WebSphere® software

IBM

Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture

- Integrate ESBs in WebSphere V6 and Message Broker V5
- Patterns for integrating ESBs
- Learn by example with practical scenarios

Martin Keen
Jonathan Bond
Jerry Denman
Stuart Foster
Stepan Husek
Ben Thompson
Helen Wylie

Redbooks

ibm.com/redbooks

International Technical Support Organization

Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture

November 2005

SG24-6773-00

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

First Edition (November 2005)

This edition applies to WebSphere Application Server V6, Rational Application Developer V6, and WebSphere Business Integration Message Broker V5.

Contents

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®	Domino®	Rational®
AS/400®	@server®	Redbooks (logo) ™
CICS®	Everyplace®	Redbooks™
Cloudscape™	ibm.com®	SupportPac™
DB2 Connect™	IBM®	Tivoli®
DB2 Universal Database™	IMS™	WebSphere®
DB2®	iSeries™	xSeries®
developerWorks®	Lotus®	z/OS®

The following terms are trademarks of other companies:

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.

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

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

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

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

Preface

The Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying e-business applications. This IBM® Redbook focuses on how you can integrate Enterprise Service Bus (ESB) implementations in a service-oriented architecture (SOA). We discuss patterns for integrating ESBs and provide step-by-step instructions for integrating ESBs implemented in WebSphere® Business Integration Message Broker V5 and WebSphere Application Server V6. However, the ESB integration patterns and concepts apply to ESBs implemented with any product.

Part 1 introduces SOA and ESB concepts, and discusses the ESB capabilities of WebSphere Business Integration Message Broker V5 and WebSphere Application Server V6. It describes guidelines for determining when integration of ESBs is necessary, and describes patterns for integrating ESBs.

Part 2 describes the business scenario used throughout this book and explains the key technologies relevant to SOA and ESB.

Part 3 guides you through the process of integrating ESBs. Two scenarios are described: integration of homogeneous ESBs and integration of heterogeneous ESBs. The homogeneous ESB scenario describes the integration of two ESBs implemented in WebSphere Application Server V6. The heterogeneous ESB scenario describes the integration between an ESB implemented in WebSphere Application Server V6 and an ESB implemented in WebSphere Business Integration Message Broker V5.

The IBM Enterprise Service Bus strategy:

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

- WebSphere Enterprise Service Bus V6

  Delivers an ESB with Web services connectivity and data transformation.

- WebSphere Message Broker V6

  Delivers an advanced ESB with universal connectivity and data transformation.

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

For more information about the IBM ESB strategy see:

http://www.ibm.com/software/info1/websphere/index.jsp?tab=landings/esb

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Figure 1 The redbook team (left to right): Helen, Stepan, Ben, Jonathan, Stuart, Jerry, Martin

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

Jonathan Bond is a Software Engineer working on the Service Integration Bus and Web Services Gateway in IBM Hursley, UK. He has three years of experience in the Web services and SOA field. He holds a Bachelor of Science degree from the University of Bath.

Jerry M Denman is an Executive IT Architect with IBM Global Services in Orlando, Florida. He has 20 years of consulting experience. His area of technical expertise includes EAI, SOA, Service Oriented Integration, Web services architecture, and enterprise architecture. His primary industries are travel and transportation, consumer package goods, and retail companies. He is a member

of the IBM SOA and Web services Center of Excellence and a frequent conference speaker.

Stuart Foster is a Certified Software IT Specialist. He has worked for IBM for 20 years, focusing on Java™ and WebSphere since 1997, and he currently works in the WebSphere Technical Sales team in the UK. He holds a BSc degree in Management Studies from Aston University, Birmingham, but this was gained a long time ago. Stuart was a co-author of the IBM Redbook *Accessing the AS/400® with Java* published in 1997.

Stepan Husek is an IT Specialist at IBM Global Services, Prague, Czech Republic. He has more than four years of experience in integration technologies starting with business-to-business integration based on XML and messaging. He is a specialist in WebSphere Business Integration products and has experience with service-oriented architecture, service oriented integration, and business processes. Stepan holds an MSc degree in Computer Systems, Distributed Systems and Networking from Czech Technical University, Prague.

Ben Thompson is an Advisory IT Specialist in IBM Software Group EMEA Laboratory Services in Hursley, UK. He has worked with distributed transactional middleware for five years and has extensive experience designing and implementing solutions using the WebSphere product portfolio with IBM customers worldwide. He frequently deals with ESB and Web Service architectures and is also a recognized expert in the WebSphere Business Integration Message Broker. Ben holds a bachelors degree in Natural Science (Physics) from the University of Cambridge.

Helen Wylie is a certified Consultant IT Architect in Hursley Architectural Services in the UK. She has many years of experience in various aspects of the IT industry with the past decade being focused primarily in integration architectures. She holds a Methematics degree from the Open University and a post graduate diploma in Computer Science from Cambridge University. Her recent areas of expertise include service-oriented architecture, integration patterns, integration projects using a range of IBM products, and most recently the combination of patterns and models to support automated generation of services and deployment artifacts for deployment to an ESB.

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

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

Rick Robinson
Architecture Services, IBM EMEA WebSphere Lab Services, IBM UK

Dr. Keith Jones
Senior Software Architect & Specialist, IBM Software Group, US

Bob Dill
Distinguished Engineer, IBM Global Services, US

Emily Plachy
Distinguished Engineer, IBM Global Services, US

Tapas Som
Distinguished Engineer, IBM Global Services, US

Jake Thorwart
ITSO Intern, Penn State University

Become a published author

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

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

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

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

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

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

  `ibm.com/redbooks`

- Send your comments in an email to:

  `redbook@us.ibm.com`

- Mail your comments to:

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

Part 1

Patterns for e-business and SOA

This part contains the following chapters:

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, developing an outline of the desired solution, and considering any special requirements that need to be factored into that solution. The architect then takes this input and designs the solution, which can include one or more computer applications that address the business problems by supplying the necessary business functions.

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

The IBM Patterns for e-business help facilitate this reuse of assets. Their purpose is to capture and publish e-business artifacts that have been used, tested, and proven to be successful. The information captured by them is assumed to fit the majority, or 80/20, situation. The IBM Patterns for e-business are further augmented with guidelines and related links 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 reuse of components and solution elements from proven successful experiences. The Patterns approach is based on a set of layered assets that can be exploited by any existing development methodology. These layered assets are structured in a way that each level of detail builds on the last and include:

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

Figure 1-1 on page 5 shows these assets and their relationships to each other.

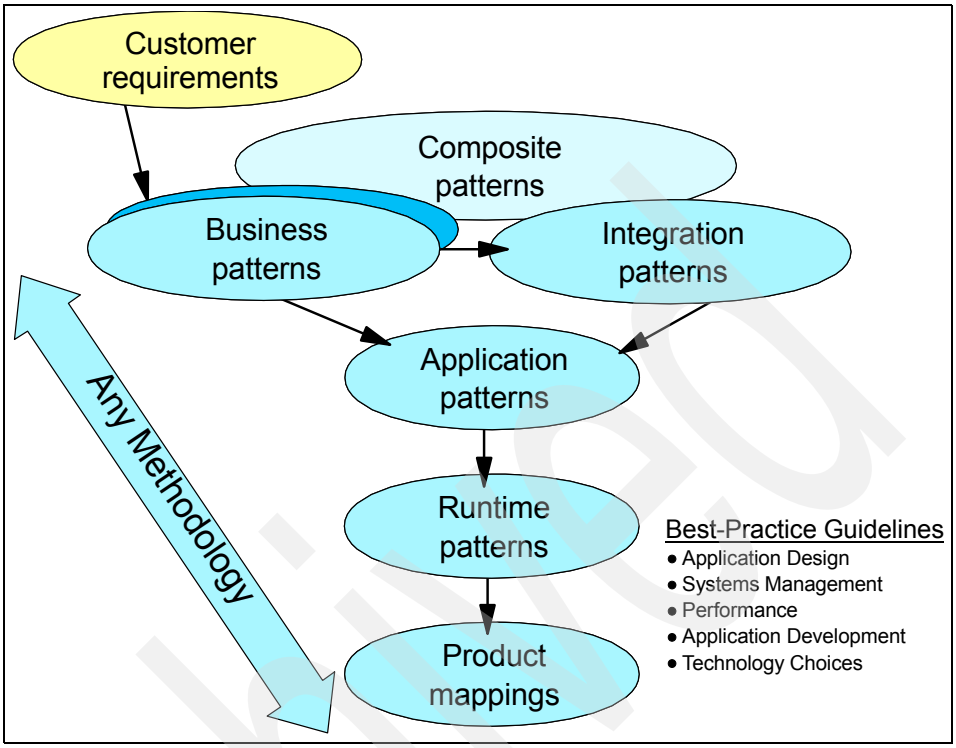

Figure 1-1 The Patterns for e-business layered asset model

Patterns for e-business Web site

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

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

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

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

1.2 How to use the Patterns for e-business

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

Composite patterns appear in the hierarchy shown in Figure 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. This section discusses how to use the layered structure of Patterns for e-business assets.

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

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

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

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

Business patterns

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

There are four primary Business patterns, that are explained in Table 1-1.

Table 1-1 The four primary Business patterns

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

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

Integration patterns

Integration patterns enable you to tie together multiple Business patterns to solve a business problem. Table 1-2 describes the Integration patterns.

Table 1-2 Integration patterns

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

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

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

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

Custom design

Figure 1-2 illustrates the use of a Custom design to address a business problem.

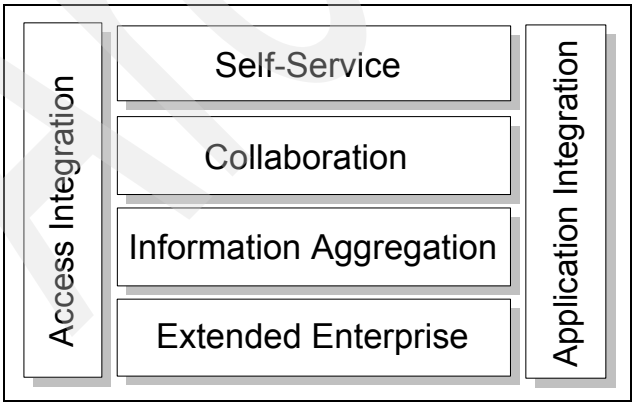

Figure 1-2 Patterns representing a Custom design

If you do not use any of the Business or Integration patterns in a Custom design, you can show the unused patterns as lighter blocks than those patterns that you do use. For example, Figure 1-3 shows a Custom design that does not have a Collaboration or an Extended Enterprise business pattern for a business problem.

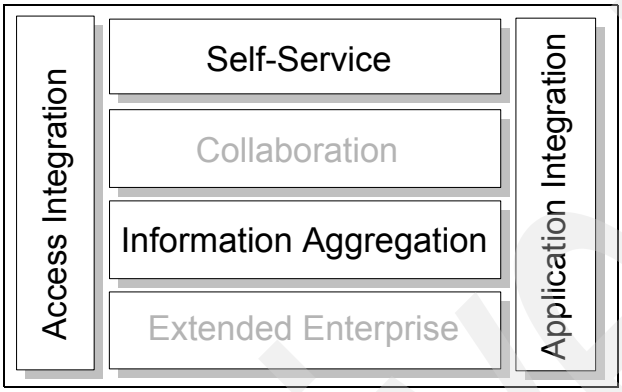

Figure 1-3 Custom design showing unused patterns

If a Custom design recurs many times across domains that have similar business problems, then it can also be a Composite pattern. For example, the Custom design in Figure 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. Table 1-3 shows the identified Composite patterns.

Table 1-3 Composite patterns

Composite patterns	Description	Examples
Electronic Commerce	User-to-online-buying	• www.macys.com • www.amazon.com
Portal	Typically designed to aggregate multiple information sources and applications to provide uniform, seamless, and personalized access for its users.	• Enterprise intranet portal providing self-service functions such as payroll, benefits, and travel expenses. • Collaboration providers who provide services such as e-mail or instant messaging.
Account Access	Provides customers with around-the-clock account access to their account information.	• Online brokerage trading applications. • Telephone company account manager functions. • Bank, credit card, and insurance company online applications.
Trading Exchange	Enables buyers and sellers to trade goods and services on a public site.	• Buyer's side: interaction between buyer's procurement system and commerce functions of e-Marketplace. • Seller's side: interaction between the procurement functions of the e-Marketplace and its suppliers.
Sell-Side Hub (supplier)	The seller owns the e-Marketplace and uses it as a vehicle to sell goods and services on the Web.	www.carmax.com (car purchase)
Buy-Side Hub (purchaser)	The buyer of the goods owns the e-Marketplace and uses it as a vehicle to leverage the buying or procurement budget in soliciting the best deals for goods and services from prospective sellers across the Web.	www.wwre.org (WorldWide Retail Exchange)

The makeup of these patterns is variable in that there will be basic patterns present for each type. However, you can extend the Composite 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

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

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

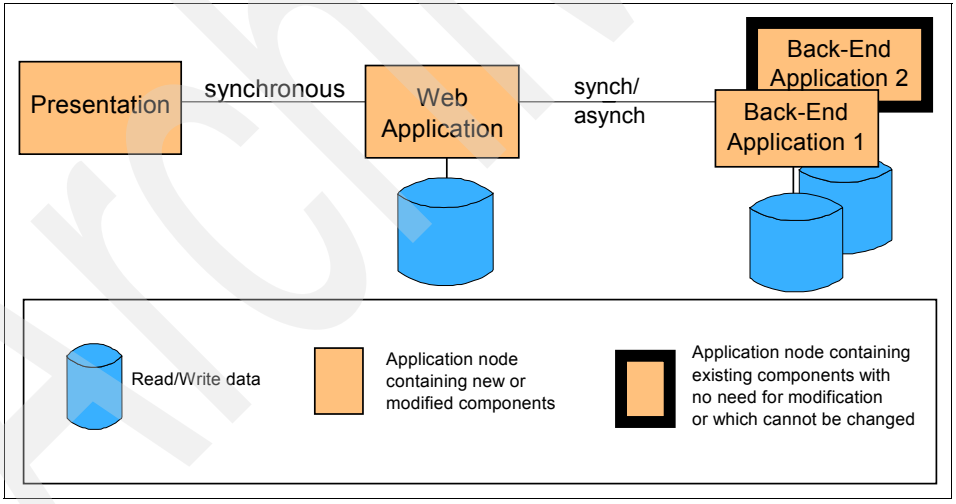

Figure 1-4 Self-Service::Directly Integrated Single Channel pattern

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

response, then next request/response) or asynchronous (multiple requests and responses intermixed).

Suppose that the situation is a little more complicated. Let's say that the automobile policies and the homeowner policies are kept in two separate and dissimilar databases. The user request actually needs data from multiple, disparate back-end systems. In this case, there is a need to break the request down into multiple requests (decompose the request) to be sent to the two different back-end databases, then to gather the information that is sent back from the requests, and put this information into the form of a response (recompose). In this case, the Self-Service::Decomposition application pattern (as shown in Figure 1-5) would be more appropriate.

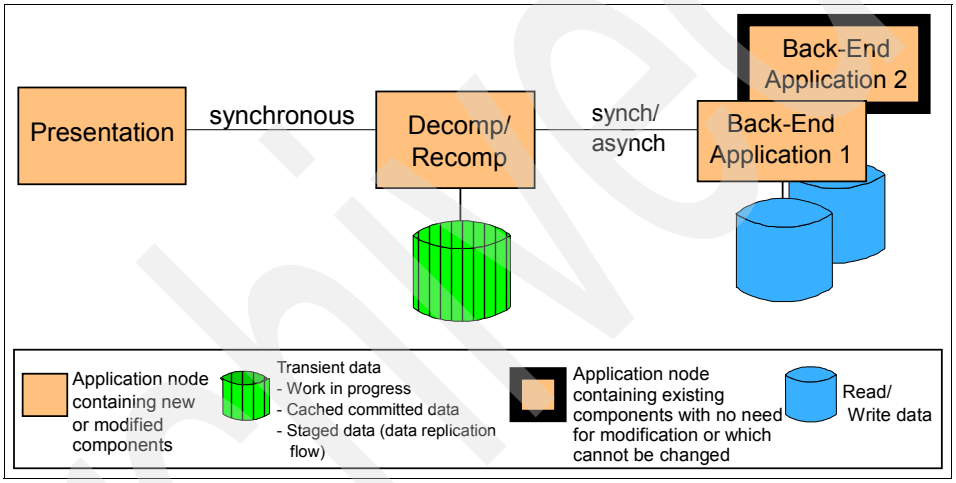

Figure 1-5 Self-Service::Decomposition pattern

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

1.2.3 Review Runtime patterns

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

As an example, suppose that our customer has determined that their solution fits into the Self-Service business pattern and that the Directly Integrated Single

Channel pattern is the most descriptive of the situation. The next step is to determine the Runtime pattern that is most appropriate for the situation.

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

The customer also needs to determine the functional nodes that are required to implement the application and security measures. Figure 1-6 shows the Runtime pattern that is one option.

