drivers apply to the Stage Two scenario. Two Parallel Process Workflow variation implementations are described for this scenario: one for WebSphere Process Choreographer and one for WebSphere MQ Workflow. This is described in following chapter:

Chapter 11, “Creating processes with human interaction” on page 267

6.2.3 Stage Three: Ordering on demand with multiple wholesalers

In the third stage of the internal implementation, ITSO Electronics wants to extend its internal retail and wholesale ordering processes. It now requires the extension of the retail stock replenishment order process to order a part from two wholesalers simultaneously. When these parts are combined, they form a single part for ITSO Electronics to resell. For example, ABC may require a toy from one wholesale system and batteries to accompany it from the second wholesale system. If either part cannot not be ordered, the entire order is automatically
cancelled. Additionally, ABC now requires the ordering process to span the entire
duration of the order, from when the order is placed with the wholesale systems
to when the order arrives by the receiving department.

For Stage Three of our business scenario, we identify an additional actor:
  - Receiving manager

We can also identify an additional *use case*:
  - Place Order with Multiple Wholesalers

**Actors**

Table 6-6 provides details on the Receiving Manager.

*Table 6-6 Receiving manager actor details*

<table>
<thead>
<tr>
<th>Actor name</th>
<th>Receiving manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief description</td>
<td>A manager from the receiving manager of the retail department who is responsible for notifying when an order has been received by the receiving department.</td>
</tr>
<tr>
<td>Status</td>
<td>Primary</td>
</tr>
<tr>
<td>Relationships</td>
<td></td>
</tr>
<tr>
<td>Associations to use cases</td>
<td>003 Place Order with Multiple Wholesalers</td>
</tr>
</tbody>
</table>

**Use case 003: Place order with multiple wholesalers**

Table 6-7 provides details on the place order with multiple wholesalers use case.

*Table 6-7 Use case 003: Place order with multiple wholesalers*

<table>
<thead>
<tr>
<th>Use case name</th>
<th>003 Place Order with Multiple Wholesalers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject area</td>
<td>Wholesale ordering</td>
</tr>
<tr>
<td>Business event</td>
<td>An item sold by the retail division needs to be replaced from two individual parts from two wholesale systems. The process should span the entire duration of the order.</td>
</tr>
<tr>
<td>Actors</td>
<td>Retail system, wholesale systems, workflow manager, receiving manager</td>
</tr>
<tr>
<td>Use case overview</td>
<td>The retail system places an order for one part with wholesale system A and an order for a related part with wholesale system B. If either part cannot be ordered, the entire order should be compensated. The business process must also span the entire length of the order, requiring the receiving manager to notify when both parts arrive.</td>
</tr>
</tbody>
</table>
The Stage Two use case model is shown in Figure 6-6.

<table>
<thead>
<tr>
<th>Use case name</th>
<th>003 Place Order with Multiple Wholesalers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>The retail system supplies two part numbers.</td>
</tr>
<tr>
<td>Termination outcome 1</td>
<td>The retail inventory system records the confirmation number of both orders.</td>
</tr>
<tr>
<td>Notes</td>
<td></td>
</tr>
</tbody>
</table>

The Stage Two use case model is shown in Figure 6-6.

![Figure 6-6 Stage III use case model](image)

Selecting an Application pattern

Figure 3-1 on page 41 and Figure 3-2 on page 42 list the drivers for selecting the Workflow variation of the Process-focused Application Integration::Parallel Process Application pattern.

All drivers apply to the Stage Three scenario. Two Parallel Process Workflow variation implementations are described for this scenario; one for WebSphere Process Choreographer and one for WebSphere MQ Workflow. This is described in following chapter:

   Chapter 12, “Creating processes with events and compensation” on page 315
Technology options

In this chapter we consider some of the key technologies relevant to the implementation of the Process-focused Application Integration pattern. The following technology areas are discussed:

- **Flow languages**
  Business processes are described by flow languages. This section discusses three of these languages and describes which Process Managers use them.

- **Web services**
  Web services are a common component of any new business process activity implementation. This section introduces some of the fundamental issues associated with Web services.

- **Java Message Service**
  Messaging is a key component of most Process Managers. This section introduces the Java messaging standard.
7.1 Flow languages

This section discusses the business process languages relevant to the WebSphere Process Choreographer and WebSphere MQ Workflow Process Managers:

- Flow Definition Language (used by WebSphere MQ Workflow)
- Flow Definition Markup Language (used by WebSphere Process Choreographer)
- Business Process Execution Language for Web Services

7.1.1 Flow Definition Language

Flow Definition Language (FDL) is a text-based process definition format that provides an interface between (1) process modeling and development tools and (2) the process engine of WebSphere MQ Workflow.

An FDL document describes the make-up of a process flow. Although FDL theoretically could be written by hand, it is commonly generated using WebSphere MQ Workflow Buildtime. As FDL became an established standard, other process modeling and development tools came on the market that were capable of exporting FDL and thus are suitable for use with WebSphere MQ Workflow. This includes third parties such as Holosofx®, who have since been acquired by IBM. The Holosofx product is now marketed as IBM WebSphere Business Integration Workbench. It offers facilities to generate FDL.

To see a sample of FDL code, consult the scenario chapters in this redbook.

7.1.2 Flow Definition Markup Language

Despite its similar-sounding name to FDL, Flow Definition Markup Language (FDML) is actually based on the Web Services Flow Language (WSFL) standard introduced by IBM. FDML introduces extensions to the WSFL standard and is the flow language used to describe a WebSphere Process Choreographer process.

WebSphere Studio Application Developer Integration Edition uses FDML to describe WebSphere Process Choreographer business processes.

FDML describes the behavior of the process. It is an XML-based format. An FDML file describes elements of a process such as the flow model, activities, control links, compensation, messages, and variables.

A sample of an FDML file is shown in Example 7-1 on page 121.
Example 7-1  Sample FDML file

```
wf:flowModel autoDelete="true" canRunInterrupted="true"
canRunSynchronous="false" name="EventCompensationProcess"
requiresCompensationSphere="true" validFrom="2003-01-01T12:00:00">
  <wf:messageType name="MessageType_11" typeSystem="WSDL2">
    <service:messageRef name="processOrderInput"
      namespace="http://process.itso.ibm.com/">
      <service:import
        location="file://com/ibm/itso/process/EventCompensationProcessInterface.wsdl"/>
    </service:messageRef>
  </wf:messageType>
```

For more information about FDML, please refer to the IBM Redbook *WebSphere V5.0.2 Application Developer V5.1 Web Services Handbook*, SG24-6891.

### 7.1.3 Business Process Execution Language for Web Services

Business Process Execution Language for Web Services (BPEL4WS) is another notation for describing business process behavior that is based on Web Services. Processes in BPEL4WS export and import functionality by using Web service interfaces exclusively. It represents a convergence of the ideas in the Microsoft's XLANG and IBM WSFL specifications. Both XLANG and WSFL are superseded by the BPEL4WS specification.

BPEL4WS defines business processes and how they relate to Web services. This includes specifying how a business process makes use of Web services to achieve its goal, as well as specifying Web services that are provided by a business process. Business processes specified in BPEL4WS are fully executable and portable between BPEL4WS-conforming environments. A BPEL4WS business process interoperates with the Web services of its partners, whether or not these Web services are implemented based on BPEL4WS. Finally, BPEL4WS supports the specification of business protocols between partners and views on complex internal business processes.

WebSphere Business Integration Server Foundation V5.1 supports BPEL4WS and a number of BPEL4WS IBM extensions in WebSphere Process Choreographer. The BPEL4WS specification is available at:


For more information about BPEL4WS, please refer to the IBM Redbook *WebSphere V5.0.2 Application Developer V5.1 Web Services Handbook*, SG24-6891.
7.2 Web services

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

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

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

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

Let us take a closer look:

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

- **Web services are self-describing.**
  Neither the client nor the server knows or cares about anything besides the format and content of request and response messages (loosely coupled application integration).
  The definition of the message format travels with the message. No external metadata repositories or code generation tools are required.

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

- **Web services can be published, located, and invoked across the Web.**
  The standards required to do so are:
  - Simple Object Access Protocol (SOAP), also known as service-oriented architecture protocol, is an XML-based RPC and messaging protocol
  - Web Service Description Language (WSDL) is a descriptive interface and protocol binding language
  - Universal Description, Discovery, and Integration (UDDI), a registry mechanism that can be used to look up Web service descriptions
Web services are language independent and interoperable.
The interaction between a service provider and a service requester is designed to be completely platform- and language-independent. This interaction requires a WSDL document to define the interface and describe the service, along with a network protocol (usually HTTP). Because the service provider and the service requester have no idea what platforms or languages the other is using, interoperability is a given.

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

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

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

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

For detailed information about Web services, refer to the following:

- IBM Redbooks:
  - *WebSphere Version 5 Web Services Handbook*, SG24-6891

### 7.2.1 Static and dynamic Web services

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

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

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

### 7.2.2 Web Services Invocation Framework

The Apache Web Services Invocation Framework (WSIF) provides a standard Java API to invoke services, no matter how or where the service is provided, as long it is described in WSDL.

WSIF enables the developer to move away from the native APIs of the underlying service and interact with representations of the services instead. This allows the developer to work with the same programming model regardless of how the service is implemented and accessed.

WSIF is WSDL-driven, and it provides a uniform interface to invoke services using WSDL documents. Therefore, if a SOAP service you are using becomes available as an EJB (for example), you can change to RMI/IOP by just modifying the WSDL service description. You will not need to modify your applications that use the service.

This API is used by WebSphere Process Choreographer and the IBM Web Services Gateway to construct and manipulate services defined in WSDL documents. The architecture allows new bindings to be added at runtime.

WSIF has the following advantages:

- Multiple bindings can be offered for services, and bindings can be decided at runtime.
- Services can be used either by a set of stub classes (static) or by a dynamic interface invocation (dynamic).
- You have the flexibility to switch protocols, location, etc., without having to recompile your client code.

For more details on the Web Services Invocation Framework see:

7.2.3 Web services and the service-oriented architecture

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

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

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

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

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

7.2.4 Web services security

In April 2002, IBM and Microsoft proposed a technical strategy and roadmap for “addressing security within a Web service environment.” The Web services security specifications define a comprehensive Web service security model that supports, integrates, and unifies several popular security models, mechanisms, and technologies (including both symmetric and public key technologies) in a way that enables a variety of systems to securely interoperate in a platform- and language-neutral manner.

The Web services security specification provides a broad set of specifications that cover security technologies including authentication, authorization, privacy, trust, integrity, confidentiality, secure communications channels, federation, delegation, and auditing across a wide spectrum of application and business topologies. These specifications provide a framework that is extensible and flexible. Thus, it maximizes existing investments in security infrastructure. By leveraging the natural extensibility that is at the core of the Web services model, the specifications build upon foundational technologies such as SOAP, WSDL, XML Digital Signatures, XML Encryption, and SSL/TLS.
As shown in Figure 7-1, this set includes a message security model (WS-Security) that provides the basis for the other security specifications. Layered on this, we have a policy layer that includes a Web service endpoint policy (WS-Policy), a trust model (WS-Trust), and a privacy model (WS-Privacy). Together, these initial specifications provide the foundation upon which we can work to establish secure interoperable Web services across trust domains.

**Figure 7-1  The evolving WS-Security roadmap**

For more information, see the IBM developerWorks article *Security in a Web Services World: A Proposed Architecture and Roadmap*:


**WS-Security**

Web Services Security (WS-Security) Version 1.0 was jointly developed by IBM, Microsoft, and VeriSign, and was released in April 2002. It was submitted to OASIS by 18 companies, and now involves over 50 companies.

WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity and message confidentiality. Also, this specification defines how to attach and include security tokens within SOAP messages. Finally, a mechanism is provided for specifying binary encoded security tokens (for example, X.509 certificates). These mechanisms can be used independently or in combination to accommodate a wide variety of security models and encryption technologies.

For more information, see the IBM developerWorks article *Web Services Security (WS-Security)*:

7.2.5 Advantages of Web services

Web services technology enables businesses to:

- Deliver new IT solutions faster and at lower cost by focusing their code development on core business and using Web services applications for non-core business programming
- Protect their investment in IT legacy systems by using Web services to wrap legacy software systems for integration with modern IT systems
- Integrate their business processes with customers and partners at less cost: Web services make this integration feasible by allowing businesses to share processes without sharing technology. With lower costs, even small business will be able to participate in B2B integration.
- Enter new markets and widen their customer base: Web services listed in UDDI registries can be discovered and thus are visible to the entire Web community.

7.2.6 Disadvantages of Web services

Some Web services issues to consider are:

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

7.2.7 Comparing Web services with CORBA and RMI

There are often comparisons made between CORBA, RMI and SOAP, but such comparisons can be misleading. CORBA and RMI are technologies describing the whole communication process for publishing, finding, and invoking methods on remote objects.

In contrast, SOAP describes only the format of the data exchange for communication. Even the transport protocol is not compulsory. To understand comparisons between these technologies, it is important to realize that SOAP does not cover the whole interaction process, as the other technologies do.

There are many articles on the Web describing the pros and cons of using one of these technologies. One aspect worthy of further consideration is object references. There are no object references in SOAP Web services! This basic characteristic of SOAP has far-reaching consequences for the programming
model, for security, and for the coupling strength between requestor and provider. The simplest effect is that parameters in Web services are in-parameters and never ever in-out-parameters. To return a response value, you have to use the return value of the method.

The security impact results from the fact that in the case of CORBA or RMI the client is remotely acting in the address space of the server application. Reference errors can lead to server-side memory access exceptions or data corruption for example.

Furthermore, in the case of CORBA and RMI the structure of the stub objects is identical to those on the server side. Changing the object structure on the server side needs a refactoring of the clients. In the case of SOAP, no object structures are exposed, except for methods that can have objects as parameters and return values. Changing the object structure on the server side need not lead to a refactoring of the client as a direct consequence.

### 7.3 Java Message Service

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

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

According to the JMS 1.0 specification, a message provider is integrated in an application server. As shown in Figure 7-2 on page 129, the integrated message provider makes it possible to communicate asynchronously with other WebSphere applications without installing separate messaging software like IBM WebSphere MQ. WebSphere Application Server’s integrated JMS server is based on IBM WebSphere MQ.
An important new feature of EJB 2.0 is message-driven beans (MDB). Message-driven beans are designed specifically to handle incoming JMS messages. Further information about message-driven beans can be found in the IBM Redbook *EJB 2.0 Development with WebSphere Studio Application Developer*, SG24-6819.

### 7.3.1 What messaging is

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

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

- One-to-one, or point-to-point model
- Publish/subscribe model
7.3.2 JMS and IBM WebSphere MQ

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

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

7.3.3 Advantages of JMS

The JMS standard is important because:

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

7.3.4 Disadvantages of JMS

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

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

Further information about JMS can be found in the IBM Redbook MQSeries Programming Patterns, SG24-6506.
Process manager capabilities

This book focuses on two Process managers; WebSphere Process Choreographer and WebSphere MQ Workflow. In this chapter, we discuss the capabilities of these Process managers, using the following categories:

- Product features capabilities
  This section discusses the functional capabilities offered by both Process managers.

- QoS capabilities
  This section discusses the non-functional capabilities offered by both Process managers.
8.1 Process manager summary

The WebSphere MQ Workflow is a proven technology providing high performance, scalability, reliability, and availability over the years. It is based on a WebSphere MQ messaging architecture. It offers a top-down approach to managed process-integration solutions, including systems and people and process monitoring via WebSphere Business Integration Monitor. WebSphere MQ Workflow is available with native implementation on the z/OS platform, exploiting z/OS features and quality of service.

The WebSphere Process Choreographer is fully integrated into WebSphere Application Server Enterprise and WebSphere Studio Application Developer Integration Edition. It is based on a Service Oriented Architecture approach. It exploits WebSphere Application Server features such as clustering and load balancing for scalability, availability, and failover. Based on full support of the J2EE programming model and open architecture, it allows J2EE programmers to rapidly create complex composite services and processes involving long-running workflows with human interaction.

8.2 Product features capabilities

This section describes and compares the major features offered by the WebSphere Process Choreographer and WebSphere MQ Workflow process managers. Many of these features have been used in the scenarios in this book. For a more in-depth description of many of these features, consult the relevant scenario chapter.

8.2.1 Process modeling and monitoring features

The following product features relate to the modeling and monitoring of processes.

Process modeling and simulation
Process modeling and simulation are enabled in these process manager applications as follows:

- WebSphere Process Choreographer
  The WebSphere Process Choreographer uses an internal flow language called Flow Definition Markup Language (FDML) in V5.0 to define processes. There are currently no Business Process Modelling tools that provide the capability to model and simulate FDML processes.
IBM will replace FDML with the BPEL4WS open standard. WebSphere Studio Application Developer Integration Edition V5.1 provides a tool to migrate FDL processes to BPEL4WS. BPEL4WS will likely be supported by several Business Process Modeling tools.

- **WebSphere MQ Workflow**
  
  The WebSphere MQ Workflow uses a flow language called Flow Definition Language (FDL) to define processes.

  WebSphere Business Integration Workbench is a Business Process Modelling tool that supports FDL processes. It provides modeling and simulation capabilities. During simulations, it is possible to isolate bottlenecks while changing the staffing and other parameters.

**Process monitoring**

Process monitoring is enabled in these process manager applications as follows:

- **WebSphere Process Choreographer**
  
  The WebSphere Process Choreographer provides a Web client with a tabular process monitor that you can use to monitor states of processes and activities. You are also able to repair stopped activities by using the process monitor. For example, if the activity encounters a connection failure, you can restart this activity.

- **WebSphere MQ Workflow**
  
  The WebSphere MQ Workflow provides several types of interfaces that offer monitoring capability. These include the following:

  - The WebSphere Business Integration Monitor referenced at:
  - The WebSphere MQ Workflow GUI client for Windows clients
  - The WebSphere MQ Workflow Web Client

**8.2.2 Process manager architecture**

The following product features relate to the architecture of a process manager.

**Pluggable runtime architecture**

These process manager applications use the following runtime architecture:

- **WebSphere Process Choreographer**
  
  The WebSphere Process Choreographer provides a pluggable runtime architecture with three major components: Process Navigation, People Interaction, and a Factory for persistence. There are different plug-ins for staff
resolution, activity invocation, and so on. These are internally pluggable, allowing IBM to remove and add components to the runtime in future releases.

Individual activities in a process can easily be replaced with different implementations without major change, for example from a Web service implementation to an Enterprise bean.

▶ **WebSphere MQ Workflow**

The WebSphere MQ Workflow provides a pluggable runtime architecture. At each activity in the process, different programs can be plugged in, some without even changing the process template. Thus, an activity that is initially implemented as a human interaction step to prompt a user to go to system X and enter data can eventually be changed to have an automatic activity perform the integration. This scenario is common because many organizations may not have the budget to get all the benefits a workflow solution can deliver in the initial implementation. Therefore, they embark on performing additional system integration with future releases.

### Short- and long-running processes

These process manager applications use the short- and long-running processes as follows:

▶ **WebSphere Process Choreographer**

The WebSphere Process Choreographer provides support for short-running processes (called non-interruptible processes) and long-running processes (called interruptible processes). Long-running processes can incorporate human interaction.

▶ **WebSphere MQ Workflow**

The WebSphere MQ Workflow is best suited to long-running processes. These processes contain more than a single activity, and a persistent process state is maintained.

### 8.2.3 Process and activity invocation

The following product features relate to how process instances and activities are invoked and managed.

#### Process invocation

These process manager applications perform process invocation as follows:

▶ **WebSphere Process Choreographer**

The business process engine in WebSphere Process Choreographer has two major interfaces. The first is a session bean-based API, and the second is a
message-driven bean-based API for listening to queues accessible using JMS. There are two ways to invoke the process, either synchronous (two way: request/reply) or asynchronous (one way: request only).

► **WebSphere MQ Workflow**

The WebSphere MQ Workflow accepts several methods to create an instance of a process model. These include the following:

- From one of the supported programming languages (Java, C++, C and Visual Basic), use one of the various CreateAndStartInstance, CreateInstance, or ExecuteProcessInstance APIs. Both synchronous and asynchronous process invocations are supported.

- From an application that you create, use a WebSphere MQ PUT of an XML message with the ProcessTemplateCreateAndStartInstance or ProcessTemplateExecute formats to the workflow execution server input queue EXEXMLINPUTQ. The XML interface supports both synchronous and asynchronous process invocations.

- From a GUI client on a LAN or from a Web client, use your template worklist. This interface is built on WebSphere MQ Workflow API’s

### Process instance user interface

These process manager applications employ the following process instance user interfaces:

► **WebSphere Process Choreographer**

The WebSphere Process Choreographer provides several types of interface to work with process instances:

- A Web client, based on the standard Struts technology: This Web client allows a customer to customize how a specific message is rendered to the end user.

- A Java API, available through a session bean and a message-driven bean

► **WebSphere MQ Workflow**

The WebSphere MQ Workflow provides several types of interface to work with process instances:

- GUI client for Windows clients
- WebSphere MQ Workflow Web Client
- WebSphere MQ Workflow Portal client
- APIs provided in Java, C++, C, XML, COBOL for zSeries® and Visual Basic to allow customer clients written in any of these languages
Activity invocation
These process manager applications perform activity invocation as follows:

* **WebSphere Process Choreographer**
  There are two categories of activity invocations: synchronous (Web services using SOAP over HTTP, Java beans, Enterprise Java Beans, or J2EE Connectors) or asynchronous (JMS, or Web services using SOAP over JMS) invocations. These invocations are wrapped into WSIF (Web Services Invocation Framework) calls to provide a unified way of invoking activities. Therefore, they provide an important step to a Service Oriented Architecture.

* **WebSphere MQ Workflow**
  The WebSphere MQ Workflow has several interfaces to invoke an activity. These activity invocation methods use either APIs or XML messages. All of these interfaces use WebSphere MQ messages. XML activity invocation messages can be modeled to use synchronous or asynchronous invocation.

  Use the WebSphere MQ Workflow Web Services Toolkit to obtain Web services invocation for an activity.

8.2.4 Debugging and auditing
The following product features relate to process debugging and auditing.

Process debugging
These process manager applications perform debugging as follows:

* **WebSphere Process Choreographer**
  The WebSphere Studio Application Developer Integration Edition provides the capability to debug business process flows through the Process Debugger. This debugger allows you to step into activities and inspect process instance variables.

* **WebSphere MQ Workflow**
  In order to debug MQ Workflow process models, it is necessary to import them into a runtime test environment. Within the test environment, all activities can be stubbed to ensure that the basic flow of connectors and data is correct within the process template. The activities can be stubbed with either the windows show container tool (fmcnshow.exe), the default activity viewer that comes with the Web client or, in the case of a UPES activity, the WebSphere MQ API Exerciser tool to retrieve and put messages on a queue.
Audit logs
These process manager applications process audit logs as follows:

- **WebSphere Process Choreographer**
  The audit log structure is published to a database. Hence, a consumer can easily create a rendering for it. With an upcoming release, a sample of an audit trail viewer is planned to show how this can be done.

- **WebSphere MQ Workflow**
  The audit trail in WebSphere MQ Workflow can be published to a WebSphere MQ queue and/or written to the audit trail table within the process manager's database. The level of auditing is very flexible within WebSphere MQ Workflow. Predefined settings for full or condensed audit can be set. For complete control over which events get recorded, a setting of filter can be applied so that the process modeler can define exactly which events are logged. Additionally, the audit settings can be set at a process level or inherited from any one of the system, system group, or domain levels within the WebSphere MQ Workflow network hierarchy.

  The audit trail can be viewed through a number of interfaces: the WebSphere MQ Workflow Web client, the WebSphere Business Integration Monitor using a predefined database view, or SQL that you write for yourself.

8.2.5 Activity related features

The following product features relate to the activities running in a process.

Events
These process manager applications handle events as follows:

- **WebSphere Process Choreographer**
  An event is an activity in WebSphere Process Choreographer that stops the business process until a certain outside activity occurs. Events can be triggered by using the process interface.

- **WebSphere MQ Workflow**
  There is no event activity native to WebSphere MQ Workflow. However, the handling of events can be easily modeled into a process template by modeling a wait activity that waits for the event to happen. This wait activity can either be dummy activity or a UPES activity. If desired, the process can be suspended while it is waiting.
Suspend and resume
These process manager applications handle suspending and resuming processes as follows:

- **WebSphere Process Choreographer**
  There is no current support to suspend the process instance after every activity invocation. Stopped activities can be restarted by the Web client if something goes wrong in the process.

- **WebSphere MQ Workflow**
  A process instance can be suspended and resumed after every activity invocation. The SuspendUntil API suspends a process until a given time, so the process can resume if the until time is exceeded.

Scheduler
These process manager applications perform scheduling as follows:

- **WebSphere Process Choreographer**
  A scheduler is fully integrated into the WebSphere Application Server V5 runtime environment and is used by WebSphere Process Choreographer for all time-driven actions.

- **WebSphere MQ Workflow**
  The WebSphere MQ Workflow incorporates a scheduling server for all timed actions such as checking activity expiry or checking whether processes or activities are past due for completion. The scheduling server also allows user configuration of its parameters, including how often the scheduling server should check for the events that it is interested in.

Compensation
These process manager applications handle compensation as follows:

- **WebSphere Process Choreographer**
  Compensation is an important out-of-the-box feature in WebSphere Process Choreographer for long-running workflows to undo transactional and non-transactional activities. A compensating activity is provided for every activity where compensation can occur.

- **WebSphere MQ Workflow**
  The WebSphere MQ Workflow does not include a compensation feature, but if desired, compensation can be modeled into a process template. This means that reversing transactions need to be written and invoked as part of the compensation.
Connectors
These process manager applications handle connectors as follows:

- **WebSphere Process Choreographer**
  The WebSphere Process Choreographer provides support for J2EE Connector Architecture resource adapters, including CICS Transaction Server and SAP. The WebSphere Business Integration Adapters are also supported.

- **WebSphere MQ Workflow**
  If you create user-defined execution servers, then you can include J2EE Connector Architecture resource adapters and WebSphere Business Integration Adapters as part of the infrastructure.

Activity expiration
These process manager applications handle activity expiration as follows:

- **WebSphere Process Choreographer**
  Activity expiration is available for staff and event activities. It uses the scheduler provided in the WebSphere Application Server runtime.

- **WebSphere MQ Workflow**
  Activity expiration is available for all activities. The WebSphere MQ Workflow scheduling server checks activities that have expiry set.

Process and activity notification
These process manager applications handle process and activity notification as follows:

- **WebSphere Process Choreographer**
  Notification is not supported in the WebSphere Process Choreographer. Staff and event activities can be set to expire, and manual notification could be implemented for these expired activities by using a staff activity with a late binding.

- **WebSphere MQ Workflow**
  If a process or activity does not complete in the specified interval, a notification work item is generated to the staff specified. A notification is an additional item that is generated by the scheduling server. When the notification time is reached, the original work item remains in place and an additional item is generated to the target user rather than the process continuing as it does with expiry. This notification item is typically sent to a supervisor or manager to alert him of a potential problem. Once the corresponding work item is completed, the notification item is removed.
automatically by the WebSphere MQ Workflow servers (if not already deleted).

The WebSphere MQ Workflow also supports a second notification for an activity that is not acted on following the first activity notification. Notification intervals can be set at fixed intervals or a data container can hold the value at runtime.

**External staff repository**

These process manager applications handle external staff repository processing as follows:

- **WebSphere Process Choreographer**
  
  The staff resolution plug-in in the WebSphere Process Choreographer runtime architecture works in conjunction with any of the three supported WebSphere user registries: local OS, LDAP, and custom user registry.

- **WebSphere MQ Workflow**
  
  The WebSphere MQ Workflow staff repository is internal to the runtime database, but an LDAP Bridge allows synchronization of the external directories with the runtime database.

**Substitution and absent flagging**

These process manager applications handle substitution and absent flagging as follows:

- **WebSphere Process Choreographer**
  
  Substitution is not supported in the current release of WebSphere Process Choreographer.

- **WebSphere MQ Workflow**
  
  For each user defined within the WebSphere MQ Workflow staff directory, the user can specify who serves as a substitute. Related to this is, the ability to declare a user absent. If a user is absent when the runtime server attempts to send him a work item, the work item is sent to the substitute instead.

**Work item transfer**

These process manager applications work item transfer as follows:

- **WebSphere Process Choreographer**
  
  This feature is not directly available. However, a work-around for the work item transfer can be done programatically (using API calls).
Chapter 8. Process manager capabilities

8.3 QoS capabilities

This section discusses the QoS capabilities offered by the WebSphere Process Choreographer and WebSphere MQ Workflow process managers.

8.3.1 Availability

For a definition of the availability QoS, see “Availability” on page 26.

WebSphere Process Choreographer

The process container of WebSphere Process Choreographer is implemented as a stateless process engine. Many instances of it can run in parallel on a single node or distributed in a cluster. It can exploit both the clustering capabilities of WebSphere Application Servers for IIOP (Internet Inter-Orb Protocol) requests and the clustering capabilities of WebSphere MQ for JMS-based requests.

For non-interruptible processes, there is an affinity between a particular process instance and the thread that executes it. Many instances of a particular process template can run in parallel on parallel threads.
For interruptible processes, no such thread affinity exists. A given process instance is simply reflected by a set of tuples persisted in the process database. Threads work on segments of process instances. Multiple threads, even on multiple nodes, can work concurrently on parallel branches of a process.

**WebSphere MQ Workflow**

WebSphere MQ Workflow can provide node redundancy and failover capability to eliminate single-points-of-failure. One way to do this is to use a WebSphere MQ cluster. WebSphere MQ clusters with multiple nodes configured in queue managers in a cluster provides some load-balancing and failover capability. However, you must consider that in a failover of one WebSphere MQ Workflow server node a message could be held in the failing machine. Refer to:


For a true high-availability such as in an AIX® environment, it is possible to configure WebSphere MQ Workflow, WebSphere MQ, and DB2 as described in:


### 8.3.2 Operability

For a definition of the operability QoS, see “Operability” on page 26.

**WebSphere Process Choreographer**

Log and trace facilities are provided by standard WebSphere Application Server functions. Logs are written to the files SystemOut.log and SystemErr.log. Tracing can also be turned on for specific classes. For example you can trace all calls to the com.ibm.bpe.* classes (the runtime classes of the WebSphere Process Choreographer container) and org.apache.wsif.* classes (the WSIF classes, used for process and activity invocation).

**WebSphere MQ Workflow**

Tracing facilities for WebSphere MQ Workflow are enabled by using the WebSphere MQ Workflow application fmczchk. The WebSphere Application Server logs (SystemOut.log and SystemErr.log) and the SOAP monitor are also useful tools when using the Web Services Toolkit.

### 8.3.3 Performance

For a definition of the performance quality of service see “Performance” on page 142.
**WebSphere Process Choreographer**

WebSphere Process Choreographer is part of WebSphere Application Server Enterprise 5. Therefore, all of the caching mechanisms and performance tools provided by WebSphere Application Server can be used. Additionally all WebSphere performance best practices can be used with WebSphere Process Choreographer implementations.

WebSphere Process Choreographer’s business process container distinguishes two kinds of processes, interruptible and non-interruptible.

Interruptible processes are not as fast as non-interruptible processes because they interact with queues and the database to reliably hold the state of the process. Non-interruptible processes on the other hand run in memory on one thread and are faster by a factor of 10 - 100 times, compared to an interruptible process.

**WebSphere MQ Workflow**

There are a large number of factors to consider when planning, installing, programming for, and using WebSphere MQ Workflow. As WebSphere MQ Workflow is a mature product with an established customer base, much work has been done on considering the performance for all of these areas. For details regarding all aspects of performance in WebSphere MQ Workflow, refer to the following documents:


### 8.3.4 Security

For a definition of the security QoS, see “Security” on page 143.

**WebSphere Process Choreographer**

WebSphere Process Choreographer leverages the authentication capabilities of WebSphere Application Server V5. The plug-in for the WebSphere Process Choreographer runtime architecture that does the staff resolution works in conjunction with any of the three supported WebSphere user registries: local OS, LDAP, and custom user registry. This plug-in can be used to secure a process, a staff work item, or an event.

You can use also use LTPA tokens for single sign-on or use an external security server by utilizing a trust association interceptor for authentication.
Authorization support is provided for processes, work items, and events. This support allows you to limit who has access to these resources.

**WebSphere MQ Workflow**

WebSphere MQ Workflow has its own internal security checking and validation. Besides the sign-on security checking, there are also authorization checks for tasks that require:

- Transfer authority
- Authority to terminate, suspend, resume, force finish, and force restart instances
- Authority to create and start processes from templates

In addition to the internal security discussed above, WebSphere MQ Workflow allows an exit to be loaded at startup time based on its machine profile. With this exit loaded, one could use LDAP for the authentication, for example. When a user performs a logon, a logon API using credentials is responsible for the authentication. If the logon is authorized by the authentication exit, the userid can be mapped to an WebSphere MQ Workflow userid. This means that WebSphere MQ Workflow still requires staff objects to be maintained in the database even if the security exit is used. However, they supply an LDAP Bridge that can compare the staff maintained and make updates as required using a batch program.

**8.3.5 Standards compliance**

For a definition of the standards compliance QoS, see “Standards compliance” on page 144.

**WebSphere Process Choreographer**

WebSphere Application Server V5 has completed the full J2EE certification test suite. The product supports all of the J2EE 1.3 APIs. WebSphere Process Choreographer is a component that is part of WebSphere Application Server Enterprise, Version 5. That means you have a J2EE compliant basis for deploying and running enterprise applications.

**WebSphere MQ Workflow**

WebSphere MQ Workflow's design follows the Workflow Management Coalition reference model. The coalition has a framework for the establishment of workflow standards. Refer to:

http://www.wfmc.org/standards/conformance.htm
8.3.6 Transactionality

For a definition of the transactionality QoS, see “Transactionality” on page 145.

**WebSphere Process Choreographer**

The WebSphere Process Choreographer relies on global J2EE transactions in non-interruptible processes where the transaction manager, resource manager and backend systems work together to achieve a single transaction that can be committed and rolled back in a two-phase commit transaction.

In addition to the J2EE transaction model, the WebSphere Application Server Enterprise also provides capabilities for involving one non-two-phase commit capable resource within a two-phase commit transaction. This feature is called Last Participant Support.

This concept is extended by building interruptible processes that can also include, for example, human activities without locking resources. An interruptible process consists of a set of *stratified transactions*. This means that in a process that contains a set of activities each navigation step is performed in its own transaction. If an activity is non-transactional, that is, its activity implementation does not participate in the two-phase-commit protocol of the transaction, compensation-based recovery can be used to undo changes performed by the activity if a failure occurs.

**WebSphere MQ Workflow**

The WebSphere MQ Workflow uses WebSphere MQs and DB2 resources that have resource managers for the two-phase commits. In a single WebSphere MQ Workflow activity, these resource managers determine whether a transaction is to be committed or rolled back.

Because WebSphere MQ Workflow processes are typical long running business processes and sub-processes, a series of activities could already be committed at the time an event or activity signaled that all the activities should be rolled back. This type of compensating transaction interface is not a built-in feature with WebSphere MQ Workflow, but it is possible to design either (1) by using compensating processes initiated by human action or (2) by designing user-defined execution servers that locate the instances in-flight and terminate them (while also creating an instance of the compensating process).
Part 3

Process manager scenarios

This part contains scenario implementations of the Serial and Parallel Process patterns. Each scenario is implemented twice: once using a WebSphere Process Choreographer Product mapping and once using a WebSphere MQ Workflow Product mapping. Each chapter builds upon the concepts of the last. Therefore, we recommend reading them in the following order:

- Chapter 9, “Creating serial processes” on page 149
- Chapter 10, “Creating parallel processes” on page 245
- Chapter 11, “Creating processes with human interaction” on page 267
- Chapter 12, “Creating processes with events and compensation” on page 315
- Chapter 13, “Process manager interoperability” on page 357
Creating serial processes

This chapter describes how to build a process that uses the Serial Process pattern. The chapter is split into the following parts:

- **Business scenario**
  Describes a business problem that ITSO Electronics wishes to solve

- **Business process model**
  Describes the business process model to solve the business problem

- **General design guidelines**
  Provides product-agnostic design guidelines for building a process conforming to the Serial Process pattern

- **WebSphere Process Choreographer guidelines**
  Describes how this Serial Process was implemented using WebSphere Process Choreographer

- **WebSphere MQ Workflow guidelines**
  Describes how this Serial Process was implemented using WebSphere MQ Workflow

- **Best practices**
  Describes general- and product-specific best practices for configuring a suitable runtime environment
9.1 Business scenario

ITSO Electronics wishes to integrate its internal retail and wholesale departments to create an order-replenishment process. The retail department can replenish items of stock from two wholesale departments, called Wholesaler A and Wholesaler B. In this scenario, we make two assumptions about ITSO Electronics:

- Wholesaler A is ABC Electronic's preferred supplier.
- Both wholesalers offer each part at an identical price. The cost of the part does not factor into ABC Electronic's decision as to which supplier the parts are sourced from.

If this were a real-life scenario, the order clerk at ITSO Electronics would likely use a Web application to place an order. The order clerk would enter the part number and the quantity for the order. Once the clerk is satisfied that the order is correct, he would submit the order request and would expect a response back within a few seconds. The response includes which wholesaler was used, the confirmation number, and the delivery days.

Since Wholesaler A is the preferred supplier for ITSO Electronics, a product 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. Then, the order is placed with the wholesaler who has the shortest delivery date.

Finally, it is important to ITSO Electronics to have a historical record of the orders that are placed. This record needs to include who placed the order, the part number and quantity that is being ordered, and which supplier the order was placed with.

9.2 Business process model

In this scenario, we are going to implement the process using the Serial Process pattern because ABC Electronic's current business policy dictates that its preferred supplier is Wholesaler A. ABC only contacts Wholesaler B for a delivery date when Wholesaler A cannot provide a delivery date in one week or less. Therefore, the process sequentially calls Wholesaler A for its best date and only calls Wholesaler B when necessary.

A generic swimlane diagram can be used to depict the process (Figure 9-1 on page 151).
Figure 9-1  Serial Process swimlane diagram

This Serial Process implementation is a relatively small process involving only automated system integration activities. Once the order is submitted by the clerk, the following sequence of events takes place:

- Wholesaler A is contacted for its delivery date.
- If the delivery date is one week or less, the order is placed with Wholesaler A.
- If the delivery date returned from Wholesaler A is greater than one week, then Wholesaler B is contacted for its delivery date.
- Once Wholesaler B responds, the system can determine who the best wholesaler is (that is, which wholesaler can deliver the most quickly).
- The order is placed with the best wholesaler.
- After the order is successfully placed, the best wholesaler, confirmation number, and the deliver days are returned to the order clerk via the application that used to start the process.
9.3 General design guidelines

### Process or workflow?

The term *process* is used throughout this redbook to generically describe a business process. A process may or may not contain human interaction.

The term *workflow* is taken to mean a *long-running process*. A distinction of a workflow or long-running process is that it persists process instance data within the process manager’s database during the execution of the process. A workflow may or may not include human interaction.

A *short-running process* can be defined as a process that contains all automated steps and will complete in a very short period of time. Contrary to a long-running process, a short-running process does not persist any process instance data. A WebSphere Process Choreographer non-interruptible process falls into the category of a short-running process.

In this scenario, it is acceptable to have a user initiate a short-running process and await a response when the process completes. Precautions need to be taken to ensure that the process does not stop during execution. If any exceptions occur during the processing, the process should complete and the response from the process should indicate that there was some form of failure. For a short-running process, timeouts must be included on all of the activities to ensure that the process does not get stranded.

In this Serial Process pattern scenario, there is no human intervention other than from the order clerk who initiates the process. We can assume that the order clerk will launch the process when he submits an order and that he will receive a response when the order process is completed.

### 9.3.1 Design overview

This scenario is an implementation of the Process-focused Application Integration::Serial Process pattern.

For information about the Serial Process Application pattern see:

3.4.6, “Serial Process Application pattern” on page 55

For information about the Serial Process Runtime pattern see:

5.1.1, “Serial Process Runtime pattern” on page 84
For information about the Product mappings of the Serial Process Runtime pattern see:

5.2.1, “Serial Process Product mappings” on page 93

This scenario is an implementation of Stage One of the ITSO Electronics integration project. For information about the use case and actors used in Stage I, see:

6.2.1, “Stage One: Internal ordering on demand” on page 110

9.3.2 Design considerations

We recommend that you consider the following design considerations when planning any type of process implementation:

- Synchronous verses asynchronous process invocation
- The need for human interaction in processes
- Rules engines
- Security
- The impact of performance and capacity analysis
- Audit trail

**Synchronous verses asynchronous process invocation**

Most process managers support the two basic process invocation methods:

- synchronous invocation
- asynchronous invocation

A synchronous invocation of a process means that a response is not returned from the process manager until the process completes (Figure 9-2 on page 154). However, the completion of the process does not mean that it completed successfully. The reply needs to be inspected to determine the success or failure of the process.
An asynchronous invocation of a process receives an immediate response, but this response only indicates whether the process instance was started successfully or not (Figure 9-3). Depending on the process manager, a response may or may not be sent back to the user at the completion of the process.

Choosing an invocation method

The characteristics of the two process invocation methods are used to determine which method to choose for a given process template.

A synchronous invocation should usually be chosen when the following circumstances are true:

- The process is short-running.
- The calling application will receive 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 usually be chosen when 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. However, there are exceptions to these guidelines, as follows:

- **Sub-processes**
  A process that is spawned from another process is called a sub-process. Even though the sub-process may be long-running (possibly even taking months to complete), it is quite common to call this sub-process synchronously to ensure that the higher level process does not continue until the sub-process completes.

- **Calls to other process managers**
  A process may contain an activity that invokes a process running on a second process manager. This call might also be synchronous, regardless of whether the process running in the second process manager is short- or long-running. Using a synchronous call ensures that the initiating process does not continue until the invoked process has completed. See the design considerations in Chapter 13, “Process manager interoperability” on page 357.

**Applying the guidelines to ITSO Electronics**
In this Serial Process pattern scenario, it was decided to invoke this short-running process using a synchronous invocation. This decision was based on the following factors:

- The process should complete in a very 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.

**The need for human interaction in processes**
Typically, any long-running process will need to have human interaction activities defined as part of the process. This can be the case even if the long-running process contains only automatic system integration activities. Some form of human interaction may be required in the process in case of a failure returned from an integrated component or of non-responses from the automatic system integration.
If the starter of the process is expecting an imminent response, then it is imperative that this short-running process generate a response quickly. This implies that there cannot be any human interaction during the life cycle of this process.

Caution needs to be taken to ensure that a process called synchronously does not stop unexpectedly. There are two approaches to handling failure or timeouts of automatic system activities within a process:

- Ending the process whenever an exception occurs
- Use human involvement when an exception occurs

**Ending the process whenever an exception occurs**

This approach has the following characteristics:

- No human intervention is used.
- The process can remain a short-running process, unless compensation is required.
- A process instance is unable to recover from an error situation.

This approach suggests that if an error is returned by any of the activities in the process, the process must end immediately and the error be returned to the user. If any data has been updated before the error or timeout occurred, some level of compensation must be applied before the process can end. This ensures that the data is in a consistent state. If no updates are performed, then no compensation is required and the process can end.

For more information about compensation, see Chapter 13, “Process manager interoperability” on page 357.

With this approach, the process template must be defined to support having the process end at each and every activity in the process if a failing condition is encountered. The modeler should take special care to ensure that this is the case. Every activity must be allowed to go to the end of the process if the activity does not complete successfully.

The process could complete with the following responses:

- The process completed successfully.
- An error occurred during the invocation of one of the integrated components.
- No response was received from one of the integrated components.
Use human involvement when an exception occurs
This approach has the following characteristics:

- A human is used to correct an error within the process.
- The process becomes long-running.
- A process instance may be able to recover from an error and complete the process successfully.

In this scenario, human interaction activities are added to a process. They are only invoked if any of the automatic system integration points either responds with an error or does not respond at all. The human interaction activities provide the opportunity for a human to correct a problem within the process. This approach is known as people-based exception handling.

For more information about people-based exception handling refer to “People-based exception handling” on page 272.

Applying the guidelines to ITSO Electronics
ITSO Electronics does not require us to handle error situations in Stage One of the integration project. Therefore, we assume that all of the automatic system integration points will complete successfully and in a short period of time. We assume that if an error or timeout occurs, then the process will immediately end. No compensation is required since we are only querying dates of the two wholesalers prior to the execution of the single place order activity. If this activity fails, then there is no harm in having executed the get delivery date functions for the wholesalers.

We also ignore a troublesome scenario that could potentially occur. It is possible that an order can be placed with either Wholesaler A or Wholesaler B and complete successfully on the wholesaler side, but the successful reply back can never reach the process manager (for whatever reason). In production implementations, special care is required to ensure that proper steps are taken in these circumstances.

In summary, we do not have the requirement to add human interaction activities to this process. If we did need human interaction, we would follow the Workflow variation of the Serial Process pattern and invoke the process as a long-running process with an asynchronous invocation.

Rules engines
Within a process template, it is common to have one or more decision points where the flow of the process can traverse one or more control paths depending on data contained within the process at runtime. Process managers natively
support having the flow of the process conditionally branch based on data at runtime.

If any decision logic in the process template is prone to change (for example, for financial authority limits), then implementing a rules engine within the process should be considered. This has the following advantages:

- The decision logic in a process can be changed at runtime without changing the process template.
- Migration issues in updating a process template are avoided. There is a period of time after the decision logic in a process template is changed when some of the many running process instances follow the old process decision logic and some follow the new logic. This may be confusing for the users because anyone looking at a specific process instance may not be able to immediately determine which process logic this process is following.

There are times when it is best to include decision logic as part of the process template instead of using a rules engine. Such times include:

- When the decision logic that is unlikely to change: Adding a rules engine to such decision logic would add unnecessary overhead.
- When the decision logic supports exceptions or error codes of automated activities

**What a rules engine does**

A rules engine separates the routing rules from the application code and keeps the routing rules external to the process. When the decision logic or a rule requires changing, a user interface can be provided to allow qualified users to make the required changes.

The process template includes an activity that integrates with the rules engine (Figure 9-4 on page 159). Data from the process instance is passed in to the rules engine, a determination is made based on the rules, and a response is sent back to the process manager. This response must include a decision variable that is returned to the process manager to allow it to determine the control flow.
Benefits of using a rules engine

A major benefit of changing a rule within a rules engine is that the change is immediate and does not take effect only with new process instantiations. Another advantage of a rules engine is that rule changes do not require a process model change. This factor may be especially appealing because a change to the process model is usually viewed as a change to code and is therefore subjected to a more rigorous testing cycle. Finally, rule changes with a rules engine can be considered less intrusive and may only require a subset of test cases before they can be deployed into production.

With these reasons in mind, the use of a rules engine within processes allow the business to be more responsive to changes. These changes to the rules lend themselves to allow customers to be an on-demand business.

IBM’s rules engine is called Business Rules Beans (commonly known as BR Beans), and it is included with WebSphere Application Server Enterprise.

Applying the guidelines to ITSO Electronics

A major decision point in the Serial Process implemented in this chapter is whether to get the input of a second wholesaler if the first wholesaler could not deliver in seven days or less. This seven day decision point could have been externalized to a rules engine. The decision of the best wholesaler would have been implemented as an activity in the process that integrates with the rules engine to determine the control path to be executed in the process.
ITSO Electronics does not expect the seven day decision point to change. Therefore, it does not wish to add the additional overhead of a rules engine. However, to illustrate how a rules engine could be added, the Serial Process implementation for the WebSphere Process Choreographer is extended to use a rules engine in “Process overview: second iteration” on page 163.

**Security**

There are two levels of security to consider when working with processes:

- **Process-related security**
  
  This level of security defines who can start, edit, and view process instances. Authenticated users can be assigned roles such as process starter and process editor, and authorization checks can be made against these roles whenever a user attempts to work with a process instance.

- **Activity-related security**
  
  Typically whenever automatic system integration is used, the components or services that are invoked require some level of authentication to ensure that the requests are from a valid source. Additionally, external event activities and work items may require certain authorization privileges.

When designing a process, both of these security considerations should be taken into account, and roles should be defined accordingly.

When a process is started, the initiator’s userid and credentials can be captured and checked that they have sufficient authorization to start a process instance. These credentials can also be passed to activities within the process. However, some existing services may not allow the typical user to perform the type of operation requested in the process. In this case, his userid would not be acceptable to authenticate the request. Situations like this require that some sort of system userid be used. This system userid can be passed as part of the process flow to each service.

**Applying the guidelines to ITSO Electronics**

ITSO Electronics does not require process-related security to limit who can work with process instances. It is assumed that the only people who have access to the process instances have already cleared some external authorization check. Additionally, the wholesaler activities do not require any activity-related security because these services can be invoked by anybody within the internal network.

**The impact of performance and capacity analysis**

As with any system implementation, some level of a performance and capacity analysis should be performed. This analysis needs to estimate the load on the system during peak periods to ensure that the system will perform as expected.
Although it is not the purpose of this redbook to address performance and capacity issues, it should be noted that once the analysis is completed, the design of a process may change.

**Applying the guidelines to ITSO Electronics**

Performance and capacity are not considered as factors in the design of this scenario. It should be noted that in the Parallel Process pattern scenario does address a performance issue by triggering two activities in parallel to facilitate potentially quicker process completion time.

**Audit trail**

Using a process manager produced audit trail can prove to be invaluable to the different parties interested in the scenario. The audit trail can be utilized by both the business and IT staff alike.

During the development and testing phases, the IT staff (such as developers and testers) may choose to use the audit trail to review which activities completed during the execution of a specific process. This review of the audit can assist the developers and testers in verifying that the application code and process models are working as expected.

Once a process template is implemented into a production environment, the business staff can use the audit trail as the basis for a process tracking component of the system. Among other things, the audit trail provides information about which activities were completed, by whom, and when they were completed.

**Applying the guidelines to ITSO Electronics**

ITSO Electronics can use the process manager audit trail to track whether a given order was placed with Wholesaler A or Wholesaler B. This is achieved by enabling the audit trail in the process manager. The audit events are written to the audit table in the process manager’s database.

### 9.4 WebSphere Process Choreographer guidelines

This section describes how to implement the Serial Process pattern to create a process that meets the requirements set by the ITSO Electronics business process model. It uses the Runtime pattern and Product mappings described in “Serial Process Product mappings” on page 93 for WebSphere Process Choreographer.
You can download the completed process and run it in WebSphere Application Server Enterprise, or you view it in WebSphere Studio Application Developer Integration Edition. See “Sample scenarios setup” on page 398.

9.4.1 Design guidelines

This section discusses WebSphere Process Choreographer-specific design guidelines when creating a process that conforms to the Serial Process pattern.

Process overview: first iteration

The first iteration of the process meets the business objectives of ITSO Electronics as defined in “Stage One: Internal ordering on demand” on page 110. The completed process is shown in Figure 9-5.

The process has the following characteristics:

- Synchronous interface
- Non-interruptible (short-running) process
- Serial execution path

The process implementation makes calls to two systems external to the process manager:

- Wholesale system A, which exposes business logic with a Web service interface
- Wholesale system B, which exposes business logic with an Enterprise JavaBean interface
Process overview: second iteration
The second iteration incorporates all of the functions provided in the first iteration, and adds one additional characteristic:

- Externalization of business rules

The business rules are externalized using a rules engine. The completed process is shown in Figure 9-6.

![Diagram of Process Overview: Second Iteration](image)

Figure 9-6 Process overview: second iteration

The process contains a new block called areDelvrDaysGoodEnough_Block. The activity in this block externalizes the rule if the delivery days are greater than seven. You can change this business rule dynamically without redeploying the process template.

Process interface definition
Processes built in WebSphere Process Choreographer employ a Service Oriented Architecture. In this architecture, you separate the interface from the service and the binding information to provide a standard way in which services can be described and called.

The Web Services Description Language (WSDL) is used to describe the WebSphere Process Choreographer interface, the service, and parts of the binding. The specification can be found at:

http://www.w3.org/TR/wsd1

Within the binding you will find protocol specific references to, for example, the SOAP, EJB, or JMS specification.

The typed system that is used in WebSphere Process Choreographer is the XML Schema Definition (XSD). The description of the data types that can be used as build-in-types for the parts can be found at:

http://www.w3.org/TR/xsdlschema-2/
Figure 9-7 shows the process interface used in this scenario. The process consists of an input message that contains the message parts needed to start the process (partNo and qty) and an output message that contains the message parts the process will return (confNo, bestWholesale, and bestWholesaleDays). This interface definition matches the business requirements identified in “Business process model” on page 150.

Interruptible verses non-interruptible processes

The WebSphere Process Choreographer uses the following terminology:

- A short-running process is referred to as a non-interruptible process.
- A long-running process is referred to as an interruptible process.

Non-interruptible processes

- Are short-running process flows that are composed of multiple operations combined to form a single business operation
- Run in a global J2EE transaction, which means that you must take care of resources that are not two-phase-commit capable
- Cannot have asynchronous activity implementations or activity implementations that involve human interaction (For example, you cannot wait for an event or a staff activity because this non-interruptible process is executed in one single transaction.)
- Run in a single thread with all activities sharing the same thread context
- Maintain no state during execution

Interruptible processes

- Are long-running process flows that are composed of business services
- Perform every activity in its own transaction
- Support all types of activities (including events, human interactions, and asynchronous invocations)
- Are not as fast as non-interruptible processes because they interact with queues and a database to reliably hold the state of the process
- Support compensation
- Are completely forward recoverable

**Choosing the correct process type**

If we take a look at these two alternatives, a non-interruptible process is the best fit for this scenario. It is a process with no human interaction and does not require the persistence offered by an interruptible process. A non-interruptible process offers all of the functionality we need, and it allows us to avoid the performance impact of interruptible processes.

**Note:** The Parallel Process pattern behavior (a call to two or more activities simultaneously) is only supported by interruptible processes.

Non-interruptible processes operate in a single thread, so each parallel branch is executed in a serial manner. The order in which each branch is executed cannot be predicted at development time.

Interruptible processes use an internal message queue. Messages for each activity to execute in parallel are sent to this queue. A message-driven bean listens for new messages on this queue and processes them. This bean has a thread pool associated with it. If there are enough free message-driven bean threads, each of the activities on the queue will be invoked in parallel.

**XSLT Transformer verses Java Snippets**

For the data mapping between activities, you could either use a Transformer service (based on XSLT) or write your own Java snippets to perform data mapping between process variables.

The Transformer service offers the following advantages:
- A graphical drag-and-drop interface for creating mappings
- No Java coding required
- Automatic data conversion

The Java snippet approach offers the following advantages:
- Significantly more performant than Transformer services
- Supports the ability to manipulate process variables beyond simple data mapping
- Able to send data to the System.out output stream
Although the attraction of avoiding Java code development may appeal to some, we recommend using Java snippets in the majority of instances to benefit from the increased performance and the ability to add messages to the output stream that are useful for debugging and logging.

**Usage of a rules engine**

The use of a rules engine allows you to define and change decision points without modifying the process template. Rule engines are described in “Rules engines” on page 157.

A rule engine can be implemented in a WebSphere Process Choreographer process by using the Business Rules Engine shipped with the WebSphere Application Server Enterprise V5. Its architecture is illustrated in Figure 9-8.

As you can see in Figure 9-8, the business rule engine consists of three parts:

- The Rule Client is represented as a session EJB.
- The BRBeans are EJBs that provide the business rule persistence and provide query functions for the rules.
- The Rule Implementors provide the implementation of the business rules. As a recommendation you should reuse the already existing rules like RuleAnd, RuleOr, RuleEqual, RuleLessThan, and so on. You also can combine these rules. We will show you how to use the already existing rule RuleGreaterThan for our scenario in the development guidelines.

We use a rules engine in our second process iteration to externalize the business rule of when to query Wholesaler B for a delivery date.
9.4.2 Development guidelines

**Note:** We assume that the reader has built a WebSphere Process Choreographer process before, therefore, we have focused only on the steps that can lead to confusion and explained them in detail so far as they are relevant to our business scenario.

We used WebSphere Studio Application Developer Integration Edition V5.0.1 to create our processes.

**Naming conventions for processes in all scenarios**
A WebSphere Process Choreographer process can quickly become a confusing web of control links, activities, blocks, loops, and variables. We recommend defining a naming convention for each element type in a process to help clarify the process structure. We used the following naming conventions for all of the WebSphere Process Choreographer processes used throughout this redbook:

- **Name of Operation:** processOrder
- **Name of Input Message:** <Name of Input Operation> + Request
- **Name of Output Message:** <Name of Output Operation> + Response
- **Java Snippets:** pre_<Name of activity> and post_<Name of activity>
- **Blocks:** <Name of Activity>_Block
- **Loops:** <Name of activity>_Loop
- **Sub-process invocation:** <Name of process called>
- **Activities (Java, EJB and services):** <Name of operation used> in general and <Name of operation used>_WholesaleA and <Name of operation used>_WholesaleB in our particular scenarios
- **Variables:** <operation+WholesaleA/B+Request/Response>, for example GetDeliveryDaysWholesaleARequest.
- **Faults:** <Name of exception>
- **Service Project Names:** ITSO<Name of scenario>
- **Package Names:** com.ibm.itso.*

**Team programming considerations**
Keep the following points in mind as you build the process:
- Make sure that you use a versioning system like CVS or ClearCase® to store your processes and source code.
- Do not copy processes between different projects, because project-specific information is held in the FDML (Flow Definition Markup Language) file. If you want to reuse certain parts of a process, encapsulate it in a sub-process in a separate project.

Web services considerations
The process in this scenario uses a Web service developed in WebSphere Studio Application Developer V5.1. As such, the Web service uses the Web Services for J2EE support provided in WebSphere Application Server V5.0.2 and above.

The enterprise application for Wholesaler B contains references to the Web Services for J2EE classes. These class references cannot be resolved in the default WebSphere Studio Application Developer Integration Edition environment. Therefore, you must upgrade the environment to include webservices.jar from a WebSphere Application Server V5.0.2 installation in order to work with the Wholesaler B EJB.

Building the process
The following steps describe how to build the first iteration of the WebSphere Process Choreographer Serial Process implementation.

1. Create a business process named SerialProcess within a package named com.ibm.itso.process within a service project named ITSOSerialProcess.
2. Select the input node and generate the WSDL interface of the process.
3. Create the request and response messages according to the interface definition in the design guidelines (message structure and so on). Create a port type and a synchronous request-response operation, and associate this one with the messages that you have created. After all these activities, your process interface should look like Figure 9-9 on page 169.
Example 9-1 shows the WSDL that is generated for this process interface definition.

Example 9-1 Generated WSDL code for process interface

```xml
<message name="processOrderRequest">
  <part name="partNo" type="xsd:string"></part>
  <part name="qty" type="xsd:int"></part>
</message>

<message name="processOrderResponse">
  <part name="confNo" type="xsd:string"></part>
  <part name="bestWholesale" type="xsd:string"></part>
  <part name="bestWholesaleDays" type="xsd:int"></part>
</message>

<portType name="processOrderPortType">
  <operation name="processOrder">
    <input name="processOrderRequest" message="tns:processOrderRequest"></input>
    <output name="processOrderResponse" message="tns:processOrderResponse"></output>
  </operation>
</portType>
```

This scenario has both an input and output message because we defined our process to implement a synchronous request-reply operation called...
processOrder. In an asynchronous interruptible process, you would define a one-way operation to invoke the process.

4. Navigate to the Interface tab in the process editor and select the file SerialProcessInterface.wsdl.

5. Create a new variable based on the process order request message on the input node. Additionally, create a new variable for the output node based on the process order response message. You can delete the automatically generated input and output variables because they are not needed.

6. Outline the process with blocks and textual conditions (as shown in Figure 9-5 on page 162). Next, implement the repeating pattern: pre-Java snippet, activity, post-Java snippet in each block (adhering to the naming conventions) as shown in Figure 9-10.

![Figure 9-10 Data mapping pattern](image)

You could either use Java snippets or a visual XSLT transformer service to perform the process variable manipulation. The XSLT alternative requires no Java programming, but it is slower than the Java snippet transformation and less flexible.

7. As a reminder, Wholesaler A is represented by a Web service and Wholesaler B by an EJB invocation. We will use a static discovery of the Web service, so import the WSDL service description of Wholesaler A and the EJB class files (including the stubs) of Wholesaler B into the workspace.

8. The EJBs used by Wholesaler B make use of the Web services support in WebSphere Application Server V5.0.2. You need to upgrade your workspace to allow the compilation of this enterprise application. Copy webservices.jar from the lib directory of a WebSphere Application Server V5.0.2 installation to the lib directory of test environment of WebSphere Studio. By default, the lib directory of the test environment is located at:

C:\Program Files\IBM\WebSphere Studio\runtimes\base_v5\lib

Next, open the properties of the EJB project (ITSOTargetAppBEJB) and, in the Java Build Path, add the variable WAS_50_PLUGININDIR and extend it to lib/webservices.jar. Create a second pointer to WAS_50_PLUGININDIR and extend it to lib/qname.jar

Additionally, update the Web project (for example ITSOTargetAppBEJB_HTTPRouter) to add webservices.jar to the Java build path.
9. Change the invocation type of getDeliveryDays_WholesaleB to an EJB invocation with the following steps:

   a. Change the invocation type of getDeliveryDays_WholesaleB activity from an empty node to an Enterprise Java Bean invocation.

   b. Browse to the remote interface you have exported in the step before and choose the method you would like to invoke.

   c. After that, click **Apply** so that the input terminals for this method invocation are created for you (Figure 9-11). A new package is created with a WSDL file that includes the message types. Additionally there are Java classes created that are WSIF (Web Services Invocation Framework) message implementations. These are used in the Java snippets to pass data to the defined EJB method and to get and set the global data context.

![Figure 9-11 Generation of input terminals based on signature of EJB remote interface](image-url)
10. Now, right-click the activity and select **Assign Variable → Input → New**. You should see the window shown in Figure 9-12. Set the fields as shown and click **OK**. Also, create the variable for the output terminal.

![Add Variable](image)

**Figure 9-12** Creation of new variable based on input terminals

11. After that, you should see your input terminals associated with the corresponding created variables. A good place to look at the variable structure is the Outline view (when the process is open in the process editor). See Figure 9-13.

![Outline](image)

**Figure 9-13** Overview of the variable structure based on the message type
12. To complete the information needed by the WSIF EJB invocation, you need to find out the JNDI name of the home interface of the EJB you want to call. Just use the dumpnamespace tool provided by WebSphere Application Server to find it. In our case, it is `ejb/com/ibm/itso.ejb/wholesaleb/InventoryHome`. If you want to call an EJB on a system that is neither on the same box nor in the same cluster, you have to either use a local wrapper EJB that calls the remote EJB or change the bindings manually to directly invoke the remote EJB.

Enter the JNDI name as shown in Figure 9-14.

![Implementation](image)

**Figure 9-14 Completed EJB activity with JNDI name and variables created**

13. Now we have to use the Java snippet before the actual EJB activity invocation to set the required data. Use the code shown in Example 9-2.

**Example 9-2 Java snippet to pass data to actual invocation**

```java
ProcessOrderRequestMessage processOrderReqMsg = getProcessOrderRequest();
com.ibm.itso.ejb.wholesaleb.Inventory_msg.GetDeliveryDaysRequestMessage
getDelvReqB = getGetDeliveryDaysWholesaleBRequest();

getDelvReqB.setPartNo(processOrderReqMsg.getPartNo());
setGetDeliveryDaysWholesaleBRequest(getDelvReqB);
```

This is also a repeating pattern:

- Get the request message (message A) you want to retrieve data from (Remember, it is a global data context.)
- Get the request message (message B) you want to pass data to
- Set the necessary data in message B from message A
14. Repeat these steps for the Web services invocation getDeliveryDays of Wholesaler A. The only difference is that you do not point to an EJB home interface, but to a WSDL interface description.

a. View the source of the WSDL file for Wholesale A. Note that the service location points to:

   http://wholesalera:9080/ITSOTargetAppAEJBRouter/servlet/rpcrouter

   The wholesalera part of the URL must map to the IP address of the WebSphere Application Server server instance where the Wholesaler A enterprise application is installed. Use the Windows hosts file to map wholesalera to this IP address.

b. Create the input terminals, variables and snippets.

15. The next step is to add our first transition condition in this scenario (as in Figure 9-15).

16. First of all, we have added a textual description of the transition condition to get an overview of the business process. Now, we are going to add a concrete Java transition condition that checks whether the delivery days of Wholesaler A is less than or equal to seven days. Right-click the control link and choose Properties. Then, set the transition condition as a Java transition condition (out of the following alternatives: built-in, true, false or otherwise). See Figure 9-16 on page 175.
Finally, add the transition code as shown in Example 9-3.

**Example 9-3  Java transition condition source example**

```java
int deliveryDaysA = getGetDeliveryDaysWholesaleAResponse().getDeliveryDays();
ProcessOrderResponseMessage processOrderRespMsg = getProcessOrderResponse();

if (deliveryDaysA <= 7) {
    processOrderRespMsg.setBestWholesaleDays(deliveryDaysA);
    processOrderRespMsg.setBestWholesale("A");
    result = true;
} else {
    result = false;
}
```

17. Set a built-in transition condition of otherwise from `getDeliveryDays_WholesaleA_Block` to `getDeliveryDays_WholesaleB_Block`. That means that if no other transition condition evaluates to true, the otherwise path is chosen. For our business scenario, it means that if Wholesale A needs longer than seven days to deliver the goods, Wholesale B is also asked.

18. Set the Java transition condition shown in Example 9-4 from `getDeliveryDays_WholesaleB_Block` to `placeOrder_WholesaleB_Block`.

**Example 9-4  Comparison of two delivery days; access to global data context.**

```java
int deliveryDaysA = getGetDeliveryDaysWholesaleAResponse().getDeliveryDays();
int deliveryDaysB = getGetDeliveryDaysWholesaleBResponse().getResult();
ProcessOrderResponseMessage processOrderRespMsg = getProcessOrderResponse();

if (deliveryDaysB < deliveryDaysA) {
    processOrderRespMsg.setBestWholesale("B");
```
processOrderRespMsg.setBestWholesaleDays(deliveryDaysB);
result = true;
else {
    processOrderRespMsg.setBestWholesaleDays(deliveryDaysA);
    processOrderRespMsg.setBestWholesale("A");
    result = false;
}

19. Set a built-in transition condition of otherwise from getDeliveryDays_WholesaleB_Block to placeOrder_WholesaleA_Block.

20. Complete the process regarding the placeOrder activities and set the necessary messages. See the variable outline to determine which parts are necessary.

21. We need to deal with exception handling. The placeOrder EJB and Web service activities both expose a red fault terminal that will be navigated if a certain exception is thrown. This fault terminal must be mapped with a control link. For now, send the fault terminal to a Java snippet. (Although we could have used a fault node to expose the fault outside of the block.) Handle the fault terminal for both placeOrder blocks. WholesaleB is shown in Figure 9-17.

![Figure 9-17 Exception handling for PartNotAvailableException](image)

22. In the PartNotAvailableException Java Snippet, shown in Figure 9-17, we set the confirmation number to FAILED. Note you cannot name both Java snippets PartNotAvailableException because activity names within a process must be unique.

**Extending the scenario by externalizing business rules**

In the second design iteration, we saw that it is a better approach to externalize the business rules within our process to Business Rule Beans provided by WebSphere Application Server Enterprise V5 under certain circumstances.

For detailed instructions on setting up the rule browser and deploying enterprise applications with business rule beans, see the redbook *WebSphere Application Server Enterprise V5 and Programming Model Extensions*, SG24-6932.
The difference in our approach from the other redbook is that we use the predefined Java implementor rules instead of creating our own.

Example 9-5 shows the method of the session EJB that is used to implement the Wholesale client. The method does not use caching for test purposes, and it is tested to work with the RuleGreaterThan Java rule implementor.

Example 9-5  Rule client example for business rule bean

```java
public boolean isAskingAnotherWholesaleRequired(Integer deliveryDays) {
    boolean result = false;
    try {
        // create new trigger point
        TriggerPoint tp = new TriggerPoint();
        tp.setCombiningStrategy(CombiningStrategy.RETURN_FIRST, TriggerPoint.ALL_RULES);
        tp.disableCaching();

        Object[] pList = new Object[1];
        pList[0] = deliveryDays;

        String ruleName =
        "com/ibm/itso/brb/rules/AskingAnotherWholesaleRule";

        // call the rule
        Object resultObject = tp.trigger(null, pList, ruleName);

        // get the result (wrapped in a ConstraintReturn class!)
        if (resultObject != null) {
            if (resultObject instanceof ConstraintReturn) {
                ConstraintReturn constraintReturnObj = (ConstraintReturn) resultObject;
                result = constraintReturnObj.result;
            }
        }
    } catch (Exception e) {
        // do exception handling
        return result;
    }
}
```

Next, we create a rule called AskingAnotherWholesaleRule in the Rule Browser shipped with WebSphere Application Server Enterprise V5. See Figure 9-18 on page 178.
In Figure 9-19, you can see the details of the rule where you can define the period of time in which this rule is valid.

In Figure 9-20 on page 179, you can see the details of the shipped Java rule implementor that is used:

```java
com.ibm.websphere.brb.implementor.RuleGreaterThan
```

You also can see the delivery days that we have defined (seven days). Therefore, the rule checks whether the value of the parameter that gets passed to the rule implementor is greater than seven days. If this is the case, a boolean value of true is returned.
Many different rules are already provided, including:

- RuleAND
- RuleOR
- RuleConstant
- RuleConvert
- RuleIfThenElse
- RuleEqual
- RuleGreaterThan
- RuleGreaterThanEqual
- RuleLessThan
- RuleLessThanEqual

### 9.4.3 Runtime guidelines

These guidelines give you an overview of how to test your developed processes. They also provide you with business scenario-specific information about runtime issues.

**Deploying a process**

Before you are able to run a process in WebSphere Application Server Enterprise, you need to deploy it as an enterprise application. This deployment process serves the following two purposes:

- Packaging the process and service project into an enterprise application that can be installed as an EAR file in WebSphere Application Server Enterprise.
- Creating an inbound binding interface that can be used to start the process instance
When you deploy a process, you are required to enter the inbound binding type even if you do not ultimately intend to start the process using the inbound binding. The three inbound binding types are EJB, SOAP, and JMS.

**Starting a process instance**

Once a process has been deployed and installed in WebSphere Application Server Enterprise, you can create process instances from it in the following ways:

- **Process Web Client**
  
  The Process Web Client is an application supplied with WebSphere Application Server Enterprise that allows you to start, view, monitor, and terminate process instances. Additionally, it can be used to manipulate work items generated by staff activities. The Process Web Client can also be customized using JSPs.

- **Inbound binding**
  
  An inbound binding is always created during deployment, regardless of whether you intend to use it or not. Each inbound binding offers a different facade to the process:
  
  - An EJB binding contains a session bean facade, with methods to start a process and, if applicable, send events.
  
  - A SOAP binding contains a servlet that sits in front of the session bean facade and processes SOAP requests to work with processes.
  
  - A JMS binding contains a message-driven bean facade which listens on a given queue for messages.

- **WebSphere Process Choreographer API**
  
  An API is provided to work directly with process instances. The API contains two interfaces: a session bean and a message-driven bean. Using either of these interfaces, you can work with process instances, work items, and process variables. The APIs allow you to effectively build your own Process Web Client or create clients to perform functions such as sending events (which is a feature not provided by the Process Web Client). API features include:
  
  - Starting and terminating a process instance
  
  - Cancelling an activity
  
  - Forcing the completion of an activity
  
  - Retrying the execution of a stopped activity
  
  - Managing work lists
EJB as the inbound binding type

We choose to deploy the processes developed in this scenario using an EJB inbound binding. The choice of inbound binding is, for us, ultimately irrelevant because we will use the Process Web Client to test the process. The Generate Deploy Code wizard is shown in Figure 9-21.

![Generate Deploy Code](image)

Figure 9-21   Deployment choices regarding the inbound binding type

So what gets generated to provide the EJB inbound binding? Beside the WSDL interface definition, which is used to capture the messages that are going to be sent to the process, and the parts, which made up a message, there also are, a WSDL service definition and a WSDL binding definition.

The service and binding WSDL files are generated by the WebSphere Studio tooling. The service file imports the binding file and the binding file imports the interface file. Ultimately, then, the WSDL definitions are split between three files to accommodate role considerations. See Example 9-6 on page 182.
Example 9-6  WSDL service file - EJB inbound binding

```
<service name="processOrderPortTypeService">
  <port binding="binding1:processOrderPortTypeEJBBinding"
       name="processOrderPortTypeEJBPort">
    <ejb:address
      className="com.ibm.itso.process.ProcessOrderPortTypeServiceHome"
      jndiName="com/ibm/itso/process/processOrderPortTypeServiceHome"/>
  </port>
</service>
```

The service is defined in the WSDL specification as “a collection of related endpoints,” while the port is a “single endpoint defined as a combination of a binding and a network address”. In an EJB binding, this is the package and class name of the EJB home interface and the JNDI name that can be used to call the EJB.

Example 9-7 shows the WSDL binding that gets generated.

Example 9-7  WSDL binding file: EJB example

```
<binding name="processOrderPortTypeEJBBinding"
         type="interface1:processOrderPortType">
  <ejb:binding/>

  <format:typeMapping encoding="EJB" style="Java">
    <format:typeMap formatType="int" typeName="xsd:int"/>
    <format:typeMap formatType="java.lang.String" typeName="xsd:string"/>
  </format:typeMapping>

  <operation name="processOrder">
    <ejb:operation
      methodName="processOrder" parameterOrder="partNo qty"
      returnPart="confNo"/>
    <input name="processOrderRequest"/>
    <output name="processOrderResponse"/>
  </operation>

</binding>
```

The binding is a concrete protocol and data format specification for a particular port type. For example, there is a type mapping between the XSD and Java types defined, as well as the mapping between the abstract operation and the concrete method that is used.
The third artifact that gets generated is the stateless session EJB itself that a client can use to call the process. The generated EJB remote interface simply takes the part number and the quantity as defined in the process interface and returns the ProcessOrderResponseMessage whose structure we have defined in the design guidelines (Example 9-8).

**Example 9-8  Remote interface of generated stateless session ejb**

```java
public interface ProcessOrderPortTypeService extends javax.ejb.EJBObject {

            java.lang.String argPartNo, int argQty)
            throws java.rmi.RemoteException;
}
```

**Using the Process Web Client to test the process**

**Note:** If you intend to run your processes in WebSphere Application Server Enterprise V5.0.2, we recommend installing each process on this server directly for testing rather than using the test environment in WebSphere Studio Application Developer Integration Edition V5.0.1.

The test environment uses a WebSphere Application Server Enterprise V5.0.1 runtime. WebSphere Process Choreographer and the Process Web Client were significantly enhanced in WebSphere Application Server Enterprise V5.0.2.

Additionally, the use of Web Services for J2EE support in the Wholesaler A and B enterprise applications means that a WebSphere Application Server V5.0.2 environment is required to run these enterprise applications.

After you have installed the EAR file in WebSphere Application Server Enterprise, you can test it using the following URL:

```
http://<hostname>:9080/bpe/webclient
```

This starts the WebSphere Process Choreographer Process Web Client See Figure 9-22 on page 184.
This is the new Struts-based Web client in WebSphere Application Server Enterprise V5.0.2, which allows you to test your project and take advantage of the new client capabilities. To get an overview of how you can define your own JSPs, please refer to:


If you select Process Template Lists → My templates you should see a list of all available process templates, as shown in Figure 9-23.
If you start our SerialProcess template according to the process interface, a user interface gets generated that can be used for testing the process, as in Figure 9-24. This Web client can be adapted using customized JSPs.

Test the process by entering a part number and quantity. To aid with testing, you can determine the delivery days of each wholesaler by interpreting the part number entered. The rules for Wholesaler A are:

- If the last character of the part number is a Z, then a PartNotAvailable fault is returned.
- Otherwise the first character of the part number is converted into a number if necessary (for example, A to 1), and this value is returned as the delivery days.

Therefore a part number of BCDE would return delivery days of 2, and a part number of BCDEZ would return a PartNotAvailable fault.

Wholesaler B uses similar logic, except it checks if the second to last character is a Z (if so, PartNotAvailableException is returned); otherwise, the second character of the part number is converted to a number and returned.

This logic is shown by example in Figure 9-25 on page 186.
You can test the different paths in the process by sending differing part numbers.

**Note:** At the time of writing, there is a problem in mapping Web service SOAP fault messages to activity fault terminals in WebSphere Process Choreographer. Therefore, the process will fail if the Wholesaler A placeOrder Web service returns a fault. At the time of writing, this matter was being addressed by the WSIF service team.

The result of our request using part number 98WLANCARD is shown in Figure 9-26.
Tracing
If something goes wrong while testing a process instance, it is helpful to enable tracing of the WebSphere Process Choreographer process engine and the WSIF providers. To turn this on in WebSphere Application Server Enterprise, perform the following:

1. Select **TroubleShooting → Logging and Tracing** in the Administrative Console. Click the server you wish tracing to be enabled on (most likely server1), then click **Diagnostic Trace**.

2. Click **Runtime**. This will ensure that any changes made to tracing will apply without restarting the server. Check **Save runtime changes to configuration as well** if you want these settings to apply every time the server is started.

3. The Trace Specification section allows you to specify which classes to trace. To turn on tracing for the process engine and WSIF classes, enter:

   ```
   com.ibm.bpe.*=all=enabled:org.apache.wsif.*=all=enabled
   ```

4. Note the location where the trace file will be output, as specified in the File field. See Figure 9-27.

5. Click **Apply**, and then save your changes.

![Figure 9-27  Trace settings](image)
6. Use the Process Web Client to run another process instance. Then, check the trace output in the directory and file specified. (By default, this will be server1\logs\trace.log.)