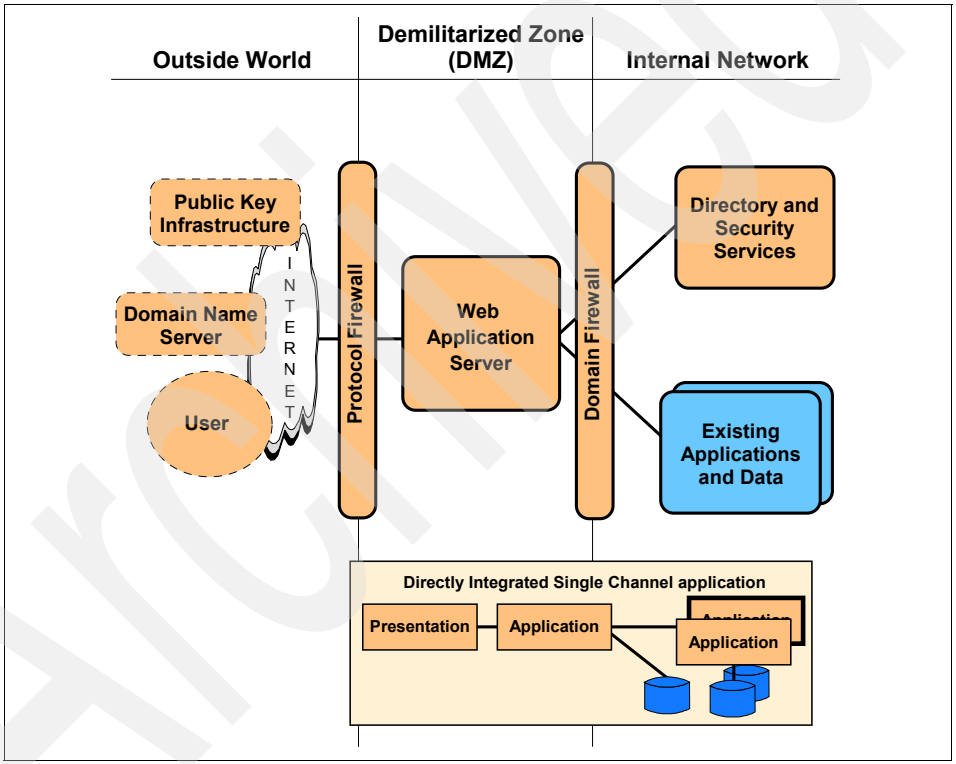

Figure 1-6 Directly Integrated Single Channel application pattern::Runtime pattern

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

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

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

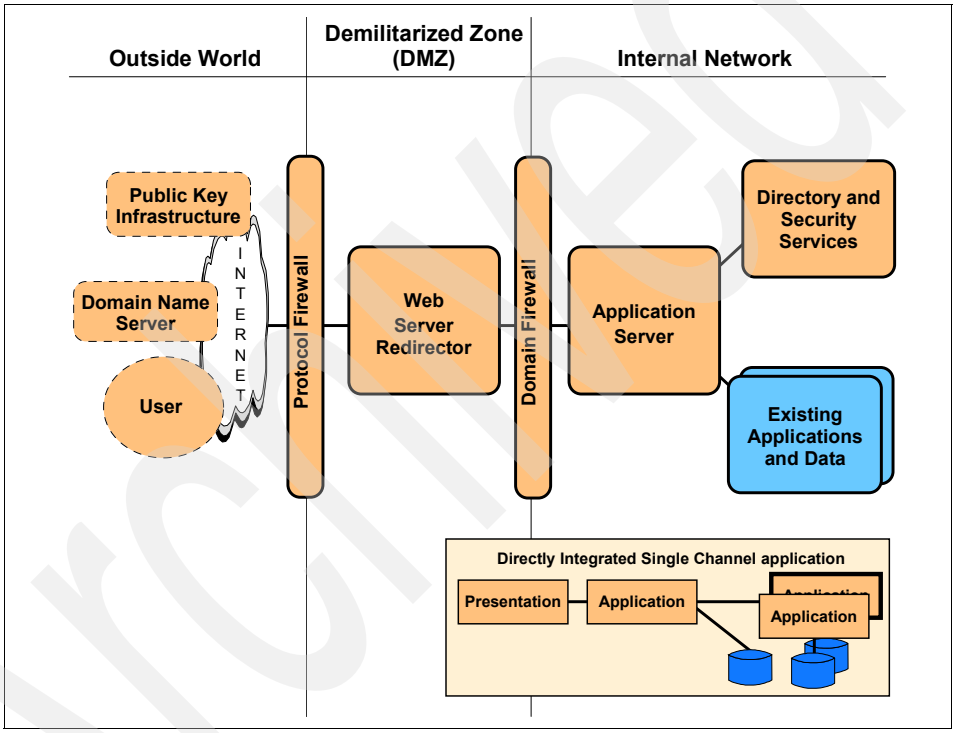

Figure 1-7 Directly Integrated Single Channel application pattern::Runtime pattern

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

1.2.4 Reviewing Product mappings

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

For example, you could implement the runtime variation in Figure 1-7 on page 14 using the product set that is depicted in Figure 1-8.

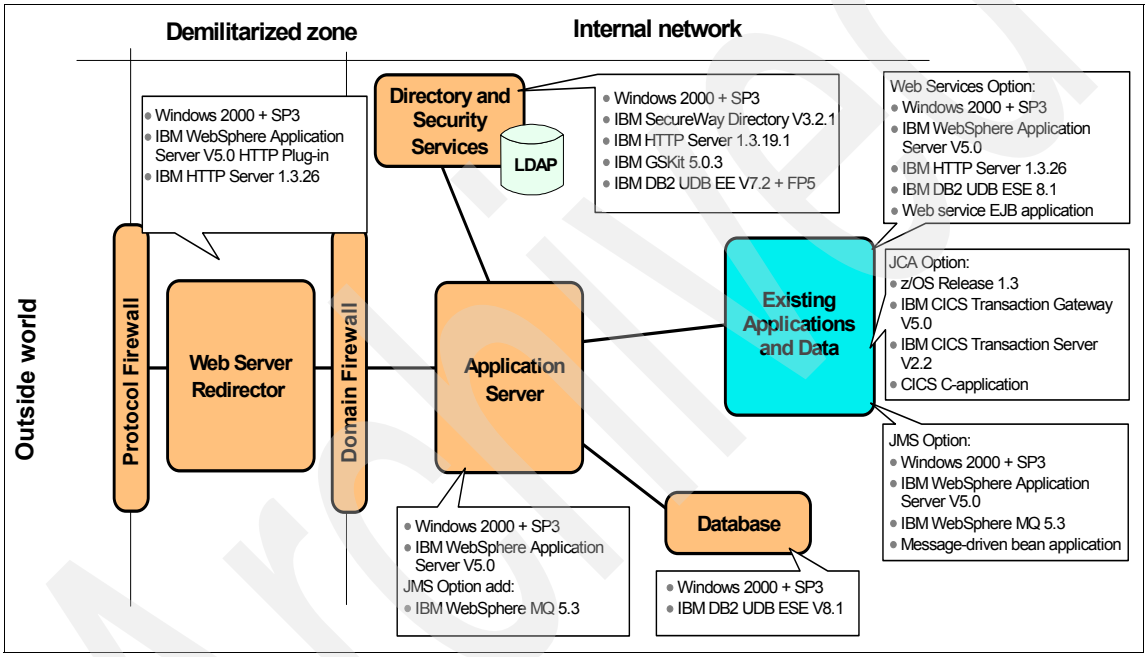

Figure 1-8 Directly Integrated Single Channel application pattern: Windows 2000 Product mapping

1.2.5 Reviewing guidelines and related links

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

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

1.3 Summary

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

2

Product descriptions

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

- WebSphere Application Server Version 6
- WebSphere Business Integration Message Broker Version 5
- WebSphere MQ Version 5.3
- WebSphere Enterprise Service Bus V6
- DB2® Universal Database™ Enterprise Server Edition Version 8.2
- IBM Rational® Application Developer V6.0

2.1 Runtime product descriptions

This section describes products that are discussed and used throughout this book for runtime functionality.

2.1.1 IBM WebSphere Application Server V6

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

The foundation of the WebSphere brand is the application server. The application server provides the runtime environment and management tools for J2EE and Web services-based applications. Clients access these applications through standard interfaces and APIs. The applications, in turn, have access to a wide variety of external sources, such as legacy systems, databases, and Web services, that can be used to process the client requests as shown in Figure 2-1.

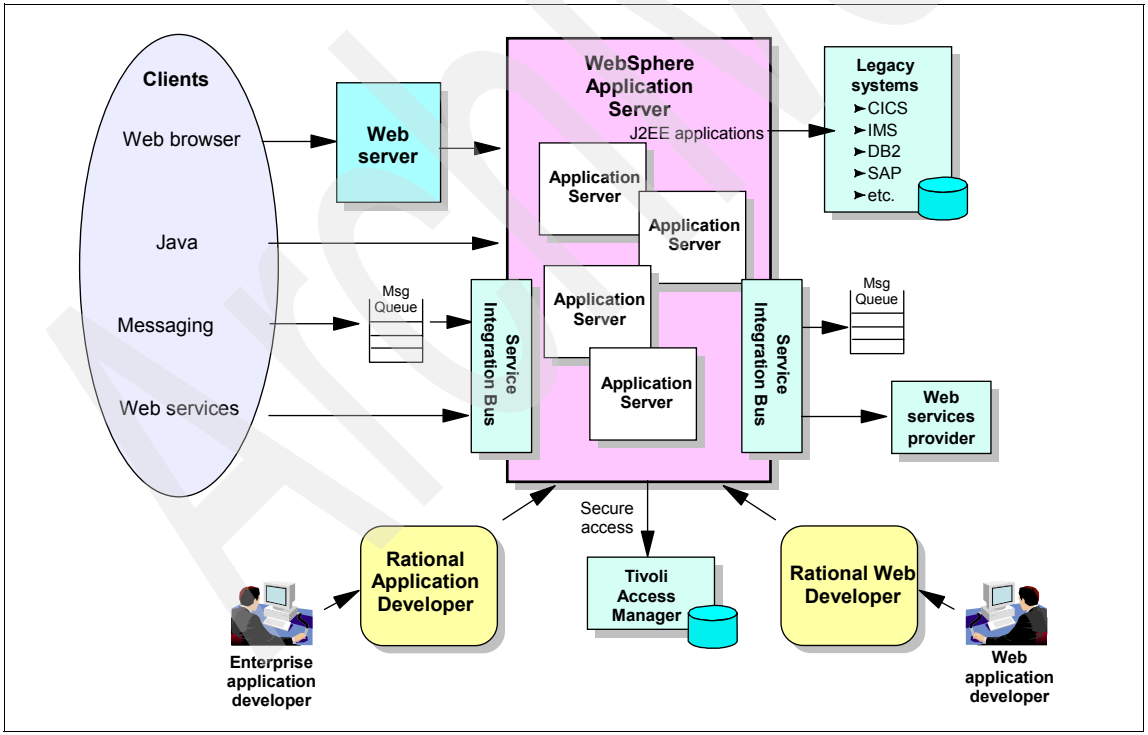

Figure 2-1 WebSphere Application Server product overview

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

Highlights and benefits

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

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

- J2EE V1.4 support.
- High performance connectors to many common back-end systems to reduce the coding effort required to link dynamic Web pages to real line-of-business data.
- Application services for session and state management.
- Web services that enable businesses to connect applications to other business applications, to deliver business functions to a broader set of customers and partners, to interact with marketplaces more efficiently, and to create new business models dynamically.
- The service integration bus infrastructure to complement and extend WebSphere MQ and the application server. It is suitable for those who are currently using the WebSphere Application Server V5 embedded messaging and for those who need to provide messaging capability between WebSphere Application Server and an existing WebSphere MQ backbone.

  The service integration bus features include:

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

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

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

You can find more information about using IBM WebSphere Application Server V6 in *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

WebSphere Application Server is currently packaged in the following configurations:

- WebSphere Application Server Express V6
- WebSphere Application Server V6
- WebSphere Application Server Network Deployment V6
- WebSphere Application Server Extended Deployment V5.1

WebSphere Application Server - Express V6

IBM WebSphere Application Server - Express Version 6 is a tightly integrated development tool and application server that provides an easily affordable entry point to e-business for companies creating dynamic Web sites utilizing a single

server deployment model. It supports the full J2EE 1.4 programming model and extensions including Servlets, JSPs, EJBs, and Web services.

WebSphere Application Server V6

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

WebSphere Application Server Network Deployment V6

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

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

In addition to all of the features and functions within WebSphere Application Server, this configuration delivers advanced deployment services that include clustering, edge-of-network services, Web services enhancements, and high availability for distributed configurations.

WebSphere Application Server Network Deployment provides the following features:

- Provides UDDI V3, enabling you to describe and discover Web services in more secure manner
- Delivers advanced Web services security to enhance the security of Web services interaction
- Provides the Web Services Gateway, which enables Web services invocation by users from outside the firewall with the benefit of robust security protection
- Supports advanced failover and clustering capabilities
- Simplified administration using the development tool interface
- Browser-based administration for remote administration across firewalls

- Convenient administration through embedded administrative console
- Intelligent workload distribution across a cluster
- Failure bypass
- Clustering support
- Edge Server Component, which delivers sophisticated load balancing, caching, and centralized security capabilities

Because WebSphere Application Server Network Deployment is designed for distributed configurations it made the ideal choice for deployment of WebSphere Application Server in the scenarios used in this book.

IBM WebSphere Web services gateway

The WebSphere Web services gateway, a component of WebSphere Application Server Network Deployment V6, provides a single point of control, access, and validation of Web service requests, and enables you to control which services are available to different groups of Web service users. You use the gateway to make available controlled sets of Web services for use within your organization and by external users. The services that each gateway instance makes available as Web services can be a mixture of internal services that are directly available at service integration bus destinations and external Web services. This approach provides the following benefits:

- The gateway service is made available at a Web address different from the target service, so you can replace or relocate the target service without changing the details for the associated gateway service.
- You can have more than one target service (that is, more than one implementation of the same logical service) for each gateway service.
- The gateway service can be made available on a different service integration bus from the target service.
- The gateway provides a common interface to the services in each set. Your gateway service users need not know where each underlying service is located, or whether the underlying service is being provided internally or sourced externally, or whether there are multiple target services available for a single gateway service.

In WebSphere Application Server Versions 4 and 5, the Web services gateway was a separable component with its own user interface. In WebSphere Application Server Network Deployment Version 6, the gateway is fully integrated into the IBM service integration technologies. It is not available in the WebSphere Application Server - Express V6 nor WebSphere Application Server packages.

WebSphere Extended Deployment V5.1

WebSphere Extended Deployment Version 5.1 delivers add-on features to WebSphere Application Server Network Deployment that provide a dynamic, goal-directed, high-performance environment for WebSphere applications. These extended capabilities help you to optimize the utilization and management of your deployments and enhance the quality of service of your business-critical applications. The following features and functions extend the capabilities of WebSphere Application Server Network Deployment:

- WebSphere Extended Deployment, delivering on demand responsiveness, simplified administration, and high-performance enhancements.
- WebSphere resource virtualization and pooling using node groups and dynamic clusters
- Dynamic adjustment of WebSphere resources through application placement
- Integration with Tivoli® Intelligent Orchestrator (an optional component, available separately) for enterprise-wide autonomic provisioning
- Introduction of operational policies to distributed WebSphere environments and intelligent routing and dynamic workload management according to established goals
- Visualization of operational environment and application-level performance against business goals
- Application partitioning technology and design patterns for improved performance and scalability for high-end transactional WebSphere applications
- High-availability services for increased reliability for business-critical applications

You can find more information about the WebSphere Application Server packages at:

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

2.1.2 IBM WebSphere Business Integration Message Broker V5

WebSphere Business Integration Message Broker V5 extends the messaging capabilities of WebSphere MQ (described in 2.1.3, “IBM WebSphere MQ V5.3” on page 27) by adding message routing, transformation, and publish/subscribe features. WebSphere Business Integration Message Broker provides a runtime environment that executes message flows, which consist of a graph of nodes that

represent the processing that is needed for integrating applications. The message flows can be designed to perform a wide variety of functions, including:

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

WebSphere Business Integration Message Broker also provides these features:

- Simplified integration of existing applications with Web services through the transformation and routing of SOAP messages, as well as logging of Web services transactions.
- Mediation between Web services and other integration models as both a service consumer and a service provider.
- Compliance with standards such as Web Services Definition Language (WSDL), Simple Object Access Protocol (SOAP), and Hypertext Transfer Protocol (HTTP).
- Integrated WebSphere MQ transports for enterprise, mobile, real-time, multicast, and telemetry endpoints.
- Standards-based metadata including XML schema and WSDL.

Architecture

WebSphere Business Integration Message Broker provides both the runtime and development environment necessary to provide broker functionality. Actions taken on messages are done by message flows executing in the runtime component. As messages arrive from applications over a supported transport, they are processed by the appropriate message flow, then sent on to their destination.

Figure 2-2 on page 25 illustrates the basic runtime and development architecture.

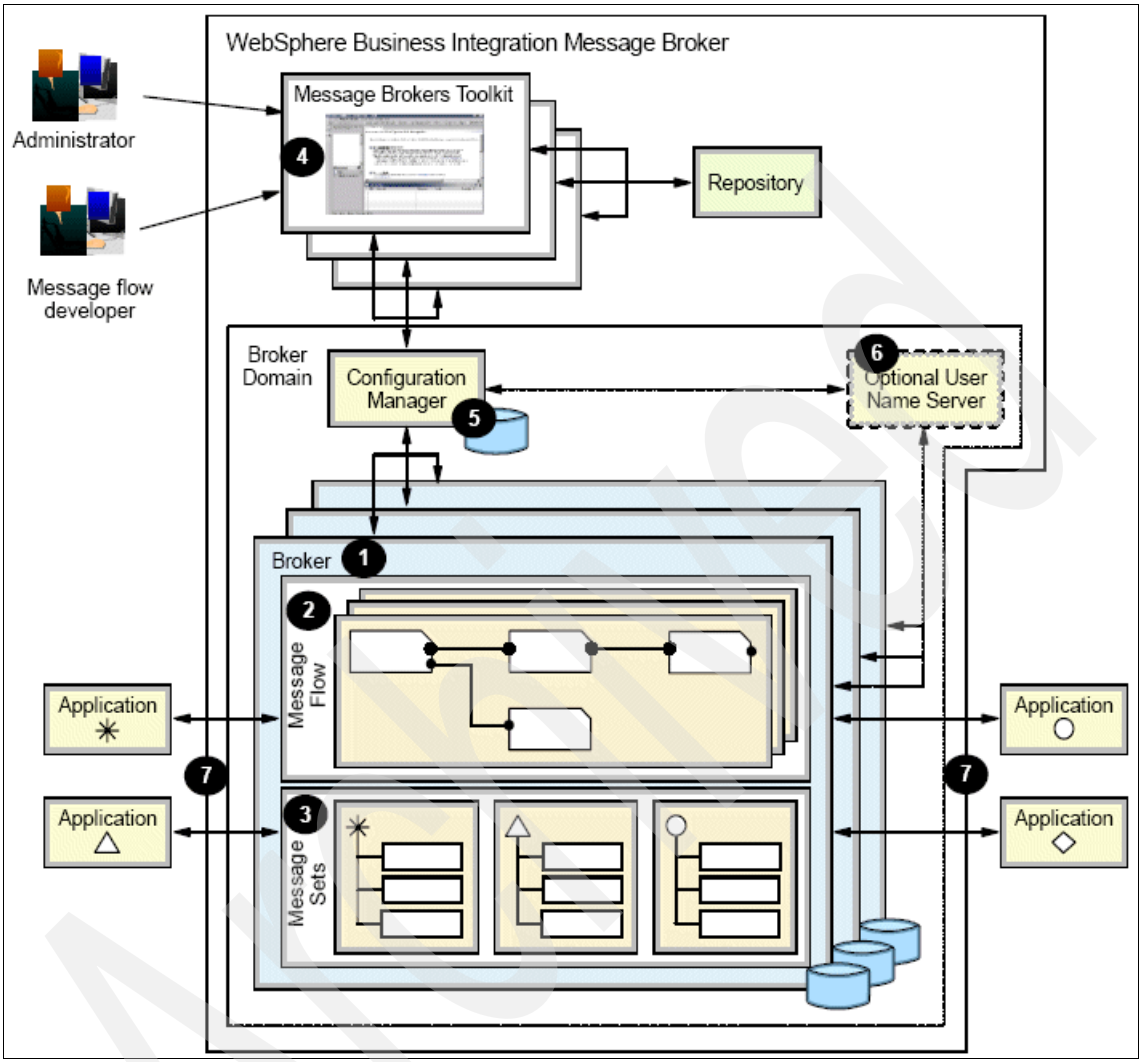

Figure 2-2 WebSphere Business Integration Message Broker architecture

1. The primary runtime component is the broker. Brokers contain a number of execution groups, which are processes in which message flows are run. Each broker uses a database to store the information it needs to process messages at runtime.
2. Messages are processed by message flows. Message flows are developed to provide specific functionality by wiring a series of nodes together. Each node has a specific job to do within the scheme of the message flow. Nodes for input and output are designed to take the messages from specific transport

types. Other nodes can perform computations or message enhancement, or make routing decisions.