Audit Trail
There is a statement in the business requirements of this scenario that the process needs to have an audit trail so that important events are logged. WebSphere Process Choreographer allows you to configure which information is written to the audit trail based on a process or on activities. In Figure 9-28 you can see how you can enable the audit trail for the complete process in the server settings of the process editor.

![Figure 9-28 Process audit trail](image)

If you want to have particular activities included in your audit trail, mark these in the properties of the corresponding activity in the server settings, as in Figure 9-29.

![Figure 9-29 Activity audit trail](image)

The output of the audit trail is written into a database table called audit_t_log. It provides information about different types of events so that you can see which path through a process was navigated at a certain time.
This information is only accessible using SQL or a tool that provides a view into a database like the Control Center for DB2 or TOAD for Oracle.

The semantics of the database table are described in the WebSphere Application Server Enterprise Infocenter.

**Deployment of a rule-based process alternative**

Make sure that you have configured the environment (database, environment variables, and so on) as described in Chapter 10 of the redbook *WebSphere Application Server Enterprise V5 and Programming Model Extensions*, SG24-6932. During the installation of the process enterprise archive (including the business rule beans and the rule client), deploy the EJBs and enter the schema for the CMP EJBs. Then, enter the JNDI names for the datasource (2.0 and 1.1), and assign the security roles gathered in the enterprise applications to your user realm.

**General runtime guidelines**

If you want to uninstall an enterprise application containing a business process from WebSphere Application Server Enterprise in your development or test environment, do the following:

1. Check whether the process template has any processes instances still running. If so, try to find the reason why the process has not completed. For example the activity may be in a stopped state instead of a finished state (meaning a problem occurred). Try to resolve the issue and restart the activity so that the process can continue and end properly. Once all process instances are stopped, delete them.

2. Stop the process template in the WebSphere Application Server Enterprise Administration Console.

3. Stop the enterprise application.

4. Uninstall the enterprise application.

**Monitoring SOAP messages**

If you want to monitor the SOAP messages sent to a Web service (such as those sent to Wholesale A), you can use a SOAP monitor that is provided in WebSphere Application Server Enterprise V5. See Figure 9-30 on page 190.
9.5 WebSphere MQ Workflow guidelines

This section describes how to implement the Serial Process pattern to create a process that meets the requirements set by the ITSO Electronics business.
process model. It uses the Runtime pattern and Product mappings described in “Serial Process Product mappings” on page 93 for WebSphere MQ Workflow.

You can download the completed process and run it in WebSphere MQ Workflow or view it in WebSphere Business Integration Workbench. See “Sample scenarios setup” on page 398.

9.5.1 Design guidelines

This section discusses WebSphere MQ Workflow specific design guidelines to use when creating a process that conforms to the Serial Process pattern.

Process overview

The process meets the business objectives of ITSO Electronics as defined in “Stage One: Internal ordering on demand” on page 110. The completed process is shown in Figure 9-31.

![Serial process overview](image)

The process has the following characteristics:

- Serial execution path
- Contains only automated activities

The process implementation makes calls to two systems external to the process manager: Wholesale systems A and B. Both of these systems expose business logic using a Web service interface.

The Web service operations are invoked using the WebSphere MQ Workflow Web Services Toolkit. This toolkit is provided in SupportPac™ WA07.

Designing processes

Before you modeling a process, there are several things to consider:

- Identifying processes in an organization
Selecting a process or processes to be created or to be re-engineered
Consulting with people who understand a process or processes about the scope and process flow
Documenting the collected information from them
The collected information includes key elements to be used in modeling a process. These are:
- Organization unit where a process is performed
- Role or staff information to decide who is doing what
- Activity or task that represents the work performed in a business process
- Interface of each activity that can be considered as input and output
- Decisions and choices that are used to determine a routing path during the execution of a process
Applications that are regarded as executable forms such as programs
Cost-related information including elapsed time of each task, cost rate of roles, and other resources

**Process interface definition**
Process interfaces are described in data structures and data fields in WebSphere MQ Workflow. A process will contain:
- A process input data structure (or input container) that contains data fields relevant to starting a process instance
- A process output data structure (or output container) that contains data fields relevant to the completion of a process instance

For our scenario, we define two data structures (as in Figure 9-32 on page 193):
- OrderInput, which contains data fields for the part number to order and the quantity
- OrderOutput, which contains data fields for the confirmation code of the order, the name of the wholesaler who fulfilled the order, and the number of days until the order is scheduled to be delivered
Task definition
A process in WebSphere MQ Workflow primarily consists of a collection of connected tasks. These tasks can be associated with an application or role, can be automated or manual, and can have properties such as notification and expiration. A User Program Execution Agent (UPES) can be assigned to a task to invoke an underlying application.

There are two types of automated tasks used in this scenario:

- Web services tasks
- NOOP tasks

**Note:** Task is the terminology used in WebSphere Business Integration Workbench to mean a component in a process. This is equivalent to the term activity.

**Web services tasks**
Our scenario contains tasks that execute business logic on the Wholesaler A and B systems. Both Wholesaler A and B expose this business logic using a Web service with the following operations:

- getDeliveryDays, which returns the number of days a specific part can be delivered in
- simplePlaceOrder, which places an order for a specific part and returns a confirmation code
cancelOrder, which cancels a placed order (This operation is not used in this scenario.)

These Web service operations are invoked using the WebSphere MQ Workflow Web Services Toolkit. The toolkit acts as a Web services UPES, sending and receiving SOAP over HTTP messages.

In our scenario, tasks are implemented for getDeliveryDays and simplePlaceOrder for both Wholesalers A and B.

**NOOP tasks**

A NOOP task (short for NO OPeration) is a dummy task used to store intermediate data. It is implemented by the FMCINTERNALNOOP program and effectively performs no function. A data structure is assigned to the NOOP task, and other tasks use the NOOP task to get and set data fields within this data structure.

There are two types of NOOP tasks:

- **Default NOOP task**
  
  This task is optionally automatically added as the first task in a process. It uses the input data structure of the process for its input and output. It is connected to all starting tasks and objects within a process.

- **User-defined NOOP task**

  This task is added manually by the user. Any data structure can be assigned as the input and output of the process. Other tasks can be manually connected to this NOOP task to get and set data fields within the assigned data structure.

In this scenario, we use user-defined NOOP tasks to store a data structure. This data structure is referenced by decision objects and the output phi.

**Task interface definition**

Each task in a process will use one or more data structures. Our scenario contains tasks that are implemented by Web service request-response operations. Therefore, each task has an input and output data structure assigned to it based on the WSDL message and part definitions of the Web service operation. The WebSphere MQ Workflow Web Services Toolkit creates these data structures.

Figure 9-33 on page 195 shows the data structures created for each Web service operation.
The NOOP task contains a data structure with fields to record the delivery days returned by Wholesaler A and B, along with the name of the best Wholesaler, as in Figure 9-34 on page 196.
**General guidelines when using tasks**

In considering a WebSphere MQ Workflow task that invokes an application, there are certain points to evaluate both from a modeling standpoint, user interaction, failure recovery, and retention of certain application data. The following list highlights some of these points:

- It is not typical to use UPES activities without any human intervention in case of failure. There is usually a human interaction activity for each UPES activity. The human interaction activity is enabled only if there is an error or a timeout in the UPES activity.

- You could also use UPES activities without these human intervention activities, but administration of this implementation is harder to manage. UPES failures and UPES non-responses are more difficult to detect because there is no event to notify someone of the failure. Even when the failure or timeout is noticed, it is difficult to retry the UPES request. Also, safeguards must be taken to ensure that the original WebSphere MQ message is not consumed from the queue before a response has been successfully placed on the reply to queue.

- Historical records of the process or activity input data (such as the order quantity and part number for ITSO Electronics) can be surfaced in the WebSphere MQ Workflow audit trail in a couple of different ways:
  
  a. The process context can be utilized to contain both the part number and the quantity.

  b. A global container can be defined that contains not only the quantity and part number, but also the output data structure members of best wholesaler, best wholesaler days, and the confirmation number.

---

**Figure 9-34** Data structure assigned to the NOOP activities

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancelOrderRequest</td>
<td>Structure</td>
</tr>
<tr>
<td>cancelOrderResponse</td>
<td>Structure</td>
</tr>
<tr>
<td>Default Data Structure</td>
<td>Structure</td>
</tr>
<tr>
<td>getDeliveryDaysRequest</td>
<td>Structure</td>
</tr>
<tr>
<td>getDeliveryDaysResponse</td>
<td>Structure</td>
</tr>
<tr>
<td>OrderInput</td>
<td>Structure</td>
</tr>
<tr>
<td>OrderOutput</td>
<td>Structure</td>
</tr>
<tr>
<td>placeOrderRequest</td>
<td>Structure</td>
</tr>
<tr>
<td>placeOrderResponse</td>
<td>Structure</td>
</tr>
<tr>
<td>Wholesale</td>
<td>Structure</td>
</tr>
<tr>
<td>daysA</td>
<td>Long</td>
</tr>
<tr>
<td>daysB</td>
<td>Long</td>
</tr>
<tr>
<td>bestWholesaler</td>
<td>char</td>
</tr>
</tbody>
</table>
Both the global container and the process context can be written to the WebSphere MQ Workflow audit trail.

**Web Services Toolkit**

The IBM WebSphere MQ Workflow Web Services Toolkit for dynamic-business (referred to in this redbook as the Web Services Toolkit) provides an environment to integrate Web services with business processes.

The toolkit can be in used in two ways:

- To define tasks that are implemented by Web service operations: These tasks use the UPES supplied with the toolkit to communicate between the WebSphere MQ Workflow runtime and the Web service operations.
- To expose a WebSphere MQ Workflow process as a Web service (known as a Flow Service): Web service clients can then create and work with instances of the Flow Service.

The toolkit is supplied with WebSphere MQ Workflow SupportPac WA07, which can be downloaded at:

http://www.ibm.com/software/integration/support/supportpacs/

The toolkit is a J2EE Web application and requires an application server that supports the Java Servlet V2.2 or later specification (such as WebSphere Application Server V5). It is based on the Apache AXIS V1.1 SOAP implementation.

In our scenario, we use the toolkit to implement tasks that communicate with the Wholesaler A and B Web services.
Staffing and organization models

The staffing and organization models play a key part in the modelling of a business process in WebSphere MQ Workflow. These aspects are used to assign the scope of the work and assign responsibility of tasks.

This scenario defines three organizational units:

- **Retail**, which is the organization executing the business process
- **Wholesaler A**, which offers services to get the delivery days for a particular part and to place an order for that part
- **Wholesaler B**, which is a separate wholesaler who offers the same services as Wholesaler A

The staffing model of each organization can also be modeled. In this scenario, we choose not to use Web service operations with faults. The placeOrder operation does return a fault, so we have created a simplePlaceOrder operation that takes the same input and output without returning a fault.

Mapping between tasks

For WebSphere MQ Workflow, the data flow between tasks has to be defined or mapped. A Phi object is used to map data field elements from one data structure to data field elements of another structure. The mappings that we must consider are:

- From the process input (source) to each task, so that the task can use the input data

Note: Careful consideration needs to be given to using the Web Services Toolkit with Web service operations that thrown faults. At the time of writing, if a fault is returned from a Web service operation during the execution of a task, an exception message is sent from the Web Services Toolkit. Additionally, the task is set to the In Error state. The fault message is not mapped to the relevant data field in the output data container.

Therefore, we do not recommend using WSDL faults to communicate business exceptions with the Web Services Toolkit. For the purposes of our scenario, we choose not to use Web service operations with faults. The placeOrder operation does return a fault, so we have created a simplePlaceOrder operation that takes the same input and output without returning a fault.

Support for mapping WSDL faults to task output data containers will be added to a later release.
From the result of a task to other tasks, so that the other task can use the result data to perform activities and decisions

From previous tasks to the final task (sink), to collect results of the process

**Control objects**
Control objects are used to send control down one of multiple process paths. The control object contains a logical decision that determines which path is followed. Several decisions need to be made in this scenario:

- Check whether the delivery days of Wholesaler A is within a week. If the delivery days returned by Wholesaler A are equal to or less than seven, the best wholesaler is A. Control should be passed to the placeOrder task for Wholesaler A.
- If the delivery days of Wholesaler A are greater than a week, we need to retrieve the delivery days from Wholesaler B, and compare both values to determine which wholesaler can fulfill the order most quickly. The relevant placeOrder task should be executed.

### 9.5.2 Development guidelines
This section describes the development points that affect the WebSphere MQ Workflow process model, interaction with users, and type of interfaces both from a WebSphere MQ Workflow internal and external perspective. Some of the development points apply to any WebSphere MQ Workflow process. Others are specific to the interaction of the retail department and wholesale departments in our scenario.

**Modeling tool**
A WebSphere MQ Workflow process is described in an FDL format. There are a number of different modeling tools that can create FDL processes, including:

- **WebSphere MQ Workflow Buildtime**
  This modeling tool is supplied with WebSphere MQ Workflow and is used to create WebSphere MQ Workflow specific processes.

- **WebSphere Business Integration Workbench**
  This is an industry-independent tool that allows realistic and visual modeling of a business process. It includes support for FDL and WebSphere MQ Workflow. In addition to process modeling, the tool provides analysis, animated simulations, and detailed reporting of processes.

The WebSphere Business Integration Workbench provides a relatively generic business and technical view of a process, while the WebSphere MQ Workflow
Buildtime is aimed at people with a good understanding of WebSphere MQ Workflow.

In this redbook, we will use WebSphere Business Integration Workbench V4.2.4 to model our processes for WebSphere MQ Workflow. For more information see “WebSphere Business Integration Workbench” on page 80.

For detailed information about how to use WebSphere Business Integration Workbench to model a process, consult Continuous Business Process Management with HOLOSOFX BPM Suite and WebSphere MQ Workflow, SG24-6590.

Creating the serial process scenario
In order to build a process implementation for the serial process scenario, the following steps must be taken:

1. Install and configure the Web Services Toolkit with the WSDL of each Web service that will be used.
2. Import the Web service processes and data structures to the WebSphere Business Integration Workbench repository.
3. Build the process in WebSphere Business Integration Workbench.

This section describes how to perform these steps.

Installing and configuring the Web Services Toolkit
As described in “Web Services Toolkit” on page 197, we have selected the WebSphere MQ Workflow Web Services Toolkit (WA07) to integrate with the Web service operations. This section describes how to install and configure the toolkit.

Installing the Web Services Toolkit
The WA07 SupportPac provides detailed information about how to install the toolkit. This section describes how we achieved a successful installation:

1. Install the toolkit enterprise application into WebSphere Application Server.
   a. Locate mqwfwstk.ear in the WA07 SupportPac and use the WebSphere Administrative Console to install it (Applications → Install New Application).
   b. During the installation, you are required to check the checkbox to map the Web module to default_host as shown in Figure 9-35 on page 201.
You need to add the JAR file fmcojagt.jar to a container-wide shared library. To do this, add a shared library and assign it to a classloader.

2. Click Environment → Shared Libraries in the Administrative Console.

b. Define a new shared library for fmcojagt.jar. You will find this JAR file in the bin/Java3xxx directory of your WebSphere MQ Workflow installation. Additionally, set the native library path to the bin directory of your WebSphere MQ Workflow installation. See Figure 9-36.

![Shared Libraries](image-url)
c. Click **Servers → Application Servers → [server name]** and under the Settings section click **Classloader**.

d. Define a new classloader, and add a library reference to this classloader for fmcojagt. See Figure 9-37.

![Figure 9-37 Setting the fmcoagt shared library for a classloader](image)

3. The toolkit has a large list of prerequisite classes that it needs to function. These classes are listed in the WA07 documentation. The majority of the classes are supplied by the toolkit Web application and WebSphere Application Server. However, you also need to add fmcoutil.jar to the toolkit’s Web application classpath. This JAR file is located in the bin directory of your WebSphere MQ Workflow installation. You can either follow the process above to create a shared library and add it to the classloader, or you can simply copy the file into the Web application lib directory (WebSphere\AppServer\installedApps\<host_name>\mqwfwstk.ear.ear\mqwf wstk.war\WEB-INF\lib).

**Note:** JAR files added to the WEB-INF\lib directory of an enterprise application will be deleted if the enterprise application is reinstalled.

4. The toolkit also requires the Apache AXIS API V1.1 JAR file. This file is not supplied with the SupportPac. Download axis-1_1.zip it from the Apache AXIS Web site:

   http://ws.apache.org/axis/download.cgi

   Extract axis.jar from this ZIP file and place it in the WEB-INF\lib directory of the toolkit Web application, or create a shared library for it.

5. Start the mqwfwstk.ear enterprise application in the Administrative Console. There should be no errors during startup in the SystemOut.log file.
Verifying the Web Services Toolkit Installation

Start the Web Services Toolkit application with the following URL:

http://localhost:9080/mqwfwstk

A Web page should display, giving more information about the toolkit. Check to ensure that the toolkit has all of the required Java classes in the classpath by clicking on the Installation Verification link, as in Figure 9-38.

Figure 9-38 Needed components for the toolkit

The toolkit will inform you whether you have all needed components and where these components were found. If any of the needed components are missing, the Web page will tell you where to find these classes. Optional components (such as XML security) are not mandatory.

Configuring the Web Services Toolkit

The toolkit configuration allows you to configure the names of queues, queue managers, and methods of access to these objects. Refer to Figure 9-39 on page 204.
For a full description of each of the configuration options, see the WA07 documentation. There are three sections to the interface:

- **WebSphere MQ Workflow section**
  
  We select the default queue manager (FMCQM) and default queue (EXEXMLINPUTQ). EXEXMLINPUTQ is the queue that WebSphere MQ Workflow uses for XML messages.

- **Flow Service Provider section**
  
  Maps deploy Flow Services to the WebSphere MQ Workflow process templates. The Flow Service provider transforms messages from WebSphere MQ Workflow XML format to those expected by the Web Service, which are in a SOAP format. Flow Service also transforms the Web Service SOAP format messages into the WebSphere MQ Workflow XML format for the responses.

- **Web Service UPES section**
  
  This section invokes the Web service operations based on WebSphere MQ Workflow XML messages from the process activity.

  You can elect to use an MQ or JMS pipe. If you select the JMS pipe, you must configure the JMS settings in WebSphere Application Server. The MQ pipe requires no additional configuration. Select the UPES version to match the version of WebSphere MQ Workflow you are using.
The WebSphere MQ Workflow Web Services Toolkit must be enabled to listen for messages on the specified queues that you desire for the Web service UPES. In the \textit{UPES is} row of the Web Services section, you must click \textbf{Enabled} before you execute the Update Configuration command.

\textbf{Tip:} The WebSphere MQ Workflow Web Services Toolkit will create the additional queues that you name in the Flow Service Provider and Web Service UPES sections of the configuration interface. The queue manager must already exist before you perform the Update Configuration command.

Once you have entered the configuration information, click \textbf{Update Configuration}. Refer to Figure 9-40, which shows the success message. You should also see two new queues created in the queue manager, by default FMCWFLOW and FMCWUPES.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure9-40.png}
\caption{Update configuration success message}
\end{figure}

We found during our testing of the toolkit that queues within an WebSphere MQ cluster give exceptions when the toolkit deploys while running the Update Configuration action. We found those exceptions logged in the WebSphere Application Server SystemOut.log file. If you encounter an WebSphere MQ exception in this system log, try to create the queues in a queue manager outside the WebSphere MQ Workflow queue manager cluster. We did this using remote queue definitions within the workflow FMCQM.

\textbf{Importing the WSDL Web service}

\textbf{Note:} Before executing this step, you should ensure that the Web service enterprise applications (ITSOTargetAppA and ITSOTargetAppB) are not installed in the WebSphere Application Server server instance. Once this step is complete, it is safe to install them.
The next step is to import the Web service WSDL files for Wholesaler A and B. Click **Web Services UPES Administration**.

To load the WSDL, click **Browse** and locate the Wholesaler A WSDL file. With the WDSL filename selected, click **Import Endpoints**. You should see a confirmation that the endpoint is deployed. Repeat this step for the Wholesaler B WSDL file. You should see both Web services, as shown in Figure 9-41.

![Imported WSDL documents](image)

**Figure 9-41** Imported WSDL documents

### Updating the mapping rules

After performing the configuration setups, deploying the FDL, and importing the WSDL files, the next stage is to update the mapping between the FDL names and the Web service names.

Click **Mapping Rules**. Figure 9-42 on page 207 shows the left navigation bar of the mapping rules screen. The endpoint classes (WSDL ports) and parameter types (WSDL messages) are displayed here.
Figure 9-42 Mapping rules navigation bar

Figure 9-43 The shows properties section of the mapping rules. The appropriate endpoint or parameter properties are shown here.

### EndPoint InventoryPort_A

Defined by: file:///C:/was/WebSphere/AppServer/installedApps/m23x2501/mcwfwstkcar.ear/mcwfwstkcar/WEB-INF/wsd/IInventory_WholesaleA.wsdl

<table>
<thead>
<tr>
<th>Endpoint Name</th>
<th>FDL Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>InventoryPort_A</td>
<td>InventoryPort_A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>FDL Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>simplePlaceOrder</td>
<td>PlaceOrderA</td>
</tr>
<tr>
<td>cancelOrder</td>
<td>CancelOrderA</td>
</tr>
<tr>
<td>placeOrder</td>
<td>placeOrder</td>
</tr>
<tr>
<td>getDeliveryDays</td>
<td>GetDeliveryDaysA</td>
</tr>
</tbody>
</table>
This mapping step allows you to map FDL resources to Web services resources. In the process it does the following:

- Maps Web service endpoints to FDL program names
- Maps Web service messages and parts with FDL structures and fields

We will change the name of the FDL program names to illustrate how mapping works. Update the FDL program names to match Table 9-1, and be sure to click Update Mapping Rule for each endpoint mapping.

**Note:** There is a placeOrder and a simplePlaceOrder operation for Wholesalers A and B. The placeOrder operation places an order and returns a fault if the order cannot be placed. The simplePlaceOrder operation also places an order, but it does not return a fault. We will only map the simplePlaceOrder operation.

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Operation Name</th>
<th>FDL Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>InventoryPort_A</td>
<td>getDeliveryDays</td>
<td>GetDeliveryDaysA</td>
</tr>
<tr>
<td>InventoryPort_A</td>
<td>simplePlaceOrder</td>
<td>PlaceOrderA</td>
</tr>
<tr>
<td>InventoryPort_A</td>
<td>cancelOrder</td>
<td>CancelOrderA</td>
</tr>
<tr>
<td>InventoryPort_B</td>
<td>getDeliveryDays</td>
<td>GetDeliveryDaysB</td>
</tr>
<tr>
<td>InventoryPort_B</td>
<td>simplePlaceOrder</td>
<td>PlaceOrderB</td>
</tr>
<tr>
<td>InventoryPort_B</td>
<td>cancelOrder</td>
<td>CancelOrderB</td>
</tr>
</tbody>
</table>

**Generating an FDL file**

Once you complete the mapping rules update, click Web Service UPES Administration. We need to get the Web services generated data structures, structure members, and program definition into the WebSphere Business Integration Workbench repository. The WebSphere MQ Workflow Web Services Toolkit does not support an import directly into the WebSphere Business Integration Workbench. Therefore, we export the data structures, structure members, and program definitions into an FDL file. We will import this FDL file into the WebSphere Business Integration Workbench repository later.

1. For InventoryPort_A:
   a. Click Show FDL on the Web Services UPES Administration interface for InventoryPort_A.
   b. On the browse screen, click Edit → Select All and then click Edit → Copy.
c. In a new Windows Notepad window, click **Edit → Paste** and then click **File → Save As**.

d. In the file dialog, save the file as UPESA.FDL with a save as type of All Files.

2. For InventoryPort_B, complete the same steps, saving to file UPESB.FDL.

**Building the process**

We have chosen to build the process in WebSphere Business Integration Workbench V4.2.4. This section describes how to build the process by creating a new organization, importing the Web service programs and structures into the repository, and constructing the process flow.

We created a new organization in WebSphere Business Integration Workbench called ITSOElec.org.

**Editing modes**

WebSphere Business Integration Workbench is used for many purposes, such as building process models, documenting policies, and integrating with workflow engines. The level of detail and data elements required differs for each purpose of model development. The Editing Modes feature of WebSphere Business Integration Workbench configures the toolbar, windows, and menus so that items not relevant for the desired purpose will be hidden from view.

To review the editing modes, click **Format → Editing Mode**. See Figure 9-44.

![Editing Mode](image)

*Figure 9-44  Editing modes*

The editing mode required to create processes for WebSphere MQ Workflow is the Integration Mode.

**Activity decision flow diagram**

The activity decision flow diagram (ADF) models a process visually and shows how elements of the process are interrelated. The ADF contains a toolbar that
contains all of the tools required to model a process. The content of the toolbar is dependant on the editing mode selected.

The toolbar for integration mode is shown in Figure 9-45. Objects used in this scenario such as Task, Interface, and Decision can be found in the toolbar. For more information about each object in the toolbar, see the *Modeling Guidelines* of WebSphere Business Integration Workbench V4.2.4.

![Figure 9-45  Toolbar in integration mode](image)

When you use the toolbar and the Repository data, the process is modeled as a diagram, as shown in Figure 9-31 on page 191. Each object has a dialog interface so that more information related to the object can be defined. Objects used in this process are described in this section.

**Adding the Web service FDL to the repository**

After generating the FDL files in the WebSphere MQ Workflow Web Services Toolkit, we import the UPESA.FDL and UPESB.FDL files into WebSphere Business Integration Workbench.

In WebSphere Business Integration Workbench, there are functions to import FDL as shown in Figure 9-46 on page 211.
We use the following steps to get application information from the Web services to the process defined in WebSphere Business Integration Workbench.

1. Import UPESA.FDL and UPESB.FDL using the function shown in Figure 9-46.

2. Create an org file for UPESA.FDL, and save it as UPESA.org.

3. Create an org file for UPESB.FDL, and save it as UPESB.org.

4. Import each process into the ITSOElec.org organization. Select File → Import/Export → Import Processes. Select UPESA.org. Then, highlight CancelOrderA, GetDeliveryDaysA, and PlaceOrderA (as in Figure 9-47 on page 212) and click OK.
5. Repeat the process for UPESB.org.

6. The Web service data structures are now imported into the repository. You can see them by clicking Repository → Process Data → Data Structures.

7. The Web service applications are also imported into the organization. To see them click Repository → Organization Data → Applications.

8. Finally a process is created for each Web service application. You can view these using File → Open Process (as shown in Figure 9-48 on page 213). We will not use these processes, but we could have used them as sub-processes in our process flow instead of defining tasks. Alternatively, these processes can be used to test each Web service operation in isolation.

Figure 9-47 Importing processes into an organization
Creating organizations and staff

As described in the design guidelines, there are three organization units of note: Retail, Wholesaler A and Wholesaler B. Create these organization units in the repository by clicking Repository → Organization Data → Organization Units. You can assign colors to these units, which will dictate the color of a task that is assigned to that unit. See Figure 9-49.
WebSphere MQ Workflow requires each organization unit to be assigned at least one manager. This serial process scenario does not use any human interaction, so we will assign the minimum staff requirements to satisfy WebSphere MQ Workflow.

Click Repository → Organization Data → Employees and create an employee called MGR to the Retail unit. Create a MGRA and MGRB for the Wholesaler A and B units respectively. See Figure 9-50.

![Figure 9-50 Creating employees for organization units](image)

Assign the employees to be managers of their organization. Click Repository → Organization Data → Organization Units. Select each unit, and choose the relevant manager from the Manager pull down menu.

**Adding data structures and fields to the repository**

As discussed in the design guidelines, we need to add data structures and data fields to the repository. Use Repository → Process Data → Data Fields to add data fields and new structures, and Repository → Process Data → Data Structures to assign data fields to these structures.

The following data structures should be defined. Refer to the design guidelines to see the fields in each structure

- OrderInput
- OrderOutput
- Wholesale
Creating a new process
Create a new process by selecting File → New Process. Make sure you are in Integration Mode (the keyboard shortcut to switch to this mode is Alt-3). Click Process → Info to open the process properties. Perform the following:

- In the General tab, set the process name and MQ Workflow name to SerialProcess.
- In the Fields tab, set the Input structure to OrderInput and the Output structure to OrderOutput.

Task object definitions
This process contains a number of task objects. Each of these tasks (with the exception of the NOOP task) are implemented by Web service operations, using a Web services UPES.

Create a task called getDeliveryDaysA as follows:

1. In the ADF toolbar select Task, and add a task to the ADF diagram.
2. Open the properties of the task by selecting the pointer tool and then double-clicking on the task. For our scenario, we only define the data in the tabs called General, Automation, and Data of the Task object properties window.
3. In the General tab (Figure 9-51 on page 216), the following needs to be defined:
Figure 9-51  Task properties: General tab

a. **Task Name and MQ Workflow Name**: Both to getDeliveryDaysA

b. **Organization Unit**: Defines the organization this task is applicable to (Select *Wholesaler A*, from the pull-down menu.)

c. **Application**: Assigns which application will be executed by this task (Select *InventoryPort_A_GetDeliveryDaysA* from the pull-down menu.)

d. **Options**: Uncheck User Program Execution Agent and click the **Server** radio button. Select *FMCSYS* as the system name and *FMCUPES* as the server name from the two pull-down menus. These settings were imported from the UPESA and UPESB FDL files and point to the settings specified in the Web Services Toolkit. You can view these settings in the repository by selecting **Repository → Organization Data → Systems**.

e. **Mode**: Select **Synchronous**.

4. In the Automation tab (Figure 9-52 on page 217), we set the starting and end of the task as **Automatic** because this process contains only automated tasks. In addition, since the scenario is performed in a Serial Process pattern, each task can be executed as long as one input is delivered to it.
5. In the Data tab (Figure 9-53 on page 218), the data structures used by the input and output of the task are defined. These should already be set to getDeliveryDaysRequest for input and getDeliveryDaysResponse for output. No data mapping is required.
Create tasks, using the technique above, for the following tasks. Remember to assign the correct organization unit and application:

- getDeliveryDaysB
- placeOrderA
- placeOrderB

We need to create the NOOP tasks that are used as a data container for the Wholesale structure. Before creating the tasks, we need to add a NOOP application to the repository as follows:

1. Click **Repository → Organization Data → Applications**.
2. In the General tab (Figure 9-54 on page 219), define a new application called FMCINTERNALNOOP. This is the WebSphere MQ Workflow NOOP application. Set the appropriate active platform and check **Run Unattended**.
3. In the Parameters tab (Figure 9-55), set the input and output data structure to Wholesale. Uncheck Program requires these options.
Create the NOOP task as follows:

1. In the ADF toolbar select **Task**, and add a task to the ADF diagram.
2. Open the properties of the task by selecting the pointer tool and then double-clicking on the task.
3. In the General tab (Figure 9-56), define the following:

   ![Diagram](image)

   **Figure 9-56 NOOP task properties: General tab**

   a. **Task Name and MQ Workflow Name**: NOOP
   
   b. **Organization Unit**: Retail
   
   c. **Application**: FMCINTERNALNOOP
   
   d. **Options**: Uncheck the **User Program Execution Agent** option. This activates the other options in this section. Set Mode to **Asynchronous** and then recheck the **User Program Execution Agent** option.

4. In the Automation tab, set the **start and end execution** to **Automatic** and select **One Input**.

5. In the Data tab (Figure 9-57 on page 221), ensure that the input and output structure are both set to **Wholesale**. We want to map the input structure to the output structure. There are two types of mappings supported: (1) Default, which passes data from the input container to the output container, and (2) Loop which passes data from the output container to the input container.
Specify **Default**, and create a mapping between source and target for _STRUCT (which represents all data fields in the Wholesale data structure).

![Activity Decision Flow Diagram](image)

**Figure 9-57  NOOP task properties: Data tab**

Create two more NOOP tasks. Create these in exactly the same way, except assign the MQ Workflow name as NOOP_01 and NOOP_02. In the Data tab of NOOP_01, create a Default mapping of _STRUCT to _STRUCT and a Loop mapping to set bestWholesaler to an initial value of A (Figure 9-58 on page 222). Similarly, in NOOP_02, create the same Default mapping and a Loop mapping to set bestWholesaler to an initial value of B.
**Decision object definition**

In this scenario, we defined the following two decisions:

- To check whether the delivery days from Wholesaler A is within a week
- To compare which wholesaler can fulfill the order the fastest

Add the first decision object as follows:

1. In the ADF toolbar, select **Decision** and add a decision to the ADF diagram.
2. Create a connector between getDeliveryDaysA and the decision object. This connector is required so that we can use the output of getDeliveryDaysA.
3. Open the properties of this object add and a binary decision. We gave the decision a name of `delDays<=7`.
4. Click **True Expression**. In the Data Fields area, highlight `getDeliveryDays`. Modify the expression text to read `deliveryDays<=7` (as shown in Figure 9-59 on page 223).
5. Set the false expression to OTHERWISE.

6. The decision object now has two outputs. Control will flow to a connector leaving from the right side of the decision object if the decision evaluates to true. Control will flow to a connector on the bottom of the decision object if the decision evaluates to false. Create a connector to placeOrderA for a true decision, and a connector to getDeliveryDaysB for a false decision.

Create a second decision object called daysA<=daysB and connect it to the NOOP activity. Set the true expression to daysA<=daysB and the false expression to OTHERWISE. Connect to placeOrderA for a true decision, and to placeOrderB for a false decision.

Data mapping between tasks
Data mapping is achieved using Phi objects. Create an input phi object as follows:

1. In the ADF toolbar select Phi, and add a phi to the ADF diagram.

2. Open the properties of the phi, and then click Phi type. Define a new type called orderInfo. Set the category to Electronic Document. At this point, you can also assign a color and bitmap to your phi type.

3. Back in the phi properties, set the phi name to order and the phi type to OrderInfo. You will use this phi name and type for all of your phi objects in this process.

4. Click OK to create the Phi.
5. Create connectors between the phi object and the following tasks: `getDeliveryDaysA`, `getDeliveryDaysB`, `placeOrderA`, and `placeOrderB`.

6. Open the phi object properties again and click **Data Flow**. This section lets you add mappings between the input to the Phi (in this case, the Source structure or process input) and the tasks the phi is connected to.

7. Set the Source Task to **Source** and the Target Task to `getDeliveryDaysA`. Map the partNo from `OrderInput` to the partNo of `getDeliveryDaysRequest` (Figure 9-60). The partNo supplied at the start of the process will now be available to the `getDeliveryDaysA` Web service operation.

![Figure 9-60 Data mapping between process input and getDeliveryDaysA](image)

8. Now select `getDeliveryDaysB` as the Target Task and again map the partNo data field between the two structures. Also uncheck **Control Connector**. This means that we want the connector between the phi and `getDeliveryDaysB` to only flow data. The control connector will now show as a dotted line.

9. Map partNo and qty for `placeOrderA` (Figure 9-61 on page 225) and `placeOrderB`. Again, these are only data mappings, so be sure to uncheck **Control Connector**.
Mappings also need to be created between different tasks in the process:

- From getDeliveryDaysA to NOOP to pass deliveryDays to daysA (not a control connector)
- From getDeliveryDaysA to NOOP_01 to pass deliveryDays to daysA (not a control connector)
- From getDeliveryDaysB to NOOP to pass deliveryDays to daysB (control connector)
- From getDeliveryDaysB to NOOP_02 to pass deliveryDays to daysB (not a control connector)
- From placeOrderA to the Sink to pass confirmCode to confirmCode
- From placeOrderB to the Sink to pass confirmCode to confirmCode

Finally, the output phi object needs to be created. Two output phis are required because there are two possible process paths. Place these at the end of the process, and perform the data mapping as follows:

- From NOOP_01 to the Sink to pass daysA to bestWholesalerDays and bestWholesaler to bestWholesaler (control connector). See Figure 9-62 on page 226.
- From NOOP_02 to the Sink to pass daysB to bestWholesalerDays and bestWholesaler to bestWholesaler (control connector).
Exporting the process to the runtime database

Once the process is built in WebSphere Business Integration Workbench, it should be exported into the WebSphere MQ Workflow runtime database. To do this perform the following:

1. Verify that the process has no errors. Click Process → Process Validation → MQ Workflow Model. Fix any errors that you find.

2. Export the process from WebSphere Business Integration Workbench using the Workflow icon on the toolbar. Select WebSphere MQ Workflow (3.4) in the Workflow Engine panel. Deselect Export NOOP (which refers to the default NOOP task, not the NOOP tasks you added to this process) and click OK.

3. The process is shown in an WebSphere MQ Workflow view. Select the Export icon from the toolbar. Leave the defaults to export organization and process data, and click OK.

4. The FDL file describing your process and organization is created. Save this file, and import it into the WebSphere MQ Workflow runtime using the following command (replacing anything in square brackets with the relevant parameter):


9.5.3 Runtime guidelines

This section describes the runtime environment used to test the process. We discuss the different ways a process instance can be started. Then, we
demonstrate this using two methods: using an XML message and using the WebSphere MQ Workflow Web client.

**Process instance creation**

WebSphere MQ Workflow accepts several methods to create an instance of a process model. These include:

- **WebSphere MQ Workflow Client**
  
  This is a GUI client that connects to WebSphere MQ Workflow across a LAN. This client allows you to create process instances from a process template, monitor process instances, and manage work items.

- **WebSphere MQ Workflow Web Client**
  
  This offers the same functions as the regular GUI client, but it runs as a Web application in a J2EE application server. Thus, it can be accessed from a wider range of clients.

- **XML message**
  
  You can use an WebSphere MQ PUT of an XML message with the ProcessTemplateExecute format to the WebSphere MQ Workflow execution server input queue EXEXMLINPUTQ from an application that you create (or from the API Exerciser).

**Starting process instances with XML messages**

To test the serial process in WebSphere MQ Workflow, we PUT an XML message with the ProcessTemplateExecute format to the WebSphere MQ Workflow execution server input queue EXEXMLINPUTQ. We used the API Exerciser to do this. This method simulates an application performing the PUT without actually writing the application. The steps to perform this are:

1. From the desktop, select **Start → Programs → WebSphere MQ → First Steps**.
2. Select **API Exerciser**.
3. Click **Launch API Exerciser**.
4. With the FMCQM queue manager displayed in the local queue manager drop-down, click **MQCONN**. The status window should display an `[OK(0)]` reason code.
5. Click **Queues**.
6. In the Selected Queue drop-down, select **EXEXMLINPUTQ**.
7. Click **Advanced mode**.
8. Click **MQOPEN**. In the MQOPEN Selectable Options, select **MQOO_SET_IDENTITY_CONTEXT** and **MQOO_OUTPUT**. Refer to
Figure 9-63. Leave the other default options. Click OK. The Status window should display a reason code [OK(0)].

**Figure 9-63  MQOPEN selectable options**

9. Click MQPUT. In the message window, paste the XML message as shown in Example 9-9. This message requests a process instance called SerialProcess01 to be created from process template SerialProcess. The process input data (partNo and qty) are also specified. You can modify the values of the process input data to test different execution paths in your process.