3. Messages must have a defined structure that is known and agreed to by the sender and the receiver. In order for the broker to process messages, it must also understand their format. A message set contains message definition files that describe the messages.
4. The Message Brokers Toolkit for WebSphere Studio provides an integrated development environment for message flow development and runtime administration. Message flows and message sets can be developed using the Message Brokers Toolkit Workbench, then packaged for execution and deployed to the runtime environment via a connection established between the Workbench and the Configuration Manager.
5. The Configuration Manager coordinates all activity (for example, changes to a message set) between the Workbench and brokers within its domain. Brokers are grouped into broker domains. Each domain is coordinated by a Configuration Manager, which uses a database as a repository to store information relating to its broker domain.
6. If you have applications that use the publish/subscribe services of a broker, you can apply an additional level of security to the topics on which messages are published and subscribed. This additional security, known as topic-based security, is managed by the User Name Server. It provides administrative control over who can publish and who can subscribe.
7. Transport support facilities provide the interface between the client applications and the message flows. One or more WebSphere MQ queue managers provide the underlying transport infrastructure for the WebSphere Business Integration Message Broker. IBM WebSphere MQ messaging is used between the Workbench, the Configuration Manager, and the brokers. WebSphere MQ is also one of the transports supported for communication between applications and brokers as well.

WebSphere Business Integration Message Broker provides function and transport capabilities that support and facilitate enterprise-level business integration.

End-user applications can connect to the broker with the following transports:

- WebSphere MQ clients connect using the WebSphere MQ Enterprise Transport.
- WebSphere MQ Everyplace® clients (pervasive devices) connect using the WebSphere MQ Mobile Transport.
- Multicast JMS clients connect using the WebSphere MQ Multicast Transport.
- Real-time JMS clients (direct TCP/IP) connect using the WebSphere MQ Real-time Transport.

- SCADA clients (remote devices) connect using the WebSphere MQ Telemetry Transport.
- Web services clients (HTTP) connect using the WebSphere MQ Web Services Transport (Message Broker only), allowing message flows to be invoked as Web services

You can find more information about IBM WebSphere Business Integration Message Broker V5 at:

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

2.1.3 IBM WebSphere MQ V5.3

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

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

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

WebSphere MQ supports a variety of application programming interfaces (including MQI, AMI, and JMS), which provide support for several programming languages as well as point-to-point and publish/subscribe communication models. In addition to support for application programming, WebSphere MQ provides several connectors and gateways to a variety of other products, such as Microsoft Exchange, Lotus® Domino®, SAP/R3, CICS, and IMS™, to name just a few.

You can find more information about IBM WebSphere MQ at:

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

2.1.4 IBM WebSphere Enterprise Service Bus V6

WebSphere Enterprise Service Bus is a new product designed to provide an ESB for IT environments built around open standards and SOA. It delivers robust and easy-to use-functionality built on the proven messaging and Web services technologies of WebSphere Application Server. It is aimed at businesses looking for Web services based connectivity and service oriented integration.

WebSphere Enterprise Service Bus provides the following features:

- Provides Web services connectivity, JMS messaging, and service oriented integration by including support for:
  - SOAP/HTTP
  - SOAP/JMS
  - WSDL V1.1
  - UDDI V3.0
- Provides support for building an ESB with integration logic such as:
  - Protocol conversion for messages received over HTTP, JMS, and IIOP
  - Format transformation between XML, SOAP, and JMS message standards, and many more when used with adapters
  - Mediation capabilities, including pre-built mediations for the following functions:
    - Message logging
    - Flow of business events
    - Use of WebSphere Adapters to capture and disseminate business events

You can find more information about IBM WebSphere Enterprise Service Bus at:

`http://www.ibm.com/software/integration/wsesb/`

2.1.5 IBM DB2 Universal Database Enterprise Server Edition V8.2

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

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

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

- Broadened autonomic computing solutions that automate and simplify potentially time consuming and complex database tasks.
- A significant amount of new capabilities as well as further integration of DB2 tooling into the Microsoft .NET and WebSphere Java environments. These new capabilities simplify the development and deployment of DB2 applications and enable application developers to take advantage of the openness, performance, and scalability of DB2 without regard to the back-end database or the chosen application architecture
- Integration of industry-proven high availability disaster recovery technology enable line-of-business managers and the enterprise itself to benefit because applications face less risk of downtime.

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

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

2.2 Development product descriptions

This section describes products that are discussed and used throughout this book for development.

2.2.1 IBM Rational Application Developer V6

Rational Application Developer is an integrated development environment with full support for the J2EE programming model including EJB development, Web services, Web applications, and Java. In previous releases this product was known as WebSphere Studio Application Developer. This tool includes integrated portal development, UML editing, code analysis, automated test and deployment tools, built-in version control, and team tools. Everything you need to be productive and to make sure written code is well designed, scalable, and ready for production is included in Rational Application Developer. Additionally, everything is provided for version control and protection when developers work in large teams or on complex projects. Rational Application Developer is optimized for IBM WebSphere software.

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

test, debug, and deploy applications in support of the application development life cycle.

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

- Rational Web Developer tools allow accelerated use of portal, SOA, and J2EE.
- You can shorten the Java learning curve by using drag-and-drop components and point-and-click database connectivity.
- You can improve code quality by using automated tools for applying coding standard reviews, component, and Web service unit testing and multi-tier runtime analysis.
- Business applications can be integrated with Web services and SOA.

You can find more information about IBM Rational Application Developer at:

`http://www.ibm.com/software/awdtools/developer/application`

3

SOA runtime patterns and Product mappings

This section describes the Runtime patterns and Product mappings that are relevant to a service-oriented architecture (SOA) with specific focus on the Enterprise Service Bus (ESB).

This chapter describes the following SOA runtime patterns:

- Direct Connection using a Service Bus runtime pattern
- ESB runtime pattern
- ESB Gateway runtime pattern
- BSC runtime pattern
- ESB, BSC composite pattern
- Exposed ESB Gateway runtime pattern (for inter-enterprise)
- Exposed ESB Gateway, BSC composite pattern (for inter-enterprise)

This chapter also provides Product mappings for the following Runtime patterns:

- ESB runtime pattern
- ESB Gateway runtime pattern
- BSC runtime pattern
- Exposed ESB Gateway runtime pattern (for inter-enterprise)

You can find an overview of the products used in the Product mappings in Chapter 2, “Product descriptions” on page 17.

3.1 Runtime patterns

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

The Runtime patterns that are illustrated in this chapter give some typical examples of possible solutions. However, these examples should not be considered exhaustive.

3.1.1 Direct Connection using a service bus

The Direct Connection runtime pattern (Figure 3-1) shows a service consumer that is connected to two other service providers via a simple service bus. The Application pattern overlays in this figure show that multiple Direct Connection application patterns can be deployed using the service bus.

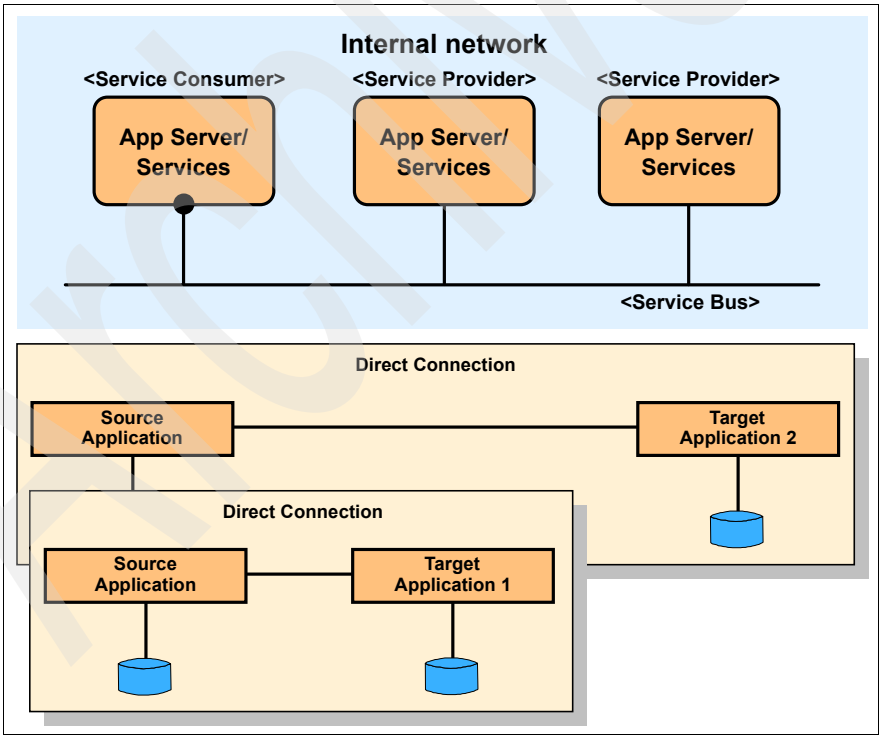

Figure 3-1 Direct Connection using a simple service bus

Note: The figure shows the relationship between the Application and Runtime patterns as an example. For clarity, the remainder of this section concentrates on the Runtime patterns and does not show the associated Application patterns. Find full mapping between Application and Runtime patterns at:

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

The service consumer (or source application) can use the service bus to initiate direct connections to two service providers — one to Target Application 1 and the other to Target Application 2.

In order to focus on the service bus concept, we do not explicitly model adapter connectors or connection rules in Figure 3-1 on page 32. The service bus concept is, however, an extension of the Direct Connection with federated adapter connectors runtime pattern that enables a set of connected Direct Connections. The service bus approach:

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

The service bus can span multiple system or application tiers, and can extend beyond the enterprise boundary. A rules repository node can also be included to model a service directory, allowing services to be discovered within and outside of the enterprise.

Note: The very simple service bus described here provides just a small subset of the integration capabilities of a true ESB as described in the remainder of this chapter.

3.1.2 ESB runtime pattern

The Runtime pattern shown in Figure 3-2 provides the highest-level view of the ESB.

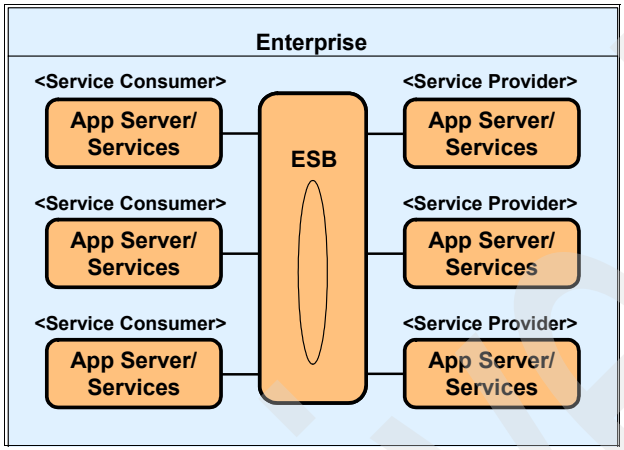

Figure 3-2 ESB runtime pattern: Level 0

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

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

The Runtime pattern shown in Figure 3-3 on page 35 represents a first-level decomposition of the major components that make up an ESB.

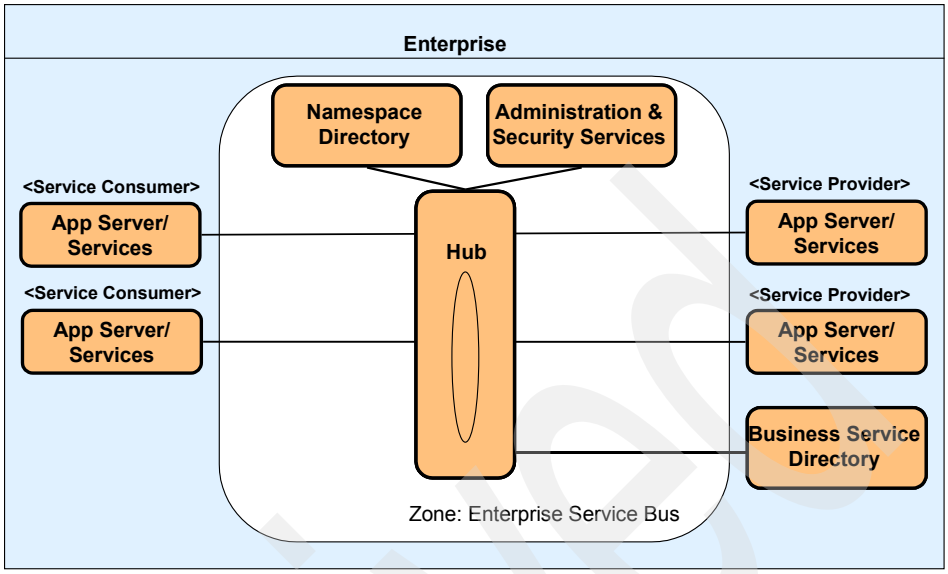

Figure 3-3 ESB runtime pattern: Level 1

This basic topology leverages the nodes with their associated responsibilities as described in the following sections.

App server/services node

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

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

Hub node

This node supports the key ESB functions and, therefore, fulfills a large part of the ESB capabilities. The hub has a fundamental service integration role and should be able to support various styles of interaction. There are two interaction styles (that are covered in detail in Part 3) that the hub supports. Those styles are the Router and Broker interaction patterns. The Router interaction pattern is where a request is routed to a single provider. The Broker interaction pattern supports requests that are routed to multiple providers, including aggregation and disaggregation of messages. The hub must contain rules for routing messages, and in the case of hubs that support the Broker interaction pattern,

the rules must also describe how messages should be disaggregated or aggregated.

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

- Routing

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

- Addressing

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

- Messaging styles

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

- Transport protocols

  The hub should support at least one transport that is or can be made widely available, such as HTTP/S. The hub can provide protocol transformation. If a protocol transformation is required that is not supported by the hub, then a specific connector can be used to perform the transformation. (See “Connectors” on page 38).

- Service interface definition

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

- Service messaging model

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

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

- Integration

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

- Quality of service

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

- Message processing

  The hub can support more advanced message processing capabilities such as encoded logic, content-based logic, message and data transformations, message/service aggregation and correlation, validation, intermediaries, object identity mapping, service/message aggregation, and store and forward.

- Modeling

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

- Service level

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

- Infrastructure intelligence

  More advanced infrastructure capabilities can be provided. These include:

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

Namespace directory

This node provides routing information in order for the hub to perform routing of service interactions. This node could be implemented as a routing table in the more simple implementations of an ESB.

Administration and security services

This section covers both administration and security services.

Administration

An ESB should be controlled by a single administration infrastructure. This node provides these administration services which, at a minimum, should support service addressing and naming.

The key services that must be provided by this node are:

- ESB configuration
- Service provisioning and registration
- Logging
- Metering
- Monitoring
- Integration with systems management and administration tooling

More advanced administration features that can be provided by this node include self-monitoring and self-management.

Security

In a mission-critical environment and, depending on the confidentiality, integrity, and availability requirements of the applications, the hub should support security capabilities such as authentication, authorization, non-repudiation, confidentiality, and security standards, such as Kerberos and WS-Security.

Business service directory

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

The vision of Web services defines an open-standard UDDI registry that enables the dynamic discovery and invocation of business services. However, although technologies mature toward that vision, more basic solutions are likely to be implemented in the near term.

Connectors

If we model the connectors that facilitate the interactions between service consumer/providers and the ESB, as shown in Figure 3-4 on page 39, we find that we might require that some of these are both *adapter connectors* and *path connectors*, while other service consumer/providers need only a path connector to the ESB.

An adapter connector is concerned with enabling logical connectivity by bridging the gap between the context schema and protocols used by the source and target applications (in this case, between the service consumer/providers and the ESB).

A path connector is concerned with providing physical connectivity between source and target applications. It can be very complex (for example, the Internet) or very simple (an area of shared storage).

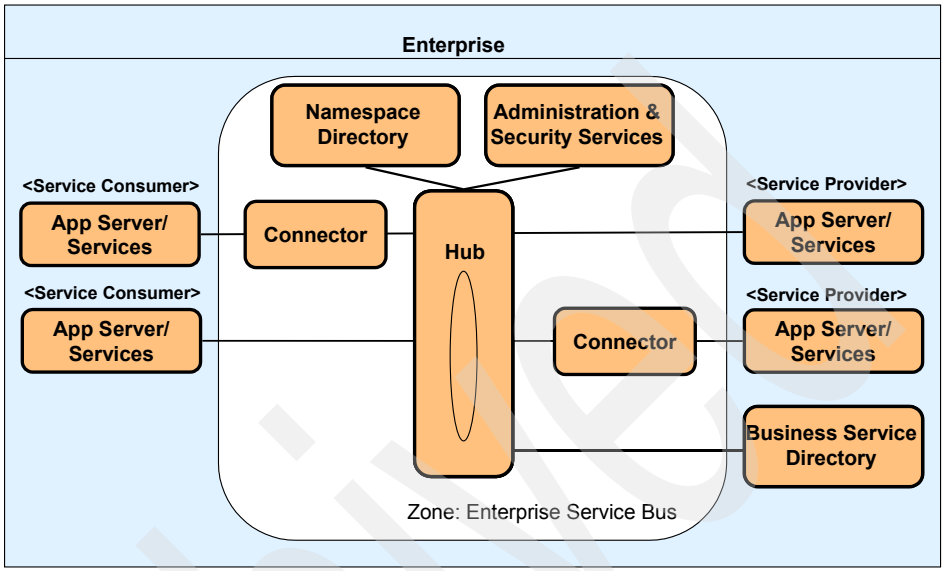

Figure 3-4 ESB level-two diagram showing adapter connectors

Adapter connectors facilitate integration in a heterogeneous environment with diverse technology, protocols, application types, and integration styles. Adapters perform the following key types of functions:

- Technology adaptation

  This type of adapter handles service consumers and providers that are built using technologies that are not natively supported by the hub. Examples of technologies that can be supported via adapters are CORBA, COM, JDBC, JMS, and EJB. Some of these technology adapters can use data handlers for particular data formats such as EDI, SOAP, XML, and various text formats.