**Example 9-9  XML to create a process instance of SerialProcess**

```xml
<?xml version="1.0" standalone="yes"?>
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>Yes</ResponseRequired>
  </WfMessageHeader>
  <ProcessTemplateName>SerialProcess</ProcessTemplateName>
  <ProcInstName>SerialProcess01</ProcInstName>
  <KeepName>false</KeepName>
  <ProcInstInputData>
    <OrderInput>
      <partNo>ABCDE</partNo>
    </OrderInput>
  </ProcInstInputData>
</WfMessage>
```
10. Click **Message Descriptor**. Click **Next** until Page 6 is available as shown in Figure 9-64. Enter the name of your ReplyToQ, ReplyToQMgr, and the userID **ADMIN**. We create a queue MyQueue for our ReplyToQ in the default WebSphere MQ Workflow queue manager FMCQM.

![Figure 9-64 MQPUT Descriptor option page 6](image-url)

11. Click the **Put Message Options** tab. Click **MQPMO_SET.IDENTITYCONTEXT** as shown in Figure 9-65 on page 230.
12. Click **OK** on the MQPUT window. The status window should show [OK(0)] for the MQPUT. The XML message has now been put on the EXEXMLINPUTQ queue.

After the message is successfully PUT, WebSphere MQ Workflow creates a process instance. WebSphere MQ Workflow provides a process instance finish message on the ReplyToQ that you specify in the MQPUT options once the process reaches a finished state.

To view the response message on the ReplyToQ, perform the following:

1. In the API Exerciser main window, open the pull-down list called Selected Queue and select the ReplyToQ you specified earlier. (We use MyQueue.)

2. Click **MQOPEN**. Leave the settings to defaults, and click **OK**.

3. Click **MQGET**. Again, leave all settings to defaults and click **Execute**. This will retrieve a reply message off the ReplyToQ if the process instance has completed. If successful, you will see an XML message, similar to Figure 9-66 on page 231.
4. The output message should contain the contents of the process output data structure OrderOutput. There are three data fields in this structure: the name of the best wholesaler (either A or B), the number of days this wholesaler can fulfill the order in, and the confirmation code of the placed order.

To aid with testing, you can determine the delivery days of each wholesaler by interpreting the part number entered. The rules for Wholesaler A are:

- If the last character of the part number is a Z, then a confirmCode of zero is returned.
- Otherwise, the first character of the part number is converted into a number if necessary (for example, A to 1) and this value is returned as the delivery days.

Therefore a part number of BCDE would return delivery days of 2, and a part number of BCDEZ would return zero.

Wholesaler B uses similar logic, except that it checks whether the second-to-last character is a Z (which returns zero). Otherwise, the second character of the part number is converted to a number and returned.

This logic is shown by example in Figure 9-67 on page 232.
Test the different paths in the process by sending differing part numbers.

**Note:** We could have improved our model by checking whether the confirmCode is `zero` and whether it informs the process starter than the order failed. We will implement this logic in a later scenario.

---

**Starting and monitoring process instances with the Web client**

The WebSphere MQ Workflow Web client provides a graphical monitor to view the status of a process instance. It also allows you to create and start process instances from a given process template. This offers an alternative to sending XML messages to EXEXMLINPUTQ.

This section describes how to start and monitor the serial process described in this chapter.

**Creating persistent process instances**

By default, once a process instance completes, it is deleted. For monitoring purposes, it is useful if we can view process instances even after completion, allowing us to see the path of execution a particular instance took. Therefore, we need to modify the process template of SerialProcess to specify that all process instances from this template are persisted after completion.

To do this, open SerialProcess in WebSphere Business Integration Workbench and select **Process → Info.** Click **Process Set.** In the Keep Finished Process row, uncheck **Inherited** and click **Forever** (Figure 9-68 on page 233). Click **OK;**
save the process template, and then export it to the WebSphere MQ Workflow runtime database again.

![Activity Decision Flow Diagram Info](image)

**Figure 9-68  Persisting process instances**

**Installing the Web client**

The Web client is installed by using the WebSphere MQ Workflow Configuration Utility. We used this utility to configure the Web client on a WebSphere Application Server V5 server instance.
**Note:** If you install the Web client and the Web Services Toolkit in the same WebSphere Application Server server instance, you need to make the following modification to the Web client installation to allow both applications to co-exist:

Locate the Web client properties files at:

```<WAS_HOME>\installedApps\<nodename>\MQWF_Web_Client_FMC.ear\fmcohcli.war\WEB-INF\WebClient.properties```

Look for the following line:

```AgentConfiguration=FMC```

Change it to read:

```AgentConfiguration=```

Restart the application server for the change to take effect.

---

**Using the Web client**

To start the WebSphere MQ Workflow Web client, enter the following URL:

```http://localhost:9080/MQWFClient/RTC.html```

The logon page should be displayed. Log in using your WebSphere MQ Workflow userid and password. We use a userid of `ADMIN` and a password of `password`.

The first time you log in to the Web client, you do not see any views (unless you have already created them using the WebSphere MQ Workflow Client). We will create views to list all process instances and process templates. Perform the following:

1. Click **Create a list** (under Action) to create a new list.

**Note:** To determine the name of an icon, hold your mouse pointer over it.

2. Select **Process Instance List**. Then, name the list **ProcessInstances** and click **OK** (Figure 9-69 on page 235). You could additionally specify filters to only display certain process instances, but we display all instances.
3. In the Navigate pull down menu, select List of Lists and click Create a list again. This time create a process template list called ProcessTemplates.

4. The main window should now be showing the ProcessTemplates view, and the SerialProcess process template should be displayed (Figure 9-70).

5. Click the Create and start instance icon for SerialProcess. Give the instance a name of SerialProcess01, and set partNo to JDDGS and qty to 2 (Figure 9-71 on page 236). Click Create and start instance.
Create and start Instance from Template "SerialProcess"

6. Move to the ProcessInstances view by selecting Instance List: Process Instances from the Navigation pull-down menu. Once in this view, click Refresh. The SerialProcess process instance should be displayed (Figure 9-72). Depending on how quickly you refreshed the Web client, the process instance may be in a state of running or finished.

7. Continue to click Refresh until the process instance reaches a state of finished. To examine the properties of the process instance, click on the Properties icon of SerialProcess01. To view the output data structure of the process, click Output Container (Figure 9-73).
8. You can also monitor process instances in the Web client. Click **Back To Instance List** to return to the ProcessInstances view. To monitor the flow of execution in the process instance, click **Instance Monitor** (which is a picture of a movie camera). The monitor should display (as shown in Figure 9-74). The green arrows represent the flow of control that has been navigated. If the process is still running, the running activity is highlighted.

![Figure 9-74 A completed process instance](image)

9. You can view information about any activity by clicking on it. This is a useful way of debugging problems if a particular activity fails to complete. To inspect the confirmation code returned by placeOrderB, click it. Then click **Output Container** (Figure 9-75).

![Properties of Activity Instance "SerialProcess01.placeOrderB"](image)

**How the process works at runtime**

There are four automatic Web services activities and three automatic NOOP activity used in this process. The execution server checks the state of these automatic start activities when one of them is the next task to start. If the execution server determines that the activity is ready to start because all the required control connectors leading into that activity have been evaluated, it changes the activities state to a running state. Since these activities use a UPES, the execution server creates an XML message that includes the following:

- A workflow header
The program name for the activity
The correlation ID for this activity
The data structure name specified for the activity
The data structure member name and its current value

**Note:** The WebSphere MQ message header contains information set by WebSphere MQ Workflow such as the replyToQueue and replyToQueueManager for any responses.

This execution server performs a WebSphere MQ PUT of this XML message to the queue manager and queue specified in the server definition, as defined in the WebSphere Business Integration Workbench repository.

For Web service activities where a message is PUT into the queue, it is the job of the WebSphere MQ Workflow Web Services Toolkit to:

- Perform the WebSphere MQ GET operation.
- Transform the message to a SOAP message.
- Send the SOAP message as a request to the relevant Web service address and port.
- Handle the response from the Web service operation.

Figure 9-76 shows the message flow of WA07 (the Web Services Toolkit).

---

**Figure 9-76  Message flow**

- **EXEXMLINPUTQ**
- **FMCOAXIS**
- **MQWF**
- **FMCWUPES**
- **WA07**
- **FMCOAXIS**
- **Web Service**
Because we modeled these activities as a WebSphere MQ Workflow synchronous activity, the WebSphere MQ Workflow execution server waits until either a reply is received or an expire time is reached (if one is set). In this scenario, we do not set an expiration value. Therefore, WebSphere MQ Workflow waits forever for the reply from the external application that is listening for the message that WebSphere MQ Workflow PUT to the queue (FMCWUPES). WebSphere MQ Workflow provides a correlationID in the message that it expects in the returned message. The correlationID allows WebSphere MQ Workflow to match the reply to the original activity in WebSphere MQ Workflow.

The application that is listening on the queue that WebSphere MQ Workflow put the message on is the WebSphere MQ Workflow Web Service Toolkit. The message obtained in the MQGET from the queue is an XML message that the application passes to the Apache AXIS queue for transformation into a SOAP message format. There, the application generates the Web service request message.

When a Wholesaler processes a Web service request, it generates a response message back to the WebSphere MQ Workflow Web Service Toolkit Apache AXIS queue (FMCOAXIS). The WebSphere MQ Workflow Web Services Toolkit transforms the message back into an XML format that WebSphere MQ Workflow can process. This message is combined with the correlationID that WebSphere MQ Workflow sent to the application, and an WebSphere MQ PUT message is generated to WebSphere MQ Workflow’s XML input queue (EXEXMLINPUTQ).

The WebSphere MQ Workflow execution server processes the message on the EXEXMLINPUTQ. If no formatting errors are detected, the state of the automatic activity is updated by the execution server to a finished state. WebSphere MQ Workflow holds the deliveryDays or a confirmCode result in its output container, depending on the request type of the automated activity. We can use methods provided by WebSphere MQ Workflow to access this result in the model.

**WebSphere MQ Workflow Web Services Toolkit SOAP monitor**

The WebSphere MQ Workflow Web Services Toolkit includes a SOAP monitor that allows you to see the request and responses as messages are sent and received from the Web services. To use the monitor, click **SOAP Monitor** on the WebSphere MQ Workflow Web Services Toolkit. The SOAP monitor is shown in Figure 9-77 on page 240.
The WebSphere Application Server logs includes entries for the WebSphere MQ Workflow Web Services Toolkit. The default names for the log files are SystemOut.log and SystemErr.log. The logs are important for us during the setup of the WebSphere MQ Workflow Web Services Toolkit and help identify problems detected during our configuration steps when the toolkit needs to access queues and queue managers. You can locate these logs at:

```plaintext
<websphere_install_dir>\logs\server1\n```

The Diagnostic Trace Service is useful to trace application components. The trace specifications to enable the trace are:

```plaintext
com.ibm.ws.webservices.*=all=enabled
```

In addition to the above trace and logs, the SOAP monitor is useful for observing the SOAP message request and response between the WebSphere MQ Workflow Web Service Toolkit and the Web service implementations. Tracing facilities for WebSphere MQ Workflow are enabled by using the WebSphere MQ Workflow application fmczchk. Once the trace parameters are in the WebSphere MQ Workflow profile, you must restart WebSphere MQ Workflow so that the trace parameters are loaded from the profile. To obtain parameter details for fmczchk, enter the following command from a prompt:

```plaintext
```
9.6 Best practices

This section describes general best practices when configuring components used in a Serial Process pattern for a WebSphere Process Choreographer or WebSphere MQ Workflow Product mapping.

9.6.1 General best practices

The best practices in this section may apply to either WebSphere Process Choreographer or WebSphere MQ Workflow because of the DB2 and WebSphere MQ interfaces that they use.

**Database**

The following are best practices to consider for the DB2 database server:

- Consider locating the DB2 logs and database tables in file systems that are not contending for file system resources.
- Consider locating tables on the file system based on demand for access and updates.
- Consider a fail-over node for the DB2 node.
- Evaluate the DB2 configuration resources needed to provide proper response to the WebSphere MQ Workflow runtime node. As minimum, consider the following:
  - Agent stack size
  - Query heap size
  - Log file size
  - Primary log parameter
  - Secondary log parameter
  - Database heap size
  - Default application heap size
  - Number of page cleaners
  - Log buffer size
  - Lock list setting
  - Buffer pool size
  - Number of processors that can be configured for DB2.
- Develop a database backup plan that does not conflict with the internal processes of WebSphere MQ Workflow and WebSphere Process Choreographer.

- Database statistics should be scheduled on a regular interval and a re-organization scheduled on a regular bases. After re-organizing a DB2 database used by WebSphere MQ Workflow runtime, it may be necessary to perform a rebind of the runtime executables.

- Develop a plan for regular checks of the DB2DIAG log for errors, and allow for corrective action by technicians as soon as possible.

- Monitor the corrective service packages available that might influence the system level currently running on your platform.

- Before applying corrective service packages, check whether this corrective service impacts other service for the operating system, WebSphere MQ, WebSphere MQ Workflow, or WebSphere Process Choreographer.

- Before applying a corrective service package, develop a plan to back up the system before applying the upgrade.

- Develop a plan to install and test without impacting any production system. Then, plan the production system upgrade at the appropriate time.

- Ensure that there is a plan in place to move the audit trail data off the process manager database into a data warehouse or similar data store where queries can be performed.

**WebSphere MQ configuration**

The following are best practices to consider for the WebSphere MQ runtime maintenance:

- Locate WebSphere MQ logs on separate file systems away from contention with other file system usage.

- Attempt to limit use of persistence queues when they are not necessary.

- Develop a plan for regular checks of the WebSphere MQ error logs. Allow for corrective action by technicians as soon as possible.

- Monitor the corrective service packages available that might influence the system level currently running on your platform.

- Before applying corrective service packages, check whether this corrective service impacts other service for the operating system or other applications such as DB2, WebSphere MQ Workflow, or WebSphere Process Choreographer.

- Before applying a corrective service package, develop a plan to back up the system before applying the upgrade.
- Develop a plan to install and test without impacting any production system. Then plan the production system upgrade at the appropriate time.
- If you use WebSphere MQ circular logs, both WebSphere MQ and DB2 should be backed up at the same time.

### 9.6.2 WebSphere Process Choreographer specific best practices

For information about how to configure WebSphere Process Choreographer in a distributed environment, refer to:


A collection of WebSphere Process Choreographer-related articles is located at IBM developerWorks at:

http://www.ibm.com/developerworks/websphere/zones/was/wpc.html

### 9.6.3 WebSphere MQ Workflow specific best practices

There are several SupportPacs available with detailed information related to best practices for WebSphere MQ Workflow. Refer to:

http://www-3.ibm.com/software/integration/support/supportpacs/individual/wa0b.html

In addition for best practices related to production rollout, refer to the following:


For additional details regarding best practices related to modeling with WebSphere MQ Workflow, refer to the following SupportPac:


A SupportPac to help understand performance and capacity for an WebSphere MQ Workflow system is available from the following URL:

Creating parallel processes

This chapter describes how to build a process that uses the Parallel Process pattern. The chapter is split into the following parts:

- **Business scenario**
  Describes a business problem ITSO Electronics wishes to solve

- **Business process model**
  Describes the business process model to solve the business problem

- **General design guidelines**
  Provides product-agnostic design guidelines for building a process conforming to the Parallel Process pattern

- **WebSphere Process Choreographer guidelines**
  Describes how this Parallel Process was implemented using WebSphere Process Choreographer

- **WebSphere MQ Workflow guidelines**
  Describes how this Parallel Process was implemented using WebSphere MQ Workflow
10.1 Business scenario

In this scenario, we build upon the Serial Process scenario from the previous chapter. The scenario remains essentially the same, with one small change. Now the request to get the delivery dates from the wholesalers happens in a parallel fashion. The order clerk from ITSO Electronics will place 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.

10.2 Business process model

In this scenario we are going to implement the process using the Parallel Process pattern, ensuring that we retrieve the delivery days for a particular part from Wholesaler A and B in parallel.

A generic swimlane diagram can be used to depict the process (Figure 10-1).

The Parallel Process implementation is similar to the Serial Process implementation described in the previous chapter. The only difference is
Wholesaler A and Wholesaler B are contacted at the same time. The scenario is as follows:

- The order clerk places an order for a part number and a quantity.
- Wholesaler A is contacted for its delivery date.
- At the same time, Wholesaler B is contacted for its delivery date.
- Once both wholesalers respond, the system can determine who the best wholesaler is (that is, which wholesaler can deliver the most quickly).
- The order is placed with the best wholesaler.
- After the order is successfully placed, the best wholesaler, confirmation number and the deliver days are returned to the order clerk via the application that was used to start the process.

### 10.3 General design guidelines

All of the design considerations listed in the previous scenario chapter apply to the Parallel Process scenario as well. By getting the delivery dates from the two wholesalers in parallel, we are potentially shortening the time it takes to complete the process.

When the defined business process allows it, having the process manager split the process into two or more paths of parallel processing is often desirable. For long-running processes, cycle time reduction is in constant focus for business executives. Since business processes should support a customer in some way, shorter cycle time in processes usually equates to an organization being more responsive to its customer.

### 10.3.1 Design overview

This scenario is an implementation of the Process-focused Application Integration::Parallel Process pattern.

For information about the Parallel Process Application pattern, see:

3.4.8, “Parallel Process Application pattern” on page 61

For information about the Parallel Process Runtime pattern, see:

5.1.3, “Parallel Process Runtime pattern” on page 88

For information about the Product mappings of the Parallel Process Runtime pattern, see:

5.2.3, “Parallel Process Product mappings” on page 98
This scenario is an implementation of Stage One of the ITSO Electronics integration project. For information about the use case and actors used in Stage One, see:

6.2.1, “Stage One: Internal ordering on demand” on page 110

10.3.2 Design considerations

The following design decisions need to be made when considering whether to use the Serial or Parallel Process pattern.

How much parallelism?
In our scenario, when the decision to contact the two wholesalers in parallel was made, it was relatively easy to implement. Splitting the workflow into two parallel paths is quite common and does not introduce much complexity.

However, consider a scenario that the logical model (that is, the to-be business process) dictates that the process can run in, say, 10 parallel paths. Should the implementation of the process support the 10 parallel paths?

Similar to a data model, there should be some level of normalization of the process that needs to occur for both performance and complexity reasons. In an automated system process (either a short or long-running process), if there is no urgency to have the process complete as quickly as possible, the process should be normalized into some serial and some parallel processing paths. This is easy to accomplish when there isn’t an urgency to complete the process as quickly as possible.

If there is an urgency to complete the process quickly, then consideration for the performance and complexity of implementing a process this way needs to be considered.

Performance
From a performance point of view, the modeler must understand what resources are required when the parallel processing occurs. You should ask yourself the following questions:

- If more than one of the activities executing in parallel are requesting the same resource, have we just created a bottle-neck in the process?
- Can the requested resource handle all of the requests from this process at once?
- What if there are more than one of these process instances running at the same time? Have we overloaded the resources we are calling with too many concurrent requests?
In addition, the type of resource you consider calling in parallel is significant. For example, if you are considering making parallel calls to Web service operations, you should ask the following:

- Is the Web service outside of your organization and therefore outside of your control?
- Is it possible that the Web service may become overloaded if called concurrently?
- Is the length of time it takes to complete a Web service call significantly long enough to consider making the call in parallel, bearing in mind the extra complexities of parallel execution?

The answers to these types of questions help the solution architects and modelers to determine how much parallelism should be included in the process template. All of these factors substantiate the need to set aside time to do a proper performance and capacity analysis on the system. (See “The impact of performance and capacity analysis” on page 160).

**Note:** A WebSphere Process Choreographer non-interruptible process only executes in a single thread. A decision to have a parallel process in WebSphere Process Choreographer means that the process needs to be defined as interruptible. It is worthwhile to note that a change from a non-interruptible to an interruptible process does have a negative impact on performance.

**Complexity**

With each activity that is processed in parallel, complexity seems to increase exponentially if the processing does not complete successfully. In using parallel execution, you need to 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?
- Should the failure of one of the activities prevent processing to continue along the other parallel paths?
Should the process be put into a state that calls a human to investigate the problem?

**Human interaction**
When human interaction is introduced with significant parallel processing, the complexity also increases. This is especially true if one parallel processing path can influence what happens in another (parallel) path.

Sometimes it may be better to spawn what can be called *loosely-linked processes* to handle this behavior. A loosely-linked process is a separate process (not a subprocess) that is spawned from the master process. The processes are linked by a relationship in an application database and this link is usually accomplished with the master process ID. That is, a separate application table may be used to keep track of the master process and the process IDs that it has spawned.

As an example, if the master process ID is 12345, then its loosely linked processes may be named 12345-1, 12345-2, etc. With WebSphere MQ Workflow, you can use the external context feature to store 12345 for all processes that are related to this master process. Using the external context makes it easy to relate all the processes.

**Conclusion**
If you encounter a business process that is being automated that calls for a significant amount of parallel processing, the solution architects and modeler need to approach the situation with caution. There are no hard and fast rules that apply, but blindly modeling all of the parallel paths as defined in the logical model may lead to problems down the road. It is really requires case-by-case analysis to determine how much parallelism is acceptable and reasonable.

### 10.4 WebSphere Process Choreographer guidelines

This section describes how to implement the Parallel Process pattern to create a process that meets the requirements set by the ITSO Electronics business process model. It uses the Runtime pattern and Product mappings described in “Parallel Process Product mappings” on page 98 for WebSphere Process Choreographer.

You can download the completed process and run it in WebSphere Application Server Enterprise or view it in WebSphere Studio Application Developer Integration Edition. See “Sample scenarios setup” on page 398.
10.4.1 Design guidelines

This section discusses WebSphere Process Choreographer-specific design guidelines when creating a process that conforms to the Parallel Process pattern. The design decisions identified in the previous chapter are also relevant here.

Process overview

This process meets the business objectives of ITSO Electronics as defined in “Stage One: Internal ordering on demand” on page 110. The completed process is shown in Figure 10-2.

The process has the following characteristics:
- Asynchronous interface
- Interruptible (long-running) process
- Parallel execution paths

Parallel execution paths

A parallel execution path is represented by two or more control links from an activity output terminal, where each of the control links evaluate to true. The execution of the parallel path depends on the type of process:
- Non-interruptible process

Each parallel path will be executed sequentially. The order in which the paths will be executed is variable. For example, consider a process with two parallel paths. In one given process instance, path A is executed and upon completion of this path, path B is executed. The next process instance may execute path B before path A. Therefore you must design your non-interruptible parallel processes to allow for the sequential execution of parallel paths, making no assumption on which path is executed first.
Interruptible process

Each parallel path is executed simultaneously, assuming the WebSphere Process Choreographer runtime has sufficient message-driven bean threads to process each execution path.

In addition to parallel processing of parallel execution paths, interruptible processes provide other benefits such as process instance data persistence. These benefits are described in “Interruptible processes” on page 164.

Applying the guidelines to ITSO Electronics

We used a parallel execution path to process Web service operation calls to Wholesaler A and B. We required these paths to be processed in parallel, and, therefore used an interruptible process.

In this scenario, moving from a Serial Process pattern to a Parallel Process pattern has degraded process instance performance. An interruptible process is significantly slower than a non-interruptible process. However, the interruptible process offers higher qualities of service and can, for example, persist a WebSphere Application Server Enterprise server restart.

10.4.2 Development guidelines

This section describes how to build a Parallel Process pattern implementation for ITSO Electronics. This process is essentially the same as the process built in the previous scenario, with the addition of parallel execution paths.

Process case study

The process will take the following as input:

- The part number of a part to order
- The quantity of the part to order

The process will perform the following tasks:

1. The process will get the number of days required by Wholesaler A to deliver the part. It will invoke a Web service operation to achieve this.

2. In parallel it will make the same call to Wholesaler B. It will invoke an Enterprise JavaBean method to achieve this.

3. The two results from the above steps will be compared to determine the wholesaler who can fulfill the order in the shortest time.

4. An order will be placed with the wholesaler who can fulfill the order in the shortest time. The relevant Web service operation or Enterprise JavaBean method will be invoked to achieve this.
The completed process will return the following information:
1. The confirmation number for the order (in case of success)
2. The name of the best wholesaler
3. The number of days required by the best wholesaler to deliver the product

Building the process
This section describes how to build the parallel process. Many of the steps are identical to the previous chapter and are therefore not repeated here.

1. Create a business process named ParallelProcess within a package name com.ibm.itso.process. Place it inside of a service project named ITSOParallelProcess.
2. Create the activities shown in the diagram Figure 10-2 on page 251.
3. Generate an interface for this process, being sure to select an Asynchronous process type (Figure 10-3).

![Generate WSDL Interface](image)

*Figure 10-3  Asynchronous process interface*

4. Create messages for the process input and output. These take the same structure as those used in the previous chapter.
5. Create a port type and two asynchronous one-way operations. Assign messages to these operations. The process interface is shown in Figure 10-4.

![Process interface definition](image)

**Figure 10-4  Process interface definition**

The complete source of the WSDL is shown in Example 10-1.

**Example 10-1  Generated WSDL code for the process interface**

```xml
<message name="processOrderRequest">
  <part name="partNo" type="xsd:string"/>
  <part name="qty" type="xsd:int"/>
</message>
<message name="processOrderResponse">
  <part name="confNo" type="xsd:string"></part>
  <part name="bestWholesale" type="xsd:string"></part>
  <part name="bestWholesaleDays" type="xsd:int"></part>
</message>
<portType name="parallelProcessPortType">
  <operation name="ProcessOrder">
    <input message="tns:processOrderRequest" name="ProcessOrderRequest"/>
  </operation>
  <operation name="OutputOperation">
    <input message="tns:processOrderResponse" name="OutputOperationRequest"/>
  </operation>
</portType>
```
6. Create the input and output variables based on the input and output messages. Then, assign them to the process input and output nodes.

**Note:** The blocks in the process are similar to those in the previous chapter. As mentioned earlier, the process invokes a Web service operation to interact with Wholesaler A and an Enterprise JavaBean method to interact with Wholesaler B. Only differences from the previous scenario are discussed here.

7. In this scenario, we use a Java snippet to compare the delivery days from Wholesaler A and Wholesaler B. We will store the best wholesaler in a process variable. In the Variables tab, create a new variable called `bestWholesale` and set its type to **String** (as shown in Figure 10-5).

![Figure 10-5 bestWholesale process variable](image)

8. In the `getBestWholesale` Java snippet, enter code to compare the delivery days returned from Wholesalers A and B and store the wholesaler name with the shortest date in the `bestWholesale` process variable. The source for this snippet is shown in Example 10-2.

**Example 10-2 getBestWholesale Java snippet**