  These adapters can also support different application server environments such as J2EE and .NET and different language interfaces such as Java, C, C++, and C#.

- Application adaptation

  This type of adapter facilitates integration with package solutions. Many examples of package solutions provide application adapters, such as Siebel, PeopleSoft, and SAP, among others.

- Legacy adaptation

  This type of adapter facilitates exposing valuable enterprise applications as services. These enterprise applications can be implemented using technologies such as CICS Transaction Server, IMS Transaction Manager and ADABAS amongst others.

  Development-time support can also be provided in order to develop custom adapters.

As an example, the connectors in Figure 3-4 on page 39 that are modeled (that is that are represented as a *connector* node) can support a Siebel Customer Service application that acts as a service consumer to the ESB and requests a service that is provided by an enterprise application running under CICS Transaction Server. In this scenario, the connector might be a Siebel *application adaptation* adapter connector and a l*egacy adaptation* adapter connector that supports CICS Transaction Server. The connectors that are not modelled (that is, that are only represented by a line) in this example could support the interaction between applications that use a SOAP/JMS to interface with the ESB and, therefore, only require a path connector.

Not all connectors are necessarily within the ESB Zone. Figure 3-5 shows the possible placement options for connectors that support interaction between application server and services and the ESB. These application server and services can be service consumers or service providers.

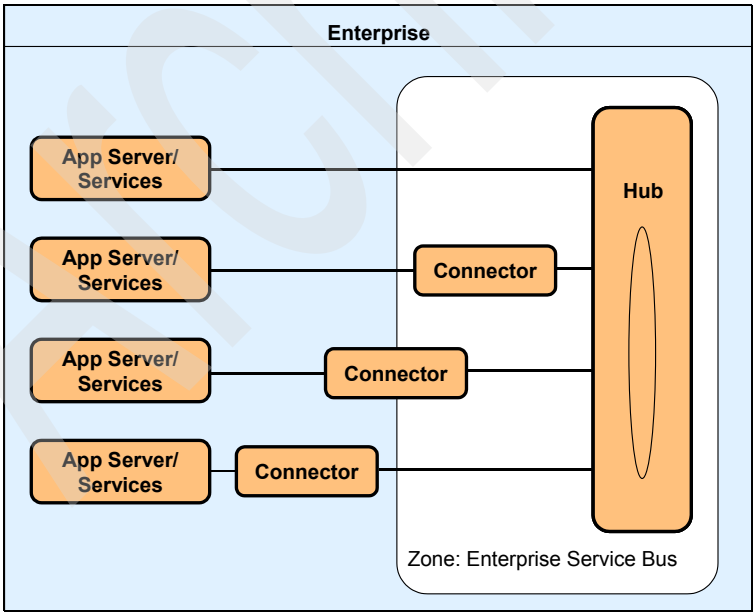

Figure 3-5 Placement of adapter connectors

The placement options for connectors are:

- Inside the ESB Zone
- On the boundary of the ESB Zone
- Outside the ESB Zone

In general, IT artifacts (such as nodes, connectors, and client APIs) have some configuration that determines their behavior. If this configuration is managed by the ESB management infrastructure, the artifact is inside the ESB Zone.

In some instances, an artifact such as a connector can be either outside or on the boundary of the ESB Zone, depending on whether it is managed or partly managed by the ESB infrastructure. An example of a partly managed connector could be an ESB that is built on WebSphere V6 with J2C Adapter to CICS Transaction Server using CICS Transaction Gateway in Server or Client mode. CICS Transaction Gateway runs as a separate process, or at least with its own configuration and management, but the J2C end is within WebSphere control.

In the scenario described previously, if we build the ESB using WebSphere Business Integration Message Broker, linking to the J2C adaptor that is running in WebSphere Application Server, the J2C adaptor is outside the WebSphere Business Integration Message Broker management and, therefore, outside of the ESB Zone.

3.1.3 ESB Gateway runtime pattern

The Runtime pattern shown in Figure 3-6 provides the highest-level view of the ESB Gateway.

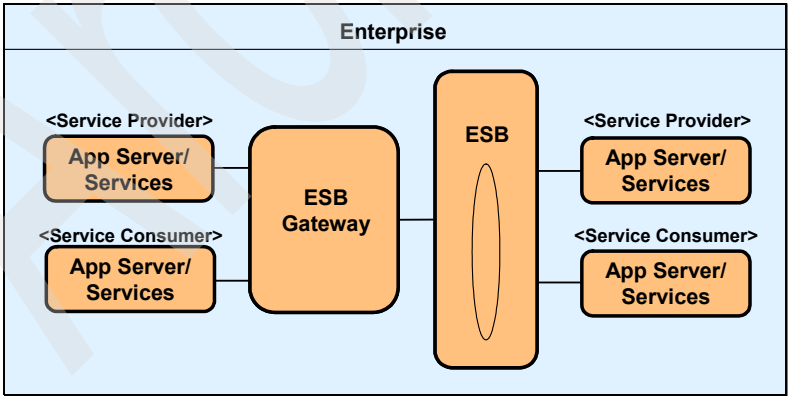

Figure 3-6 ESB Gateway runtime pattern: Level 0

The ESB Gateway acts as a proxy to provide controlled access to the ESB. A common use of the ESB Gateway is to expose services to external parties as well as allow internal applications to access external services in a secure and controlled manner. This section discusses a generic ESB Gateway pattern. For information about the Exposed ESB Gateway runtime pattern, see “Exposed ESB Gateway runtime pattern” on page 48.

Figure 3-7 represents a first-level decomposition of the major nodes that make up the ESB Gateway.

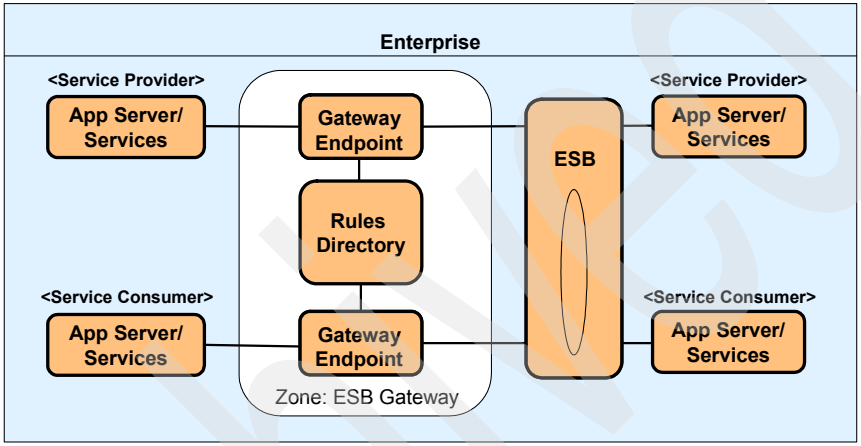

Figure 3-7 ESB Gateway runtime pattern: Level 1

This basic topology leverages the nodes with their associated responsibilities as described in the following sections.

App server/services node

You can find information about this node in “App server/services node” on page 35.

ESB

The ESB is a key enabler for an SOA because it routes and transports service requests from the service consumer to the correct service provider. For information about this node, see “ESB runtime pattern” on page 34.

Rules directory

This node contains the necessary configuration information that the ESB Gateway needs to support secure and controlled access to services. The rules directory has configuration rules that can include mapping of service interface

definitions to gateway endpoints, mapping of ESB gateway-provided service names to destination service names, and access control lists.

The configuration rules can also include information about service level policies to control throughput. These rules protect associated service implementations from operating beyond the established capacity levels.

Gateway endpoint

This node is the entry point into services that the ESB provides or that are external to the ESB. It provides the address where messages are received, and it is mapped to particular protocols that the ESB Gateway supports (for example, HTTP/S). The gateway endpoint controls access to and from the ESB based on configuration rules that include access control lists and service level policies. It maps requests to the appropriate service and facilitates the interaction.

3.1.4 BSC runtime pattern

The Runtime pattern shown in Figure 3-8 provides the highest-level view of this pattern.

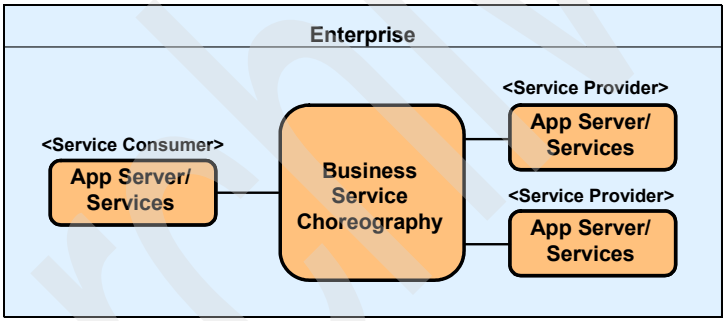

Figure 3-8 BSC runtime pattern – Level 0

With the Business Service Choreography (BSC) runtime pattern, you can develop and execute business process flow logic that governs the sequence and control of service invocations. The business process is controlled centrally and is not part of the program logic in individual applications. Therefore, rather than having the business process defined in multiple applications and within the interactions between these multiple applications, the business process can be modeled and implemented by a central function. The Business Service Choreography facilitates the implementation of changes to the business process and monitoring and analysis of business process execution.

Figure 3-9 represents a first-level decomposition of the BSC node.

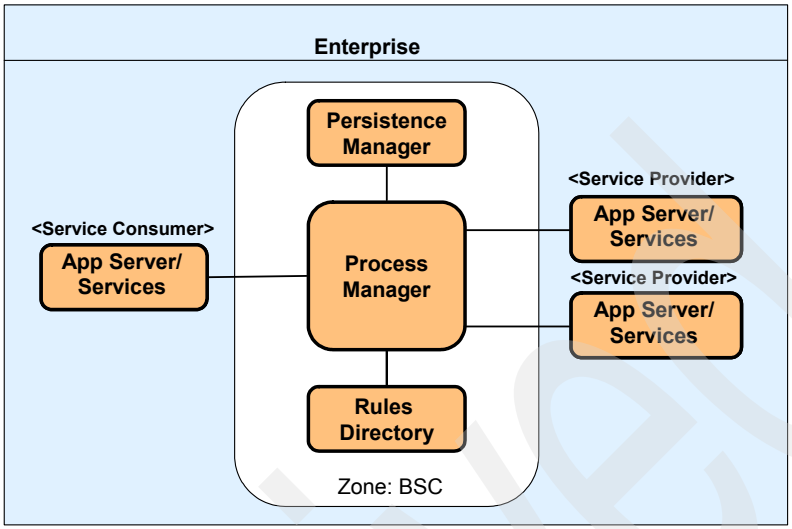

Figure 3-9 BSC runtime pattern – Level 1

This basic topology leverages the nodes with their associated responsibilities as described in the following sections.

App server/services

This node is described in “App server/services node” on page 35. In this pattern, the app server/services node provides services to the process manager.

Process manager

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

These processes can span multiple applications and organizational boundaries within an enterprise. The node maintains state and tracks sequencing through the process flow. In doing so, it often leverages the persistence manager to store intermediate results. Finally, it invokes target services as necessary via the ESB.

The process manager node can support serial processes in which there is a sequential execution of process steps and parallel processes where process steps or subprocesses can execute concurrently.

The process manager should support the following key capabilities:

- Process definition standards, such as WS-BPEL, and the ability to execute process definitions that have been defined and exported from a modeling tool.
- Monitoring and analysis of processes by capturing information about process execution for historical analysis. It should also support integration with system management and administration tools.
- Ability to meet non-functional requirements such as performance, availability, and scalability will be important for mission-critical enterprise applications. Other key non-functional requirements are security and transaction management, particularly supporting the integrity and recovery of long-running business processes.
- Multiple levels of process abstractions.
- Correlation of events or incoming messages with existing process instances.
- Support for branching, parallel branch execution, and recomposing if the process manager supports parallel process execution.

Persistence manager

This node provides a persistent data storage service in support of the process flow execution. It holds results from the execution of certain activities within the context of an end-to-end process flow. These can be intermediate results that are valid within the context of a particular process flow and process data for the purpose of process monitoring and analysis. The intermediate results are necessary to support state management.

The implementation of this node typically involves a persistent data technology, such as a DBMS. In some cases, you can use non-persistent storage to store the intermediate results.

Rules directory

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

3.1.5 ESB, BSC composite pattern

The Runtime pattern shown in Figure 3-10 provides the highest-level view of this Composite pattern.

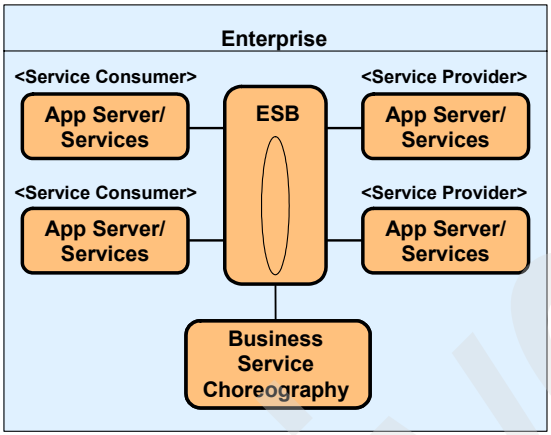

Figure 3-10 ESB with BSC composite pattern – Level 0

The BSC node is implemented as a service consumer or service provider of the ESB. The BSC node is focused on process management function, and the ESB node provides the integration capabilities with other services. This pattern generally provides a loosely coupled and more functionally cohesive architecture where functional responsibility of nodes is clearly defined. The business process governs the sequence and control of service invocations that are mediated through the ESB.

Figure 3-11 on page 47 represents a first-level decomposition of the major nodes that make up this pattern.

The BSC has two core components, the process manager and repository nodes, that support the development and execution of business process flow logic. This logic is controlled centrally outside the application logic. Shielding the applications from the business process flow facilitates the implementation of changes to the business process and the monitoring and analysis of business process execution.

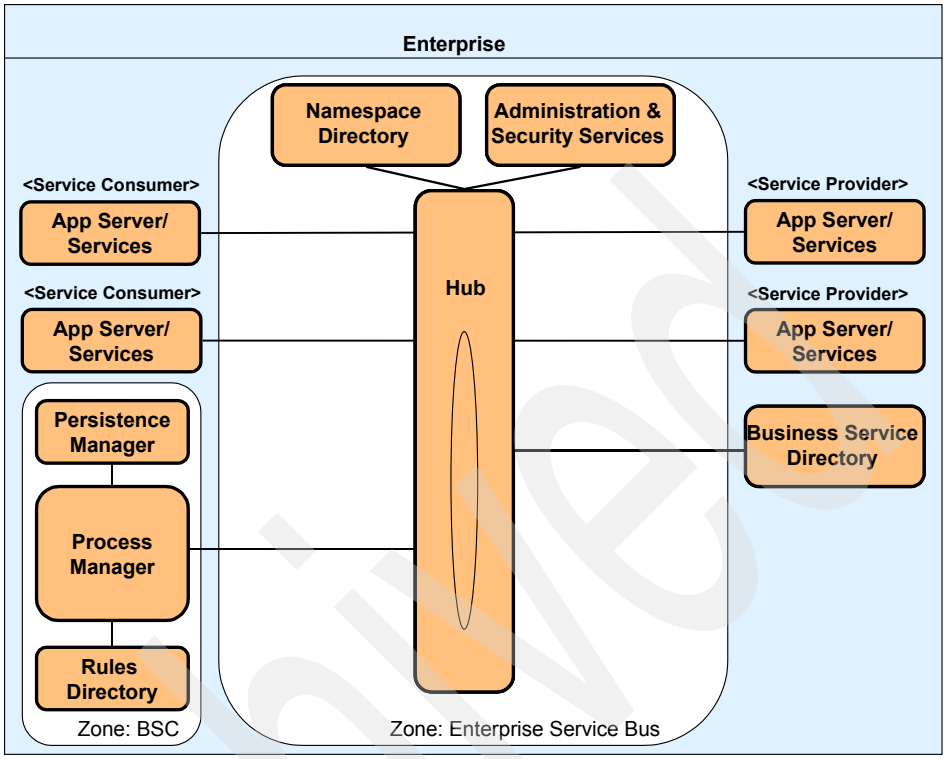

Figure 3-11 ESB with BSC composite pattern – Level 1

Note: Only a connector to the ESB is required, as opposed to a connector to each app server/services node that is involved in the process flows as described in "BSC runtime pattern" on page 43. The service integration function is subsumed by the ESB, leaving the BSC to perform its core process management function.

The BSC and ESB nodes are described in "ESB runtime pattern" on page 34. Therefore, this section provides only a brief description of these nodes.

BSC

This node is limited to process management and contains only the process manager, rules directory, and persistent manager nodes. The BSC relies on the ESB for integration and security functionality, and both receive requests from the ESB and send requests to the ESB via the hub node. The ESB can issue a request to the BSC to start execution of a process. The process execution will in

turn most likely require services to be invoked as part of the process flow. Therefore, the BSC will request services from the ESB.

For a description of the BSC nodes, see “BSC runtime pattern” on page 43.

ESB

The ESB nodes are described in “ESB runtime pattern” on page 34. As far as the ESB is concerned, the BSC is another application that can both request and provide services.

3.1.6 Exposed ESB Gateway runtime pattern

Note: This pattern applies to inter-enterprise solutions.

Figure 3-12 shows a runtime pattern that supports secured and controlled access to enterprise services from outside of the enterprise and that enables enterprise applications to access external services. The two major nodes in this pattern, the Exposed ESB Gateway and ESB, are described in “ESB Gateway runtime pattern” on page 41 and “ESB runtime pattern” on page 34.

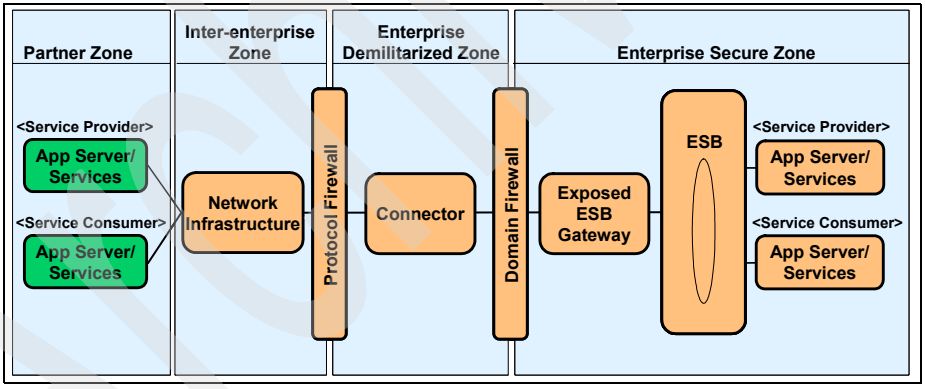

Figure 3-12 Exposed ESB Gateway runtime pattern – Level 0

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

This basic topology leverages the nodes with their associated responsibilities as described in the following sections.

App server/services

This node is described in “App server/services node” on page 35.

Note: The app server/services node that interacts directly with the ESB Gateway could be an ESB in the other enterprise.

Connector

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

Exposed ESB Gateway

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

This node is described in “ESB Gateway runtime pattern” on page 41.

ESB

The ESB is a key enabler for an SOA as it provides the capability to route and transport service requests from the service consumer to the correct service provider.

This node is described in “ESB runtime pattern” on page 34.

3.1.7 Exposed ESB Gateway, BSC composite pattern

Note: This pattern applies to inter-enterprise solutions.

The Runtime pattern shown in Figure 3-13 adds BSC to the Exposed ESB runtime pattern that is described in “Exposed ESB Gateway runtime pattern” on page 48.

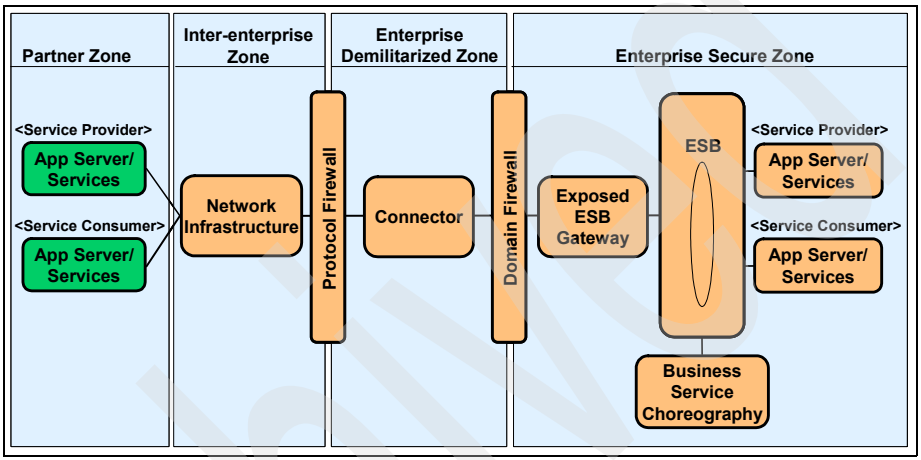

Figure 3-13 Exposed ESB Gateway, BSC composite pattern: Level 0

This Runtime pattern supports scenarios where business process services should be provided to both external and internal requesters. The BSC node is added to expose enterprise business processes to the enterprise, clients, partners, and suppliers.

This Runtime pattern also enables internal requesters to have controlled and secure access to services external to the enterprise, which can also include business processes, depending on the capabilities implemented by the external organization. Therefore, this Runtime pattern, when combined with appropriate process definition standards such as BPEL4WS, enables inter-enterprise processes.

3.2 Product mappings

After choosing a Runtime pattern, you need to determine the actual products and platforms that you will use. The Product mappings in this section are suggested mappings, and they address both of the scenario implementations that Part 3 of this book discusses. These Product mappings are also typical product mappings that are used for production systems.

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

Other considerations that influence the product selection include:

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

Note: The product mappings in this section do not include hardware nodes and operating systems. The sample scenarios in Part 3 of this book were implemented on xSeries® Servers running the Windows® 2000 Operating System.

3.2.1 ESB runtime pattern::Product mappings

Figure 3-14 shows a Product mapping for the ESB runtime pattern.

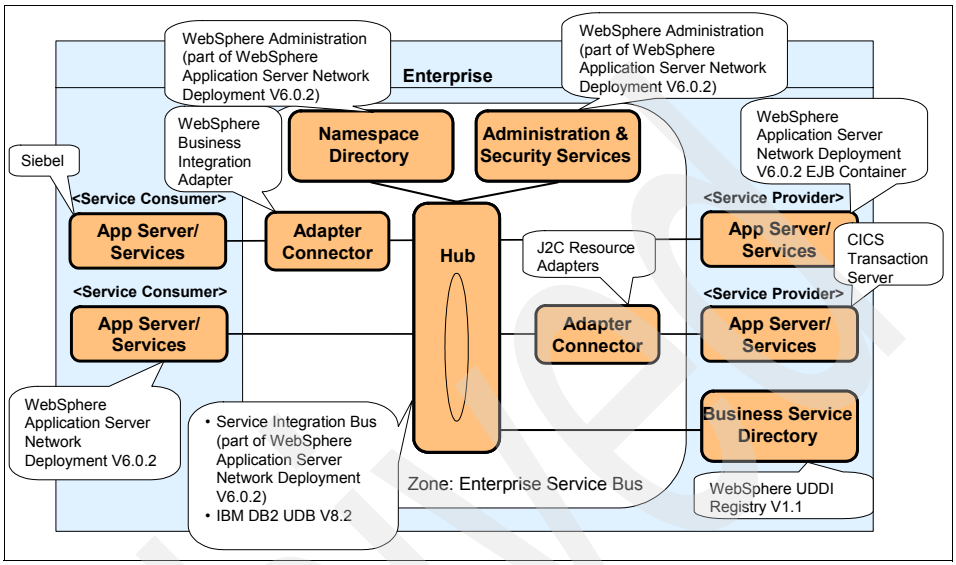

Figure 3-14 ESB runtime pattern::Product mapping=WebSphere Application Server V6

This Product mapping uses WebSphere Application Server Network Deployment V6.0. Using WebSphere Application Server Network Deployment, you can implement a scalable clustering of multiple WebSphere Application Server servers. If the clustering capability is not required, you should use the base WebSphere Application Server V6 offering.

The service consumer applications that are supported are not only Java applications that issue SOAP/HTTP requests but are also packages or applications that are built on other technologies, using other protocols (for example, the Siebel package in shown in Figure 3-14). For this purpose, the WebSphere Business Integration Adapters are used to implement the adapter connector node. The ESB hub is run on WebSphere Application Server Network Deployment, which acts as a broker between the requester and multiple provider applications that are also running under WebSphere Application Server Network Deployment.

The J2EE Connector Architecture (J2C) resource adapter is used to implement Adapters to access services that are implemented under enterprise resources such as CICS Transaction Server. The WebSphere UDDI Registry is used to implement the business service directory. The advantage of using a UDDI registry is that there is a central location where all available services are

published, which should encourage reuse of services within an enterprise. The Administration Services and namespace directory are provided by WebSphere Application Server Network Deployment. A local DB2 database is used to store the SDO repository.

3.2.2 ESB Gateway runtime pattern::Product mapping

Figure 3-15 shows the Product mapping for the ESB Gateway runtime pattern.

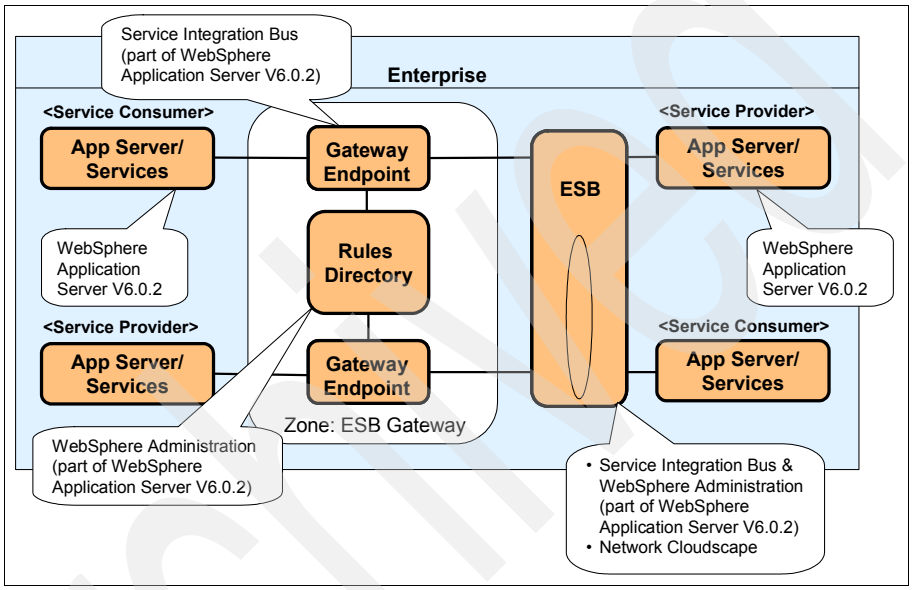

Figure 3-15 ESB Gateway::Product mappings

The service consumer application in this scenario is implemented using WebSphere Application Server V6. However, it could be implemented in other technologies and, in fact, could be another ESB. The service consumer initiates a service via the Gateway using either SOAP over HTTP or SOAP over JMS. The gateway endpoint node in the gateway is implemented using WebSphere Application Server, and the rules directory is implemented using the file system.

The ESB Gateway verifies that it is a valid request via the access control list held in the Repository and maps the request to a service that is provided by the ESB. The request to the ESB uses SOAP over HTTP or JMS.

Network Cloudscape™ database is used to store the SDO repository. The network configuration of Cloudscape is required to enable the ESB Gateway and the ESB to share the same repository.

The service provider application can be implemented using the EJB container of WebSphere Application Server. However, the ESB supports a heterogeneous environment through the use of adapters. Therefore, the services can be legacy enterprise applications, other non-J2EE application servers, or software packages.

3.2.3 BSC runtime pattern::Product mapping

Figure 3-16 shows the Product mapping for the BSC zone of the ESB, BSC composite pattern. You can find the Product mapping for the ESB in 3.2.1, “ESB runtime pattern::Product mappings” on page 52.

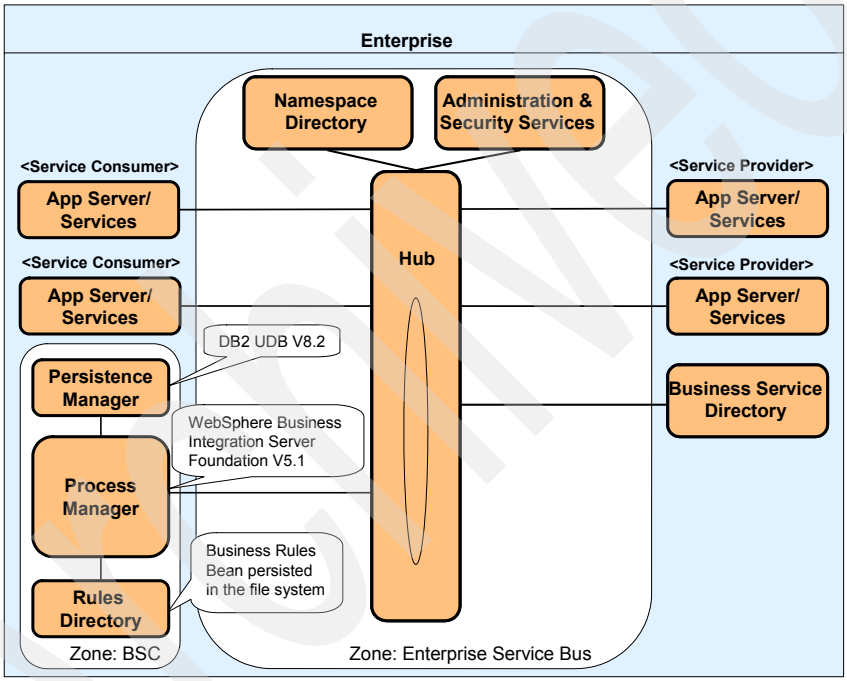

Figure 3-16 BSC runtime pattern::Product mappings

The process manager is implemented using the Business Process Choreography component that is part of the WebSphere Business Integration Server Foundation product. The process manager controls the process execution and invokes services from the ESB via the hub using SOAP over HTTP or JMS.

The process manager uses the persistence manager implemented using the DB2 database to store process results and a business rules bean persisted in the file system to implement the business process flow rules as part of the rules directory node.

For more information about BSC, refer to *Patterns: Serial and Parallel Processes for Process Choreography and Workflow*, SG24-6306, and to *Patterns: Using Business Service Choreography In Conjunction With An Enterprise Service Bus*, REDP-3908.

3.2.4 Exposed ESB Gateway Product mapping

Figure 3-17 shows the Product mapping for the Exposed ESB Gateway runtime pattern.

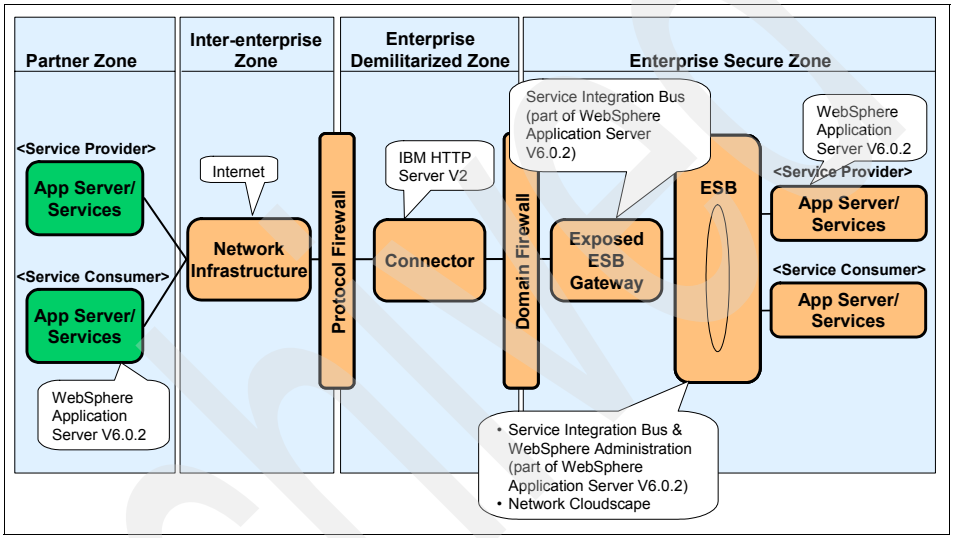

Figure 3-17 Exposed ESB Gateway::Product mappings

This scenario represents an external service consumer accessing services from an Enterprise over the Internet. The service consumer in this scenario is implemented using WebSphere Application Server V6. However, it could be implemented in any technology capable of issuing HTTPS requests. The figure also shows an external service provider that is implemented using WebSphere Application Server V6 and provides Web Services that use SOAP/HTTPS.

The requests are received by an HTTP server that is located in the DMZ, which receives all incoming requests and sends them to the Exposed ESB Gateway. The Exposed ESB Gateway verifies and maps the request to a service provided by the ESB. The ESB is implemented using WebSphere Application Server (as described in “Exposed ESB Gateway” on page 49) using a network setup for the Cloudscape database that holds the SDO repository. Finally, the internal service provider application is implemented using WebSphere Application Server.

4

Technology capabilities for an additional ESB

This chapter describes two scenarios that illustrate the technology decisions that must be made to integrate multiple Enterprise Service Bus (ESB) infrastructures in a single enterprise. The chapter focuses on:

- Minimum and extended ESB capabilities
- Business-related ESB capabilities and considerations

Following this we outline two examples for an additional ESB that lead to different technology choices:

- WebSphere Business Integration Message Broker V5
- WebSphere Application Server V6

As this redbook is about integrating ESBs, we look at the capabilities and examples for adding a new ESB infrastructure.

Note: We do not recommend creating additional ESBs in an enterprise if there is no need to do so. A single ESB eliminates the complexities of integrating additional ESBs. This chapter describes scenarios that require the integration of multiple ESBs due to business requirements.

4.1 ESB capabilities and decision attributes

The capabilities that drive the implementation of an ESB infrastructure are described in *Patterns: Implementing an SOA using an Enterprise Service Bus in WebSphere Application Server V6,* SG24-6494. These are described in terms of minimum and extended capabilities.

4.1.1 Minimum ESB capabilities

Minimum ESB capabilities are an agreed-upon list of capabilities that define an ESB. If a given infrastructure does not meet all of these requirements, it cannot be termed an ESB. Table 4-1 summarizes the minimum ESB capabilities.

Table 4-1 Minimum ESB capabilities

Category	Capabilities	Reasons
Communications	► Routing ► Addressing ► At least one messaging style (request / response, pub/sub) ► At least one transport protocol that is or can be made widely available	Provide location transparency and support service substitution
Integration	► Several integration styles or adapters ► Protocol transformation	Support integration in heterogeneous environments and support service substitution
Service interaction	► Service interface definition ► Service messaging model ► Substitution of service implementation	Support SOA principles, separating application code from specific service protocols and implementations
Management	► Administration capability	► A point of control over service addressing and naming

Communication

An ESB must supply a communication layer to support service interactions. It should support communication through a variety of protocols. It should provide

underlying support for message and event-oriented middleware and integrate with existing HTTP infrastructure and other enterprise application integration (EAI) technologies. An ESB should be able to route between all of these communication technologies through a consistent naming and administration model.

Particularly in an integrated ESB scenario, the additional ESB must be able to support service interactions provided by the original ESB over one or more available protocols.

Service interaction

An ESB must support SOA concepts for the use of interfaces and support declaration service operations and quality-of-service requirements.

An ESB should also support service messaging models consistent with those interfaces and be capable of transmitting the required interaction context, such as security, transaction, or message correlation information.

Integration

An ESB should support linking to a variety of systems that do not directly support service-style interactions so that a variety of services can be offered in a heterogeneous environment.

This includes legacy systems, packaged applications, and other EAI technologies. Integration technologies might be protocols (for example JDBC, FTP, EDI) or adapters such as the J2EE Connector Architecture resource adapters or WebSphere Business Integration Adapters. It also includes service client invocation through client APIs for various languages (Java, C+, C#) and platforms (J2EE, Microsoft .NET), CORBA, and RMI.

Management

As with any other infrastructure component an ESB must have administration capabilities to enable it to be managed and monitored and so to provide a point of control over service addressing and naming. In addition, it should be capable of integration into systems management software.

4.1.2 Extended ESB capabilities

In addition to the minimum ESB capabilities, there is a set of extended ESB capabilities. The detailed requirements of any particular scenario drive extended ESB capabilities that can then be used to select specific, appropriate products.

In particular, the following types of requirements are likely to lead to the use of more sophisticated technologies, either now or over time:

- Quality of service
- Integration
- Security
- Service level
- Modeling
- Infrastructure intelligence
- Management and autonomic

The following sections discuss these requirements in a little more detail.

Quality of service

An ESB may be required to support service interactions that require different qualities of service to protect the integrity of data mediated through those interactions. This may involve transactional support, compensation, and levels of delivery assurance. These features should be variable and driven by service interface definitions.

Integration

As additional integration capabilities could be supported, the ESB should be capable of connectivity to a wide range of different service providers, using adapters and EAI middleware. They should be capable of data enrichment to alter the service request content and destination on route, and map an incoming service request to one or more service providers.