```java
com.ibm.itso.ejb.wholesalea_msg.GetDeliveryDaysResponseMessage
delDaysA = getGetDeliveryDaysWholesaleAResponse();
com.ibm.itso.ejb.wholesaleb.Inventory_msg.GetDeliveryDaysResponseMessage
delDaysB = getGetDeliveryDaysWholesaleBResponse();
```
StringMessage bestWholesale = getBestWholesale();

if (delDaysA.getDeliveryDays() <= delDaysB.getResult())
    bestWholesale.setValue("A");
else
    bestWholesale.setValue("B");

setBestWholesale(bestWholesale);

9. After the getBestWholesale snippet completes, we want to continue processing down only one of the two outbound control links. We use conditional control links to decide which path of execution is followed. The control link between getBestWholesale and placeOrder_WholesaleA_Block checks whether the best wholesaler evaluated to Wholesaler A. This control link also sets the best wholesaler name and delivery days in the output message. Use a Java condition as shown in Example 10-3.

Example 10-3  Conditional control link for best wholesaler

ProcessOrderResponseMessage processOrderRespMsg = getProcessOrderResponse();
if (getBestWholesale().getValue().equalsIgnoreCase("A")){
    result = true;
    processOrderRespMsg.setBestWholesale("A");
    processOrderRespMsg.setBestWholesaleDays(getGetDeliveryDaysWholesaleAResponse().getDeliveryDays());
} else {
    result = false;
    processOrderRespMsg.setBestWholesale("B");
    processOrderRespMsg.setBestWholesaleDays(getGetDeliveryDaysWholesaleBResponse().getResult());
}
setProcessOrderResponse(processOrderRespMsg);

10. Set the control link between getBestWholesale and placeOrder_WholesaleA_Block to a condition of otherwise.

11. This process contains parallel paths that we wish to be executed concurrently. Therefore, we need to define this process to be interruptible. In the process editor, click Server and check Run Process as Interruptible. Additionally, ensure that Delete Process on Completion is not checked. This enables us to see the result of the process in the Process Web Client. See Figure 10-6 on page 257.
10.4.3 Runtime guidelines

These guidelines give you an overview of how to test the process described in this chapter.

Deploying a process and starting a process instance
As described in the previous chapter, a process must be deployed as an enterprise application. During this deployment, an inbound binding will be created. The process instance can be started either by using this inbound binding or by using the Process Web Client or WebSphere Process Choreographer APIs.

JMS as the inbound binding type
We choose to deploy the processes developed in this scenario using a JMS inbound binding. The choice of inbound binding is ultimately irrelevant for us because we will use the Process Web Client to test the process.

The result of a JMS inbound binding is a message-driven bean that has a message selector listening on a specific queue for a message containing a specific header property. Use the deployment wizard to specify these values. The message selector will check the value of the WSDLBinding header property (as in Figure 10-7 on page 258). Optionally, you can turn off the message selector if you want the message-driven bean to process every message received on the queue.
The deployment wizard allows you to specify which parts of the WSDL input message should be included in a generated JMS message. It also allows you to set values for other header properties. One such header property is JMSReplyTo, which specifies which JMS destination for a reply message. Finally, you can specify the JMS port information that includes the connection factory and queue JNDI names (Figure 10-8).

The following artifacts are generated for the JMS inbound binding:

- **Message-driven bean**
  The message driven bean contains an onMessage() method that initiates a new process instance based on values in the JMS message it is processing. The bean’s deployment descriptor describes the message selector (if one is used), along with the connection factory and queue where the message-driven bean is listening for messages.

- **Binding and service WSDL files**
  The WSDL files expose the JMS binding as an enterprise service. They describe the location of the JMS queue where the message driven-bean is
listening. They also describe the format of the JMS message expected by the bean.

### Using the Process Web Client to test the process

Starting a process instance with an asynchronous interface in the Process Web Client is somewhat different from starting a synchronous one. After the Process Web Client initiates a request to start the process instance, it does not wait for the process instance to complete. Therefore, it does not show the output message of the process upon completion.

The process built in this chapter uses an asynchronous interface. To start a process instance of it and view its output message, perform the following steps:

1. Start the Process Web Client and create a new process instance of ParallelProcess. You will notice that in addition to specifying the input message part values, you can also enter a process instance name (Figure 10-9). This allows you to assign a meaningful name to a long-running process instance, which will be useful when trying to identify the process instance later. The process name is optional, and if it is not specified the Process Web Client will use the process ID to identify the process instance.

![Figure 10-9](image)

2. The Process Web Client will send a request to initiate this process instance. However, unlike a process with a synchronous interface, it will not wait until the process completes. Instead, you will be taken to the My To Dos page.
Click **Created By Me** to view the status of the process instance (Figure 10-10). All running process instances that you started are displayed here (along with completed process instances that have not been deleted).

![Figure 10-10   Viewing the status of a running process instance](image)

3. The Created By Me page lists the status of the process instances and each block within the process instances. You can view the activity that is being executed in a running block by clicking on the process ID link of the relevant block. Keep clicking **Created By Me** to refresh the page. If all goes well, the process will reach a stage of Finished. Click the process name to view the output message of the completed process (as shown in Figure 10-11 on page 261).
4. If a problem occurs such as a block never completes execution, you can click on the block to view which activity is causing the problem. You can terminate or force complete troublesome activities using the Process Web Client.

5. To remove a process instance from the database and the Process Web Client, select the instance and click **Delete**.

### 10.5 WebSphere MQ Workflow guidelines

This section describes how to implement the Parallel Process pattern to create a process that meets the requirements set by the ITSO Electronics business process model. It uses the Runtime pattern and Product mappings for WebSphere MQ Workflow described in “Parallel Process Product mappings” on page 98.
You can download the completed process and run it in WebSphere MQ Workflow or view it in WebSphere Business Integration Workbench. See “Sample scenarios setup” on page 398.

10.5.1 Design guidelines

This section discusses WebSphere MQ Workflow-specific design guidelines when creating a process that conforms to the Parallel Process pattern. The design decisions identified in the previous chapter are also relevant here.

Process overview

This process meets the business objectives of ITSO Electronics as defined in “Stage One: Internal ordering on demand” on page 110. The completed process is shown in Figure 10-12.

The process has the following characteristics:

- Parallel execution
- Contains only automated activities

The process shares many characteristics with the previous scenario (the serial process). This scenario shares the following:

- The organizational structure
- The process interface
- The data structures
- The Web services implementations
**Parallel execution paths**
The WebSphere MQ Workflow runtime supports multiple paths of execution running in parallel. Two options are provided when the parallel execution paths rejoin:

- Continue processing when all parallel paths have completed
- Continue processing when the first parallel execution path has completed

For this scenario, we will retrieve the delivery days from both wholesalers in parallel. We require both paths to complete before continuing the process.

### 10.5.2 Development guidelines

This section describes how to build a Parallel Process pattern implementation for ITSO Electronics. This process is essentially the same as the process built in the previous scenario, with the addition of parallel execution paths. You should have completed the serial process described in the previous chapter before following the steps in this section.

**Building the process**

This section describes how to build the parallel process. Many of the steps are identical to the previous chapter and are therefore not repeated here. You will continue to use the repository data already defined in the previous scenario.

1. Open the organization created in the previous chapter in the WebSphere Business Integration Workbench.

2. Create a new process. Switch the editing mode to Integration mode, then open the process properties (Process → Info). In the Info window, set the following:
   a. Set the Process name and MQ Workflow process name to **ParallelProcess**.
   b. Set the Input structure to OrderInput and the Output structure to **OrderOutput**.
   c. Set Keep Finished Process to **Forever**.

3. Use the ADF toolbar and Repository data to model the process as shown in Figure 10-12 on page 262. Refer to the previous chapter for information about how to define the tasks and data mappings.

4. The first NOOP task must be set to ensure that the tasks getDeliveryDaysA and getDeliveryDaysB are completed before processing continues. Open the NOOP task properties, and select the **Automation** tab. Ensure that the task waits for both incoming control connectors by setting Automated Execution Wait for to **All Inputs** (as shown in Figure 10-13 on page 264).
5. Once the process is built, verify it by selecting **Process → Process Validation → MQ Workflow Model**. Correct any errors reported here.

6. Export the process to a FDL file that can be used in WebSphere MQ Workflow V3.4 (deselect Export NOOP).

7. Import the FDL file using following command to WebSphere MQ Workflow runtime:

   ```bash
   ```

### 10.5.3 Runtime guidelines

These guidelines give you an overview of how to test the process described in this chapter.

**Starting process instances with XML messages**

As with the scenario described in the previous chapter, a process instance of ParallelProcess can be started by placing an XML message on EXEXMLINPUTQ. If a ReplyToQ is specified, the process output data structure
will be described in an XML message in this queue upon process instance completion.

**Starting and monitoring process instances with the Web client**

The Web client can also start process instances. It has the added advantage of providing monitoring facilities. Once ParallelProcess is added to the WebSphere MQ Workflow runtime, it should be visible in the Web client template list.

Figure 10-14 shows a completed process instance of ParallelProcess. The green arrows indicate that getDeliveryDaysA and getDeliveryDaysB were executed in parallel.

![Completed process instance in the process monitor](image)

The process output data structure is shown in Figure 10-15.

![Completed process instance output container](image)
Creating processes with human interaction

This chapter describes how to build a process that uses the Parallel Workflow variation pattern with human interaction. The chapter is split into the following parts:

- **Business scenario**
  Describes a business problem that ITSO Electronics wishes to solve

- **Business process model**
  Describes the business process model to solve the business problem

- **General design guidelines**
  Provides product-agnostic design guidelines for building a process conforming to the Parallel Process Workflow variation pattern

- **WebSphere Process Choreographer guidelines**
  Describes how this process was implemented using WebSphere Process Choreographer

- **WebSphere MQ Workflow guidelines**
  Describes how this process was implemented using WebSphere MQ Workflow
11.1 Business scenario

The third scenario is an extension of the Parallel Process scenario from the last chapter. In this scenario, the order clerk once again places an order for a part by specifying the part number and the quantity. See “Business scenario” on page 246.

What is new is that human interaction is introduced in this scenario in two places:
- If a certain delivery criteria is not met by either of the wholesalers, a manager approval is required to order the part.
- At the end of the process, we notify the originator of the result of the order they placed. Possible results include that the order was placed with either wholesaler or that the order was canceled.

All of the human interaction activities are delivered to the users through a work list. The work list contains all of the eligible work (work items) for the logged-on user.

After the order is successfully placed, the order clerk is notified about which wholesaler the order has been place with. The status notification that the order clerk receives is a work item delivered by the process manager to the clerk’s work list.

11.2 Business process model

In this scenario, we are going to implement a workflow that involves human interaction in a couple of places in the process. The process is similar to what has been used in the first two scenarios.

However, a manager's approval to order the part is required if delivery dates from the wholesalers exceed the company policy of seven days. At the end of the workflow, we are also going to send a work item to the order clerk who initiated the process. The order clerk will receive confirmation of the order status and, if applicable, the confirmation number, best wholesaler, and delivery date.

The swimlane diagram to depict this process is shown in Figure 11-1 on page 269.
The details for this scenario are as follows:

1. The order clerk places an order for a part number and a quantity.
2. A request is sent to both Wholesaler A and Wholesaler B in parallel, requesting the delivery date.
3. Once a response from both wholesalers is received, a determination of the best wholesaler is made.
4. If both wholesalers return a delivery date of 7 days or less, the best wholesaler is determined and the part is ordered.
5. If the best wholesaler has a delivery date of greater than 7 days, the part is not ordered immediately. Instead, the order is sent to a manager for his decision.
6. The manager decides whether the order should be placed with the best wholesaler or canceled.
7. If the order is canceled, a work item is sent to the originator of the request to inform him of the cancellation.
8. If the manager approves the order with the best wholesaler, then the order is placed with that wholesaler.

9. After the order is placed with either wholesaler, a work item is sent to the originator. The work item contains the confirmation number, the identity of who the best wholesaler, and the delivery date.

11.3 General design guidelines

In this scenario, we revisit the synchronous versus asynchronous process invocation discussion and expand upon the options available. The section “An alternative solution” on page 273 elaborates on another solution to the one that was implemented in our scenario. Human interaction is also discussed in this section, including people-based exception handling.

11.3.1 Design overview

This scenario is an implementation of the Process-focused Application Integration::Parallel Process Workflow variation pattern.

For information about the Parallel Process: Workflow variation Application pattern, see:

3.4.9, “Parallel Process: Workflow variation” on page 63

For information about the Parallel Process: Workflow variation Runtime pattern, see:

5.1.4, “Parallel Process: Workflow variation Runtime pattern” on page 90

For information about the Product mappings of the Parallel Process: Workflow variation Runtime pattern, see:

5.2.4, “Parallel Process: Workflow variation Product mappings” on page 100

This scenario is an implementation of Stage Two of the ITSO Electronics integration project. For information about the use case and actors used in Stage Two, see:

6.2.2, “Stage Two: Internal ordering on demand with approval workflow” on page 113

11.3.2 Design considerations

This section discusses a number of design decisions relevant to the Parallel Process Workflow variation pattern. The design considerations for the Serial and Parallel Process patterns in the previous scenario chapters are also relevant.
Synchronous vs. asynchronous process invocation

Processes that use the Workflow variation pattern (either Serial or Parallel) are long-running. They contain an element of human interaction and also other potentially long-running tasks. Therefore, in the majority of cases, these processes will have an asynchronous interface.

In this scenario, there is a choice to be made about how to start this process. If the order is placed and at least one of the wholesalers can deliver the part within a week, this process could have been implemented with a synchronous call to start the process. In this case, the notify status activity at the end of the process would have to have been removed, and this status update could have been implemented as the reply message of the synchronous process invocation. This scenario would have been similar to the parallel process scenario in the previous chapter.

However, since there is the possibility that a manager from ITSO Electronics will need to get involved to approve the order, the process should not be called synchronously. Once the decision that this process will be started asynchronously has been made, other activities that involve human interaction can be introduced. In fact, it is common that people-based exception handling is used with processes that involve automatic system integration to ensure that a process is not stranded because of a system failure. Refer to “People-based exception handling” on page 272.

Human interaction

When human interaction is introduced into a process, the process manager needs to determine who the eligible users are for the activity instance. When possible, this determination of the eligible users should be a dynamic determination at runtime. This late-binding allows for the most current staff definitions for the organization to be applied.

The delegation of the work item to a manager can be implemented in a number of ways. Possible methods of delegating this work item are:

- To a manager role within a certain organization
- To the manager of the user who started the process
- To a specific workflow user that is identified at the start of the process
- To the co-ordinator of a role

In this scenario, we choose to implement this activity with a delegation to the role/group of Retail Manager.

In this scenario, we want to introduce human intervention to handle the situation that arises when the delivery dates exceed the company policy of seven days. As
previously stated, this policy could be implemented with a rules engine. For the purposes of this scenario, a rules engine is not used: we simply put the evaluation condition within the process template.

**Status updates at the end of a process**

This scenario also introduces a second human interaction activity. We assume in this scenario that it is important for the order clerk to receive a confirmation when the process is completed. The decision to include a status update at the end of the process would have been defined in the process requirements as part of the business process re-engineering that usually takes place before a workflow solution is designed. The documented *to-be process* is used as the basis for the workflow design.

The confirmation to the order clerk contains the same data as in the previous two scenarios. However, because an immediate reply may not be available when the clerk initiates the order, an asynchronous process invocation will be used.

Therefore, we need an alternative method to notify the clerk of the details of the completed process. Here are several of the most often-used methods for updating status at the end of a process.

- The most common method is to have the last activity within the workflow process be a staff activity. This activity is delegated to the clerk who initiates the workflow process. We selected this method for our scenario.

- Another possible method is to have the final activity in the workflow process be an automated system activity that sends an email to the order clerk. Sending an email to a participant in the workflow is not desirable in most situations because it is outside the sphere for workflow. Therefore, no auditing can be performed on this activity. Thus, this method cannot easily track the time when the clerk looked at the status notification. If this information is important, do not use this method.

- A third method is to again automate the final activity in the process. However, this time the activity could write the confirmation details to an application database (that is, to a database outside of the process manager database).

The latter two methods are less appealing than the first one. The first method is preferable because it allows the process manager to control the flow and notify the appropriate people based upon the defined business process.

**People-based exception handling**

When you have long-running workflow, timeouts and conditions can be used to ensure that only successfully-completed automatic system activities allow the process to move forward in the business process. If any exceptions occur (that is, if an automated activity returns an error), the process can be routed to the
appropriate support personnel to handle. Typically, this pattern of exception handling is applied to all automatic system activities within a workflow.

Depending on the process manager used, timeouts or non-responses from automatic system activities can be handled in one of two ways:

- A notification can be placed on each automatic system activity. If an automated system activity does not respond in a specified period of time, an activity notification is generated. The modeler builds the process template in such a way that notifications for automated system activities are sent to the work list of a support person.

- The modeler can place an expiry on each automatic system activity. Therefore, if a response is not received in the specified period of time, the activity expires and the process continues down a control connector path that puts a work item on a system administrator’s work list to investigate. Once the problem is corrected, the administrator retries the automatic system activity by completing the work item and having the process manager take control.

The difference between a notification and the activity expiry is that the original request to the automatic system activity is still valid with a notification and that activity waits until it receives a response. In the expiry, the original (timed-out) request to the automatic activity is deemed disposable once the activity expires. The request is eventually re-sent after the system administrator investigates the problem.

In this scenario involving human interaction, we have not applied either of these patterns to the automatic system activities for the sake of simplicity. In reality, some sort of pattern should be applied to all of the automatic system activities to ensure that the process does not become stranded.

### 11.3.3 An alternative solution

A slightly more complex alternative to the one proposed in this chapter exists and may be quite viable. You can consider putting two process templates together to solve the same business problem. Why would you want to have two process templates solve this problem rather than one?

Having the order clerk receive an immediate response containing the order confirmation information is very appealing. With the existing solution, however, the human interaction could possibly delay the completion of the process. Therefore, it is not good practice to have the user wait for a response.

An alternative solution could break this business process into two smaller processes. The first process (Figure 11-2 on page 274) is short-running and have no human intervention. The order clerk always gets an immediate
response. The second process is needed when a manager is required to approve the order.

**The short-running process**
The short-running process is shown in Figure 11-2.

![Figure 11-2 Alternative solution: Short-running process](image)

When the order clerk initiates an order, a short-running process is started. This process will:

1. Get the delivery dates from Wholesaler A and Wholesaler B concurrently
2. Determine the identity of the best wholesaler
3. Place the order with the wholesaler who has the earliest delivery date if either wholesaler can deliver within seven days or less
4. Execute a new activity that will start a long-running process to handle the human interaction part of the process if both wholesalers return a delivery date of greater than seven days: (This activity to trigger the long-running process completes once the process has been started.)

If the process successfully places the order to one of the wholesalers or if any of the activities within the process encounters an error, the process ends and the order clerk is notified immediately. If both of the wholesalers return a delivery date greater than seven days, then a *fire and forget* message triggers the long-running process.
With this alternative, the order clerk now gets an immediate response in all cases. The response the order clerk receives will be one of the following:

- Your order was placed successfully with Wholesaler X. The confirmation number is `<order_num>` and the delivery date is `yyyy-mm-dd`.
- Your order was not placed. An error occurred. The error details are: `<error_detail>`.
- Your order request has been sent to a Retail Manager because the earliest delivery date exceeds the company policy of seven days. You will be notified in your work list when the manager decides the fate of the order.

**The long-running workflow**

The long-running workflow is shown in Figure 11-3.

![Diagram showing the long-running workflow](image)

*Figure 11-3 Alternative solution long-running process*

If required, the long-running process is started by the short-running one. The process steps are as follows:

1. The process is started at the manager approval activity. The manager has two choices: approve the order for the best wholesaler or reject the order.
2. If the order is approved, then the order is placed with the best wholesaler. Once the order is successfully placed, a work item is sent to the order clerk to inform him of the order details.
3. If the order is rejected, then a work item is sent to the order clerk to inform him of the decision.
Why design it this way?

So why would you take a simple business process and add complexity by breaking it into two processes? Some knowledge of the business and actual statistics can help decide which is the best way to implement this business process.

If it is known that a high percentage of the time either Wholesaler A and Wholesaler B return a delivery date of one week or less, then this should factor into how the solution is implemented. Why would you want your order clerk to place an order via application number one (the order entry system) and then expect him to go look in application number two (the work list) for a reply? Most of the time, a work item will already be waiting for him that confirms his order with one of the wholesalers. Would it not make more sense to provide the order clerk with an immediate response if the order usually gets placed immediately? A strong case can be made to implement a simpler solution.

If however, the manager is required to make a decision a high percentage of time, you may not want to implement this solution. You may even wish to consider changing the company’s one week policy.

This solution could also be implemented using an interoperability scenario that involves two process managers. The short-running process may be best served by making this a non-interruptible process in WebSphere Process Choreographer, while the process involving the staff definitions may be better served by WebSphere MQ Workflow. The interoperability scenario is the basis of Chapter 13, “Process manager interoperability” on page 357.

11.4 WebSphere Process Choreographer guidelines

This section describes how to implement the Parallel Workflow variation pattern with human interaction by creating a process that meets the requirements set by the ITSO Electronics business process model. It uses the Runtime pattern and Product mappings described in “Parallel Process: Workflow variation Product mappings” on page 100 for WebSphere Process Choreographer.

You can download the completed process and run it in WebSphere Application Server Enterprise or view it in WebSphere Studio Application Developer Integration Edition. See “Sample scenarios setup” on page 398.

11.4.1 Design guidelines

This section discusses WebSphere Process Choreographer-specific design guidelines used when creating a process that conforms to the Parallel Workflow variation pattern with human interaction.
Process overview: First iteration
The first iteration of the process meets the business objectives of ITSO Electronics as defined in “Stage Two: Internal ordering on demand with approval workflow” on page 113. The completed process is shown in Figure 11-4.

The process has the following characteristics:
- Asynchronous interface
- Interruptible (long-running) process
- Parallel execution paths
- Human interaction with LDAP-user repository

Process overview: second iteration
The second iteration incorporates all of the functions provided in the first iteration and adds one additional characteristic:
- Decoupled sub-process for management approval using JMS.

This second iteration is illustrated in Figure 11-5 on page 278.
This second iteration describes a WebSphere Process Choreographer tailored design for the architectural discussions raised in “An alternative solution” on page 273. The advantages gained by using a synchronous interface and a non-interruptible process include a quick response without checking your work list and better performance when most of the orders do not require an approval.

The invocation of the approval process is achieved by using an asynchronous JMS activity. This decouples the process with a JMS binding. A late binding should be used to retrieve the process starter, and this should be propagated to the child process (the approval process). This information is need by the child process in order to return a notification to the process starter.

**Process interface definition**
The first part of the process interface definition is shown in Figure 11-6 on page 279.
The processOrder operation is separate from the processOrder response operation. Both operations only take an input, with no output. This is automatically generated if you choose to generate an asynchronous, instead of a synchronous, interface.

Figure 11-7 shows the other messages defined in the process interface.

In addition to the input and output message structure for the process, we define:

- A request and response message for our retail manager approval
- A one-way message for our explicit notification
Interruptible verses non-interruptible processes
We need an interruptible process because the business scenario includes:
- Parallel execution of paths that we wish to be processed concurrently
- Human interaction

The process has an asynchronous interface because the process is long-running, potentially taking hours or days to complete while the process pauses for the management approval activity. If, for some business reason, you want to have an interruptible process with a synchronous interface, only the JMS inbound binding supports this combination.

LDAP design
Figure 11-8 shows a partial design structure of the ITSO Electronics organization, as defined using LDAP (Lightweight Directory Access Protocol) notation.

![LDAP design diagram]

Figure 11-8  LDAP design used for our staff resolution

Our focus for this scenario is on the Retail Manager and Retail Staff group. Users in these groups interact with the WebSphere Process Choreographer Process Web Client.

Staff resolution with LDAP
As shown in Figure 11-9 on page 281, the WebSphere Process Choreographer supports different types of user repositories to integrate an existing user directory into an interruptible process with human interaction. We use the LDAP Plug-In in this scenario. One task of the Staff Support Service component is to transform staff queries into concrete LDAP search queries in our example.
Modelling the staff resolution

Figure 11-10 shows how a staff query is resolved in a process model.

There is a comprehensive set of predefined staff verbs available that define abstract definitions used in assigning specific work items. One approach is to
completely specify the information at modelling time with, for example, a distinguished name like cn=George Washington, ou=Retail, o=ITSOElec, c=US. This is called *early binding* in a WebSphere Process Choreographer context.

As an alternative, you can use expressions within the parameters of your chosen staff verb, for example `%wf:process.starter%` within the `users` verb. This is called *late binding*.

Below are a few examples of supplied staff verbs:

- Users (full distinguished name)
- Users by user ID (uid attribute)
- Group members (full distinguished name)
- Manager of employee

Alternatively, you can define expressions that get evaluated at runtime. (For example, you can use them within the `users by user id` verb set):

- `%wf:process.starter%`
- `%wf:process.administrator%`
- `%wf:activity(-enter activity name-).potentialOwners%`
- `%wf:activity(-enter activity name-).readers%`

It is also possible to use custom attributes to set, for example, a value for a Users by user ID verb from the output of a process variable. However this requires use of the custom attributes with the WebSphere Process Choreographer API.

You parameterize *staff verb sets* by adding arguments for a staff activity. The corresponding *verb set mapping file* translates the parameterized staff verb set into an executable query. This executable query is input for the staff support service that produces the concrete executable LDAP query.

**Roles in WebSphere Process Choreographer**

Figure 11-11 on page 283 shows the work item actions that are authorized for each role in the WebSphere Process Choreographer Process Web Client. This refers to roles that are defined for staff activities within a process.
Chapter 11. Creating processes with human interaction

283

Figure 11-11  Work item actions and roles

In our scenario, the Retail Manager has to decide whether to approve or reject an order request. Which actions does the manager (person or group) need to execute to fulfill this approval? Well, the standard process is:

1. The Manager gets a work item in his work list assigned by the work item manager component of WebSphere Process Choreographer. According to our staff verb definition, there is a work item created for the Retail Manager group.

2. The Manager views the work item. Here we simply send the message defined in the process interface so that the manager has the necessary information to make a decision.

3. The Manager claims a work item. Once the manager has done this, another potential owner is not able to claim this particular work item.

4. The Manager can save information within the work item if he does not want to complete it instantly.

5. The Manager completes a work item if he wants to save and complete his work item at once. After that, the work item is no longer assigned to the manager. It is removed from his work list, and the interruptible process continues with the next activity.

As a consequence, the manager needs to have the role of a potential owner in WebSphere Process Choreographer because he needs at least to display, save, claim, and complete the work item.

Processes can also have roles associated with them to limit who can perform certain actions with a process instance. See Figure 11-12 on page 284.
The starter of the process instance automatically becomes the process administrator of the process instance if no one else is specified. That is why you can terminate or delete a process instance even if you are not explicitly defined as process administrator.

For the sake of simplicity, we have no explicit definition of process roles in our scenario.

**Organizational changes in the company**

Organization structures can change on a relatively frequent basis. Therefore, we recommend putting staff activities in a loop and setting an expiration for the staff activity. If the staff activity times out, check for expiration in the loop and continue the loop again so that a new work item gets created. Each time a new work item is created, it will be based on the most current organizational structure.

### 11.4.2 Development guidelines

This section describes how to build the process in WebSphere Process Choreographer. Only the steps specific to this scenario are discussed here.
Building the process

1. Create an interruptible process with a process interface and a flow as defined in the design guidelines. Be sure to set the XSD type for each WSDL part. If you forget to set the type, process variables will not be generated.

2. Create a process as described in the previous chapter, with the modifications shown in Figure 11-4 on page 277.

3. Create a flow within the managerApproval_Block (as shown in Figure 11-13):

![Figure 11-13](image)

*Figure 11-13  Manager approval block*

4. Define two new variables to pass data to the staff activity and to get data from the staff activity (as shown in Figure 11-14). A simple user interface is automatically generated for you.

![Figure 11-14](image)

*Figure 11-14  Passing data to a staff activity and getting data from it*

5. Define the staff query for the staff activity managerApproval. By using the staff query verb *Group Members* with the parameter `cn=RetailManager,ou=Retail,o=ITSOElec,c=US` a work item is created for each member in the Retail Manager group. This procedure is called *early binding* because you specify concrete arguments during development time. Make sure that you do not include spaces in your parameter. See Figure 11-15 on page 286.
6. Create the same construct for the notification.

7. Define the staff query for the notification. By using the staff query verb `Users by user id` and the parameter `%wf:process.starter%` a work item is created for the process starter with all of the aggregated information as defined in the design guidelines. This more dynamic approach is called *late binding*. See Figure 11-16.

8. Make sure that you point to the correct JNDI name regarding the staff plugin provider. Within the properties of the service project, there is a staff tab where you can set this property (as shown in Figure 11-17 on page 287).
9. In the server settings of the process editor, ensure that the process will run as interruptible. Now that we have added notification there is no need to retain completed process instances, so check Delete Process on Completion.

**Team Programming Considerations**

Make sure that the VerbSet.xml of your service project is also committed to CVS, if modified from the default. If you do not ensure this, someone who checks out your service project will not get the content of the staff tab of your activity.

**Hints for implementing the second iteration**

Let's assume that both the short- and long-running processes are implemented in WebSphere Process Choreographer. Create a new interruptible process with an asynchronous interface that does the approval, notification, and order placing in case an approval is needed. Once it is complete, generate the deployed code for this new process with a JMS inbound binding.

Now simply copy the `<processname>Interface.wsdl`, `<processname>JMSBinding.wsdl`, and `<processname>Service.wsdl` to the parent process after you have made sure that the `<jms:address>` in the `<processname>JMSService.wsdl` is pointing to the correct connection factory and destination queue.

Change the original process to a non-interruptible process with a synchronous interface. Then, simply drag and drop the `<processname>Service.wsdl` to your canvas and prepare the input messages.

See the runtime guidelines for additional hints for the deployment.

### 11.4.3 Runtime guidelines

You should make sure that your business process container is properly configured for using human interaction. This involves the following:
- Setting up a user registry and configuring security in WebSphere Application Server Enterprise. (We use LDAP.)

- Configuring the business process container (using the wizard) and the LDAP staff plug-in provider. We recommend using DB2 as the database and Embedded Messaging or an external WebSphere MQ for usage with WebSphere Process Choreographer.

In order to test the complete functionality of the process, the test cases shown in Table 11-1 must be considered:

<table>
<thead>
<tr>
<th>Best Wholesale</th>
<th>Delivery Days</th>
<th>Manager decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>( \leq 7 )</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>( \leq 7 )</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>&gt; 7</td>
<td>accepted</td>
</tr>
<tr>
<td>A</td>
<td>&gt; 7</td>
<td>rejected</td>
</tr>
<tr>
<td>B</td>
<td>&gt; 7</td>
<td>accepted</td>
</tr>
<tr>
<td>B</td>
<td>&gt; 7</td>
<td>rejected</td>
</tr>
</tbody>
</table>

We will work through one of these test cases in this section. We will use the test case where the calculated best wholesaler is A, the delivery days over seven days, and the manager accepts this order.

We will use one member of the retail management group and one member of the retail staff group to run our test case in the Process Web Client. The LDAP definition for the process starter, Phil Redding, is shown in Example 11-1.

**Example 11-1   Process starter as defined in LDAP**

```
dn: cn=Phil Redding, ou=Retail, o=ITSOElec, c=US
objectclass: ePerson
objectclass: inetOrgPerson
objectclass: organizationalPerson
cn: Phil Redding
userpassword: fun4you
uid: predding
manager: cn=Jane Baxter, ou=Retail, o=ITSOElec, c=US
```

The LDAP definition for the retail manager, Jane Baxter, is shown in Example 11-2 on page 289. Additionally we have defined a group called ProcessManager that includes Jane Baxter as member of the management team.
Example 11-2  Group of retail manager as defined in LDAP

dn: cn=RetailManager, ou=Retail, o=ITSOElec, c=US
objectclass: groupOfNames
objectclass: top
description: Retail Manager
cn: RetailManager
owner: cn=Gary Washington, ou=Retail, o=ITSOElec, c=US
member: cn=Gary Washington, ou=Retail, o=ITSOElec, c=US
member: cn=Jane Baxter, ou=Retail, o=ITSOElec, c=US

We log in to the Process Web Client as user predding and create a new process instance. View the process instance by selecting Process Instance Lists → Created by me (as shown in Figure 11-18). This view shows you the state of process instances and the state of its activities. All activities shown here are in a state of finished except for the HumanInteraction.managerApproval_Block.

![Created By Me](image)

**Figure 11-18  Interruptible process in the Process Web Client**

Next, we log in to the Process Web Client as jbaxter and check her work list, as shown in Figure 11-19 on page 290.
Jane Baxter has a work item associated to her by the work item manager because she is a potential owner as defined in the verb set on the manager approval staff activity. If you click on the work item (which has a state of *ready*), you are shown the aggregated information passed in the manager approval request message, as shown in Figure 11-20.

If the manager wishes to approve this request, she has to claim this work item. After she has claimed the work item, no other potential owner is now able to claim it. The the state is now *claimed* (as shown in Figure 11-21 on page 291).
The final step is to complete the activity output message with a decision of accepted (as shown in Figure 11-22).

If you have completed this activity, the next staff activity takes place and sends a notification in the form of a work item to the starter of the process. If the process starter (Phil Redding) logs in again, he will see the notification regarding his order and its detail if he views the work item (as shown in Figure 11-23 on page 292).
Hints for deploying the second iteration
Define a listener port and a queue for the approval process. Do not use the WebSphere Process Choreographer destination BPEApiQueue because this is used by the internal process engine message-driven bean.

11.5 WebSphere MQ Workflow guidelines

This section describes how to implement the Parallel Workflow pattern with human interaction. It also describes how to create a process that meets the requirements set by the ITSO Electronics business process model. It uses the Runtime pattern and Product mappings described in “Parallel Process: Workflow variation Product mappings” on page 100 for WebSphere MQ Workflow.

You can download the completed process and run it in WebSphere MQ Workflow or view it in WebSphere Business Integration Workbench. See “Sample scenarios setup” on page 398.

11.5.1 Design guidelines
This section discusses WebSphere MQ Workflow specific-design guidelines used when creating a process that conforms to the Parallel Workflow variation pattern with human interaction.
Process overview
The process is too large to show in one picture, so is split into two parts. Figure 11-24 shows the left side of the process.

Figure 11-24 Process overview: Part one

Figure 11-25 shows the right side of the process.

Figure 11-25 Process overview: Part two
The process has the following characteristics:

- Parallel execution paths
- Human interaction activities
- Automated activities

This model is an extension of the Parallel Process scenario implemented in the previous chapter.

### Staffing and organization models

The organization unit consists of subdivisions within an organization. They can be departments, divisions or sections. In this scenario, the organization is ITSO Electronics, which has three organization units: Retail, Wholesaler A, and Wholesaler B.

Roles play an important part in modeling human interaction in a workflow. An individual is assigned to a role, such as manager or auditor. In the workflow, role-based staff assignment is used to limit the authority a user has for a given process template.

In this scenario, the following roles are defined for the ITSO Electronics organizational unit:

- **Retail**, which requests that an order be placed with a wholesaler. It consists of the following three groups of roles defined for requesting, approving, and processing the order:
  - **RetailManager**: a role for the managers who approve an order
  - **RetailStaff**: a role for requesting an order and for checking the result of the order
  - **ITGroup**: a role for handling automated task system failure that can include system administrators as members

- **Wholesaler A**, which supplies parts to the Retail department and includes:
  - **WholeA_ITGroup**, which maintains the systems for Wholesaler A

- **Wholesaler B**, which supplies parts to the Retail department and includes:
  - **WholeB_ITGroup**, which maintains the systems for Wholesaler B

### Task definition

In addition to the tasks defined in the previous scenarios, there are two human interaction activities to add to the process. These tasks are:

- **GetApproval**, which gets approval from RetailManager and returns the result with a status of approved or rejected
- NotifyStatus, which shows the result of the order process to RetailStaff

**Process and task interface definition**

The process output data structure needs an additional data field to indicate whether management approval is given for an order. Also, the Wholesale data structure (which is used as a placeholder for storing data required by multiple tasks within the process) needs to store both (1) confirmation codes and (2) whether an item was approved.

To avoid interfering with the previous scenarios, we are creating new data structures called OrderOutputApproval and WholesaleApproval. Once these structures are created, we are adding new data fields them, as shown in Figure 11-26.

![Data Structures Tree]

*Figure 11-26  New data structures added to the repository*

**Decisions and choices**

We will use decision objects with multiple choices in this scenario. There are two decisions with choices that must be considered. The decisions are as follows:

- Compare two delivery days and determine whether management approval is required. Approval is required if the best delivery date is greater than seven days.
- Determine whether the RetailManager approves the order.
Applications and integration
As in the previous scenarios, the automated tasks are implemented by Web services. The two additional human interaction tasks may require Java Server Pages (JSPs) created to allow a human to input responses. Human interaction tasks can be implemented in several ways:

- Write a client using WebSphere MQ Workflow APIs in one of the API interfaces such as Java, C/C++, or ActiveX. We need to implement the work list functionality if we choose this type of implementation.

- Use the WebSphere MQ Workflow Client on a Windows workstation. This supplied client could be used if configured. It allows us to view the containers if we use fmcnshow.exe as the program to execute in the activity program definition.

- Use the WebSphere MQ Workflow Web Client. This supplied client can be modified by putting customer Java Server Pages (JSP) in the process. It programs directories of the deployed WebSphere MQ Workflow Web Client.

The Web-based approach is the most popular method to implement human interaction pages because it uses a thin client: a Web browser. We elect to use the WebSphere MQ Workflow Web Client because of the ease of creating JSPs using a SupportPac. Also, the Web client process monitor can display the input and output container of the activities to see what data is carried into and out of each activity.

11.5.2 Development guidelines
This section describes how to build a Parallel Workflow pattern implementation for ITSO Electronics. This process is essentially the same as the process built in the previous scenario, with the addition of human interaction.

Building the process
This section describes how to build the process using the WebSphere Business Integration Workbench. Use the organization repository we define in the previous two scenarios (ITSOElec.org).

Role and employee definitions
In “Staffing and organization models” on page 294, we identify a number of roles. We need to define these roles in the repository by performing the following steps:

1. Start the WebSphere Business Integration Workbench with the ITSOElec.org organization open. Switch to Integration mode.

2. Click Repository → Organization Data → Roles. You should see that a role of RetailManager is already defined.
3. Define roles for RetailStaff, ITGroup, WholeA_ITGroup, and WholeB_ITGroup (as shown in Figure 11-27).

![Figure 11-27  Role definitions](image)

The next stage is to create employees and assign them to roles and organization units. In a real model, there would be multiple employees assigned to each role. For simplicity, we will assign just one employee to each role.

1. Click Repository → Organization Data → Employees.
2. You will see a list of three employees already defined: MGR, MGRA, and MGRB. These employees are created for the previous scenarios as placeholders. We ignore these employees in our scenario.
3. Create a new employee called GWASHINGTON and assign it to the Retail organization unit and the RetailManager role (as shown in Figure 11-28 on page 298). Click Add to add the employee to the repository.
4. Define the remaining employees as described in Table 11-2.

Table 11-2  Employees assigned to roles

<table>
<thead>
<tr>
<th>Organization unit</th>
<th>Role</th>
<th>Employee name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retail</td>
<td>RetailStaff</td>
<td>JCAMPBELL</td>
</tr>
<tr>
<td>Retail</td>
<td>ITGroup</td>
<td>WASADMIN</td>
</tr>
<tr>
<td>Wholesaler A</td>
<td>WholeA_ITGroup</td>
<td>WHOLEA</td>
</tr>
<tr>
<td>Wholesaler B</td>
<td>WholeB_ITGroup</td>
<td>WHOLEB</td>
</tr>
</tbody>
</table>

Adding data structures and fields to the repository

As discussed in the design guidelines, we need to add new data structures and data fields to the repository. Use Repository → Process Data → Data Fields to add data fields and new structures and Repository → Process Data → Data Structures to assign data fields to these structures.

The following data structures should be defined. Refer to the design guidelines to see the fields in each structure.

- OrderOutputApproval
- WholesaleApproval
Creating a new process

Create a new process by selecting File → New Process. Make sure you are in Integration mode. (The keyboard shortcut to switch to this mode is Alt-3.) Click Process → Info to open the process properties. Then, perform the following:

- Select General. Then, set the process name and MQ Workflow name to HumanInteraction.
- Select Fields. Then, set the Input structure to OrderInput and the Output structure to OrderOutputApproval.
- Select Process. Then, check Keep finished processes forever.

Application definitions

The management approval and notify status tasks both need applications defined for them. We define two applications: SgetApproval and SnotifyStatus. To create the SgetApproval application, perform the following:

1. Click Repository → Organization Data → Applications.
2. Create a new application with a name of SgetApproval. Set the active platform to Windows NT® and ensure Run unattended is not selected. See Figure 11-29.

3. Click Parameters. We want the manager to see the WholesaleApproval data structure to allow him to make a decision on whether to approve or reject an
order. Therefore select **WholesaleApproval** as the Input Structure and Output Structure, and check **Program requires these** (as shown in Figure 11-30).

![Figure 11-30](image)

**Figure 11-30  Defining SgetApproval: Parameters tab**

4. Click **Windows NT**. Set the path to `fmcnshow.exe`.

Repeat the process to define `SnotifyStatus`. This application should use the `OrderOutputApproval` data structure instead of `WholesaleApproval`.

**Task object definitions**

Referring to the process overview in Figure 11-24 on page 293, add the following Web service tasks and configure them as you have done for the previous scenarios. (Remember to set them to only wait for one input.)

- `getDeliveryDaysA`
- `getDeliveryDaysB`
- `placeOrderA`
- `placeOrderB`

Define five NOOP tasks. All NOOP tasks share the following characteristics:

- Belonging to the Retail organization unit
- Using the FMCINTERNALNOOP application
Using the WholesaleApproval data structure
- Being asynchronous
- Being automated
- Containing a default mapping that maps _STRUCT to _STRUCT

In addition, the following NOOP tasks have the following special properties:
- NOOP waits for all inputs before beginning automatic execution.
- NOOP_01 and NOOP_03 have a loop mapping that sets bestWholesaler to A.
- NOOP_02 and NOOP_03 have a loop mapping that sets bestWholesaler to B.

Create a management approval task that is used by employees in the RetailManager role to decide whether an order should be placed based on the delivery days from the best Wholesaler.

1. Create a new task with a Task name and MQ Workflow name of getApproval. Assign this task to the Retail organization and set the application to SgetApproval (as shown in Figure 11-31).

2. Click Staff Assignment. You can assign this task to a role. Set the role to RetailManager (as shown in Figure 11-32 on page 302).
3. Click the **Automation** tab and set the task to **start manually** on one input.

4. Click the **Data** tab and set the input and output data structure to **WholesaleApproval**. Create a default mapping for `_STRUCT`.

Create a notify status task that is used by the person who started the process to view the output of the process.

1. Create a new task with a Task name and MQ Workflow name of **notifyStatus**. Assign this task to the **Retail** organization, and set the application to **SnotifyStatus**.

2. Click **Staff Assignment**. This time, we will assign the work item to the person who started the process instead of setting a dynamic staff assignment. Set the Staff Assignment to **Process Starter** (as shown in Figure 11-33 on page 303).
3. Click Automation and set the task to start manually on one input.
4. Click Data and set the input and output data structure to OrderOutputApproval. Create a default mapping for _STRUCT.

**Decision and choice object definitions**

There are two decision objects with choices to add to the process. Add the first decision object as follows:

1. In the ADF toolbar, select Decision and add a decision to the ADF diagram.
2. Create a connector between NOOP and the decision object. This connector is required so that we can use the output of NOOP.
3. Name the decision object Check days and set it to a type of Multiple. See Figure 11-34 on page 304.
4. Add four choice objects to the ADF diagram and create control connectors between the choice objects and the decision object.

5. Open the properties of the first choice object. This object is selected if the best Wholesaler is determined to be Wholesaler A and the delivery days value of Wholesaler A is less than one week (thus not requiring management approval). Set the choice object with the name and expression shown in Figure 11-35.

6. Set the remaining choice objects with the values shown in Table 11-3 on page 305.
Perform the following steps to create a second decision object with choices to determine the flow after the management approval task:

1. Add a new decision object, called `Management decision` and set it to a type of `Multiple`.
2. Create a control connector between the `getApproval` task and the decision object.
3. Add three choice objects and connect them to the decision object.
4. Set the choice objects with the values shown in Table 11-4.

### Table 11-4  Choice object settings for Management decision

<table>
<thead>
<tr>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved and best=A</td>
<td>approved=&quot;Y&quot; AND bestWholesaler=&quot;A&quot;</td>
</tr>
<tr>
<td>Rejected</td>
<td>approved=&quot;N&quot;</td>
</tr>
<tr>
<td>Approved and best=B</td>
<td>approved=&quot;Y&quot; and bestWholesaler=&quot;B&quot;</td>
</tr>
</tbody>
</table>

Create control connector links for all of the choice objects, as shown in Figure 11-24 on page 293.

**Data mapping between tasks**

Data mappings for the automated tasks are the same as in the previous scenario. The following mappings also need to be made:

- From Source to `getApproval`, passing `partNo` and `qty`.
- From `NOOP` to `NOOP_01`, `NOOP_02`, `NOOP_03`, and `NOOP_04` passing `_STRUCT`.
- From `NOOP_01` to `getApproval`, passing `_STRUCT`.
- From `NOOP_02` to `getApproval`, passing `_STRUCT`.
- From `getApproval` to `NOOP_03` and `NOOP_04`, passing approved.
- From `getApproval` to `notifyStatus`, passing approved and `bestWholesaler`.
- From `placeOrderA` to `NOOP_03`, passing `confirmCode` to `confirmCodeA`. 
- From placeOrderB to NOOP_04, passing confirmCode to confirmCodeB.
- From NOOP_03 to notifyStatus, passing daysA to deliveryDays, confirmCodeA to confirmCode, approved to approved, and bestWholesaler to bestWholesaler.
- From NOOP_04 to notifyStatus, passing daysB to deliveryDays, confirmCodeB to confirmCode, approved to approved, and bestWholesaler to bestWholesaler.
- From notifyStatus to Sink, passing _STRUCT

**Exporting the process to the runtime database**

1. Once the process is built, verify it by selecting **Process → Process Validation → MQ Workflow Model**. Correct any errors reported here.
2. Export the process to a FDL file that can be used in WebSphere MQ Workflow V3.4 (deselecting **Export NOOP**).
3. Import the FDL file using the following command to the WebSphere MQ Workflow runtime:
   ```
   ```

**Building the human interaction Java Server Pages**

We had a choice of using the WebSphere MQ Workflow Web Client default HTML pages for the human interaction or building JSPs. We elected to build the JSP using a WebSphere MQ Workflow--Rapid Deployment Wizard WA83 SupportPac available from the following URL:


To use the SupportPac, we add it as a plug-in to WebSphere Studio Application Developer 5.1. To install the support, you must extract it to:

```
<drive>:\<program files>\IBM\Websphere Studio\Application Developer\v5.1\eclipse\n```

Once you extract the SupportPac to the directory, you need to restart WebSphere Studio. Once restarted, perform the following:

1. Click **Window → Customized Perspective**.
2. Expand **File → New**.
3. Select **JSP for MQWF Web Client**.
4. Click **OK**.
5. Select **File → New → Project**.
6. Select **Web → Dynamic Web Project**.
7. Enter the project name HumanInteractionWeb and click Configure advanced options. Then, click Next.
8. Specify an EAR project of HumanInteractionEAR and click Finish.

Creating the process instance instantiation JSP

With the WebSphere MQ Workflow--Rapid Deployment Wizard installed, we are ready to create the JSP that we need when we instantiate the process in the WebSphere MQ Workflow Web Client. The steps are as follows:

1. Select File → New → JSP for MQWF Web Client
2. Click Browse and locate the FDL for the HumanInteraction process. Highlight HumanInteractionWeb and click Next.
3. The HumanInteraction process and its activities should be displayed (as shown in Figure 11-36). Unselect Generate JSP for the activity and click Next.

4. Select Force to create the JSP file and click Next.
5. Select the partNo and then select Output this field (as shown in Figure 11-37 on page 308). Do the same for qty and then click Next.
Figure 11-37  Outputting the partNo field

**Note:** We left the *Field Type* drop-down as *Text Field* even though there are other choices because we plan to alter the JSP later.

6. At this next step, we could alter the field locations, but we leave them as provided and click **Next**.

7. At this step, we could alter the action buttons but we leave them as provided and click **Finish**.

The JSP page is created. We need to modify the classpath to remove a number of compilation errors. To do this, perform the following steps:

1. In the Project Navigator view, right-click on **HumanInteractionWeb** and select **Properties**.

2. In the Properties window, select **Java Build Path**. Click **Libraries** and click **Add External JARs**.

3. You need to find fmcohcli.jar, which is located in the WebSphere MQ Workflow BIN, directly. Highlight it and click **Open**.
4. Repeat the same method to add fmcojapi.jar from the WebSphere MQ Workflow BIN\Java3400 directory.

5. Click **OK** to close the Properties window, and the compilation errors should be fixed.

You can view the JSP page by locating it in the Project Navigator and double-clicking on **HumanInteraction.jsp** to open it in the Page Designer editor (as shown in Figure 11-38).

![Figure 11-38  Created JSP in the Page Designer editor](image)

You could leave the JSP as shown, but we have decided to add our ITSO Electronics Retail staff in a picture and incorporate some tables and headings. Once your page is complete, it needs to be moved to the WebSphere Application Server installedApps directory where you have your configured WebSphere MQ Workflow Web Client. This type of WebSphere MQ Workflow Web Client JSP should be in the following default path:

```
<drive>:\<WebSphere>\installedApps\<hostname>\MQWF_Web_Client_FMC.ear\fmcohcli.war\processes
```

If you use an image as we did, the default location for the image is the following:

```
<drive>:\<WebSphere>\installedApps\<hostname>\MQWF_Web_Client_FMC.ear\fmcohcli.war\images
```
Creating the approval and notify status activity JSPs

We can use the WebSphere MQ Workflow Web Client default HTML pages for the manager's approval page and the notify status page. We have elected to build JSPs for these activities. The procedure follows almost the same steps as described in “Creating the approval and notify status activity JSPs” on page 310.

The differences arise when you reference Figure 11-36 on page 307, as follows:

1. You must select Generate JSP for the activity.
2. Select GetApproval for the activity.
3. In the Form Properties screen, select the following fields from the Input tab and select output this field:
   - partNo
   - qty
   - daysA
   - daysB
   - bestWholesaler
4. Also in the Form Properties screen, select the approved field from the Output tab and select output this field. This is the only data output container that should be editable. Make sure that the others are not set to output. The manager will update the approval field to Y or N, depending upon the decision he makes.

Repeat the steps above for the Notify Status JSP, but deselect all output data containers (so that nothing is editable). You can use Page Designer to make title or other style changes.

After both JSPs are complete, they must be copied to the default location for the WebSphere MQ Workflow Web Client activity pages, as follows:

```
<drive>:\WebSphere\installedApps\hostname\MQWF_Web_Client_FMC.ear\fmcohcli.war\programs
```

If you put any additional images in the JSP, you copy the images to:

```
<drive>:\WebSphere\installedApps\hostname\MQWF_Web_Client_FMC.ear\fmcohcli.war\images
```

11.5.3 Runtime guidelines

This is the first scenario to involve human interaction. For this scenario, we had the choice of:

- Writing our own client using WebSphere MQ Workflow APIs
We elected to use the Web client. The Web client allows us to view the custom JSPs we create for the human interaction tasks and for the process instance creation page.