Security

An ESB should ensure that the integrity and confidentiality of the services that they carry are maintained. They should integrate with the existing security infrastructures to address the essential security functions such as:

- Identification and authentication
- Access controls
- Confidentiality
- Data integrity
- Security management and administration
- Disaster recovery and contingency planning
- Incident reporting

Additionally the ESB should integrate with the overall management and monitoring of the security infrastructure. The ESB may provide security either directly or by integrating with other security components such as authentication, authorization, and directory components.

Service level

An ESB should mediate interactions between systems supporting specific performance, availability, and other requirements. They should offer a variety of techniques and capabilities to meet these requirements.

An ESB should provide support that enables technical and business service level agreements to be monitored and enforced.

Message processing

An ESB must be capable of integrating message, object, and data models between the application components of an SOA. It should also be able to make decisions such as routing based on content of service messages, in particular when the services are defined on an integrated ESB.

An ESB should have a mediation model that enables message processing to be customized. The model should also allow sequencing of infrastructure services (for example, security logging and monitoring) around business services invocations.

Mediations can be located close to consumers, providers, or anywhere in the ESB infrastructure transparent to consumers and providers. Mediations can also be chained. An ESB should be able to validate content and format.

Modeling

An ESB should support the increasing array of cross-industry and vertical standards in both the XML and Web services spaces.

It should support custom message and data models. It should also support the use of development tooling and be capable of identifying different models for internal and external services and processes.

Infrastructure intelligence

An ESB should be capable of evolving toward a more autonomic, on demand infrastructure. It should allow business rules and policies to affect ESB function, and it should support pattern recognition.

Management and autonomic

In addition to basic management capabilities the ESB should also support the migration to autonomic and On Demand infrastructure by supporting metering and billing, self-healing, and dynamic routing, and react to events to self-configure, heal, and optimize.

4.1.3 Softer attributes

The minimum and extended ESB capabilities enable the making of an informed decision for adding an additional Enterprise Service Bus to an existing ESB infrastructure, and the technology to use. However, the decision criteria for this technology should not be restricted to these minimum and extended capabilities. In many situations there will be a list of “softer” attributes that will shape the decision, and these are shown in Table 4-2.

Table 4-2 Softer attributes for an additional ESB

Attribute	Description
Existing ESB technology	What ESB technology is deployed today?
Maturity of existing ESB implementation	► How long has the existing ESB been deployed? ► How much investment has been made in its overall capability? ► How well is the ESB delivering its non-functional attributes, for example: – Performance – Reliability – Serviceability
ESB strategy	What is the strategy for the following ESB attributes: ► Single administration ► Single namespace / naming ► Single security ► Governance
Capabilities of existing ESB	How well does the existing ESB implement the minimum (and extended) ESB capabilities?
ESB technology allegiance	Are there any historical or commercial allegiances to a specific ESB technology?
Enterprise integration strategy	What is the overall integration strategy within the enterprise? ► Single / dual vendor ► Analyst ratings
Programming model	What strategic programming models and tools are used in the enterprise?
Hardware and operating system	► What is the current ESB deployed on? ► What is the enterprise strategy for the hardware and operating system?

The following section describes in more detail the additional softer attributes introduced in Table 4-2 that must be considered for adding an additional ESB to an existing infrastructure.

Existing ESB technology

It is important to understand which products and version numbers are used to implement the existing ESB infrastructure. This is not an actual attribute that will affect the decision for the additional ESB technology. However, no decision can be made without this fundamental piece of information as it may be required in further commercial discussions with its vendor and for understanding its minimum and extended capabilities. This is very important and should not be overlooked because “version n+1” of the existing ESB may have additional capabilities when compared with “version n”.

Maturity of existing ESB implementation

The enterprise may have implemented the latest and greatest version of an ESB but how long has it been in production and how is it performing in terms of functional and non-functional requirements? Understanding this may have a bearing on whether an additional ESB is implemented or whether the existing infrastructure is extended or replaced.

Established ESB example

Many enterprises will have already deployed an environment that displays all of the minimum requirements for an existing ESB (for example, using WebSphere Business Integration Message Broker).

The existing environment might have been deployed for a number of years and had a considerable investment in its overall capability within the enterprise. If we assume that it is delivering satisfactory service then it is reasonable to conclude that this ESB will be retained and the additional ESB will have to integrate with it.

Non-functioning ESB example

We can take an alternative view where the existing ESB displays the following characteristics;

- Has only been deployed for a relatively short period of time
- Has a small number of providers and consumers
- Is delivering a marginal level of service

This example may guide us down a number of different paths:

- Making additional investment in the existing ESB to bring its level of service and capability up to the required level. And then:
  - Extending it to include the requirement for the additional ESB. Therefore, no new, additional ESB is required at all.
  - Adding the additional ESB to it for reasons of governance or any other capability reason as discussed in 4.1, “ESB capabilities and decision attributes” on page 58.

- Replacing the existing ESB with the new ESB technology so that it consumes the capabilities of the existing ESB. The existing ESB is removed from the enterprise infrastructure.

Capabilities of existing ESB

How well does the existing ESB implement the minimum and extended ESB capabilities?

- If the existing ESB implements the minimum capabilities for an ESB and potentially some of the extended capabilities then it is likely that this ESB will be retained and the new ESB added alongside.
- However, if the existing ESB does not provide capabilities beyond the minimum ESB capabilities it may be reasonable to choose a new, additional ESB that has strong ESB capabilities and to integrate the two ESBs together using one of the patterns described in this book. This is a realistic situation as many enterprises may have implemented ESB-style capabilities using older technologies that have little investment, and which therefore are unable to grow to meet the requirements of a fully-fledged ESB.

ESB technology allegiance

Many enterprises have an allegiance to a particular vendor or technology. Irrespective of the merits of a particular technology solution for the additional ESB, the historical or commercial allegiance to the existing ESB vendor may be so strong that the decision might have little regard for the strength of other technologies.

Enterprise integration strategy

The enterprise integration strategy for the organization could have a considerable bearing on the selection of the additional ESB technology. For example:

- Some enterprises are moving to policies where they are reducing the number of core IT vendors.
- Others are continuing on a best-of-breed IT selection policy.
- Finally, some enterprises have a policy for building middleware solutions versus buying Commercial Off The Shelf (COTS) software, commonly known as “build versus buy.”

Therefore, the enterprise integration strategy could dictate what type of additional ESB technology is chosen and implemented, irrespective of the capabilities and wider decision criteria.

Programming model

The programming model and development tools used by an enterprise could also have a strong bearing on the implementation choice made for an additional ESB within the enterprise.

For example, a J2EE-centric organization might lean more toward making a WebSphere Application Server service integration bus decision because of the similarities of the programming model for application and mediation development. Whereas organizations using a longer-established programming model, for example using COBOL as the programming model and its associated programming model, might decide that WebSphere Business Integration Message Broker has a tighter fit to their programming practices.

Additionally, an organization geared toward Web services might choose to implement their additional ESB using WebSphere Application Server because of the associated tooling capabilities of Rational Application Developer. In particular they may use the Rational Application Developer wizards to build Web services components from existing J2EE components and vice-versa.

Hardware and operating system

We must not forget that the underlying hardware and operating system infrastructure could have a bearing on the additional ESB decision. For example, the enterprise may have a strategy for deploying new infrastructure deployments on Linux®, or more specifically on particular versions of Linux. These prerequisite statements might preclude specific additional ESB technologies.

4.2 A review of ESB technologies

This section provides a short review of various ESB topics that will be used when discussing the three additional ESB examples later in this chapter.

4.2.1 WebSphere Integration Reference Architecture

The WebSphere Integration Reference Architecture provides a comprehensive set of services to enable business integration. The services provide the breadth of functionality needed to solve integration requirements. More important, the component services can be implemented in stages to allow incremental evolution on a project-by-project basis while working toward an enterprise integration solution architecture. Although specific projects may not require all of these services, enterprise-level integration will require the ability to add these functional capabilities to the integration architecture. The resultant architecture provides for separation of concerns by allowing business logic, control logic, routing, and transformation logic, to be loosely coupled and, as a result, more

flexible to change. At the organizational level, this approach facilitates simpler integration solution development and enhances maintainability and operation of the solution.

The WebSphere Integration Reference Architecture shows the key integration functions that are required for comprehensive, enterprise-level solutions. These service groupings provide the ability to apply separation of concerns to enterprise integration requirements and lead to a convergence with the principles of SOA as they apply to integration.

At the core of the WebSphere Integration Reference Architecture is the Connectivity Services layer. This component provides the infrastructure to support and instantiate the Enterprise Service Bus (ESB) architectural pattern and is shown in Figure 4-1.

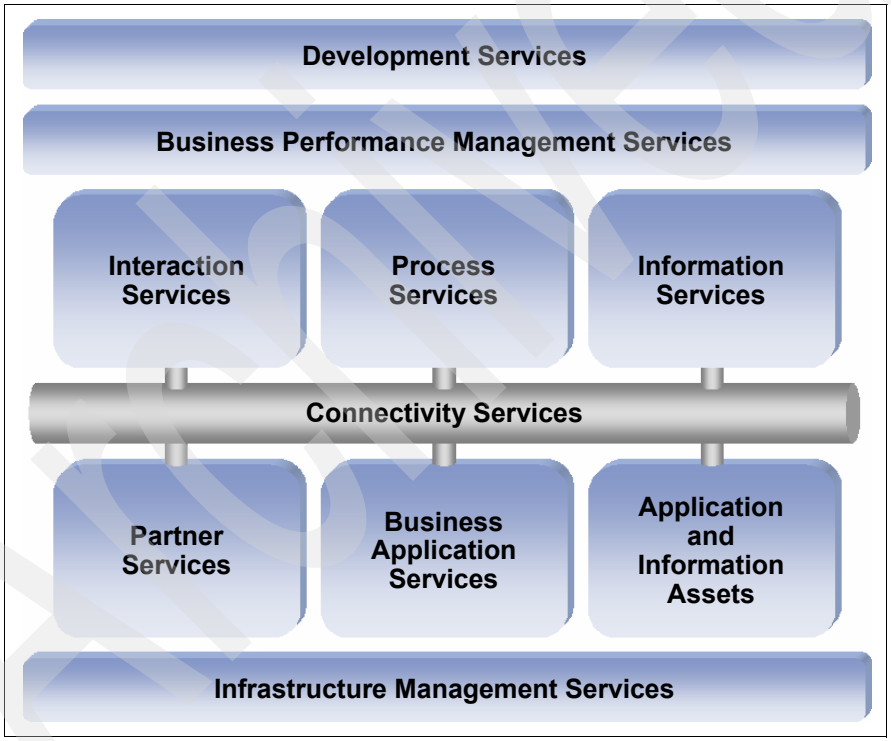

Figure 4-1 WebSphere Integration Reference Architecture

The ESB architectural pattern within the WebSphere Integration Reference Architecture can be implemented using the service integration bus component of WebSphere Application Server or with WebSphere Business Integration Message Broker. There are additional product options within the Connectivity Services layer but these are outside the scope of this book.

The IBM Enterprise Service Bus strategy:

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

- WebSphere Enterprise Service Bus V6

  Delivers an ESB with Web services connectivity and data transformation.

- WebSphere Message Broker V6

  Delivers an advanced ESB with universal connectivity and data transformation.

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

For more information about the IBM ESB strategy see:

http://www.ibm.com/software/info1/websphere/index.jsp?tab=landings/esb

4.2.2 General capability discussion

The following section provides a general discussion and background information for some of the topics that will be discussed in the two examples later in this chapter.

WebSphere MQ and JMS

WebSphere Application Server provides support for the JMS API and for using WebSphere MQ as the JMS provider. JMS defines a message format that JMS providers must support. Many JMS providers (such as WebSphere MQ) that were designed before JMS was finalized have to provide a mapping between MQ and JMS message formats. Within WebSphere MQ the MQRFH2 is optional and is omitted in many native WebSphere MQ applications, but for JMS messages it carries JMS-specific information. Therefore, it should always be included in a WebSphere MQ message when the sender knows that the receiving destination is a JMS application. Figure 4-2 on page 68 shows how the structure of a JMS message is transformed to a WebSphere MQ message and back again.

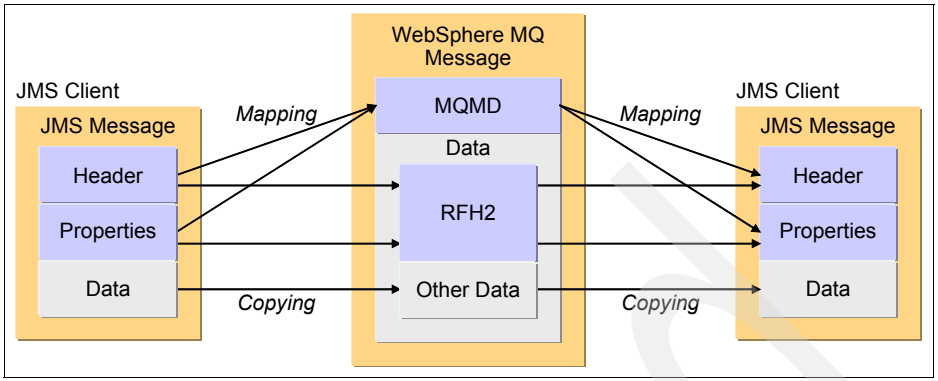

Figure 4-2 Mapping of WebSphere MQ and JMS message formats

SOAP/JMS support

When sending or receiving a JMS message as part of a Web services interaction using SOAP/JMS and using WebSphere Application Server V6 as the consumer or producer, additional properties are required. Chapter 10, “Directly Connected heterogeneous ESBs” on page 241 discusses a suggested mapping for SOAP/JMS messages flowing between WebSphere Application Server V6 and WebSphere Business Integration Message Broker. Knowing and understanding this mapping is important when deciding on an additional ESB and in the architectural design of the solution.

Message persistence levels

The WebSphere Application Server V6 messaging engine is a new server component that provides the core messaging functionality of a WebSphere Application Server service integration bus. As such it is the native JMS provider for WebSphere Application Server V6. A messaging engine manages bus resources and provides a connection point for applications. It provides additional persistence levels from the defined JMS delivery modes and from WebSphere MQ persistence levels. These persistence levels are shown in Table 4-3.

Table 4-3 Persistence level mapping

WebSphere Application Server messaging engine reliability level	JMS delivery mode	WebSphere MQ persistence level
Best effort nonpersistent	Nonpersistent	NonPersistent
Express nonpersistent	Nonpersistent	NonPersistent
Reliable nonpersistent	Nonpersistent	NonPersistent

WebSphere Application Server messaging engine reliability level	JMS delivery mode	WebSphere MQ persistence level
Reliable persistent	Persistent	NonPersistent
Assured persistent	Persistent	Persistent

Linking the service integration bus to a WebSphere MQ network

The new service integration bus component in WebSphere Application Server has been designed to allow extension to:

- Other service integration buses
- A WebSphere MQ infrastructure

The purpose of this topic is to discuss the integration between the service integration bus component of WebSphere Application Server and WebSphere MQ since these will be frequent candidates for ESBs that have to be integrated.

Using the WebSphere MQ link within the service integration bus of WebSphere Application Server you can exchange point-to-point messages, and use publish and subscribe between WebSphere Application Server and a WebSphere MQ network by viewing the WebSphere MQ network as a *foreign bus*.

The WebSphere MQ link is defined on a messaging engine in a service integration bus. It makes exchanging messages very simple by automatically converting them so their characteristics are retained or mapped to similar settings.

A WebSphere MQ link engine is the messaging engine through which other messaging engines on the same bus send messages to, and receive messages from, a WebSphere MQ queue manager. This queue manager acts as a gateway to other WebSphere MQ queue managers. This gateway queue manager is represented as a foreign bus when you configure the WebSphere MQ link.

Figure 4-3 on page 70 shows how messages exchanged between the messaging engine with the WebSphere MQ link and the gateway queue manager can be sent and received by other messaging engines on the same bus, and other queues on the gateway queue manager.

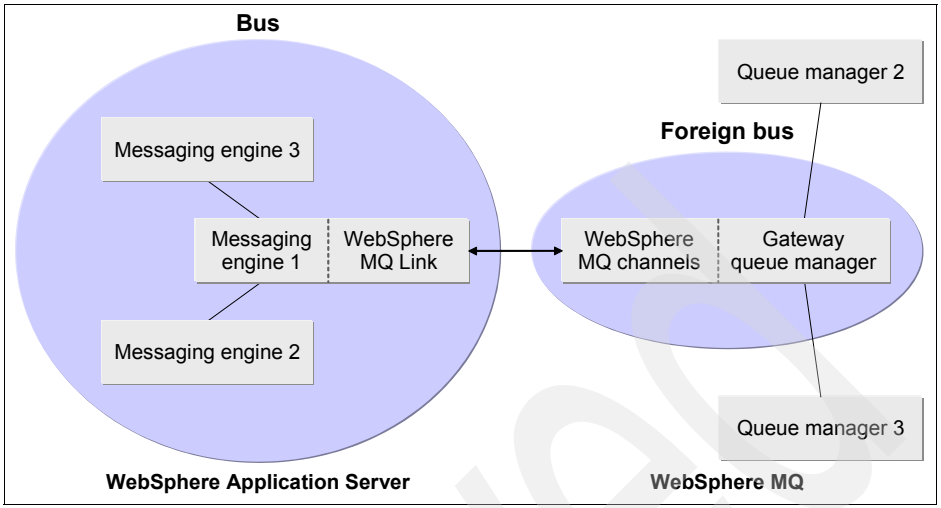

Figure 4-3 Communication between the service integration bus and WebSphere MQ

4.3 Examples of adding new ESB technology to an existing ESB infrastructure

The following two sections each give an example of a situation where an enterprise is looking to add an additional ESB to an existing ESB infrastructure. There is no exact rule or matrix that can be applied to the decision-making process about which technology to use for the additional ESB. Each scenario will lead the reader to a different conclusion. However, the only precise rule that can be applied is that where there is the possibility to use only one ESB then that course of action should be followed, unless there is very good reason not to.

The two scenarios discussed in this section are:

- "Scenario 1: Adding ESB capabilities to a WebSphere MQ infrastructure" on page 71
- "Scenario 2: Integrating ESBs in a J2EE and Web services-based infrastructure" on page 76

The assessment of the options for implementation of the new ESB will be described using a combination of a subset of capabilities described in 4.1.1, "Minimum ESB capabilities" on page 58 and 4.1.2, "Extended ESB capabilities" on page 59.

The options examined for the two scenarios include:

- Integration based on WebSphere Application Server V6 service integration bus, potentially with adapter technology to integrate with the existing ESB.
- Integration based on WebSphere Business Integration Message Broker V5, potentially with adapter technology to integrate with the existing ESB.

4.3.1 Scenario 1: Adding ESB capabilities to a WebSphere MQ infrastructure

This example is based on WebSphere MQ technologies, where an MQ-based service bus is currently in place. This service bus meets most, but not all, of the minimum ESB capabilities so it cannot be termed an ESB. We examine the options for adding an ESB to integrate with the existing MQ-based service bus.