**Starting and monitoring process instances with the Web client**

Perform the following to test one of the execution paths through the HumanInteraction process:

1. Start the Web client and log in as userid JCAMPBELL (with the password as password). This is the user we earlier defined as belonging to the RetailStaff role, with the responsibility for placing orders.

2. Create and start a new instance of the HumanInteraction process template. As this point, the custom JSP that you created for this page should be page-compiled by WebSphere Application Server and then displayed in the browser (as shown in Figure 11-39).

3. Enter a partNo of ABCDE and any value of qty. A part number of ABCDE resolves to Wholesaler A offering delivery in one day and Wholesaler B in two
days. This order will not require management approval because the best Wholesaler can deliver in under seven days. Click **Apply for Instance**.

4. Switch to the ProcessInstances view and monitor the process. You should see that (1) getDeliveryDaysA and getDeliveryDaysB are executed and (2) placeOrderA has been executed (hence skipping management approval). The process should be waiting at the notifyStatus activity. This is a human interaction activity, and a work item is created for the starter of the process.

5. To view work items, you need to create a new list. Back in the Navigate pull-down menu, select **List of Lists** and then click **Create a list**. In the Create a new list page, create a new Work List called **WorkList** (as shown in Figure 11-40).

![Figure 11-40 Defining a work list](image)

6. Switch to the WorkList view, and you should see a new work item called notifyStatus, which is assigned to owner JCAMPBELL. See Figure 11-41.

![Figure 11-41 New work item generated for notifyStatus](image)

7. Click **Check out Work item** to claim the work item. The customized JSP you created for notifyStatus should be displayed (as shown in Figure 11-42 on
This work item is for informational purposes only, so there is no data to input. To complete the work item, click Complete work item.

Summary of order

Best Wholesaler: A
Delivery days of best Wholesaler: 1
Confirmation code of order: 61153_ABCDE (note: a confirmation code of zero indicates the order was not placed)
Approved by RetailManager?

Figure 11-42 Customized JSP for notifyStatus

8. Once the work item has completed, the process will populate the output container and end. Switch to the ProcessInstances view. The state of the process instance should be set to complete. Additionally, you can examine the output container of the process. The information in the output container will be identical to the information shown in the notifyStatus work item.

Using the Web client for management approval

In this section you will create another process instance. This time, the management approval functionality is tested. Perform the following steps:

1. While still logged into the Web client as JCAMPBELL, create a new process instance for HumanInteraction, specifying a partNo of KJANH. This will resolve to Wholesaler A, which offers delivery in 11 days and Wholesaler B, which can deliver in 10 days. Because the best Wholesaler requires 10 days to deliver, management approval is required.

2. View the process instance in the monitor. You should see that (1) getDeliveryDaysA and getDeliveryDaysB are executed and (2) the process is waiting at the getApproval activity.

3. Switch to the WorkList view. Notice you do not see any work items in the view. Although a work item has been created, it is only visible to users in the RetailManager role.

4. In a separate browser window, start another instance of the Web client. This time, log in as userid GWASHINGTON. This user is part of the RetailManager role. Move to the WorkList view, if you are not already there. You should see a new work item called getApproval (as shown in Figure 11-43 on page 314).
5. Click **Check out Work item**. You should see the customized JSP page you created for getApproval. The approve data field needs to be set in this work item. Valid values are Y or N. Approve the order by entering Y into the text field (as shown in Figure 11-44) and then click **Complete work item**.

![Figure 11-43 New work item generated for getApproval](image)

### Management approval needed for part number KJANH

| Quantity | 3 |
| Delivery days for Wholesaler A | 11 |
| Delivery days for Wholesaler B | 10 |

**Do you approve this order? [Y/N]**

![Customized JSP for getApproval](image)

6. Switch back to the other Web client browser where you are logged in as JCAMPBELL. A new notifyStatus work item will be created. Notice that his work item contains a non-zero confirmation number, indicating management approval was received and the order was placed. Once the notifyStatus work item is complete, the process will be set to a state of complete.

You can experiment with other paths through the process, such as denying management approval of an order. If you encounter problems, use the process monitor to track the state and input and output container of each activity.
Creating processes with events and compensation

This chapter describes how to build a process that uses the Parallel Workflow variation pattern with events and compensation. The chapter is split into the following parts:

- **Business scenario**
  Describes a business problem ITSO Electronics wishes to solve

- **Business process model**
  Describes the business process model to solve the business problem

- **General design guidelines**
  Provides product-agnostic design guidelines for building a process conforming to the Parallel Process Workflow variation pattern

- **WebSphere Process Choreographer guidelines**
  Describes how this process was implemented using WebSphere Process Choreographer

- **WebSphere MQ Workflow guidelines**
  Describes how this process was implemented using WebSphere MQ Workflow
12.1 Business scenario

In this scenario, we introduce two concepts that are quite common in real life:

- **External events**
  
  Events that are external to the main process quite often can influence the behavior of the process. In our scenario, we model the behavior that ITSO Electronics waits for the delivery of the parts before the process can complete into the process template. This is a common scenario.

  The scenario also includes a *timer* on the activity that waits for delivery. If the delivery is not received within the expected period of time, this *wait* activity expires and a work item is delegated to the receiving department to follow up. Once the follow up occurs, the process waits again for the delivery.

- **Compensation**
  
  Although compensation does not have a great influence on the business scenario, it does impact the implementation. The idea of compensation suggests that within a certain group of activities (a sphere), if one activity (or transaction) fails, then all the previous activities that completed successfully within that sphere are reversed, or compensated.

  In this scenario, ITSO Electronics is ordering parts from two wholesalers, that when combined, form a single part for ITSO Electronics to resell. ITSO Electronics will order a toy from Wholesaler A and a battery required for the toy from Wholesaler B. Because ITSO Electronics wants to minimize its inventory carrying costs, it deems it necessary that if the order for either part cannot be fulfilled and the order for the other part should be canceled.

  This scenario also includes the receipt of the parts by the receiving department and how this event gets reflected in the process. This is necessary because the workflow does not complete until both parts are received by ITSO Electronics.

12.2 Business process model

The swimlane diagram for this workflow is shown in Figure 12-1 on page 317.
Similar to the previous scenario, the order clerk initiates the workflow by making a request. But this time, the order clerk must order two distinct parts, each with a quantity, on a single order. Once the order clerk places the order, a process is started that executes the following process logic:

- Requests are sent to both Wholesaler A and B to place the order for the toys.
- If the order for the toys is unsuccessful, then the process ends and the process starter is notified of the error.
- If the order is placed successfully, an order is placed with Wholesaler B for the batteries.
If the order for the batteries fails, then a compensating transaction/order must be placed for the toys. Once this happens, the process ends and the process starter is notified of the error.

If both orders are placed successfully, the process splits again and waits for the receipt of both parts.

If a part is not received in its expected time, the wait activity will expire and the receiving department is delegated a work item.

The receiving department follows up with the appropriate wholesaler and the process waits once again for the delivery.

Once a part is received, it is assumed that there is a link between the receiving system and this system. An external event is then triggered to let the workflow know that a particular part order has been received. Once this happens, this segment of the workflow has completed, although the workflow does not end until both parts for the same order are received.

12.3 General design guidelines

In this section we look at how introducing compensation into a process affects the process. Decisions need to be made with respect to how much to include in the sphere of compensation and how to minimize the amount of compensation required.

Additionally, in processes that include external events, the modeler and solution architect need to ensure that the process does not get stranded if the event never occurs. In this section, we examine the methods that can be used to accomplish this goal.

12.3.1 Design overview

This scenario is an implementation of the Process-focused Application Integration::Parallel Workflow variation pattern.

For information about the Parallel Process: Workflow variation Application pattern, see:

3.4.9, “Parallel Process: Workflow variation” on page 63

For information about the Parallel Process: Workflow variation Runtime pattern, see:

5.1.4, “Parallel Process: Workflow variation Runtime pattern” on page 90
12.3.2 Design considerations

This scenario introduces three design considerations that we have not yet discussed in this redbook: compensation, external events, and expiry. This section examines some of the factors that lead to the implementation in this scenario, as well as other options.

It should be noted that there are considerations that we didn’t account for in our implementation. That is, there are other events that could complicate the process. We assume that the parts shipment will always arrive. We assume that the shipment received by ITSO Electronics is always exactly what they ordered. The shipment is never shorted, nor is a wrong part ever received in error, nor does the receiving department of ITSO Electronics make an error when recording the shipment. We also assume that ITSO Electronics will never cancel an order after both the orders for the toys and the batteries are successfully placed.

Compensation

Within a process, a number of automated transactions may be modeled. These automated transactions usually alter data in data stores outside the process manager. If several of these automated transactions are deemed to be a logical transaction, then it makes sense that if one of the transactions fail, the transactions that successfully executed should be reversed to their state before the sequence of transactions began. This concept is known as compensation.

Because ITSO Electronics wants to keep inventory costs to a minimum, it only orders complimentary parts in like quantities. It is assumed that the toys cannot be used without the batteries and that the batteries are useless to ITSO Electronics without the toys. This being the case, the order for the toys is placed first. If this transaction is not successful, then only an update to the process starter is required. Once the toy order is successfully placed, the order for the batteries is placed. If this fails, then the corresponding order for the toys must be canceled. This is the compensating transaction in our scenario.
As an alternative, we could have placed the orders to the wholesalers in parallel, as we did in the second scenario. By doing this, it increases the chance of having to cancel an order by 50%. That is, if we model the process to place an order to both wholesalers at the same time, there is twice the opportunity that a transaction will fail and thus require compensation. With the way we implement the scenario, the only time compensation is involved is if the place order to Wholesaler B fails. Then, the order to Wholesaler A needs to be canceled. It is a business requirement of ITSO Electronics to model the orders serially because ITSO Electronics is levied a financial penalty by the wholesalers any time an order is canceled.

Another consideration is to determine how far the sphere of compensation extends. In other words, what makes up a logical transaction that dictates if one transaction fails, then the other transactions need to be reversed? In order to reduce complexity, you may want to consider breaking the transaction up into smaller spheres for complexity and performance reasons.

With the two process manager products we are working with, compensation is native to WebSphere Process Choreographer. Therefore, implementing compensation is relatively straight-forward. WebSphere MQ Workflow does not natively support the concept of compensation. However, it is quite common to model one or more activities to reverse transactions that already have taken place in WebSphere MQ Workflow in case a transaction fails.

**Expiry**

Putting an expiry on an activity means that within the process template, the modeler can define that if an activity does not complete within a period of time, then the activity *expires* and the process continues. Conditions are typically modeled on the control connectors from this activity that determine the path of execution depending on whether the activity expired or not. This is a way of ensuring that if an activity does not complete with the time specified by the business, then some other action should occur.

By employing a pattern that consists of a wait activity followed by a follow-up activity, detailed metrics can be obtained for ITSO Electronics’s relationship with its wholesalers. Therefore, this modeling technique provides valuable audit information gathered by the process manager.

ITSO Electronics can now query the audit trail and see how reliable its suppliers are. It can determine how quickly the parts arrive after the order is placed and calculate the average delivery date. However, the most useful metrics may be things like how often a supplier delivers a part on time, how often it delivers after the expected delivery date, and how often and how much time is spent following up with the wholesaler for late deliveries. Benefits can be gained from this modeling implementation as business continues its quest to reduce cycle time.
WebSphere MQ Workflow currently provides more flexibility with expiry because an expiry can be placed on any type of activity. With WebSphere Process Choreographer, expiry is handled natively for staff and event activities. However, activities such as Web services eventually time out, even though this is not controlled by the WebSphere Process Choreographer.

**External Events**

It is common for events that happen outside of a process to influence the process. These are typically referred to as external events. For example, you can have a customer-based order process that would end if the customer called up and canceled its order subsequent to placing it. Because you cannot model the cancellation as part of the process, an event triggers the running process to be terminated.

Another example is having a process that essentially stops until some criteria is met. In our scenario, we do not want the process to continue until both of the parts are received. To be entirely correct, our process does not continue when both parts are received. Instead, the process ends. But even modeling it this way provides the business analysts with metrics to determine the entire process cycle time.

In this scenario, the external event is modeled as two activities. First there is a wait activity. This activity serves no other purpose than to just wait for the delivery of the part. Once the delivery of the part is received for the activity that the process is waiting for, the activity is completed and the process continues. If the delivery is not received within the expected time frame, the activity expires as discussed previously and the second activity, a staff activity, is triggered to request that a human follow up the order with the wholesaler.

External events modeled in this way need to ensure that if the event is triggered while the process is at the staff activity of following up with the wholesaler, the follow-up activity no longer needs to be completed by the receiving department. The process manager simply completes this part of the process as it would if the part were received during the Wait For Delivery activity. Our scenario is simplified, so we ignore this possibility, but considerations like this are necessary in a production workflow.

### 12.4 WebSphere Process Choreographer guidelines

This section describes how to implement the Parallel Workflow variation pattern with events and compensation. It also describes how to create a process that meets the requirements set by the ITSO Electronics business process model. It uses the Runtime pattern and Product mappings described in “Parallel Process:
Workflow variation Product mappings” on page 100 for WebSphere Process Choreographer.

You can download the completed process and run it in WebSphere Application Server Enterprise or view it in WebSphere Studio Application Developer Integration Edition. See “Sample scenarios setup” on page 398.

12.4.1 Design guidelines

This section discusses WebSphere Process Choreographer-specific design guidelines when creating a process that conforms to the Parallel Workflow variation pattern with events and compensation.

Process overview

This process meets the business objectives of ITSO Electronics as defined in “Stage Three: Ordering on demand with multiple wholesalers” on page 115. The completed process is shown in Figure 12-2.

The process has the following characteristics:

- Asynchronous interface
- Non-interruptible (long-running) process
- Parallel execution paths
- External events
- Compensation pairs

Process interface definition

The first part of the process interface definition is shown in Figure 12-3 on page 323.
The processOrder operation is one-way, and it contains the placeOrderRequest input message that accepts two part numbers and two quantities. The processOrderNotification operation contains the placeOrderResponse message that provides confirmation codes for the two orders.

Figure 12-4 shows the other messages and operations defined in the process interface.

This section of the process interface describes the following:

- The receiveOrderWholesaleA operation is a one-way operation, used by the waitForDelivery_WholesaleA_Event. We do not require any data to be sent to this event, merely that the event be sent at some point. Therefore, the
receiveOrderWholesaleA message does not contain any parts. To define this message without any parts, enter the following WSDL:

```xml
<message name="receiveOrderWholesaleA"/>
```

- The receiveOrderWholesaleB operation performs the same function for waitForDelivery_WholesaleB_Event.
- The notifyProcessStarter message is used by the notifyProcessStarter staff activities and contains information about the status of an order.

**Interruptible verses non-interruptible processes**
The following characteristics require our process to run as interruptible:

- Parallel paths that we wish to be executed concurrently
- Human interaction (staff activities)
- External events (receive event activities)
- Compensation

**Serial and parallel execution paths**
A serial execution path is used for placing the order to Wholesaler A and Wholesaler B. A parallel execution path is used for waiting for an external event to be sent notifying that a part has been received from Wholesaler A and Wholesaler B.

We choose to execute the place order activities in serial to reduce the chance of compensation being required on both place order activities. This serial approach means that if Wholesaler A cannot fulfil the order, the process is terminated before placing an order with Wholesaler B. A parallel approach would require that compensation be initiated for both Wholesaler A and B whenever either of the wholesalers is unable to fulfil an order.

However, there are other business scenarios that would be better served by placing these activities in parallel. Although we did not implement these activities in parallel, the process model is shown in Figure 12-5 on page 325.
Receive event activities
Receive event activities pause a process execution path until either of the following conditions are met:

- An event is received for the process instance, identified by either the process ID or a correlation ID.
- The receive event activity expires.

Using event activities
We also implement a recommended practice for using event activities in a process:

- The event processing is implemented in a loop activity.
- If the event is sent for the event activity, the loop is exited.
- If the event expires, a staff activity is invoked that creates a work item for a human to follow up the reason for the event expiration. The loop is then reiterated, causing the process path to pause at the event activity again.

The receive event activity loop is shown in Figure 12-6 on page 326.
Sending events
An event is sent by invoking the WSDL operation assigned to it. This operation is a one-way operation. The input message of the operation can contain data, thus providing another opportunity for external sources to provide input data into a process.

There are two ways to send an event:
- Call the relevant method of the process inbound binding
- Use the WebSphere Process Choreographer API sendEvent() method

Note: The Process Web Client does not provide a facility to send events to process instances.

For our scenario, we create a Web application to work with the events in our process. The Web application uses the WebSphere Process Choreographer API to perform the following tasks:
- List all process instances that are currently waiting for an event to be received.
- Send an event, identified by the event one-way operation and the process instance where the event resides.

Compensation
WebSphere Process Choreographer provides a compensation sphere that is used to compensate, or undo, service activities. Compensation occurs in a process instance when the process is running within the compensation sphere and processing navigates to a fault activity. In this case, the process is marked as compensating, and the control links are navigated in reverse. If a service activity has been assigned a compensation service, it will be compensated.
A service activity that has been configured for compensation will consist of the following two services:

- **Primary service**
  This service is executed during normal execution.

- **Compensation service**
  This service is executed when the process instance is in a compensating state. The compensation service should perform an opposite action to the primary service, thereby undoing the work performed in the primary service.

### Compensating non-service activities

WebSphere Process Choreographer can only compensate service activities. Other activities, such as EJB and Java activities, cannot be compensated. In our scenario, Wholesaler B exposes its place order operation as an EJB method. In order to compensate it, we need to expose the EJB as a service and then add this service as a service activity into the process.

WebSphere Studio Application Developer Integration Edition can create enterprise services from EJBs. This tool generates three WSDL files that describe the EJB as a WSIF service. This WSDL can be used as the basis of a service activity in a process.

### Designing the compensation service

It is up to the developer to create a suitable compensation service. To aid in compensating the primary service, the input and output message data for the primary service can be retrieved. There are three possibilities:

- **If the compensation service requires the primary service input data:**
  The input message structure for the compensation service should be same as the input message of the primary service.
If the compensation service requires the primary service output data:
The input message structure for the compensation service should be same as
the output message of the primary service.

If the compensation service requires both input and output data of the
primary service:
The input message structure of the compensation service should consist of
the parts that make up the input and output message of the primary service.

In our scenario, the cancel order compensating service needs the confirmation
code of the order to cancel. This information is provided in the output message of
place order (the primary service). Therefore, the input message of cancel order
has the same structure (the same part name and type) as the input message of
place order. See Example 12-1.

Example 12-1 Messages used in the primary and compensating services

```xml
<message name="placeOrderResponse">
  <part name="confirmCode" type="xsd:string"/>
</message>

<message name="cancelOrderRequest">
  <part name="confirmCode" type="xsd:string"/>
</message>
```

12.4.2 Development guidelines

This section describes how to build the process in WebSphere Process
Choreographer. Only the steps specific to this scenario are discussed here. It
also outlines the steps required to build a Web application capable of sending
events to this process.

Building the process
1. Create a process named EventCompensation in the service project
   ITSOEventCompensation. Place the activities in the process as shown in
   Figure 12-2 on page 322.

2. Create a process interface definition as described in the design guidelines.

3. In order to use the placeOrder and cancelOrder EJB methods in a
   compensation service, you need to create an enterprise service for the EJB:
   a. Select File → New → Service build from.
   b. Click EJB and then click Next.
c. Navigate to the Inventory bean in ITSOTargetAppB and select the EJB methods `cancelOrder` and `placeOrder`. These methods will be exposed in the enterprise service. Click **Next**.

d. The enterprise service will generate three WSDL files. Be sure that these files are created in your service project. Also, ensure that you generate faults. Complete the window as shown in Figure 12-7 and then click **Finish**.

e. Three WSDL files are generated. Open the interface file, `Inventory.wsdl`. For compensation to work correctly, we need to ensure that the parts in the input message of `cancelOrder` match the output message of `placeOrder`. By default they will not, so change the part name of `placeOrderResponse` from `result` to `confirmCode` as shown in Example 12-1 on page 328.

f. You will also need to reflect this change in the binding WSDL file, `InventoryEJBBinding.wsdl`.
4. Implement the two `placeOrder` blocks as shown in Figure 12-8. For Wholesaler B, use the service WSDL file `InventoryEJBService.wsdl`.

![Figure 12-8 Place order block](image)

5. To assign compensation to each of the `placeOrder` activities, open the activity properties, and select Compensation. Check Assign compensation service and specify the `cancelOrder` operation as the compensating operation. See Figure 12-9.

![Figure 12-9 Assigning compensation](image)

6. Implement the two `waitForDelivery` loops, as shown in Figure 12-10 on page 331.
7. The loop activities will continue to loop until the loop condition is set to false. Create two new process variables of type boolean, called `isDeliveryPendingWholesaleA` and `isDeliveryPendingWholesaleB`. These variables will store the loop condition. Initialize the variables to a value of true in the control links connecting `placeOrder_WholesaleB` with the loop activities.

8. In each loop activity, assign the relevant variable to the loop condition, as follows in Example 12-2.

   **Example 12-2 Loop condition for waitForDelivery_WholesaleA_Loop**
   ```java
   result = getIsDeliveryPendingWholesaleA().getValue();
   ```

9. Assign the `receiveOrderWholesaleA` and `receiveOrderWholesaleB` operations to their appropriate event activities as shown in Figure 12-11 on page 332.
10. Event activities can expire if the event is not sent within a given period of time. In the event activity properties window, select **Server** and enter a duration as shown in Figure 12-12 on page 333.
11. If the event expires, all outgoing control links will be evaluated. We need to add a Java condition to check the event expiration state. If the event expires, the state of the activity will be set to STATE_EXPIRED. Set a Java condition in the control link between the event and staff activity to check for expiration. Only navigate to this control link if the event is expired. See Example 12-3.

Example 12-3  Check for event expiration

```java
int eventState = activityInstance().getExecutionState();
if (eventState != ActivityInstanceData.STATE_EXPIRED) {
    result = false;
}
```

12. If the event operation is called, the activity state is set to STATE_FINISHED. The control link between the event and output node should only be navigated if the event is received. Also use this opportunity to set the loop condition to false. See Example 12-4 on page 334. (Modify accordingly for Wholesaler B.)
Example 12-4  Check for event completion

```java
int eventState = activityInstance().getExecutionState();
if (eventState == ActivityInstanceData.STATE_FINISHED) {
    BooleanMessage isDeliveryPendingWholesaleA =
        getIsDeliveryPendingWholesaleA();
    isDeliveryPendingWholesaleA.setValue(false);
    setIsDeliveryPendingWholesaleA(isDeliveryPendingWholesaleA);
} else {
    result = false;
}
```

13. Define the staff query for the staff activity. Set the potential owner to use the staff query verb Group Members with the group name parameter as:

```
cn=RetailManager,ou=retail,o=ITSOElec,c=US
```

For more details about staff activity configurations, refer to the previous chapter.

14. The staff member assigned to follow up the order will need to know the confirmation code of the order. Therefore assign the placeOrderResponse (for Wholesaler A and B) variable to the staff activity input data. See Figure 12-13.

![Figure 12-13  Staff activity input data](image)

15. Implement the notifyProcessStarter blocks as shown in Figure 12-14. These are similar to those used in the previous chapter and should send a notification to the process starter.

![Figure 12-14  Notify process starter block](image)
16. Because this scenario uses compensation, we run this process in the compensation sphere. To do this, we need to select the Run with Compensation Sphere option from the server tab of the process as shown in the Figure 12-15.

![Server Settings](image)

**Figure 12-15  Run the process in the compensation sphere**

### Building the event Web application

The Process Web Client does not provide a feature to send events to waitForDelivery_WholesaleA_Event and waitForDeliveryWholesaleB_Event. Therefore, we have created a Web application called EventUtilityWeb that lists and sends events to a process instance. The Web application consists of servlets to work with events and a JSP to display the results.

The Web application uses the EJB facade of the WebSphere Process Choreographer API.

### Listing process instances where an event is waiting

The BusinessProcess bean provides a query() method that can retrieve a list of process instances where a specific event is waiting. Events take their name from the operation assigned to them, not the name of the event activity.

The query() method takes as parameters a select clause, specifying which objects should be returned, and a where clause, specifying a selection criteria. Our Web application retrieves all process instance IDs for every event named receiveOrderWholesaleA where the user is authenticated to be a potential owner of the work item (as shown in Example 12-5).

#### Example 12-5  Retrieve a list of process ids where events are waiting

```java
String selectClause = "EVENT.PIID";
String whereClause = "EVENT.NAME = 'receiveOrderWholesaleA' AND WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER";
```
QueryResultset resultSet =
    businessProcessBean.query(selectClause, whereClause, null, null, null, null);

The result set is navigated to extract the process ID. Information about the
process instance can be obtained from the process ID, such as the name of the
process instance and a value in a given process variable such as
placeOrderWholesaleAResponse (as shown in Example 12-6).

Example 12-6  Retrieve process name and confirm code

    //get process name
    processInstanceID = (PIID) resultSet.getOID(1);
    String processInstanceData =
        businessProcessBean.getProcessInstance(processInstanceID);
    String processName = processInstanceData.getName();

    //get confirm code from placeOrderWholesaleAResponse
    ClientObjectWrapper placeOrderVariable =
        businessProcessBean.getVariable( 
            processInstanceID, "placeOrderWholesaleAResponse");
    WSIFDefaultMessage wsifMsg =
        (WSIFDefaultMessage)placeOrderVariable.getObject();
    String confirmCode = (String)wsifMsg.getObjectPart("confirmCode");

Sending events to a process instance

The BusinessProcess bean provides a sendEvent() method to send a specific
event to a specific process instance. When sending the event, you must pass an
input message to the event. The input messages used for the events in this
process contain no parts, so they can be easily created. The code to send an
event to the receiveOrderWholesaleA event for a given process instance is
shown in Example 12-7.

Example 12-7  Sending events

    WSIFDefaultMessage defMsg = new WSIFDefaultMessage();
    ClientObjectWrapper messageObject = new ClientObjectWrapper(defMsg);
    businessProcessBean.sendEvent( 
        processName, "receiveOrderWholesaleA", messageObject);

12.4.3  Runtime guidelines

As with any process, the process must be deployed to an enterprise application.
In this specific scenario, we are limited to using the JMS-inbound binding. The
process described in this chapter contains messages that are defined without
parts. (These messages are defined for the two-event activities). Only the
JMS-inbound binding supports these types of messages. The EJB- and
SOAP-inbound bindings are not compatible with these message types and will generate errors during deployment.

This process shares many runtime characteristics with the previous human interaction scenario. In this section, we describe the following two test cases that highlight the runtime characteristics specific to this scenario:

- Compensation
- Sending an event to a process and event expiration

**Testing compensation**

In the Process Web Client, create a new process instance. In the input message set partNo1 to ABCDE and partNo2 to FGHZA. (Do not forget to specify quantities as well.) When the process is run, the place order activity to Wholesaler A completes successfully and the place order activity to Wholesaler B returns a PartNotAvailableException. At this point, a fault terminal is navigated and both place order activities are compensated.

You can see this by clicking on the Created By Me link and refreshing the page until you see the place order block for Wholesaler B marked as Failed (as shown in Figure 12-16).

![Figure 12-16 An activity in failed state](image)

The process instance is compensated once the notification staff activity has completed. The work item generated for this staff activity contains a message stating that the order failed. Claim and complete this work item and then return to the Created By Me page to see the completed process instance with a state of Compensated (Figure 12-17 on page 338).
Figure 12-17  A process instance in compensated state

Testing events
Create a new process instance (making sure neither of the part numbers contain the letter Z). Use the Created By Me page to view when the process instances reach the event loops (as shown in Figure 12-18). The process instances pause until either an event is sent to each receive event activity or the events expire.

The clock is ticking; you have five minutes to send an event before the expiration occurs. We have created a Web application to send events to this process. Assuming you have installed this Web application into WebSphere Application Server Enterprise, you can access it using the following URL:

http://localhost:9080/EventUtilityWeb/ListEventsServlet

When prompted for a userid, use the same credentials you are logged in with in the Process Web Client (probably wasadmin). You should see the process instances and confirmation codes (returned from the place order activities), as shown in Figure 12-19 on page 339.
To send an event to a process instance (to notify that a delivery has been received), click on the specific process instance radio button and click **Order Received**. The Web application confirms that an event has been successfully sent and removes the process instance from the table (as shown in Figure 12-20).

Once the event has been sent, the event activity is complete. This can be seen in the Process Web Client (as shown in Figure 12-21 on page 340). Notice that waitForDelivery_WholesaleB_Loop now has a state of Finished.
Figure 12-21  Completion of a block with an event activity

Wait for a five minute period and then click **Refresh List** in the Web application. The remaining process instance is removed from the table. This occurs because the event activity has expired. A work item is now be generated to follow up the reason for the expiration. Once you complete and claim this work item, the event reappears in the Web application.

Send an event to the process instance and complete the notify process starter work item to complete the process instance.

### 12.5 WebSphere MQ Workflow guidelines

This section describes how to implement the Parallel Process Workflow variation pattern with events and compensation. It also describes how to create a process that meets the requirements set by the ITSO Electronics business process model. It uses the Runtime pattern and Product mappings described in “Parallel Process: Workflow variation Product mappings” on page 100 for WebSphere MQ Workflow.

You can download the completed process and run it in WebSphere MQ Workflow or view it in WebSphere Business Integration Workbench. See “Sample scenarios setup” on page 398.

#### 12.5.1 Design guidelines

This section discusses WebSphere MQ Workflow specific-design guidelines when creating a process that conforms to the Parallel Process Workflow variation pattern with events and compensation.
Process overview

Figure 12-22 shows an overview of the process, as modeled in WebSphere Business Integration Workbench.

The process has the following characteristics:

- Blocks with loop conditions
- Parallel execution paths
- Human interaction activities
- Automated activities

Staffing and organization models

This scenario builds on the organization model defined in the previous scenario. In addition to the RetailManager and RetailStaff roles, two new roles are needed for the Retail organization:

- ReceivingStaff, responsible for receiving deliveries of goods
- ReceivingManager, responsible for following up on delayed orders with the responsible Wholesaler

Process and task interface definition

This process uses different process input and output data structures from the previous scenarios. The process input data structure needs the name of two part numbers and two quantities to order. The process output returns both confirmation numbers from these orders. Additionally, the Wholesale data structure (used as a placeholder for storing data required by multiple tasks within the process) needs to store both confirmation codes and whether an item was delivered.
So as not to interfere with the previous scenarios, we have created new data structures called OrderInputEvents, OrderOutputEvents, and WholesaleEvents.

The block activities require their own data structure. We define this data structure as Delivery. It contains the part number and confirmation code of the item we are waiting to be delivered, along with a delivered flag that is set to Y when the item is received.

All data structures new to this scenario are shown in Figure 12-23.

Figure 12-23  New data structures added to the repository

Blocks and loops
It is necessary in this scenario for a portion of the process to loop until a certain condition is met. WebSphere MQ Workflow supports this concept by using blocks.

To create the looping functionality using blocks, we use the following guidelines:

- Create two processes, (1) containing the main process and (2) containing the tasks and data mappings that are required in the loop portion of the process.
- Define the second process as a block.
- Add a new Process activity to the first process, and set it to the name of the block.
In the Process activity, define an end expression. This expression can, for example, check the value of one of the data fields in the output container of the block. The block will re-execute unless the end expression evaluates to true.

We created two blocks, WaitForDeliveryA and WaitForDeliveryB. Each block consists of two tasks:

- Create a work item for the ReceivingStaff role to inform when an order has arrived.
- Create a work item for the ReceivingManager role to follow up with the Wholesaler if the order is late in arriving.

The block WaitForDeliveryA is shown in Figure 12-24. The block WaitForDeliveryB is identical, except that the task names are waitForDeliveryB and followUpOrderB.

![Figure 12-24  WaitForDeliveryA block overview](image)

**Control objects**

We use two control objects in this scenario to model compensation. as follows:

- The first object checks whether the confirmation code from placeOrderA is set to zero. If so, the place order operation failed, and the process moves directly to the notifyStatus activity.
- The second object checks whether the confirmation code from placeOrderB is set to zero. If so, the place order operation failed. The order placed with Wholesaler A should be cancelled, so cancelOrderA is invoked and the process then moves to the notifyStatus activity.

Additionally, we use a control object to each of the blocks to model an event:

- This object examines the delivered data field to check whether an item was received. If it is set to Y, then the order is deemed to have been delivered. Otherwise, a staff task is invoked to follow up on the order.
12.5.2 Development guidelines

This section describes how to build a Parallel Workflow variation pattern implementation for ITSO Electronics. This section focuses on the events and compensation aspects of the process.

Building the process
This section describes the how to build the process described in this chapter.

Role and employee definitions
In “Staffing and organization models” on page 341, we identified a number of roles. We need to define these roles in the repository, as follows:

1. Start the WebSphere Business Integration Workbench, with the ITSOElec.org organization open. Switch to Integration mode.
2. Click Repository → Organization Data → Roles.
3. Define roles for ReceivingStaff and ReceivingManager.

The next stage is to create employees and assign them to roles and organization units. In a real model, there would be multiple employees assigned to each role. For simplicity, we will assign just one employee to each role.

1. Click Repository → Organization Data → Employees.
2. Create a new employee called PREDDING and assign it to the Retail organization unit and the ReceivingStaff role. Click Add to add the employee to the repository.
3. Add a second employee called JBAXTER and assign it to the Retail organization unit and the ReceivingManager role.

Adding data structures and fields to the repository
As discussed in the design guidelines, we need to add new data structures and data fields to the repository. Use Repository → Process Data → Data Fields to add data fields and new structures, and Repository → Process Data → Data Structures to assign data fields to these structures.

The following data structures should be defined. Refer to the design guidelines to see the fields in each structure:

- OrderInputEvents
- OrderOutputEvents
- WholesaleEvents
- Delivery
Creating a new process and blocks
As well as creating a process for this scenario, two blocks are also required. Create the new process by selecting File \(\rightarrow\) New Process. Click Process \(\rightarrow\) Info to open the process properties. Perform the following:

- Click General and set the process name and MQ Workflow name to EventCompensation.
- Click Fields and set the Input structure to OrderInputEvents and the Output structure to OrderOutputEvents.
- Click Process Set and specify the option to keep finished processes forever.

Create a block called WaitForDeliveryA. Click File \(\rightarrow\) New Process. Click Process \(\rightarrow\) Info to open the process properties. Define the following:

- Click General and set the process name and MQ Workflow name to WaitForDeliveryA. Set the organization unit to Wholesaler A.
- Click Fields (as shown in Figure 12-25) and set the input and output structure to Delivery. Also, set the Export Type to Block.

![Figure 12-25  Defining a block](image)

Define a second block called WaitForDeliveryB, and assign it to Wholesaler B. This also uses Delivery for its input and output structure.
**Application definitions**
The wait for delivery and follow-up order tasks both need applications defined for them. We additionally need a new notify status application.

We define two applications for the WaitForDeliveryA block:
- SwaitForDeliveryA
- $followUpOrderA$

We then define similar applications for the WaitForDeliveryB block called:
- SwaitForDeliveryB
- $followUpOrderB$

To create the SwaitForDeliveryA application, perform the following:
1. Click **Repository → Organization Data → Applications**.
2. Create a new application with a name of **SwaitForDeliveryA**. Set the active platform to **Windows NT** and ensure that Run unattended is **not** selected.
3. Click the **Parameters** tab. Select **Delivery** as the Input Structure and Output Structure, and check **Program requires these**.
4. Click the **Windows NT** tab. Set the path to **fmcnshow.exe**.

Define the applications for $followUpOrderA$, SwaitForDeliveryB, and $followUpOrderB$ in exactly the same way. Define an application called **SnotifyStatus2** using the same procedure, except set the input and output data structure to **OrderOutputEvents**.

**Task object definitions**
Referring to the process overview in Figure 12-22 on page 341, add the following Web service tasks to the main process and configure them as you have done for the previous scenarios:
- placeOrderA
- placeOrderB
- cancelOrderA

Note that a new Web service task is used called cancelOrderA. This should be set to use the InventoryPort_A_CancelOrderA application.

Define a NOOP task. This NOOP task has the following properties:
- Belongs to the Retail organization unit
- Uses the FMCINTERNALNOOP application
Uses the WholesaleEvents data structure

- Is asynchronous
- Is automated
- Contains a default mapping that maps confirmCodeA to confirmCodeA, and confirmCodeB to confirmCodeB

Define two notifyStatus tasks, which are used by whoever started the process to view the output of the process. Name these tasks notifyStatus and notifyStatus_01, and set the following properties:

- Belongs to the Retail organization unit
- Uses the SnotifyStatus2 application
- Uses the orderOutputEvents data structure
- Is manual
- Has a staff assignment of Process Starter.
- Contains a default mapping which maps _STRUCT to _STRUCT.

Leave notifyStatus to use the default of waiting for all inputs before executing, and set notifyStatus2 to wait for only one input before executing.

Define two process tasks, which invoke the WaitForDeliveryA and WaitForDeliveryB blocks. First, add a block called WaitForDeliveryA as follows:

1. Add a Process task to the ADF diagram.
2. Set the Process name and MQ Workflow name to WaitForDeliveryA.
3. Click Expressions (as shown in Figure 12-26 on page 348) and set the automatic execution to One Input. Add an end expression by clicking End Exp and entering delivered="Y". The block will loop unless the delivered data field from the output container is set to Y.
4. Click **Data** and set the Input Structure and Output Structure to **Delivery**.

Define the block **WaitForDeliveryB** in the same way.

Each block contains two tasks, one to wait for delivery and one to follow up a delayed order. Add the following tasks to **WaitForDeliveryA**:

- The **waitForDeliveryA** task belongs to the Retail organization and uses the application **SwaitForDeliveryA**. Set this as a manual task that uses the Delivery data structure and creates a default mapping from _STRUCT to _STRUCT. Assign it to the role **ReceivingStaff**.

- The **followUpOrderA** task shares the same characteristics as **waitForDeliveryA**, except it uses application **SfollowUpOrderA** and is assigned to the **ReceivingManager** role.

Define similar tasks to **WaitForDeliveryB**, using the appropriate tasks names (**waitForDeliveryB** and **followUpOrderB**) and applications (**SwaitForDeliveryB** and **SfollowUpOrderB**).

**Decision object definitions**

Two decision objects are used in the main process. Add the first decision object as follows:
1. In the ADF toolbar select **Decision** and add a decision to the ADF diagram.

2. Create a connector between placeOrderA and the decision object. This connector is required so we can use the output of placeOrderA.

3. Name the decision object **order 1 placed?** and set it to a type of **Binary**.

4. Set the true expression to **confirmCode<>"0"** and the false expression to **OTHERWISE**. The completed decision object is shown in Figure 12-27.

![Figure 12-27  Decision object definition](image)

5. If the decision object evaluates to true, control should flow to placeOrderB. Otherwise control should flow to notifyStatus_01.