Example overview

In this example there is an existing service bus through which legacy services are presented to an existing suite of applications. The data standards used in this service bus are consistent across all of the applications, and the transport used is MQ. The bus is implemented as custom, MQ-enabled infrastructure software written in-house. It enables applications to submit messages to the bus and to have these routed to the correct service based on the submission parameters and a custom message header.

The existing service bus is mature and reliable and has high performance. There is no desire currently to do a wholesale replacement of the existing bus but the bus is not flexible enough to offer support for the enterprise architecture under which many new applications supporting open standards will be integrated.

Proposed change and high-level requirements

The service oriented approach has worked well in this enterprise and a decision has therefore been made to take this further and to implement a new ESB to support future integration capabilities.

This ESB must:

- Be capable of supporting the integration of new off-the-shelf packages and existing applications.
- Support the presentation of a subset of services from the existing service bus to these new packages.
- Support a full SOA.
- Include support for open standards, in order to provide an integration environment to meet current and future requirements.

Communication

The solution must support integration of the WebSphere MQ based existing bus. Although the service integration bus in WebSphere Application Server V6 could enable the integration between itself and the existing WebSphere MQ bus (see “Linking the service integration bus to a WebSphere MQ network” on page 69), integration with WebSphere Business Integration Message Broker is thought to provide additional benefits in terms of skills and programming paradigm for this particular enterprise.

Both WebSphere Application Server and WebSphere Business Integration Message Broker provide support for HTTP. WebSphere Application Server has full support for HTTPS, but WebSphere Business Integration Message Broker has support only for messages coming into the bus as requests but not for outgoing calls. In some contexts this could be an issue but the deployment strategy in this enterprise places both ESB components and provider components in a data center with secured network communications. This deployment strategy eliminates the need for HTTPS to secure HTTP requests from the bus out to service providers and this is not therefore a deciding factor.

Routing and addressing in WebSphere Business Integration Message Broker enables content-based lookup of routing information to support a number of routing patterns, including the routing of messages to one or more target destinations that were not specified by the sending application. Mediations can be implemented in the service integration bus in WebSphere Application Server V6 to carry out similar functionality. Development of mediations requires Java implementation skills, which although not an issue in the general case would require new skills in our example IT department.

The new ESB will be required to support the interaction styles of the existing service bus and to extend these to further messaging-based interactions. WebSphere Business Integration Message Broker is able to do this using MQ and JMS protocols and can maintain full transactionality as requests pass through the ESB to provide assured delivery and transactionality across message retrieval and any associated updates executed by the service.

Integration

Both WebSphere Application Server and WebSphere Business Integration Message Broker have strong integration capabilities and support both database integration and connection to legacy systems. In both cases adapters normally are required for such connections, and these are available. In the case of WebSphere Business Integration Message Broker the use of WebSphere Business Integration adapters will provide this functionality.

WebSphere Business Integration Message Broker provides extensive data enrichment support augmenting messages by adding data from an external data

source, and distributing messages to multiple target destinations. All of this can be achieved by setting configuration options without needing to write code. WebSphere Application Server V6 can also provide data enrichment but this requires mediations to be implemented as code.

Security

The WebSphere MQ network supporting the existing service bus has been fully secured. There are no remote WebSphere MQ clients, and all WebSphere MQ servers have authorization set on all queues. Governance is already in place within the organization to securely manage the creation of new messaging resources, to apply the appropriate security controls, and to monitor any attempted breaches. This security model would extend naturally to a new ESB implemented in WebSphere Business Integration Message Broker.

WebSphere Application Server V6 supports all mandatory elements of security over both HTTPS and JMS transports. This would support all the necessary security requirements but would need its security infrastructure to be integrated with the existing WebSphere MQ–based model.

Modeling

Modeling is not seen as a significant deciding factor in the current scenario. The new ESB is not intended to support process management, and the modeling of data flowing through a WebSphere Business Integration Message Broker message flow is deemed adequate for requirements.

Both products could support the integration of additional functionality for process management and workflow equally well. However, as it is not relevant to the example, it is not considered here.

Service interaction

Both WebSphere Application Server and WebSphere Business Integration Message Broker provide support for SOA principles and separation of application code from specific service protocols and implementations.

The two technologies place a different emphasis on how the integrity of business transactions and data are maintained, with WebSphere Business Integration Message Broker tending to make heavy use of reliable messaging concepts while WebSphere Application Server will be more likely to use coordinated transactions where possible or use compensation techniques.

In this example, the skills profile of the integration team is based historically on WebSphere MQ messaging and the styles of interaction that come with that background. They would take full advantage of the opportunities for asynchronous interactions and have the design skills to move to a

message-based *broker* style of solution. This would be seen as an extension rather than denial of the achievements of the existing bus, and there would be a determination to make the solution work, which is a very strong factor in achieving ultimate success.

Service level

With regard to this capability, we will consider performance and availability characteristics.

Performance

Performance characteristics of the WebSphere MQ system are well known within the organization; they fully understand the use of WebSphere MQ clusters for workload balancing and believe that it will meet requirements. There is confidence that the extrapolation of these figures based on WebSphere MQ with their potential use of WebSphere Business Integration Message Broker would give the required performance on the new ESB.

Performance of the service integration bus using WebSphere Application Server is also well documented. For example, benchmark specifications comparing WebSphere Application Server V6 to other application servers can be found at:

`http://www.spec.org/jAppServer2004/results/jAppServer2004.html`

Therefore in performance terms, the organization would be happy with either solution.

Availability

The assessment process examined the requirements for availability and failover. Both WebSphere Application Server and WebSphere Business Integration Message Broker solutions were deemed to fully support the requirements. Given the capabilities of both of these solutions, availability is unlikely to be a deciding factor in this or other similar decision processes.

Other factors

There are often additional factors not directly related to the functional capabilities of the bus as discussed in 4.1.3, “Softer attributes” on page 62, and this is true within this enterprise. Although several options would meet the requirement, these additional factors can be as important as technological capabilities in influencing the decision for an additional ESB.

One of the key factors that would normally be a very positive factor leading to the selection of WebSphere Application Server rather than WebSphere Business Integration Message Broker is relevant only in situations where services are to be *run* on the same infrastructure as the ESB is deployed. WebSphere Application Server V6, as its name suggests, can perform the role of a highly capable ESB,

and can also host many of the provider services themselves. The development of services from WSDL, or the encapsulation of existing J2EE functionality as Web services, has significant support in WebSphere Application Server V6,

In this example the services to be provided are mainly made available by packaged software or existing service-based in-house systems, and there has been no decision to move to a J2EE platform for further service development. Therefore, one of the key benefits of a solution based on WebSphere Application Server service integration bus is not relevant for this organization.

The enterprise architecture for this example does not include process management as part of a separate layer that cooperates with the ESB. In the future this may be a separate layer that sits above the ESB. In this longer-term vision, the ESB would invoke the services that are choreographed by the process management engine. The WebSphere Application Server runtime engine and the process engine provided by WebSphere Business Integration Server Foundation share many J2EE components, administration, and programming model and these similarities could sway a decision toward WebSphere Application Server. However, process management is not part of the major requirements for this organization.

Summary

The diagram in Figure 4-4 summarizes the high-level solution that this organization decided to deploy.

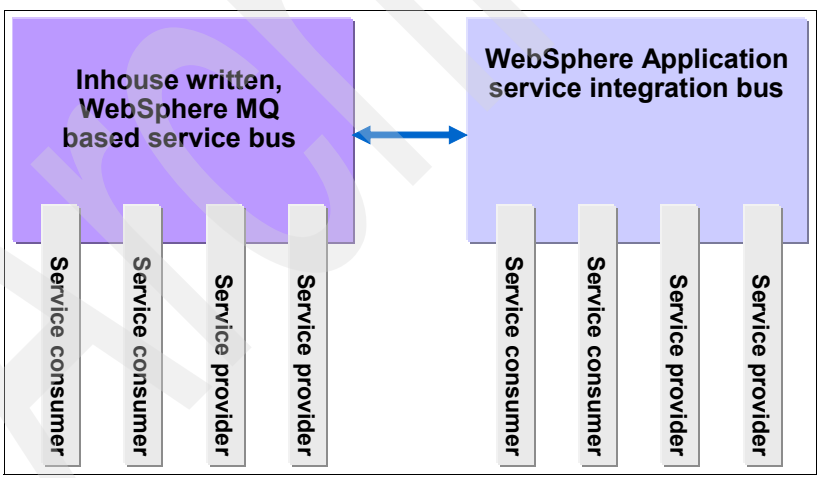

Figure 4-4 Adding ESB capabilities in WebSphere MQ scenarios

Note: Figure 4-4 does not make any reference to the ESB integration patterns made in this book. The Runtime pattern deployed would require much further analysis of the multiple ESB requirements for this organization.

4.3.2 Scenario 2: Integrating ESBs in a J2EE and Web services-based infrastructure

In this example a large organization wishes to move toward a single, standards-based ESB but recognizes that with current governance structures it will not be possible to enforce a single integration implementation across all divisions. They have therefore decided to allow different organizational units within the enterprise to own the implementation of their own autonomous ESB, and to provide a standards-based framework to ensure that the ESBs can be integrated to share services as necessary across the whole enterprise.

This enterprise has a strategy for implementing new internal and Web channel applications on J2EE technology. They have a large, existing portfolio of J2EE applications deployed on a variety of J2EE application servers in different organizational units. WebSphere Application Server is the predominant J2EE application server used, and they have recently deployed an internal portal as a proof of concept using WebSphere Portal Server.

There has been an increasing movement toward SOA over the past few years. They see investing in reusable, composable services as a strategic asset that will provide for greater flexibility and success in their marketplace. Because of the movement toward open standards and with their investment in J2EE and WSDL, they are keen to implement any business process management solution using a standards-based process management engine. The process management engine is likely to be compliant with Business Process Execution Language (WS-BPEL).

This organization does have some WebSphere MQ infrastructure, although it is currently used only for point-to-point communications. However, its implementation has seen performance, availability, ease-of-use, security, and other non-functional aspects meeting their requirements.

Proposed change and high-level requirements

The organization wishes to integrate within and between divisions of the same company using an SOA based on Web services standards.

Each division in implementing this architecture must aim for an ESB that supports not only their own legacy integration requirements but also integration with the ESBs of other divisions. A set of standards that must be supported will be defined centrally by the IT strategy group.

The initial standards for inter-ESB communications include:

- HTTPS
- SOAP

- WSDL
- Support for a central security model based on WS-Security
  - X509 certificates
  - Optionally signed messages
  - Optionally encrypted messages

Services to be integrated will include those offered by existing and planned software packages, and new services to be developed using J2EE.

Communication

The solution must support service requests over HTTPS and must support HTTPS between ESBs. WebSphere Business Integration Message Broker currently only supports HTTPS on incoming requests. Implementation of an incoming HTTPS solution for WebSphere Business Integration Message Broker would either require extra development effort or additional infrastructure. WebSphere Application Server does support protocol natively.

This enterprise is very happy with the style of communication typically used in Web services. They can see that a WebSphere Business Integration Message Broker solution built on WebSphere MQ would give them an assured delivery-based ESB infrastructure. They are looking toward emerging standards, in particular WS-Transaction and WS-Policy, to give them additional standards-based capabilities in the future. They will provide transactionality within their services and hope to extend this to external providers using J2EE Connector Architecture resource adapters. Therefore, this organization has decided that it will handle transactional requirements at a business application level until the new standards emerge.

JMS will also be used by the organization. Both WebSphere Application Server and WebSphere Business Integration Message Broker support this transport. It is therefore not a deciding factor in this example.

Routing and addressing in WebSphere Business Integration Message Broker enables content-based lookup of routing information to support a number of routing patterns, including the routing of messages to one or more target destinations that were not specified by the sending application. There will be some requirement for this functionality in mediating between the ESB implementations. This requires development of mediations in Java if WebSphere Application Server is selected. This is not a problem for the organization as Java is one of their core skill sets.

Integration

Both WebSphere Application Server and WebSphere Business Integration Message Broker have strong integration capabilities and support both database integration and connection to legacy.

However this organization wishes to use the J2EE Connector Architecture as the framework standard as they have established that J2EE Connector Architecture resource adapters exist for a number of their existing packages. Both WebSphere Application Server and WebSphere Business Integration Message Broker provide support for these resource adapters.

Both WebSphere Application Server V6 and WebSphere Business Integration Message Broker can transform between the JMS and HTTP protocols, which have been mandated as standard.

Security

WebSphere Business Integration Message Broker has no in-built support for the WS-Security features that are mandated as part of the longer-term security standards for the organization. Plug-in nodes could be developed within WebSphere Business Integration Message Broker to meet such requirements but there are no current skills with this organization to perform this task.

WebSphere Application Server already implements the mandatory parts of the WS-Security standards and can be expected to continue tracking such standards. Therefore, WebSphere Application Server fits the security model of this organization.

Modeling

Modeling is not seen as a significant deciding factor in the first instance. However support for process management is a future requirement. The capabilities of WebSphere Business Integration Server Foundation V5.1 and its use of WebSphere Application Server as its cornerstone technology have been noted.

Service interaction

Both WebSphere Application Server and WebSphere Business Integration Message Broker provide support for SOA principles and separation of application code from specific service protocols and implementations.

The two technologies place a different emphasis on how the integrity of business transactions and data are maintained, with WebSphere Business Integration Message Broker tending to make heavy use of reliable messaging. WebSphere Application Server V6 is more likely to use coordinated transactions where possible or compensation techniques.

Service level

With regard to this capability, we consider performance and availability characteristics.

Performance

Both products are assumed to meet known requirements based on published data. This is not a deciding factor.

Availability

The assessment process examined the requirements for availability and failover against known requirements and both WebSphere Application Server and WebSphere Business Integration Message Broker solutions were deemed to fully support the requirements. WebSphere Application Server V6 Network Deployment offered all of the capabilities they needed.

Other factors

One of the key factors which is very positive, leading to the selection of WebSphere Application Server V6 is the development support available, provided by Rational Application Developer V6, which provides support for:

- Building Web services using a bottom-up approach (from a J2EE implementation artifact to exposing a WSDL-defined Web service)
- Building Web services using a top-down approach (from WSDL to service or client skeletons).
- Building mediations for the WebSphere Application Server V6 service integration bus.

The enterprise has J2EE skills and sees the advantage of utilizing the same technology for all of the ESBs and for the services connected to them.

Summary

The diagram in Figure 4-5 on page 80 summarizes the high-level solution that this organization decided to deploy.

Note: Figure 4-5 does not make any reference to the ESB integration patterns made in this book. The Runtime pattern deployed would require much further analysis of the multiple ESB requirements for this organization.

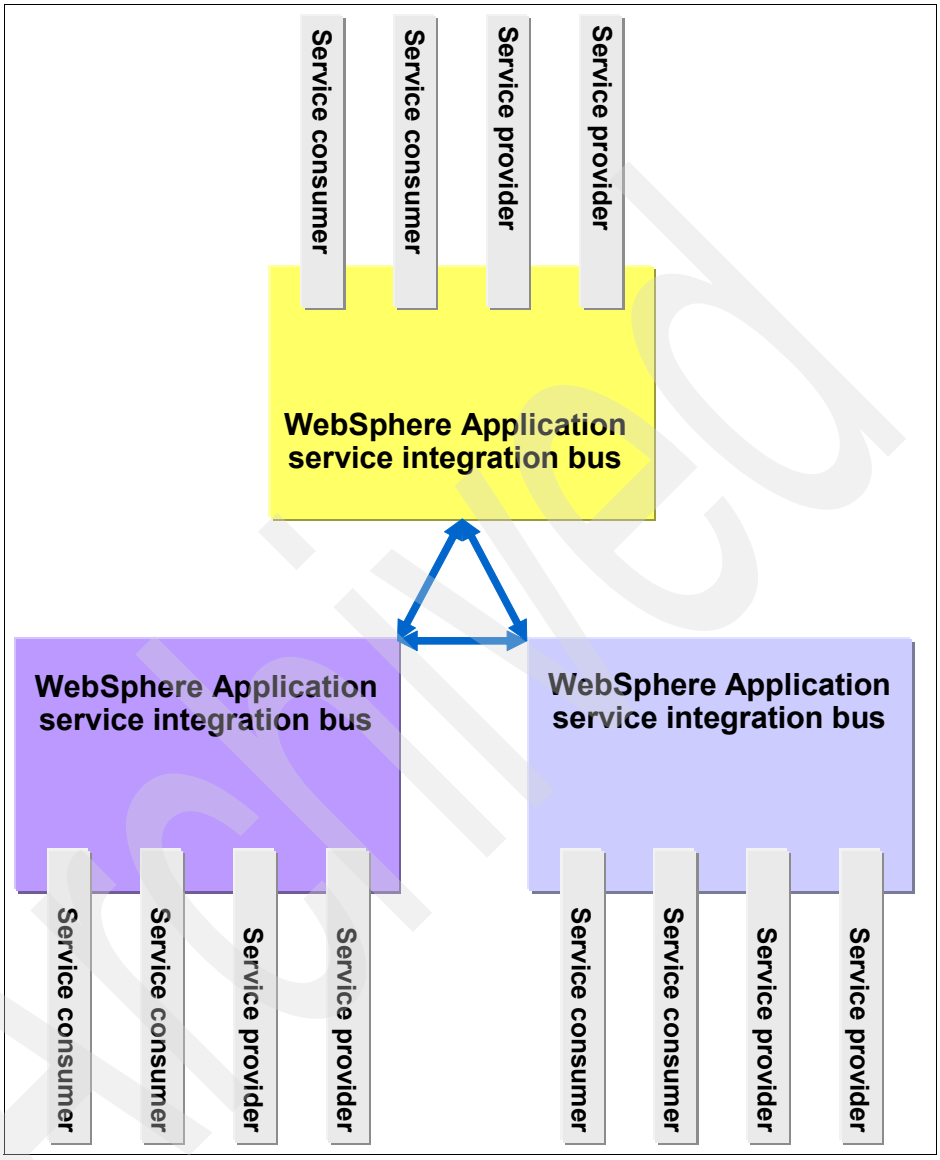

Figure 4-5 Integrating ESBs in a J2EE and Web services-based infrastructure

5

To ESB but not two ESB?

The best advice on the subject of having more than one ESB is *don't do it*. In most cases (but not all cases) the best strategy and best practice is to have only one ESB in an enterprise. The intent of the ESB is to facilitate integration across the entire enterprise, hence the name *Enterprise* Service Bus. One ESB would eliminate the need for directly connected ESBs, federated ESBs, or for any ESB integration at all.