Create a second decision object after NOOP called **order 2 placed?**. Set the true condition to **confirmCodeB<>"0"** and the false condition to **OTHERWISE**. If the decision object evaluates to true, it should flow control to the phi before the blocks. If it evaluates to false, it should flow control to the phi before cancelOrderA.

A decision object is used in each of the blocks. The purpose of this decision object is to determine if the delivery is received. Create the decision object in the WaitForDeliveryA block as follows:

1. Create a connector between waitForDeliveryA and the decision object.

2. Name the decision object **Not Delivered?** and set it to a type of **Binary**.

3. Set the true expression to **delivered="N" or delivered=""** and set the false expression to **OTHERWISE** (as shown in Figure 12-28 on page 350).
Figure 12-28  Decision object definition for the WaitForDeliveryA block

4. If the decision object evaluates to true, then followUpOrderA should be executed.

Create a similar decision object for the WaitForDeliveryB block.

**Data mapping between tasks**

Create the following data mappings in the main process:

- From Source to placeOrderA, passing partNo1 to partNo and qty1 to qty
- From Source to placeOrderB, passing partNo2 to partNo and qty2 to qty
- From Source to waitForDeliveryA, passing partNo1 to partNo
- From Source to waitForDeliveryB, passing partNo2 to partNo
- From placeOrderA to NOOP, passing confirmCode to confirmCodeA
- From placeOrderB to NOOP, passing confirmCode to confirmCodeB
- From NOOP to WaitForDeliveryA, passing confirmCodeA to confirmCode
- From NOOP to WaitForDeliveryB, passing confirmCodeB to confirmCode
- From NOOP to notifyStatus, passing confirmCodeA to confirmCodeA and confirmCodeB to confirmCodeB
- From NOOP to cancelOrderA, passing confirmCodeA to confirmCode
- From notifyStatus to the Sink, passing _STRUCT to _STRUCT
- From notifyStatus_01 to the Sink, passing _STRUCT to _STRUCT

Create data mappings in the WaitForDeliveryA block as follows:

- From Source to waitForDeliveryA, passing _STRUCT to _STRUCT
- From waitForDeliveryA to followUpOrderA, passing _STRUCT to _STRUCT
Creating similar data mappings for the WaitForDeliveryB block.

**Exporting the process to the runtime database**

1. Once the process is built, verify it by selecting **Process → Process Validation → MQ Workflow Model**. Correct any errors reported here.

2. Export the process to a FDL file that can be used in WebSphere MQ Workflow V3.4 (deselecting **Export NOOP**).

3. Import the FDL file using following command to WebSphere MQ Workflow runtime:
   
   ```
   ```

12.5.3 Runtime guidelines

As with the previous scenario, this process uses human interaction. You could create customized JSP pages for the human interaction activities, but we choose to use the Web client-default HTML pages.

There are many different execution paths through the process. We will test three in this section:

- An order that is successfully placed at both Wholesalers, but the order placed with Wholesaler B is delayed in arriving
- A failed order placed with Wholesaler A
- A failed order placed with Wholesaler B

**Starting and monitoring process instances with the Web client**

Perform the following to test the first execution path through the EventCompensation process:

1. Start the Web client and log in as userid **JCAMPBELL** (password is **password**). This is the user we have defined as belonging to the RetailStaff role, which is responsible for placing orders.

2. Create and start a new instance of the EventCompensation process template. Set partNo1 to **ABCDE** and partNo2 to **FGHIJ**.

3. Switch to the ProcessInstances view. You should see a new process instance created from the EventCompensation process template. Open the process monitor for this process instance. You should see that placeOrderA and placeOrderB are both successfully navigated and completed and that
processing is now waiting at the WaitForDeliveryA and WaitForDeliveryB blocks.

4. You can use the process monitor to examine the tasks in each of these blocks. Click **WaitForDeliveryA** to open its properties. From here, click **Show Block Monitor** to view the block. You should see that the block is waiting at the waitForDeliveryA task (as shown in Figure 12-29).

![Monitor for Block 'EventCompensation.WaitForDeliveryA'](image)

*Figure 12-29 Block monitor for WaitForDeliveryA*

5. In a new browser, open another instance of the WebSphere MQ Workflow Web client, and log in as **PREDDING**. This user belongs to the ReceivingStaff role and is responsible for completing the wait for delivery work items.

6. Switch to the **WorkList** view, where you should see work items created for waitForDeliveryA and waitForDeliveryB (as shown in Figure 12-30).

![Work Items of WorkList (2)](image)

*Figure 12-30 New work items generated from the blocks*

7. Click **Check out Work item** for waitForDeliveryA. The part number and confirmation code for this order should be displayed. This work item must be completed by specifying whether the item ordered has been received. Set delivered to **Y** to indicate that the item has been received (as shown in Figure 12-31 on page 353). This is a good example of where a customized JSP could be used. The JSP would set the partNo and confirmCode fields to read only and allow the owner of this work item to only modify the delivered field, perhaps using a radio button.
Chapter 12. Creating processes with events and compensation

Complete Work Item "waitForDeliveryA"

<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>partNo</td>
<td>STRING</td>
<td>ABCDE</td>
</tr>
<tr>
<td>confirmCode</td>
<td>STRING</td>
<td>14372_ABCDE</td>
</tr>
<tr>
<td>delivered</td>
<td>STRING</td>
<td>Y</td>
</tr>
<tr>
<td>RC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 12-31 Notifying the process that an item has arrived

8. Check out the work item **waitForDeliveryB** and set delivered to **N**. This indicates that the item was not received within a reasonable amount of time.

9. Go back to the Web client where JCAMPBELL is logged in, and view the main process instance in the process monitor. You should see that the WaitForDeliveryA block has completed and the WaitForDeliveryB block is still executing.

10. Open a third Web browser. Then, start the Web client and log in as JBAXTER. This user is assigned to the role ReceivingManager and has the task of following up delayed orders. Go to the **WorkList** view, where you should see a work item for followUpOrderB (Figure 12-32).

![Work Items of WorkList (1)](image)

Figure 12-32 Work item generated to follow up an order with Wholesaler B

11. Complete this work item. No data needs to be input, the work item should simply be completed once the user has placed a call with Wholesaler B to follow up on the reason for the late delivery.

12. The WaitForDeliveryB block completes, but the end expression of delivered=**Y** is not met, so the block restarts at the waitForDeliveryB task. Switch to the Web client where you logged in as PREDDING. You should see a new work item called waitForDeliveryB created. Complete this work item by setting delivered to **Y**.

Chapter 12. Creating processes with events and compensation 353
13. Both orders have now been received, so both blocks have completed. (You can check this in the process monitor.) Switch to the Web client where you logged in as JCAMPBELL and view the WorkList. A work item called notifyStatus is generated for this user because he is the process starter. View this work item and complete it. See Figure 12-33.

![Complete Work Item "notifyStatus"](image)

14. The process instance is now complete. You can examine the process output container by opening the process instance properties and clicking Output Container.

**Using the Web client to test compensation**

If an attempt to place an order with a Wholesaler fails, the process should cancel (compensate) any orders already successfully placed, then notify the process starter.

To test the scenario where an order to Wholesaler A fails, perform the following:

1. Log in to the Web client as user JCAMPBELL and create a new process instance of EventCompensation. Set partNo1 to ABCDEZ and partNo2 to FGHIJ.

2. Open the process monitor for this process instance. You should see that placeOrderA completes and then notifyStatus is navigated. The rest of the process is not executed because placeOrderA returns a confirmation code of zero, indicating that the order has failed.

3. Switch to the WorkList view. A work item is created for JCAMPBELL (the process starter). The confirmation code for A and B are both blank, indicating the orders were not placed (as shown in Figure 12-34 on page 355). This is another good example of where a customized JSP could be used to add some text to the notifyStatus work item indicating that blank confirmation codes indicate that the orders are not placed.
To test the scenario where an order to Wholesaler B fails, perform the same procedure, except set partNo1 to ABCDE and partNo2 to FGHIZJ. This time the display in the process monitor indicates that placeOrderA and placeOrderB have executed. The order with Wholesaler A is placed successfully, and the order with Wholesaler B fails. The cancelOrderA task is executed to compensate the order with Wholesaler A. Then, the notifyStatus work item is generated, again showing blank confirmation codes.
Process manager interoperability

This chapter focuses on how to achieve interoperability between the WebSphere Process Choreographer and WebSphere MQ Workflow process managers. The chapter is split into the following parts:

- Business scenario
- Business process model
- General design guidelines
- WebSphere MQ Workflow invoking Process Choreographer
- Process Choreographer invoking WebSphere MQ Workflow
13.1 Business scenario

From a business point of view, this scenario of using a second process manager is identical to the scenario that introduced parallel processing. The order clerk places an order specifying the part number and the quantity. An immediate response is received indicating which wholesaler the order was placed with, along with a confirmation number and a delivery date.

Behind the scenes, the user is unaware that two process managers are being used. This scenario is implemented as a WebSphere MQ Workflow process that invokes a WebSphere Process Choreographer process as a single UPES activity. For illustrative purposes, this chapter also implements a WebSphere MQ Workflow process invoking the WebSphere Process Choreographer process.

This scenario is different than the one described in “An alternative solution” on page 273. The alternative solution has one process manager spawn another process manager as the process in the first process manager ends. The process is started with a fire and forget mentality, or in other words, an asynchronous process invocation. In this scenario, we are embedding a process run in a second process manager, with a process running in the first process manager. The form of sub-process is run synchronously.

It can be speculated that since WebSphere MQ Workflow is a mature product in production at many sites around the world, an existing WebSphere MQ Workflow customer may consider using WebSphere Process Choreographer to build a non-interruptible process within an existing WebSphere MQ Workflow process. Several UPES activities may be taken out of a current WebSphere MQ Workflow process template and replaced with a single activity that calls the WebSphere Process Choreographer non-interruptible process. This could be the first step to migrating towards a WebSphere Process Choreographer runtime.

13.2 Business process model

The swimlane diagram to depict this process is shown in Figure 13-1 on page 359.
This scenario executes the following business process:

- The order clerk places an order specifying the part number and quantity and a process is started in WebSphere MQ Workflow.
- The first activity in WebSphere MQ Workflow is a UPES activity that starts a non-interruptible process in WebSphere Process Choreographer.
- In a parallel fashion, requests are sent to Wholesaler A and Wholesaler B to provide their delivery dates. Once both dates are received, the WebSphere Process Choreographer process ends and control is passed back to WebSphere MQ Workflow.
- The best wholesaler is determined by which wholesaler can supply the part the most quickly.
- The order is placed with the best wholesaler.
- The clerk who initiated the process receives a confirmation that the order was placed. This confirmation includes the name of the wholesaler and the delivery date.

To show the reverse interoperability, we will call the complete WebSphere MQ Workflow process described in this section from a WebSphere Process Choreographer process.
13.3 General design guidelines

In this section, we discuss the implications of using two process managers to run a business process.

13.3.1 Design overview

This scenario is an implementation of the Process-focused Application Integration::Parallel Process pattern.

For information about the Parallel Process Application pattern, see:

3.4.8, “Parallel Process Application pattern” on page 61

For information about the Parallel Process Runtime pattern, see:

5.1.3, “Parallel Process Runtime pattern” on page 88

For information about the Product mappings of the Parallel Process Runtime pattern, see:

5.2.3, “Parallel Process Product mappings” on page 98

This scenario is an implementation of Stage One of the ITSO Electronics integration project. For information about the use case and actors used in Stage One, see:

6.2.1, “Stage One: Internal ordering on demand” on page 110

13.3.2 Design considerations

We assume for this scenario that it is desirable for ITSO Electronics to use two process managers. In reality, it would not make a lot of sense to have an WebSphere MQ Workflow process call a WebSphere Process Choreographer process that only has the two main activities of place order with Wholesaler A and place order with Wholesaler B in it.

If we did not use two process managers in this scenario, then this scenario would not be any different than the parallel process scenario described earlier in the redbook. By introducing this interoperability scenario, it demonstrates how an organization can use its existing process manager to call another process manager. We can assume that ITSO Electronics already has a WebSphere MQ Workflow system in place, possibly even running the parallel process scenario.

WebSphere MQ Workflow invoking Process Choreographer

When an organization decides they would like to introduce WebSphere Process Choreographer into their environment, changing an existing WebSphere MQ
Workflow process to include a WebSphere Process Choreographer process may be a viable starting point. This allows the organization to take advantage of the features that a WebSphere Process Choreographer non-interruptible process brings to the table.

In order to keep our scenario process simple, we arbitrarily decide to put the place order activities into a non-interruptible process. If this were a real example, the question of how much of the process should be included in the WebSphere Process Choreographer process needs to be analyzed. Careful thought needs to be given to this because if the process fails at any time, all successful transactions can be rolled back as part of the non-interruptible process.

If staff activities are involved in both process manager processes, then consideration needs to be given to having a common work list client that can receive work items from both WebSphere Process Choreographer and WebSphere MQ Workflow. It is desirable to have a common client so that the users of the system do not need to know that some work comes from process manager A while other work comes from process manager B.

However, if many staff activities are involved in the process, perhaps this part of the process is better served in as a WebSphere MQ Workflow process. People-centric-based workflow is one of the strengths of WebSphere MQ Workflow. In contrast, a flow that is heavily relying on Web services or J2EE integration may be better implemented in WebSphere Process Choreographer.

**Process Choreographer invoking WebSphere MQ Workflow**

We create a second process in WebSphere Process Choreographer. This invokes the WebSphere MQ Workflow process described in the business process model of this chapter (which, in turns, calls another WebSphere Process Choreographer process). This is to illustrate how a WebSphere Process Choreographer activity can invoke an WebSphere MQ Workflow process. It serves no business purpose.

Because of the benefits that a WebSphere Process Choreographer non-interruptible process provides, it is desirable to have the controlling process manager be WebSphere MQ Workflow in our scenario.

There may be opportunities to have an interruptible WebSphere Process Choreographer process call a WebSphere MQ Workflow process. One example of using a WebSphere MQ Workflow process may be defining a process that contains staff activities when a WebSphere MQ Workflow environment already exists. Therefore, the existing infrastructure already includes a client that users can use to process their work items. By embedding the staff activities in an WebSphere MQ Workflow process, an organization can leverage its existing investment.
13.4 WebSphere MQ Workflow invoking Process Choreographer

This section describes how to design an MQ Workflow process that invokes a Process Choreographer process. It follows the normal phases of design, development, and runtime guidelines.

You should read the design section to get a general understanding of how this interoperability will work. The development section requires hands-on knowledge of both WebSphere MQ Workflow Buildtime and WebSphere Studio Application Developer Integration Edition. If you do not have these development tools installed, you can skip the development section and go directly to the runtime section.

13.4.1 Design guidelines

This section discusses some of the design guidelines to consider when interoperating between these two process managers.

Interoperability overview

Figure 13-2 shows the processes that we discuss and develop in this section.

The WebSphere MQ Workflow process (WMQWF2) essentially delegates the task of “getBestWholeSaler” to a WebSphere Process Choreographer process (Called). When the WebSphere Process Choreographer process returns, the WebSphere MQ Workflow process can continue processing the other activity nodes. In our case, we have an program activity which simply displays the return value.
However the process could be easily expanded to include the placeOrder program activities as described in the previous chapters.

**Note:** The focus of this chapter is on interoperability. Therefore, not all of the activities described in the business process model are actually implemented in this scenario.

This chapter will focus on some design decisions and guidelines as they relate to the interoperability scenarios. Best practices for designing either WebSphere MQ Workflow or WebSphere Process Choreographer processes, as covered in previous chapters, still apply here.

In order to understand the best practices for designing inter-operable processes, we need to first understand how the two process managers can work together.

**WebSphere MQ Workflow interface for interoperability**

As you know already, WebSphere MQ Workflow is based on WebSphere MQ, and all of its server components communicate using WebSphere MQ queues. The most important component is the execution server, which is in charge of starting and navigating WebSphere MQ Workflow processes. It is the communication end point for any applications, including WebSphere Process Choreographer, which want to exploit WebSphere MQ Workflow functionality.

The only communication interface with the execution server are messages that are sent to a WebSphere MQ queue. The execution server can process two message formats:

- **SDDS:** An internal (proprietary) format of WebSphere MQ Workflow messages that is generated by the client API functions
- **XML:** A subset of the SDDS messages in XML format: The XML messages have to be sent as the "payload" of an MQ message. The MQ message may contain additional headers.

The execution server accepts XML messages (incoming) for the invocation of certain functions, and it also uses XML message (outgoing) to invoke user-defined process execution servers (UPES). Thus, XML messages are well suited for interoperability with non-WebSphere MQ Workflow components. As you will see later, XML message exchange is the foundation of our interoperability.

**Process Choreographer interface for interoperability**

WebSphere Process Choreographer consists of several important components, including a Navigator, Factory, and a host of plug-ins. The most important component, as you can guess, is the Navigator, which is responsible for the
execution of the state machines associated with the process and activity instances, just like the execution server in WebSphere MQ Workflow. It can be considered to be the core of the process engine.

The WebSphere Process Choreographer API gives clients access to the engine. The API exists in two flavors, the synchronous RPC-style version, accessible via local Java method calls and remote Enterprise Java Beans (EJB) invocations, and the asynchronous messaging-style API via JMS using Message-Driven Beans (MDB). For each process, a message-driven bean can be defined, which means the process can be started via a JMS message. This is of particular importance for our interoperability scenario.

On the other hand, the process engine uses invocation plug-ins to talk to the outside world. By default, WebSphere Process Choreographer supports two invocation plug-ins: the WSDL Invocation Plug-in and the Java Snippet Invocation Plug-in. The WSDL Invocation Plug-in is used in the interoperability scenario.

Using Web services for interoperability
With the introduction of the Web Services Toolkit for WebSphere MQ Workflow, it is possible to invoke Web service operations from WebSphere MQ Workflow activities. A WebSphere Process Choreographer process can be exposed as a Web service operation. Therefore, the WebSphere MQ Workflow could invoke the Web service simply by calling the relevant Web service operation, which in turn starts or initiates a WebSphere Process Choreographer process. We have not implemented this procedure in this chapter.

WebSphere MQ Workflow invoking Process Choreographer
This section is divided into two sections: buildtime and runtime.

*Buildtime*
Figure 13-3 on page 365 outlines the high-level tasks a developer needs to perform when interoperating between the two process managers.

**Note:** This scenario uses the WebSphere MQ Workflow Buildtime modeling tool to build a process rather than the WebSphere Business Integration Workbench. This is because the interoperability SupportPac used in this scenario only supports WebSphere MQ Workflow Buildtime.
It is worth noting that the whole procedure starts with the WebSphere MQ Workflow Buildtime and that the generated WSDL determines the interface of the WebSphere Process Choreographer process. What this means is you may not be able to directly invoke arbitrary existing WebSphere Process Choreographer processes from a WebSphere MQ Workflow process. You would have to either develop a new WebSphere Process Choreographer process based on the WSDL interface or adapt an existing process to use this interface.

**Runtime**

WebSphere Process Choreographer process invocation is modeled as a UPES activity. When WebSphere MQ Workflow navigates to this UPES activity, a XML message with a message type of `<ActivityImplInvoke>` is sent to the queue defined by the UPES. The UPES program, in this case, is a message-driven bean, which is a facade to our WebSphere Process Choreographer process. It picks up the message, does the necessary transformation, and invokes the process.

Upon completion of the WebSphere Process Choreographer process, the runtime translates the Java object message back to an XML message with a message type of `<ActivityImplInvokeResponse>`. This message is sent to EXEXMLINPUTQ, which is monitored by the WebSphere MQ Workflow execution server. It picks up the message, notices that the UPES activity has finished, and goes on to process other activities in the flow.

This series of events is shown in Figure 13-4 on page 366.
Design Considerations

Here are some questions and hints you need to consider when implementing WebSphere MQ Workflow to WebSphere Process Choreographer interoperability:

- Always start with the WebSphere MQ Workflow design. Apply the normal best practices. Invoking a WebSphere Process Choreographer process is just an extra tool to help the overall WebSphere MQ Workflow design.

- Why do you want to invoke a WebSphere Process Choreographer process from a WebSphere MQ Workflow process?

One general guideline is if part of your WebSphere MQ Workflow process interacts with Java artifacts (Java classes, EJBs) and Web services extensively, then these artifacts may be better modelled as WebSphere Process Choreographer activities.

- Clearly define what should be done in WebSphere MQ Workflow and what needs to happen in WebSphere Process Choreographer.

- Do you want a synchronous or asynchronous invocation to the WebSphere Process Choreographer process?

If your WebSphere MQ Workflow process relies on the response from the WebSphere Process Choreographer to continue processing, then model it as synchronous. Otherwise, always model it as asynchronous.
Chapter 13. Process manager interoperability

13.4.2 Development guidelines

This section describes how to implement the interoperability between WebSphere MQ Workflow and WebSphere Process Choreographer.

Installing the Interoperability SupportPac

There is a WebSphere MQ Workflow SupportPac provided to help perform interoperability between the two process managers. We explain how to install it.

Prerequisites

In order to achieve interoperability between WebSphere MQ Workflow and WebSphere Process Choreographer, you need to install the WebSphere MQ Workflow SupportPac WA86, which you can download from:

http://www.ibm.com/software/integration/support/supportpacs/individual/wa86.html

Before installing the SupportPac, make sure you meet the following prerequisites:

- WebSphere MQ Workflow V3.4.0 CSD1 or higher
- WebSphere Application Server Enterprise V5.0.2
- WebSphere Studio Application Developer Integration Edition V5.0.1

Installing the SupportPac

You must install the SupportPac on the system where WebSphere MQ Workflow is installed. Just run the program InterOp.exe and follow the Install Shield instructions.

After the setup program finishes, you will see the directory \SMP\InterOp under your WebSphere MQ Workflow installation directory. Change to directory \SMP\InterOp\lib and extract the file com.ibm.workflow.wmqwf_1.0.0.zip to the directory \eclipse\plugins. This will install a plug-in in WebSphere Studio to generate interoperability-related artifacts.

Note: The WebSphere Process Choreographer Business Process Container must be configured to use WebSphere MQ (not Embedded Messaging).
Tools supplied with the SupportPac

Export WSDL

This WebSphere MQ Workflow Buildtime extension adds a new menu item Export WSDL. This allows you to create an XML file in the WSDL format. The WSDL file contains all the necessary information to invoke a WebSphere MQ Workflow process from WebSphere Process Choreographer.

runfdl2wsdl.bat

To invoke a WebSphere Process Choreographer process from WebSphere MQ Workflow, you must export the FDL file for your WebSphere MQ Workflow process that contains the UPES definition. Do this by using the standard Export FDL file option in Buildtime. Then you can translate the FDL file into a WSDL file using the command line tool runfdl2wsdl.bat, which you can find in your WebSphere MQ Workflow installation directory under \SMP\InterOp\bin.

Code Generator Plug-in

The interoperability plug-in consists of a Java code generator for the WebSphere MQ Workflow format handler. The generator will be automatically called during the Generate deploy code step. The selection of the appropriate format handler generator is controlled by an attribute that is set in the WSDL file (the style attribute of the <format:TypeMapping> XML element is set to the value MQWF).

Developing Interoperable Processes

The development process involves three separate tools: WebSphere MQ Workflow Buildtime, a translator, and WebSphere Studio Application Developer Integration Edition. We will start with the Buildtime.

Step 1: Workflow Buildtime tasks

As illustrated in Figure 13-2 on page 362, our WebSphere MQ Workflow process consists of only two activities: (1) an UPES activity that actually invokes our WebSphere Process Choreographer process and (2) a utility node used to display the result coming back. We follow the steps below:

1. In WebSphere MQ Workflow Buildtime, under the Processes tab, create a new Category and a new Process: in our case, we name them ITSO and WMQWF2 respectively.

2. Click Implementation and create two Data Structures: OrderInput and OrderOutput.

3. Under the Implementation tab, create a Program FMCNShow, with the properties shown in Figure 13-5 on page 369.
4. Click **Data** and select **Program can handle any data structure**. This will enable the program fmcnshow to understand the data structure for our workflow.

5. Click **Network** and expand **DOMAIN, FMCGRP, and FMCSYS**, creating a new User-Defined Program Execution Server: **INTEROP** as shown in Figure 13-6 on page 370.
6. Open the properties dialog box for our workflow: WMQWF2, and change its Input data structure to OrderInput and Output data structure to OrderOutput.

7. Open WMQWF in the Process diagram and draw the flow as shown in Figure 13-7 on page 371.

**Important:** Be sure to select JMS-compliant XML as the message format.
8. Open the Properties dialog box for the CallWPC activity, we also need to change the input data structure to **OrderInput** and the output data structure to **OrderOutput**. Click **Start** and change Start to **Automatic**.

9. Open the Properties dialog box for the Show activity and change the input data structure to **OrderOutput** and output data structure to **OrderOutput**. Click **Start** and change Start to **Automatic**.

10. Now we can save and verify our process. It should return no error. We can now export the process and associated artifacts to WMQWF2.fdl.

### Step 2: Using the translation tool

We will need to create a WSDL file which describes the interface and invocation mechanism for the WebSphere Process Choreographer process. The interface of the process is dictated by the CallWPC activity of our process in WebSphere MQ Workflow. We can run the following in a command window:

```
runfdl2wsdl WMQWF2.fdl WMQWF2.wsdl FMCQM
```

This tool will generate a WSDL file that contains an XSD description of each input and output parameter for each process and each UPES activity defined in the input FDL file. The generated WSDL will also contain one service with one port for each process and each UPES activity. Example 13-1 on page 372 shows a snippet of binding section of the WSDL file.
Example 13-1 Binding in WMQWF2.wsdl

```xml
<wSDL:binding name='CallWPC_UPES_JmsBinding' type='tns:CallWPC_UPES_PortType'>
    <jms:binding type='TextMessage'/>
    <format:typeMapping encoding='XML' style='WMQWF'>
        <format:typeMap
            formatType='com.ibm.www.workflow.schema.CallWPC_OrderInputMessageType'
            typeName='mqwf:CallWPC_OrderInputMessageType' />
        <format:typeMap
            formatType='com.ibm.www.workflow.schema.OrderOutputMessageType'
            typeName='mqwf:OrderOutputMessageType' />
    </format:typeMapping>
</wSDL:binding>
<wSDL:operation name='CallWPC_UPES'>
    <wSDL:input name='Request_UPES'>
        <jms:input parts='Container'/>
    </wSDL:input>
    <wSDL:output name='Response_UPES'>
        <jms:body parts='Container'/>
    </wSDL:output>
</wSDL:operation>
</wSDL:binding>
```

There is one binding using encoding="XML" and style="WMQWF" for each port type. This guarantees that the WebSphere MQ Workflow format handler is called to translate WebSphere MQ Workflow messages into WebSphere Process Choreographer Java objects and vice versa during runtime.

Example 13-2 shows the service definition in the WSDL file.

Example 13-2 Services in WMQWF2.wsdl

```xml
<wSDL:service name='WMQWF2Service'>
    <wSDL:port name='WMQWF2_PROCESS_Port'
        binding='tns:WMQWF2_PROCESS_JmsBinding'>
        <jms:address destinationStyle='queue'
            jndiConnectionFactoryName='jms/FMCQM'
            jndiDestinationName='jms/EXEXMLINPUTQ'/>
    </wSDL:port>
    <wSDL:port name='CallWPC_UPES_Port' binding='tns:CallWPC_UPES_JmsBinding'>
        <jms:address destinationStyle='queue'
            jndiConnectionFactoryName='jms/FMCQM'
            jndiDestinationName='jms/INTEROPQ'/>
    </wSDL:port>
</wSDL:service>
```

The second port in the WSDL tells the WebSphere Process Choreographer process where to pick up the message. The first port tells the WebSphere Process Choreographer process where to send message back.
Step 3: WebSphere Studio integration tasks

1. In WebSphere Studio Application Developer Integration Edition, create a service project named CalledProject. Import WMQWF_Formatter.jar from the lib directory of the interoperability toolkit installation to this project. We then need to add this jar to the project’s Java build path.

2. Create a java package named com.ibm.itso.interop2 and import the WMQWF2.wsdl file we generated in the previous step.

3. Right click on the imported WSDL file, select New → Business Process, in the resulting dialog box, type in Called in the file name section, and click Next.

4. On the next dialog, make sure that Use existing WSDL for business process interface is selected. Then, choose the port type name callWPC_UPES_PortType and ensure that the correct Operation name is selected as in Figure 13-8. Finally, click Finish.

Figure 13-8 Creating new process in WebSphere Process Choreographer

Note: For detailed information about how to use the translation tool and what it does, please read the documentation that comes with the SupportPac.
5. An empty process with an input and output node will show in your workspace. Note that all of the necessary Java objects for the data/message types found in WMQWF2.wsdl have been generated automatically.

6. Now, we need to build a process as shown in Figure 13-9.

![Diagram of the Called process](image)

**Figure 13-9  Diagram of the Called process**

7. In this process, we have four Java snippets, one EJB activity, and one service activity. Here are several things that we need to pay special attention to when we build this process:

- To invoke getDeliveryDaysA and getDeliveryDaysB, you also need to import the ITSOTargetAppA and ITSOTargetAppB project EAR files.

- We need to add the Java code snippet shown in Example 13-3 to the control link from Input to checkRequest. The reason we need to do this in the control link is that the generated message-driven bean does not check the incoming message type. We want to make sure that we continue to invoke our WebSphere Process Choreographer process only if we get a message with a message type of ActivityImplInvoke. We then also change the control link from Input to Output to built-in otherwise.

**Example 13-3  Check incoming message code**

```java
System.out.println("Called---Make sure we got a valid request");

_MESSAGE_CONTEXT context = null;
result = false;

CallWPC_OrderInputMessageType container = getInput().getContainer();
if ( container != null ) {
    context = container.getMessageContext();
    if (context != null) {
        if (context.getMessageType().equals(MqwfFormatter.ACTIVITY_IMPL_INVoke) ) {
            System.out.println("Called---Found valid ActivityImplInvoke message context");
        }
    }
}
```
System.out.println("Called---Continue the processing ....");
result = true;
}
else {
    System.out.println("Called---Found a invalid message type " +
    context.getMessageType() );
    System.out.println("Called---Terminating process ...");
}
}

- Add code to checkRequest to display the partNo and qty and to set the
  message context using:
  
  _MESSAGE_CONTEXT mc = getInput().getContainer().getMessageContext();
- In the pre snippets, retrieve the partNo from the message context and
  pass it to the request message of getDeliveryDays.
- In the getBestWholesaler snippet, determine the best Wholesaler and set
  the output to this value, as shown in Figure 13-4.

Example 13-4  The getBestWholesaler snippet

OrderOutputMessageMessage oomm = new OrderOutputMessageMessage();
OrderOutputMessageType oomt = new OrderOutputMessageType();

//set MessageContainer and MessageContext
OrderOutput oo = getInput().getContainer().getMessageDefaultContainer();

if (getGetDeliveryDaysWholesaleAResponse().getDeliveryDays() >
    getGetDeliveryDaysWholesaleBResponse().getResult()){
    oo.setBestWholesaler("B");
} else {
    oo.setBestWholesaler("A");
}

oomt.setMessageContainer(oo);

_ORDER_API mc = getInput().getContainer().getMessageContext();
oomt.setMessageContext(mc);

//set
oomm.setContainer(oomt);

//set
setOutput(oomm);
8. Once the process has been built, we can generate deploy code for it. Select **Called.process** and right-click **Enterprise Service → Generate Deployed code**. In the dialog window (Figure 13-10), be sure to select **Use an Existing Port** and click **Next**.

![Figure 13-10 Deploy the process Called](image)

9. In the Inbound Service Files dialog, set the service file name by browsing to **WMQWF2.wsdl**. Make sure to select **CallWPC_UPES_Port** in the port name, and click **Finish**. See Figure 13-11 on page 377.
10. We need to import the WMQWF_Formatter.jar to the generated Enterprise project CalledProjectEAR as a utility JAR and right-click CalledProject. Then, select Properties and be sure to select WMQWF_Formatter.jar in the Java Jar Dependencies in the dialog window (as shown in Figure 13-12). (This ensures that the formatter library is available to the process at runtime.) Click OK.
11. In the J2EE Hierarchy view, expand **CalledProjectEJB** and double-click on **WMQWF2ServiceMDB** to open the deployment descriptor editor. Click **Reference** and change (1) the JNDI name of ResourceRef jms/FMCQM to jms/FMCQM and (2) the JNDI name of Resource reference jms/INTEROPQ to jms/INTEROPQ. Click **Bean** and change the listener port name to INTEROPQListenerPort. Save your changes.

**Note:** You may also need to switch to the source view of deployment descriptor to remove the empty `<messageselector></messageselector>` stanza. With this empty selector, WebSphere Application Server Enterprise V5.0.2 gives the following error:

```
[11/23/03 22:51:53:760 EST] 2737f0e4 MDBListenerIm W WMSG0019E: Unable to start MDB Listener WMQWF2ServiceMDB, JMSDestination jms/INTEROPQ: javax.jms.InvalidSelectorException: MQJMS0004: JMS Client has given JMS Provider a message selector with invalid syntax
```

12. Now we are ready to export the process. In the J2EE Hierarchy view, select **CalledProjectEAR**. Then, right-click **Export EAR File** and follow the dialog to save the project to the file **CalledProjectEAR.ear**.

### 13.4.3 Runtime guidelines

This section describes how to test the interoperability between WebSphere MQ Workflow and WebSphere Process Choreographer.

#### Preparing the systems

This interoperability scenario requires some runtime configuration in order to run.

**Software installation**

In our environment, we installed the following software:

- WebSphere MQ V5.3.1
- DB2 Version V8.1
- WebSphere MQ Workflow V3.4 + fixpack 3
- WebSphere Application Server Enterprise V5 + fixpack 2

We created a default **FMC** configuration for WebSphere MQ Workflow, which uses a queue manager called FMCQM.
We created a business process container in WebSphere Application Server Enterprise, and choose also to use FMCQM as the queue manager for WebSphere Process Choreographer.

You need to download the code shipped with this redbook. The code associated with the interoperability chapter is:

- **CalledProjectEAR.ear**
  Contains the WebSphere Process Choreographer process invoked by WebSphere MQ Workflow

- **ITSOInterOpA.ear**
  Contains the WebSphere Process Choreographer process invoking a WebSphere MQ Workflow process

- **WMQWF2.fdl**
  The original FDL file for the WebSphere MQ Workflow process

- **WMQWF2.wsd1**
  The WSDL file generated by the runfdl2wsdl tool used by the Called WebSphere Process Choreographer process

- **WMQWF1.wsd1**
  The WSDL file exported by Buildtime used by the Calling WebSphere Process Choreographer process

**Configure WebSphere MQ**

We need to create one queue INTEROPQ for the UPES activity. You can do this using the MQ Explorer to create a local queue in queue manager FMCQM.

**Configure WebSphere MQ Workflow**

Import the WebSphere MQ Workflow process WMQWF2.fdl into the runtime database by using the following command:

```bash
fmcibie -u admin -p password -t -o -i WMQWF2.fdl
```

**Configure WebSphere Application Server Enterprise**

To make the interoperability work, we need to create JMS-related resources in the application server. We can do this from the WebSphere Administrative Console. These resources include one JMS connection factory, two JMS queue destinations, and one listener port.

1. Define the connection factory jms/FMCQM: In the admin console, click **Resources → WebSphere MQ JMS provider → WebSphere MQ Queue Connection Factories → New**, and provide the information shown in Figure 13-13 on page 380.
2. Similarly, we need to define two queue destinations with the properties shown in Table 13-1.

<table>
<thead>
<tr>
<th>Name</th>
<th>JNDI Name</th>
<th>Base Queue Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEROPQ</td>
<td>jms/INTEROPQ</td>
<td>INTEROPQ</td>
</tr>
<tr>
<td>EXEXMLINPUTQ</td>
<td>jms/EXEXMLINPUTQ</td>
<td>EXEXMLINPUTQ</td>
</tr>
</tbody>
</table>

3. We need to define a ListenerPort named INTEROPQListenerPort for a MDB that accepts an invocation message from WebSphere MQ Workflow, and can invoke our WebSphere Process Choreographer process. In the
administration console, go to **Servers → Application Servers → server1 → Message Listener Service → Listener Ports** and click **New**. Enter the information shown in Figure 13-14.

<table>
<thead>
<tr>
<th>General Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Initial State</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Connection factory JNDI name</td>
</tr>
<tr>
<td>Destination JNDI name</td>
</tr>
<tr>
<td>Maximum sessions</td>
</tr>
<tr>
<td>Maximum retries</td>
</tr>
<tr>
<td>Maximum messages</td>
</tr>
</tbody>
</table>

*Figure 13-14  Defining a JMS ListenerPort*

4. Now, we are ready to install our process application that we exported in a previous step into CalledProjectEAR.ear. Use the administration console, following the Install New Application Wizard, and take all default values. When installation finishes, click **Save** and restart the application server.

**Note:** You will need to install the ITSOTargetAppA.ear and ITSOTargetAppB.ear applications that contain services used by our process if you have not already done so.

**Testing our interoperable processes**

1. Start the WebSphere MQ Workflow Web client. In the process template list, right-click on **WMQWF2** and select **Create and start instance**. You will be presented with a dialog box (as shown in Figure 13-15 on page 382). Enter input values (for example, DDDDDD for the partNo and 100 for the qty) and click **OK**. The process in WebSphere MQ Workflow is invoked.
2. The first activity in the WebSphere MQ Workflow process invokes our WebSphere Process Choreographer process. When we check the System.out of the application server, we see the entries shown in Figure 13-16. These messages indicate that our WebSphere Process Choreographer process is invoked and completed successfully, returning control back to WebSphere MQ Workflow.

```
[11/24/03 23:11:36:212 EST] 820ddab30 SystemOut     O Called---ready to call Wholesale A ...
[11/24/03 23:11:36:514 EST] 820ddab30 SystemOut     O Called---the best wholesaler is A
```

Figure 13-16  Check the process execution in System log

3. Once the WebSphere Process Choreographer process returns a result to WebSphere MQ Workflow, the process instance invokes the second activity, which is the Show activity. This activity displays the window shown in
Figure 13-17. You can click on Get Data, and you will notice the value A besides the input name bestWholesaler. This value is set and returned by our WebSphere Process Choreographer process, which again indicates that we have successfully invoked the process and received a result back.

You can now click Exit to finish the process.

13.5 Process Choreographer invoking WebSphere MQ Workflow

This section continues the discussion of interoperability and focuses on how to invoke a WebSphere MQ Workflow process from a WebSphere Process Choreographer process. We will invoke the WebSphere MQ Workflow process described in the previous section from WebSphere Process Choreographer.

This section will follow the normal phases of design, development, and runtime guidelines. Any overlapping information covered in the previous section will not be repeated here.

13.5.1 Design guidelines

This section discusses some of the design guidelines to consider when interoperating between these two process managers.
Interoperability overview

Figure 13-18 shows the processes that we will discuss and develop in this section.

Figure 13-18   Process overview

The WebSphere Process Choreographer process (Calling) starts with a Java snippet to set input data and proceeds to a service activity WMQWF_PROCESS that is actually invoking a WebSphere MQ Workflow process (WMQWF2). When the WebSphere MQ Workflow process returns, another Java snippet is invoked to examine the data returned. In this scenario, we choose to reuse and call the WebSphere MQ Workflow process we developed earlier in the chapter instead of creating a new one.

Using Web services for interoperability

In the Web Services Toolkit for WebSphere MQ Workflow, it is possible to expose WebSphere MQ Workflow processes as Flow Services. A Flow Service is an WebSphere MQ Workflow process exposed as a Web-service operation. WebSphere Process Choreographer can start a WebSphere MQ Workflow process simply by calling the relevant Web-service operation of a Flow Service.

We have not implemented this procedure in this chapter.

Process Choreographer invoking WebSphere MQ Workflow

This section is divided into two sections: buildtime and runtime.

Buildtime

You design and model your WebSphere MQ Workflow process as normal. Once the process is complete, you need to export the WSDL description of the process. In WebSphere MQ Workflow Buildtime, select and right-click your process model. Then, select Export WSDL. See Figure 13-19 on page 385.
Chapter 13. Process manager interoperability

Figure 13-19  Exporting WSDL from WebSphere MQ Workflow Buildtime

The export step generates a WSDL definition that contains the message formats and a JMS binding to call the WebSphere MQ Workflow engine. When you are defining the process in WebSphere Process Choreographer, you can import the generated WSDL into WebSphere Studio Application Developer Integration Edition to define a service activity.

Runtime
WebSphere Process Choreographer executes an activity named InterOpActivity that was defined to invoke the WebSphere MQ Workflow process. The MQWFFormatHandler in the MDB converts the outbound Java message object into an XML message in the WebSphere MQ Workflow ProcessTemplateExecute format. Subsequently, the XML message is sent to the input queue of the WebSphere MQ Workflow execution server.

The execution server executes the process and returns an XML message in the WebSphere MQ Workflow ProcessTemplateExecuteResponse format to the reply queue that is specified in the request message (the WebSphere Process Choreographer internal MDB queue).

WebSphere Process Choreographer receives the response and the MQWFFormatHandler transforms it into a Java message object that can be used to process the result of the request.

This architecture is shown in Figure 13-20 on page 386.
Design considerations

Invoking WebSphere MQ Workflow processes from WebSphere Process Choreographer is straightforward. The WebSphere Process Choreographer process engine treats the WebSphere MQ Workflow process as just another JMSBinding-based service. The format handler provided with the toolkit takes care of message transformation. This kind of interoperability is easy to achieve, and it opens up many opportunities to reuse existing WebSphere MQ Workflow processes when we are developing new business processes in WebSphere Process Choreographer.

13.5.2 Development guidelines

This section describes how to create the interoperability.

Step 1: WebSphere MQ Workflow Buildtime tasks
1. Model and build the WebSphere MQ Workflow process WMQWF2 (if you have not built this already).
2. Export the WSDL of this process to WMQWF1.wsdl.

Step 2: WebSphere Studio tasks
1. In WebSphere Studio Application Developer Integration Edition, create a service project CallingProject. Import WMQWF_Formatter.jar to the project. Then, we need to add this jar to the project’s Java build path.
2. Create a java package named com.ibm.itsointerop1 and import the WMQWF1.wsdl file that we generated in the previous step.

3. Right-click on the service project and select New → Business Process. In the resulting dialog box, type Calling in the file name section. Then, click Next.

4. In the next dialog, ensure that Define business process interface later is selected and click Finish.

5. Define the process input and output structure. The process input should accept a message that consists of a part of type OrderInput. The process output should accept a message that consists of a part of type OrderOutput.

6. An empty process with an input and output node is displayed in the process editor. We will build a process as shown in Figure 13-21.

![Figure 13-21 Calling process](image)

7. In this process, we have two Java snippets and one service activity. Drop two Java snippets onto the process design diagram. Then, drag the WSDL file WMQWF1.wsdl into the process editor and drop it between the two snippets. Here are a several that things we need to pay special attention to when building this process:

   - Enter the code for the setWMQWFInput snippet, as shown in Figure 13-5.

**Example 13-5 The setWMQWFInput snippet**

```java
System.out.println("Calling---Set input for WMQWF2 process");
OrderInputMsgMessage oi = getInput();

String partNo = oi.getPart1().getPartNo();
System.out.println("Calling---the partNo is "+ partNo);
Integer qty = oi.getPart1().getQty();
System.out.println("Calling---the qty is "+ qty);

OrderInputMessageMessage oimm = new OrderInputMessageMessage();
OrderInputMessageType oit = new OrderInputMessageType();

OrderInput o = new OrderInput();
o.setPartNo(partNo);
o.setQty(qty);

遽 MESSAGE_CONTEXT context = new _MESSAGE_CONTEXT();
```
//this is the template name for Workflow process
context.setProcTemplateName("WMQWF2");
context.setUserContext("ADMIN");

oit.setMessageContainer(o);
oit.setMessageContext(context);

oimm.setContainer(oit);

setOrderInputMessage(oimm);
System.out.println("Calling---about to call WMQWF2 process");

– Enter the code for the afterCalling snippet, as shown in Example 13-6.

**Example 13-6  The afterCalling snippet**

```java
System.out.println("Calling---WMQWF2 process returned");
OrderOutputMessageMessage oumm = getOrderOutputMessage();
OrderOutput ou = oumm.getContainer().getMessageContainer();

System.out.println("Calling---the best wholesaler from MQ workflow is " +
    ou.getBestWholesaler());

OrderOutputMsgMessage o = new OrderOutputMsgMessage();
o.setPart1(ou);
setOutput(o);
```

– When you generate the interface for the process, be sure to select
  **Asynchronous** as the process type.

– Mark this process as interruptible. This is necessary to commit the
  message sent to WebSphere MQ Workflow so that it can pick up the
  message and start processing.

8. Once the process is built, we can generate deploy code for it. Select
   **Calling.process** and right click **Enterprise Service** → **Generate Deployed code**. In the dialog window (shown in Figure 13-22 on page 389), ensure that
   **Create a new port and binding** is selected. For this sample, you must select
   the inbound binding type of **SOAP**. Accept the other defaults by clicking
   **Finish**.
9. Find the generated CallingPortTypeSOAPService.wsdl in the ITSOInteropWeb project, and generate a Web service client for it. Check WebSphere Studio documentation if you are unsure how to do this.

10. We need to import the WMQWF_Formatter.jar to the generated Enterprise project ITSOInteropA as a utility JAR. Then we must right-click CallingProject and select Properties. In the dialog window, be sure to select WMQWF_Formatter.jar in the Java Jar Dependencies. This ensures that the formatter library is available to the process at runtime. Click OK.

11. Now we are ready to export the process. In the J2EE Hierarchy view, select ITSOInteropA. Then, right-click Export EAR file and follow the dialog. Save it to the file ITSOInteropA.ear.

13.5.3 Runtime guidelines

There is no special configuration for this interoperability to work, assuming that you have done all the configuration for the previous interoperability scenario. You only need to install the application ITSOInteropA.ear by using the WebSphere Administrative Console.
Testing our interoperable processes
1. Start the WebSphere MQ Workflow Web client.
2. Remember that we generated a test Web services client for our WebSphere Process Choreographer Calling process. We can access the client via a browser (Figure 13-23) using the following link:

http://localhost:9080/ITSOInteOpWeb/sample/CallingPortTypeSOAP/TestClient.jsp

3. Enter the value DDDDDD for partNo and 100 for qty and press Invoke. Our Calling process will be called. Check the SystemOut.log in the application server log directory (as shown in Figure 13-24 on page 391). If you read the log carefully, you will notice when our Calling process starts, it invokes the WebSphere MQ Workflow process WMQWF2, which in turn invokes the WebSphere Process Choreographer process Called. At the end, you can see the WebSphere Process Choreographer process Called finishes and returns to the WebSphere MQ Workflow process. This, in turn, returns control to the Calling process.
4. In the WebSphere MQ Workflow Client, we see a pop-up window. See Figure 13-25 on page 392. Click *Get Data*, and you will notice the value A beside the input name bestWholeSaler in the left pane. This value is set and returned by our WebSphere Process Choreographer process, which again indicates that the WMQWF2 process has successfully invoked the WebSphere Process Choreographer process Called and received a result back.
5. You can then change the value of bestWholeSaler to A in the right-hand pane. Then, click **Set Output Data** and click **Exit**. This completes our WebSphere MQ Workflow process and returns the control to the WebSphere Process Choreographer process Calling.

6. If you check the SystemOut.log in the application server log directory again, you notice two additional entries starting with Calling (as shown in Figure 13-26). This indicates that (1) the WebSphere MQ Workflow process has returned to the WebSphere Process Choreographer Calling process and (2) the WebSphere Process Choreographer process has inspected the returned data and finished.

| 11/25/03 1:42:01:397 EST | 3787ac23 SystemOut | Called---Merging result from A & B |
| 11/25/03 1:42:01:399 EST | 3787ac23 SystemOut | Called---The best wholesaler is A |
| 11/25/03 1:45:03:350 EST | 3787ac23 SystemOut | Called---WebSphereMQ workflow is A |

**Figure 13-26** SystemOut log when the Calling process finishes

If you get this far, congratulations! You have successfully invoked a WebSphere MQ Workflow process from a WebSphere Process Choreographer process.
Appendixes
Scenarios lab environment

In this appendix, we describe the lab setup we used when deploying our WebSphere Process Choreographer and WebSphere MQ Workflow scenarios.

We then explain how to set up each scenario, describing how to deploy each process so it can be tested and showing how to import the source of each process into the relevant development tool for inspection.
Lab setup

This section shows the topologies used by the redbook team to run the scenarios for WebSphere Process Choreographer and WebSphere MQ Workflow.

WebSphere Process Choreographer topology

The WebSphere Process Choreographer topology is illustrated in Figure A-1.

The topology shown in Figure A-1 consists of the following servers:

- **WebSphere server**
  - Hosts WebSphere Application Server Enterprise, IBM HTTP Server, and WebSphere MQ Server
Appendix A. Scenarios lab environment

- **Repository server**
  Hosts the IBM Directory Server (the LDAP server) for WebSphere Process Choreographer and the WebSphere Process Choreographer database

- **WholesaleA and WholesaleB servers**
  Hosts the Wholesaler A Web service and the Wholesaler B EJB

**WebSphere MQ Workflow topology**

The WebSphere MQ Workflow topology is illustrated in Figure A-2.

![WebSphere MQ Workflow topology](image)

**Figure A-2  WebSphere MQ Workflow topology**

The topology shown in Figure A-2 consists of the following servers:

- **Workflow server**
  Hosts the WebSphere MQ Server and WebSphere MQ Workflow server

- **WAS server**
  Hosts WebSphere MQ Server and WebSphere Application Server, which is used to run the WA07 SupportPac and the WebSphere MQ Workflow Web client

- **Repository server**
  Hosts the WebSphere MQ Workflow database (An LDAP server and a database for WebSphere Process Choreographer are also installed on this server, but not used in the WebSphere MQ Workflow configuration.)
Sample scenarios setup

This section describes how to configure the sample scenario processes developed for WebSphere Process Choreographer and WebSphere MQ Workflow. You need to obtain the additional material provided with this redbook. See Appendix B, “Additional material” on page 403.

WebSphere Process Choreographer

This section describes how to use the sample scenarios provided with this redbook. You can install the scenarios into WebSphere Application Server Enterprise and test them, or you can view the scenario processes in WebSphere Studio Application Developer Integration Edition.

Running the sample processes

To run the sample processes, you need to have WebSphere Application Server Enterprise and the business process container installed. You also need to install the Wholesaler A and B enterprise applications, located at:

\WholesalerA\ITSOTargetAppA.ear
\WholesalerB\ITSOTargetAppB.ear

You may wish to install the Wholesaler A application on a remote application server, although this is not required. If you do this, be sure to add an entry to your hosts file pointing wholesalerA to the IP address of the application server where the Wholesaler A application is installed.

Once these enterprise applications are installed, you are ready to install the relevant scenario.

Serial process scenario

Install the serial process EAR file into WebSphere Application Server Enterprise, This EAR file can be found at the following location:

\ProcChor\Chpt09-Serial\ITSOSerialProcessEAR.ear
Serial process business rules bean scenario
Install the serial process business-rules bean EAR file into WebSphere Application Server Enterprise. This EAR file can be found at the following location:

\ProcChor\Chpt09-SerialBRB\ITSOSerialProcess_BRBear.ear

Additionally install the business-rule bean. This bean file can be found at the following location:

\ProcChor\Chpt09-SerialBRB\BRBeansDB2.ear

During the installation of this enterprise application, set all CMP settings to jdbc/BRBeansDataSource.

Parallel process scenario
Install the parallel process EAR file into WebSphere Application Server Enterprise. This EAR file can be found at the following location:

\ProcChor\Chpt10-Parallel\ITSOParallelProcessEAR.ear

Human interaction scenario
Install the human interaction EAR file into WebSphere Application Server Enterprise. This EAR file can be found at the following location:

\ProcChor\Chpt11-HumanInteraction\ITSOHumanInteractionEAR.ear

Additionally install the following LDIF file into your LDAP directory:

\ProcChor\Chpt11-HumanInteraction\ITSOElec.ldif

Events and compensation scenario
Install the events and compensation EAR file into WebSphere Application Server Enterprise. This EAR file can be found at the following location:

\ProcChor\Chpt12-EventsComp\ITSOEventCompensationEAR.ear

Additionally, install the Web application to list and send events. This Web application can be found at the following location:

\ProcChor\Chpt12-EventsComp\EventUtilityEAR.ear

If you have not already done so, install the following LDIF file into your LDAP directory (it is the same file as used in the human interaction scenario):

\ProcChor\Chpt12-EventsComp\ITSOElec.ldif
Viewing the sample processes
You can view the sample processes in the WebSphere Studio Application Developer Integration Edition V5 process editor. This allows you to see how each process was constructed.

In a new workspace, import the Wholesaler A and B enterprise applications. These EAR files can be found at the following locations:

\WholesalerA\ITSOTargetAppA.ear
\WholesalerB\ITSOTargetAppB.ear

These enterprise applications make use of the Web services support in WebSphere Application Server V5.0.2. You need to upgrade your workspace to allow the compilation of these enterprise applications. Copy webservices.jar from the lib directory of a WebSphere Application Server V5.0.2 installation to the lib directory of the test environment of WebSphere Studio. By default the lib directory of the test environment is located as follows:

C:\Program Files\IBM\WebSphere Studio\runtimes\base_v5\lib

Next, open the properties of the EJB project (for example ITSOTargetAppAEJB) and, in the Java Build Path, add the variable WAS_50_PLUGINDIR and extend it to lib/webservices.jar. Create a second pointer to WAS_50_PLUGINDIR and extend it to lib/qname.jar

Additionally, update the Web project (for example ITSOTargetAppAEJB_HTTPRouter) to add webservices.jar to the Java build path.

Finally import the relevant scenario projects into the workspace. Use the import option to import an existing project into the workspace. The projects are located in the relevant \ProcChor\ sub-directory. For example, to import the events and compensation process you would need to import the following projects:

\ProcChor\Chpt12-EventsComp\ITSOEventCompensation
\ProcChor\Chpt12-EventsComp\EventUtilityEAR
\ProcChor\Chpt12-EventsComp\EventUtilityWeb

WebSphere MQ Workflow
This section describes how to use the sample scenarios provided with this redbook. You can install the scenarios into the WebSphere MQ Workflow runtime database and test them, or you can view the scenario processes in WebSphere Business Integration Workbench.

Running the sample processes
To run the sample processes, you need two software configurations installed:
Appendix A. Scenarios lab environment

WebSphere MQ Workflow V3.4 (including WebSphere MQ and DB2)
WebSphere Application Server V5.0.2 or above

The Wholesaler A and B Web services run in WebSphere Application Server. They are J2EE enterprise applications, and are found at the following location:

\WholesalerA\ITSOTargetAppA.ear
\WholesalerB\ITSOTargetAppB.ear

The Web services use hostnames wholesalera and wholesalerb to identify the IP address of the WebSphere Application Server instance where they are installed. The hostnames will be evaluated on the server running the Web Services Toolkit. Set your Windows host file on this server to map wholesalera and wholesalerb to the IP address of the WebSphere Application Server where the enterprise applications are installed (most probably localhost).

You additionally need to add each process you wish to run to the runtime database. The FDL for each process is located in the MQWorkflow directory. The serial process scenario, for example, is stored in file SerialProcess.FDL.

We used the following command to add the serial process to the runtime database:

```
fmcibie -y FMC -u ADMIN -p password -o -t -i SerialProcess.FDL
```

**Viewing the sample processes**

Each process, and the ITSO Electronics organization, can be viewed in WebSphere Business Integration Workbench. Start WebSphere Business Integration Workbench and open the following organization:

\MQWorkflow\ITSOElec\ITSOElec.org

Once this organization is open, you can open each process within the organization, such as SerialProcess and ParallelProcess.

**Interoperability sample**

You can install the interoperability sample between WebSphere Process Choreographer and WebSphere MQ Workflow by installing the files in the Interop directory. The code associated with the interoperability chapter is:

- **CalledProjectEAR.ear**
  Containing the WebSphere Process Choreographer process invoked by WebSphere MQ Workflow
- **ITSOInterOpA.ear**
  Containing the WebSphere Process Choreographer process invoking a WebSphere MQ Workflow process

- **WMQWF2.fdl**
  The original FDL file for the WebSphere MQ Workflow process

- **WMQWF2.wsdl**
  The WSDL file generated by the runfdl2wsdl tool used by the Called WebSphere Process Choreographer process

- **WMQWF1.wsdl**
  The WSDL file exported by Buildtime used by the Calling WebSphere Process Choreographer process

You can also import the WebSphere Process Choreographer processes into WebSphere Studio Application Developer Integration Edition by creating a new workspace and importing the following projects:

- CalledProject
- CalledProjectEAR
- CalledProjectEJB
- CallingProject
- ITSOInterOpA
- ITSOInterOpAEJB
- ITSOInterOpAWeb
Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246306

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the Additional materials and open the directory that corresponds with the redbook form number, SG246306.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sg246306.zip</td>
<td>Zipped scenario files</td>
</tr>
</tbody>
</table>
System requirements for downloading the Web material

The following system configuration is recommended:

**Hard disk space:** 11MB minimum  
**Operating System:** Windows 2000 or XP  
**Processor:** Pentium® 4, 1GHz or faster  
**Memory:** 512MB minimum, 1GB preferred

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.

For further instructions on how to use the sample processes included in the additional material, refer to Appendix A, “Scenarios lab environment” on page 395.
## Abbreviations and acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADF</td>
<td>Activity Decision Flow diagram</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>B2B</td>
<td>Business to business</td>
</tr>
<tr>
<td>BPEL4WS</td>
<td>Business Process Execution Language for Web Services</td>
</tr>
<tr>
<td>CICS</td>
<td>Customer Information Control System</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>CVS</td>
<td>Concurrent Versions System</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name Server</td>
</tr>
<tr>
<td>EAI</td>
<td>Enterprise Application Integration</td>
</tr>
<tr>
<td>EAR</td>
<td>Enterprise Archive</td>
</tr>
<tr>
<td>EIS</td>
<td>Enterprise Information System</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBean</td>
</tr>
<tr>
<td>ERP</td>
<td>Enterprise Resource Planning</td>
</tr>
<tr>
<td>FDL</td>
<td>Flow Definition Language</td>
</tr>
<tr>
<td>FDML</td>
<td>Flow Definition Markup Language</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>HyperText Transfer Protocol Secure</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines Corporation</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IIOP</td>
<td>Internet Inter-ORB Protocol</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>ITSO</td>
<td>International Technical Support Organization</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Platform, Enterprise Edition</td>
</tr>
<tr>
<td>JAR</td>
<td>Java archive</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java database connectivity</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>JNDI</td>
<td>Java Naming and Directory Interface</td>
</tr>
<tr>
<td>JSP</td>
<td>JavaServer Page</td>
</tr>
<tr>
<td>JSR</td>
<td>Java Specification Request</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>LOV</td>
<td>Line of visibility</td>
</tr>
<tr>
<td>LTPA</td>
<td>Lightweight Third Party Authentication</td>
</tr>
<tr>
<td>MDB</td>
<td>Message Driven Bean</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>SOA</td>
<td>Service oriented architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol / Internet Protocol</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description Discovery and Integration</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>WAR</td>
<td>Web archive</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>WIP</td>
<td>Work in progress</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>WSIF</td>
<td>Web Services Invocation Framework</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>Extensible Stylesheet Language Transformations</td>
</tr>
</tbody>
</table>
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

**IBM Redbooks**

For information about ordering these publications, see “How to get IBM Redbooks” on page 409. Note that some of the documents referenced here may be available in softcopy only.

- *Patterns: Direct Connections for Intra- and Inter-enterprise*, SG24-6933
- *WebSphere V5.0.2 Application Developer V5.1 Web Services Handbook*, SG24-6891
- *WebSphere Application Server Enterprise V5 and Programming Model Extensions*, SG24-6932
- *WebSphere Version 5 Web Services Handbook*, SG24-6891
- *EJB 2.0 Development with WebSphere Studio Application Developer*, SG24-6819
- *MQSeries Programming Patterns*, SG24-6506
- *Continuous Business Process Management with HOLOSOFX BPM Suite and WebSphere MQ Workflow*, SG24-6590

**Other publications**

These publications are also relevant as further information sources:

Online resources

These Web sites and URLs are also relevant as further information sources:

- IBM Patterns for e-business
- IBM WebSphere Application Server Enterprise
  http://www.ibm.com/software/webservers/appserv/enterprise/
- IBM WebSphere Process Choreographer at developerWorks
  http://www.ibm.com/developerworks/websphere/zones/was/wpc.html
- IBM WebSphere Studio Application Developer Integration Edition
  http://www.ibm.com/software/awdtools/studiointegration/about/
- IBM WebSphere MQ Workflow
  http://www.ibm.com/software/integration/wmqwf/
- IBM WebSphere MQ Workflow Buildtime
- IBM WebSphere Business Integration Workbench
- IBM Rational Rose XDE Modeler
  http://www.ibm.com/software/awdtools/developer/modeler/features/
- BPEL4WS specification
- World Wide Web Consortium (W3C)
  http://www.w3.org/
- Web Services Invocation Framework
  http://ws.apache.org/wsif/
- SOA and Web services
- Security in a Web Services World: A Proposed Architecture and Roadmap
- Web Services Security (WS-Security)
- Workflow Management Coalition
  http://www.wfmc.org/standards/conformance.htm
How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services
Index

Numerics
80/20 situation 3

A
Actor 110
Adapter 23
  Coupling adapter connector 23
  Flow adapter 23
  Information adapter 23
  Representation adapter 23
Adapter connector 22, 71
Aggregator pattern 67
Apache AXIS 197
Application Integration pattern 33
  Business and IT drivers 35
    Process-focused 41
  Data-focused 34, 66
  Process-focused 34
  Quality of Service
    Process-focused 43
Application patterns 4, 11
  Broker 28, 49
  Direct Connection 28, 44
  Exposed Broker 30
  Exposed Direct Connection 29
  Exposed Serial Process 30
  Parallel Process 29, 61
  Process-focused 40
    Serial Process 28, 55, 58
Application portfolio 38
Application server 70
Application server/services 70
Asynchronous 18
Asynchronous interaction 24
Audit trail 78, 188
  audit_t_log 188
Availability 26

B
BAAN 36
  Background integration 37
  Batch integration 37

  Best practices 4, 16, 241
    Database 241
    WebSphere MQ configuration 242
    WebSphere MQ Workflow specific 243
    WebSphere Process Choreographer-specific 243
  Black box 39
BPM See Business Process Management
Broker application pattern 28, 49
  Business and IT drivers 50
  Router variation 52
    Business and IT drivers 53
Broker Rules tier 51
Buildtime 80
Business and IT drivers
  Application Integration pattern 35
    Process-focused 41
  Broker application pattern 50
  Call Connection variation 49
  Direct Connection application pattern 46
  Message Connection variation 48
  Parallel Process application pattern 61
  Parallel Workflow variation 64
  Router variation 53
    Serial Process application pattern 56
    Serial Workflow variation 59
Business patterns 4, 6
Business Process Management 37
Business Process Modelling 133
Business Rules Beans 159
  Java implementor rules 177
  Rule Browser 177
  RuleGreaterThan 177
  Session EJB 177
Business Rules Engine 166
  Rule Client 166
  Rule Implementors 166
Business scenario 105
  Intra-enterprise integration 109
BusinessProcess bean 335
  query() 335
  sendEvent() 336
C
Caching Proxy Server 74
Call Connection variation 45, 49
Business and IT drivers 49
Product mapping 94–96, 98–99, 101
ClearCase 168
Collaboration 19
Compensation 316, 319
Compensating non-service activities 327
compensating transaction 319
Compensation pairs 76, 322
Compensation service 326
Fault activity 326
Logical transaction 319
Primary service 327
Sphere 316
Complex interaction 21
Composite patterns 4, 9, 35
Connection 46
Connection rules 45
Connection Rules tier 46
Connector 21, 70
Adapter connector 22
Asynchronous Interaction 24
Coupling adapter connector 23
Modelled 21
Path connector 22
Primitive 21
Synchronicity 24
Synchronous interaction 24
Connector subtypes 22
CORBA 127
Correlation id 325
Coupling
Loose 39
Tight 39
Coupling adapter connector 23
Custom design 35
Customized JSPs 306
fmcohcli.jar 308
fmcojapi.jar 309
Generate JSPs for activities 310
Install 306
installedApps directory 309
JSP for MQWF Web Client 306
Page compiled 311
Page Designer 309
Process instance instantiation JSP 307
WA83 306
CVS 168

D
Data warehouse 242
Data-focused Application Integration 34, 66
DB2 241
Corrective service packages 242
Database backup plan 242
Database statistics 242
DB2DIAG log 242
Fail-over node 241
Logs and database tables 241
Deployment Manager 74
Direct Connection application pattern 28, 44
Business and IT drivers 46
Call Connection variation 45, 49
Message Connection variation 45, 48
Direct Connection runtime pattern
Product mapping 92
Domain QoS providers 71
Dynamic Web services 123

E
EJB inbound binding 180
Enterprise Resource Planning 34, 36
ERP See Enterprise Resource Planning
Events and compensation
Swimlane diagram 316
Expiry 320
Exposed 30
Exposed Broker application pattern 30
Exposed Direct Connection application pattern 29
Exposed Serial Process application pattern 30
External events 316, 321
Listing process instances 335
Sending events 326
Sending events to a process instance 336
STATE_EXPIRED 333
STATE_FINISHED 333
Timer 316
Triggered 321
Wait activity 316

F
Federation 27
Fire and forget message 274
Flow adapter 23
Flow Definition Language 81, 120
Flow languages 120
  Business Process Execution Language for Web Services 121
  Flow Definition Language 120
  Flow Definition Markup Language 120
  Web Services Flow Language 120
FMCINTERNALNOOP 194
fmcnshow.exe 296
Foreground integration 37

G
Geographic proximity 37
Get delivery date
  Use case 114
Guidelines 4, 16

H
Human interaction
  Asynchronous 271
  Delegation 271
  Expiry 273
  Fire and forget message 274
  Interoperability 276
  Late-binding 271
  Notification 273
  People-based exception handling 272
  Status updates 272
  Swimlane diagram 268
  Synchronous 271
  Work items 268
  Work list 268

I
Inbound binding 179
  EJB binding 180
  JMS binding 180, 257
  SOAP binding 180
Information adapter 23
Integration
  Background 37
  Batch 37
  Foreground 37
  Real-time 37
  Scope 37
Integration domain
  Quality of Service capabilities 26
Integration patterns 4, 7
  Classification 24
Interaction 19–20
  Asynchronous 24
  Classification 24
  Complex 21
  Parallel 24
  Serial 25
  Synchronous 24
Interoperability 357
  Flow Service 384
  Staff activities 361
  Swimlane diagram 358
  Web services 361
  Web Services Toolkit 364
Interoperability SupportPac 367
Installing 367
InterOp.exe 367
Prerequisites 367
Tools supplied 368
  Code Generator Plugin 368
  Export WSDL 368
  runfdl2wsdl.bat 368
  WMQWF_Formatter.jar 373
Interruptible process 164
Intra-enterprise integration 109
Invocation plugins 364
  Java Snippet Invocation Plug-in 364
  WSDL Invocation Plug-in 364
ITSO Electronics 105
  Intra-enterprise integration 109
  Non-functional requirements 108
  Quality of Service 108

J
J2EE 1.3 128
J2EE Connector Architecture 139
Java Message Service 128
  Advantages 130
  Disadvantages 130
  WebSphere MQ support 130
Java Snippets 165
  JMS inbound binding 180
  JMSReplyTo 258

K
Keep Finished Process 232
L
LAN See Local Area Network
Last Participant Support 145
LDAP search queries 280
LDAP user repository 277
Lightweight Directory Access Protocol (LDAP) 280
Line of Visibility 80
Local Area Network 72
Long-running process 152
Loose coupling 39
Loosely-linked processes 250
LTPA 143

M
Manage Process pattern 67
Message Connection variation 45, 48
  Business and IT drivers 48
  Product mapping 93
Message formats 363
  SDDS 363
  XML 363
Messaging 129
Modelled connector 21
MQSeries See WebSphere MQ

N
Network Dispatcher 74
Non-functional requirements 108
Non-interruptible process 164

O
OASIS 126
onMessage() method 258
Operation latency 37
Organizational changes 284

P
Parallel interaction 24
Parallel process 246
  Compensation 249
  Complexity 249
  Cycle time reduction 247
  Human interaction 250
  Non-interruptible process 249
  Normalization 248
  Performance 248
  Swimlane diagram 246
Parallel Process application pattern 29, 61, 63
  Business and IT drivers 61
  Workflow variation
    Business and IT drivers 64
Parallel Process Product mappings 98
  WebSphere MQ Workflow 99
  WebSphere Process Choreographer 98
Parallel Process Rules tier 61
Parallel Process Runtime pattern 88
  App Server / Service nodes 90
  Connector nodes 90
  Directory and Security services node 90
  Process Manager node 89
  Repository node 90
Parallel Process runtime pattern 88
Parallel Process Workflow variation Product mappings 100
  WebSphere MQ Workflow 101
  WebSphere Process Choreographer 100
Parallel Process Workflow variation Runtime pattern 90
  App Server / Service nodes 92
  Connector nodes 92
  Directory and Security services node 92
  Repository node 92
  Staff Worklist Adapter node 92
  Workflow Manager node 91
Parallel Workflow Rules tier 64
Parallel Workflow runtime pattern 90
Path connector 22, 71
Patterns for e-business
  Application patterns 4, 11
  Best practices 4, 16
  Business patterns 4, 6
  Composite patterns 4, 9
  Guidelines 4, 16
  Integration patterns 4, 7
  Product mappings 4, 15
  Runtime patterns 4, 12
  Web site 5
People-based exception handling 157, 272
PeopleSoft 36
Performance 27
Phi object 198
Potential owner 283
Primitive connector 21
Process 152
Process manager 85
  Asynchronous invocation 153
Audit trail 161
Capacity analysis 160
Compensation 156
Decision logic 158
Imminent response 156
People-based exception handling 157
Performance 160
Process invocation methods 154
Routing rules 158
Rule engines 157
Sub-processes 155
Synchronous invocation 153
Timeouts 156
Process re-engineering 37
Process Web Client 180, 183
Asynchronous interface 259
Created By Me page 260
Long-running process instance 259
My To Dos page 259
Process templates 184
Process-centric 77
Process-focused Application Integration 34
Process-focused Application pattern 40
Product mapping
Call Connection variation 94–96, 98–99, 101
Direct Connection 92
Message Connection variation 93
Product mappings 4, 15

Q
QoS See Quality of Service
Quality of Service
Application Integration pattern
Process-focused 43
Availability 26, 43
Capabilities 26
Domain QoS providers 71
Federation 27, 43
ITSO Electronics 108
Operability 43–44
Performance 27, 43–44
Security 27, 43
Standards compliance 28, 43–44
Transactionality 28, 43–44
Queue managers 76

R
Rapid Deployment Wizard 306
Rational Rose XDE Modeler 79
Real-time integration 37
Receive event activities 325
Representation adapter 23
RMI/IIOP 127
Router Rules tier 54
Router variation 52
Rule engines 157
Rules directory 71
Runtime pattern 4, 12, 84
Parallel Process 88
Parallel Workflow 90
Serial Process 84
Serial Workflow 86

S
SAP 36
Security 27, 160
Activity-related security 160
Process-related security 160
Roles 160
Web services 125
Self-Service business pattern 35, 107
Serial interaction 25
Serial process 150
Blocks 170
Externalization of business rules 163
Static discovery 170
Swimlane diagram 150
Serial Process application pattern 28, 55, 58
Business and IT drivers 56
Workflow variation 58
Business and IT drivers 59
Serial Process Product mappings 93
WebSphere MQ Workflow 95
WebSphere Process Choreographer 94
Serial Process Rules tier 56
Serial Process Runtime pattern
App Server/Service nodes 86
Connector nodes 86
Directory and Security services node 86
Process Manager node 85
Repository node 86
Serial Process runtime pattern 84
Serial Process Workflow variation Product mappings 96
WebSphere MQ Workflow 97
WebSphere Process Choreographer 96
Serial Process Workflow variation Runtime pattern 86
   App Server/Service nodes 88
   Connector nodes 88
   Directory and Security services node 88
   Repository node 88
   Staff Worklist Adapter node 88
   Workflow Manager node 87
Serial Workflow Rules tier 59
Serial Workflow runtime pattern 86
Service-oriented architecture 125
Short-running process 152
Simple Object Access Protocol 122
SOA See Service-oriented architecture
SOAP inbound binding 180
SOAP See Simple Object Access Protocol
Source Application tier 46, 51, 54, 56, 59, 61, 64
Staff activity 285
Staff resolution 280
   Custom attributes 282
   Distinguished name 282
   Early binding 282
   Late binding 282
   Organization structures 284
   Potential owner 283
   Staff verb sets 282
   Staff verbs 281
   Verb set mapping file 282
Staff Worklist Adapter 97
Standards compliance 28
Static Web services 123
Synchronicity 24
Synchronous 18
Synchronous interaction 24

T
Target Application tier 46, 51, 54, 56, 59, 62, 65
Tight coupling 39
Transactionality 28

U
Universal Description, Discovery, and Integration 122
Update inventory
   Use case 111
Use case
   Get delivery date 114
   Update inventory 111

V
VerbSet.xml 287

W
WA07
   See Web Services Toolkit
WA83
   See Customized JSPs
WA86 See Interoperability SuportPac
WAN See Wide Area Network
Web Service Description Language 122
Web services 122
   Advantages 127
   Disadvantages 127
   Dynamic 123
   Security 125
   SOAP fault messages 186
   Static 123
   WS-Policy 126
   WS-Privacy 126
   WS-Security 126
   WS-Trust 126
Web Services Description Language 163
Web Services Flow Language 120
Web Services for J2EE 168
Web Services Invocation Framework 124
Web Services Toolkit 95
   Apache AXIS 197
   Business exceptions 198
   Configuring 203
   Container-wide shared library 201
   Default queue manager 204
   Diagnostic Trace Service 240
   EXEXMLINPUTQ 204
   Faults 198
   Flow Service 197
   fmcojagt.jar 201
   fmcoutil.jar 202
   FMCWFLOW 205
   FMCWUPES 205
   Import Endpoints 206
   In Error state 198
   Installation Verification 203
   Installing 200
   JMS pipe 204
Logs and traces 240
Mapping rules 206
Message flow 238
MQ pipe 204
mqfwstk.ear 200
Prerequisite classes 202
Show FDL 208
SOAP monitor 239
SystemOut.log file 202
Update Configuration 205
Update Mapping Rule 208
Web service endpoints 208
Web service messages 208
Web service operations 197
Web Service UPES Administration 208
Web Services UPES Administration 206
WebSphere MQ cluster 205
XML security 203
WebSphere Application Server 73
Base V5.0 73
Caching Proxy Server 74
Deployment Manager 74
dumpnamespace tool 173
Enterprise V5.0 74
Last Participant Support 145
Network Deployment V5.0 74
Network Dispatcher 74
Web Services for J2EE 168
WebSphere Application Server Enterprise 72
Business Rules Beans 159
SOAP monitor 189
WebSphere Application Server Enterprise Process
Choreographer 75
WebSphere Business Integration Adapters 139
WebSphere Business Integration Workbench 80, 199
_STRUCTURE 221
Activity decision flow diagram 209
ADF toolbar 215
Applications 212
Automated Execution Wait 263
Control connector option 224
Data Structures 212, 214
Decision icon 222
Editing modes 209
Employees 214
Exporting the process 226
Expressions tab 347
Import Processes 211
Integration Mode 209
Line of Visibility 80
New process 215–216
Organization units 213
Persistent process instances 232
Phi icon 223
Repository 208
Task icon 215
WebSphere MQ 76, 128
Circular logs 243
Corrective service packages 242
JMS support 130
Logs 242
MQCONN 227
MOGET 230
MQOO_OUTPUT 227
MQOO_SET_IDENTITY_CONTEXT 227
MQOPEN 227
MQPMO_SET_IDENTITY_CONTEXT 229
MQPUT 228
Persistence queues 242
Queue managers 76
WebSphere MQ API Exerciser 136
WebSphere MQ Workflow 72
Absent Flagging 140
Activities 193
Activity expiration 139
Audit trail 78
Authentication 144
Authorization 144
Blocks 342
Buildtime 80
Business Process Modelling 133
Client 227
Configuration Utility 233
Control objects 199
End expression 343
Execution server 363
External context feature 250
Flow Definition Language 81, 120
fmcibie 226
FMCINTERNALNOOP 194
Global container 196
Input container 192
LDAP Bridge tool 78
Looping 342
Modeling an event 343
NOOP tasks 193
Notification 139
XSLT Transformer  165

Z
zSeries  135
Patterns: Serial and Parallel Processes for Process Choreography and Workflow
Patterns: Serial and Parallel Processes for Process Choreography and Workflow

Use the Patterns for e-business to integrate business processes

WebSphere Application Server
Enterprise Process Choreographer

WebSphere MQ Workflow

The Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying Web applications. This IBM Redbook focuses on business process application integration using the Process-focused Application Integration::Serial and Parallel Process Application patterns for intra-enterprise.

Part 1 guides you through the process of selecting an Application and Runtime pattern. Next, the platform-specific Product mappings are identified based upon the selected Runtime pattern. The Runtime and Product mapping patterns in this book focus on the Serial and Parallel Process patterns.

Part 2 presents guidelines on applying the Patterns approach to a sample business scenario and on selecting application integration technologies. It also describes the capabilities of WebSphere Process Choreographer and WebSphere MQ Workflow.

Part 3 provides detailed design, development, and runtime guidelines for five scenarios, each implemented using WebSphere Process Choreographer and WebSphere MQ Workflow. These implementations focus on automated Web service activities and human interaction activities.

For more information: ibm.com/redbooks