If the recommendation is a single enterprise ESB, why would an enterprise have more than one ESB? There are several reasons, technical and organizational, which we discuss in this chapter. All reasons to have more than one ESB are tactical because the strategic direction is always a single ESB.

5.1 Tactical reasons for multiple ESBs

Reasons for implementing multiple ESB implementations in a single organization include:

- Multiple governance bodies
- Funding models
- Alignment by organizational unit
- Geography
- Business strategy
- Multiple ESB technologies

5.1.1 Multiple governance bodies

Multiple enterprise governance bodies can (and often do) result in multiple ESBs. It is often politically easier to implement multiple ESBs that align with the multiple governance bodies than to design and implement a common solution. Each governance body will define the boundary of its ESB and use the techniques in this book to integrate them.

In the Patterns for e-business terminology, these governance boundaries are called *zones*, and they define the scope of control over architecture and implementation. Every component in a zone, ESB or application, is under the same governance body.

This raises the next logical question “why more than one governance body?” and there are many answers.

Multiple governance bodies can be the result of growth through mergers and acquisitions. This may be temporary during a transition period or it may be a permanent choice. It is not uncommon to even find multiple CIO and CTO positions within an enterprise that has grown this way.

Some enterprises use a franchise or co-op business model. These enterprises are very likely to have multiple governance bodies. The franchisees remain autonomous and yet must exchange data with the corporate entity. Each has independent IT organizations but they may share common infrastructure such as wide area networks or an entire data center.

Government regulatory requirements may force an enterprise to have multiple governance bodies. An enterprise that deals with both civilian and military customers may be required to maintain mandatory separation between the two parts of the business for security reasons.

5.1.2 Funding models

How an enterprise funds projects can lead to multiple ESBs. If the enterprise does not manage the funding and governance from a central point the ESB implementations will be fragmented and disjointed.

If the technology funding is at the project level then the project team may design and deploy an ESB as part of that project's funding. The project may be something as large as a new ERP system with dozens of endpoints and several hundred interfaces or as small as a single line of business application with just a handful of endpoints. The ESB boundary is synonymous with the project boundary.

This is not an optimal way to fund integration projects nor to design and deploy an ESB. This will lead to a proliferation of ESBs that are tailored to the specific needs of an individual project. Enterprise Integration, which includes ESBs, is best funded at the enterprise level.

5.1.3 Alignment by organizational unit

Enterprises are often organized by brands or lines of business and even combinations of these. In some cases there is a central governance body but in many others the IT governance follows the organizational alignment of the enterprise.

Multiple organizational units in the same enterprise can have unique integration requirements. A highly diverse enterprise is likely to have highly diverse integration requirements. The unique integration needs may be based on the diversity of products and services they deliver to their customers. An enterprise that offers both manufactured goods and business financing may want to implement multiple ESBs based on the unique requirements of each business unit.

There can be differences in the form of government regulations that exist in one region but not another. For example, security and privacy laws differ from nation to nation and even between regions in the same nation. There may be multiple ESBs to ensure compliance with these regulations.

Government regulations may affect one enterprise organization unit but not another. An example is if an enterprise is engaged in healthcare but has a wholly owned medical device manufacturing subsidiary. In the USA, the ESB for medical device manufacturing would require Food and Drug Administration 21 CFR Part 11 certification. The initial and ongoing costs of certifying the ESB components needed for the healthcare organizational unit would not have a business benefit.

As was said before, it is always best to have a single ESB, but the time and cost to build in the flexibility to accommodate all possible requirements for all

organizational units may make the business case for the ESB more difficult to justify.

5.1.4 Geography

Geography can influence the decision to have multiple ESBs. It may be impractical to manage a single ESB across geographic boundaries. This is especially true when there is low bandwidth or unreliable communications between geographies. It can make more sense to manage them separately and use the techniques in this book to create links between them.

The architecture of an ESB supports components distributed across geographies. The problem arises with system management capabilities across geographies. Low bandwidth and intermittent communications can be challenges to managing a single global ESB.

5.1.5 Business strategy

The architecture of the ESB should be heavily influenced by the guiding principles of the business. Different parts of the business may adopt different guiding principles, which would lead to different architectural decisions for the ESB.

For example, one business unit may be in a fast-moving but high-margin industry where the agility to adapt is more important than cost. Another business unit in the same enterprise may be in a commodity business with low margins and very stable processes. In the former a highly flexible but more costly solution would be more desirable. In the latter the lowest integration cost would be more desirable than a highly flexible but more expensive solution.

ESB architecture is a series of trade-offs between cost, schedule, and quality. If all business units can agree to a common set of trade-offs then a single ESB is possible. If consensus is not possible the ESB architecture is likely to mimic the lack of consensus.

5.1.6 Multiple ESB technologies

The technology in use by one vendor to implement an ESB may not be interoperable with the technology from other vendors. An enterprise may not have the ability or desire to choose a single vendor technology platform to implement the ESB architecture. There are a number of reasons why an enterprise may have multiple ESB technologies in the same governance zone.

Many application packages are bundled with ESB technology. Trying to remove and reimplement the bundled solution would be costly and is not likely to have a

very good business case. In these cases an enterprise would end up with multiple ESB technologies that may not be interoperable.

Some enterprises do not wish to be overly dependent on a single vendor's technology. In these cases multiple ESB technologies will be intentionally introduced. In these cases it would be wise to ensure that the selected technologies are interoperable to ease integration.

5.2 Conclusion

Strategically it is always best to have a single ESB but there are tactical reasons that may conflict with this strategic objective. Each enterprise must look at the best combination of strategic and tactical issues to determine the right number of ESBs. This should be examined on a periodic basis to determine whether the original decision remains the best one. If you require more than one, the remainder of this book will help you make the best of that decision.

6

Integrating ESBs

The use of an Enterprise Service Bus architecture is becoming common in enterprises worldwide. The need to integrate multiple ESBs within the same enterprise is relatively new. This chapter is about some of the emerging patterns being used to integrate ESBs.

Integrating two or more ESBs is subject to some of the same integration patterns as connecting two or more applications; integration déjà vu. It seems logical that integrating ESBs should follow best practices similar to those for enterprise application integration (EAI). This chapter outlines where the Patterns for e-business Application Integration patterns are applied to integrating ESBs in addition to adding some new patterns.

Service-oriented architecture (SOA) best practices can also be applied to integration. The use of SOA techniques for integration is called *Service Oriented Integration* (SOI). We will discuss some emerging SOI patterns you can use to integrate ESBs.

6.1 ESB capabilities

An ESB delivers a minimum set of capabilities (defined in 4.1.1, “Minimum ESB capabilities” on page 58) and also can deliver a set of extended capabilities (defined in 4.1.2, “Extended ESB capabilities” on page 59). Examples of the capabilities are:

- Communications: routing, addressing, protocol conversion
- Security: authentication, authorization, non-repudiation
- Message Processing: transformation, mediation, validation
- Quality of Service: transaction, delivery assurance
- Management: logging, metering, monitoring, error handling, metadata

Each ESB can implement these capabilities in different ways. Integrating two ESBs is more about integrating these capabilities than about merely delivering the service request and response. It is important to present a consistent model of interaction between the service providers and service consumers on all of the ESBs involved.

It will be common to coordinate several ESB capabilities for a single inter-ESB exchange. Different inter-ESB exchanges may require coordination of different combinations of ESB capabilities. Coordinating the capabilities between different ESB implementations is where the complexity begins to surface. One exchange may require coordination of transaction, authentication, transformation, and error-handling capabilities. The next exchange may require delivery assurance and non-repudiation.

There are many differences between integration of ESBs and integration of applications but there are many similarities. Integration of ESB capabilities share many best practices with application integration, such as loose coupling, abstraction, and encapsulation. The major difference is the amount of effort to integrate aspects other than message content. New levels of integration such as security, quality of service, and management can raise issues never encountered during application integration. Many standards efforts are an attempt to address these new levels of integration.

There are both established and emerging standards for exposing interfaces and coordinating the behaviors between ESBs. The evolution of specifications such as WS-Policy, WS-Addressing, and WS-Security will standardize the service interactions between ESBs. The status of these standards range from approved specifications still in the experimitation stage to unfinished specifications. Using some of these specifications (as they exist at the time this book was published) would be risky. Some may never be formally adopted and others may see major changes before they are adopted.

Which of these you may find useful depends on how your organization adopts new technology. Early adopters may start experimenting with some specifications today, and those who prefer to wait for mainsteam adoption may have to wait quite a while longer. With so many standards under development, creating a flexible architecture can be important. A flexible architecture allows refactoring of ESB integration components and services from custom solutions to open standards based solutions as each standard matures.

6.2 ESB service request context translation

While a service request is within the domain of the ESB it carries a context. The context includes, among other things, what ESB capabilities the service request requires. The level of assured delivery, security, and logging are examples of the ESB capabilities that can be associated with the context of a particular service request. This context is understood by all ESB components used to complete the service call.

When the service request needs to pass from one ESB to another, the context as well as the content of the service request must be communicated. For example, if the originating ESB associated once-and-only-once delivery with the service request, the destination ESB must make the same association. Because two ESBs may be implemented in different technologies they cannot directly communicate the service request context. This is the same problem as passing data between applications implemented in different technologies. A COBOL application cannot understand a serialized Java object any more than the Java application could translate a COBOL copybook.

This creates a requirement to translate the service context as it passes between the ESBs. A solution to translating service request content between COBOL and Java is XML and SOAP. The data is translated into a technology-agnostic format both applications can understand. Unfortunately, the corresponding standards to translate service request context are not as mature as those for service content.

Assured delivery currently has two conflicting standards: WS-Reliability (WS-R) and WS-Reliable Messaging (WS-RM). They are both now owned by OASIS but it may take quite a while to reconcile them. WS-Policy is a specification but it is currently held by one vendor and not by a standards group like OASIS or W3C. Standards and specifications for other ESB capabilities do not yet exist.

The lack of stable or widely implemented standards for translating service request context requires custom design and development or experimentation with emerging standards. Experimentation with emerging standards can lead to a dead end because different vendors may interpret the standards in different and incompatible ways. It takes a long time for organizations such as WS-I to set

interoperability guidelines for new standards and be adopted by vendors. Even more mainstream standards like SOAP and WSDL still have some interoperability problems today.

As the standards mature, vendors implement them, and interoperability is achieved, it will be possible to replace the custom context translation with a standards-based approach. The amount of rework will depend on how the custom context passing mechanism is implemented. The principles of service orientation, low coupling, encapsulation, abstraction, and high cohesion can all affect the effort to move from custom to standards-based approaches in the future.

6.3 Introduction to ESB integration patterns

This section introduces three sets of patterns to integrate ESBs. Each instance of integrating two or more ESBs uses at least one pattern from each set. Often more than one pattern from the same set can be used in the same instance. First we introduce each of the sets of patterns and then show how they are combined.

6.3.1 ESB Topology patterns overview

The ESB Topology patterns are variations of the SOA profile in the Patterns for e-business. You can find these variations in the Patterns for e-business layered asset model under the SOA profile for the Direct Connection, Broker, and Router runtime patterns.

There are three emerging patterns for ESB Topology

- Directly Connected ESBs
- Brokered ESBs
- Federated ESB

The ESB Topology patterns describe network relationships between ESBs. Table 6-1 on page 91 describes the business drivers for these patterns, and Table 6-2 on page 91 describes the IT drivers.

Table 6-1 ESB Topology business drivers

Business drivers	Directly Connected ESBs	Brokered ESBs	Federated ESBs
Growth through mergers and acquisitions		√	√
Distributed governance model		√	√
Limited interaction between different enterprise governance zones (for example, line of business (LOB) or regions)	√		
Maximize speed and flexibility to change business processes		√	√
Support cross-organizational processes		√	√

Table 6-2 ESB Topology IT drivers

IT drivers	Directly Connected ESBs	Brokered ESBs	Federated ESBs
Route requests between two ESBs	√		
Route requests between more than two ESBs		√	√
The service consumer request requires coordination of multiple service providers on multiple ESBs			√
Central management of ESB routing rules		√	√
Few interactions between ESBs	√		
Frequent changes to the interactions between ESBs		√	√

IT drivers	Directly Connected ESBs	Brokered ESBs	Federated ESBs
Minimize changes to existing ESBs		√	
Only requires basic interactions	√		
Loose coupling between ESBs		√	√
Provide service provider location transparency to service consumers		√	√

6.3.2 ESB Governance patterns overview

There are three emerging Governance patterns for ESB integration component placement:

- Local Governance
- Intermediary Governance
- Federated Governance

The ESB Governance patterns describe the placement and control (governance) over the components used to connect the ESBs. The control may be in the form of organization, architecture, security, or operational.

It is possible to use ESB connection governance patterns with each other in any combination and in any combination with the Topology and Adapter patterns. For example, security may use a Federated Governance pattern, logging may use a Local Governance pattern, and routing an Intermediary Governance pattern.

Table 6-3 on page 93 describes the business drivers for the Governance patterns, and Table 6-4 on page 93 describes the IT drivers.

Table 6-3 ESB Governance business drivers

Business drivers	Local Governance	Intermediary Governance	Federated Governance
Business processes cross governance boundaries		√	√
Business processes are in a highly regulated environment	√		√

Table 6-4 ESB Governance IT drivers

IT drivers	Local Governance	Intermediary Governance	Federated Governance
One or both ESBs require implementation knowledge	√		√
No knowledge of other ESBs		√	
ESB integration components are controlled by a third party (either internal or external to the enterprise)		√	
Control within a single governance zone	√	√	
Tight control over external connections	√		
Information needed to coordinate behavior is in multiple governance zones			√
Coordination must be synchronous			√

6.3.3 ESB Adapter Connector patterns overview

There are two emerging patterns for ESB Adapter Connectors. These patterns can be combined with each other and with any of the previously introduced Topology and Governance patterns. The two Adapter patterns are:

- Adapter Connector
- Boundary Services Adapter Connector

Tip: You will notice that there is no Federated Services pattern. If the services of two ESBs are fully federated (all services are shared) then, by our the definition of an ESB, they cease being two separate ESBs and become one larger ESB.

The ESB Adapter Connector patterns describe how the ESBs will send and accept messages and, more important, message context. Table 6-5 describes the business drivers for the ESB Adapter patterns and Table 6-6 describes the IT drivers.

Table 6-5 ESB Adapter Connector business drivers

Business drivers	**Connector Adapter**	**Boundary Service**
More agile integration between organizational (governance) silos		√
Low initial implementation cost desired over more agility integration	√	
Agility desired over initial integration development costs		√
Static interactions between LOBs and regions	√	
Interaction between LOBs and/or regions are constantly changing		√
Reduce impact of change		√
Lower total cost of ownership		√
Accelerate integration of mergers & acquisitions		√

Table 6-6 ESB Adapter Connector IT drivers

IT drivers	**Connector Adapter**	**Boundary Services**
Binding of service request context to ESB capabilities is done at build time	√	
Binding of service request context to ESB capabilities is done at run time		√
Limited combinations of ESB capabilities needed to support service requests	√	
Wide variety of ESB capability combinations needed to support service requests		√

IT drivers	Connector Adapter	Boundary Services
Primary requirement is to translate between otherwise incompatible technologies	√	
Limit number of custom components needed		√
Process changes require minimal component changes		√
Lower initial implementation cost	√	
Service-oriented approach		√

6.4 ESB Topology patterns

This section described the following three ESB Topology patterns in more detail:

- Directly Connected ESB pattern
- Brokered ESB pattern
- Federated ESB pattern

6.4.1 Directly Connected ESBs pattern

Figure 6-1 shows the Directly Connected ESBs runtime pattern.

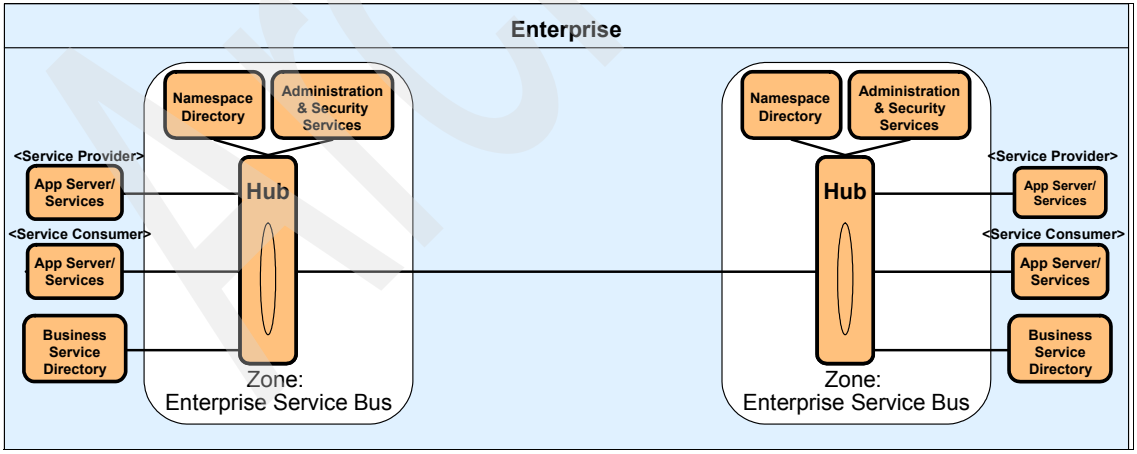

Figure 6-1 Directly Connected ESBs runtime pattern

Business drivers

- Limited interaction between different enterprise governance zones (for example, line of business (LOB) or regions)

IT drivers

- Route requests between two ESBs
- Few interactions between ESBs
- Only requires basic interactions

Pattern description

The Directly Connected ESBs pattern for integrating ESBs is based on the Direct Connection application pattern for Application Integration as described in *Patterns: Direct Connections for Intra- and Inter-enterprise*, SG24-6933.

The simplest type of interaction, this pattern is based on a point-to-point topology. Using this pattern results in the ESBs communicating directly with each other.

When directly connecting ESBs, service providers on one ESB are mapped directly to service consumers on another ESB in a point-to-point topology. As with any point-to-point topology, the endpoints contain implementation knowledge of each other. For example the ESB hosting the service consumer must know which ESB hosts the service provider, what protocol to use, and message format so it can route the request. This results in service routing rules distributed across multiple ESBs and in multiple ESB components. The challenges of managing distributed configurations are well known and often cited as a major systems management problem.

Many important lessons were learned when integrating applications that are applicable to integrating ESBs. One of the most important is that the number of links can grow exponentially as the number of components being linked together grows. This is as true for ESBs as it is for applications. Each time a service provider is linked to a service consumer on a different ESB another point-to-point relationship is created. Maintenance will require exponentially more effort as the number of links between ESBs increases.

WIth no shared infrastructure, each ESB maintains its own namespace directory, administration, and security services. They share no infrastructure so the link must coordinate all service request context between the ESBs.

6.4.2 Brokered ESBs pattern

Figure 6-2 shows the Brokered ESBs runtime pattern.

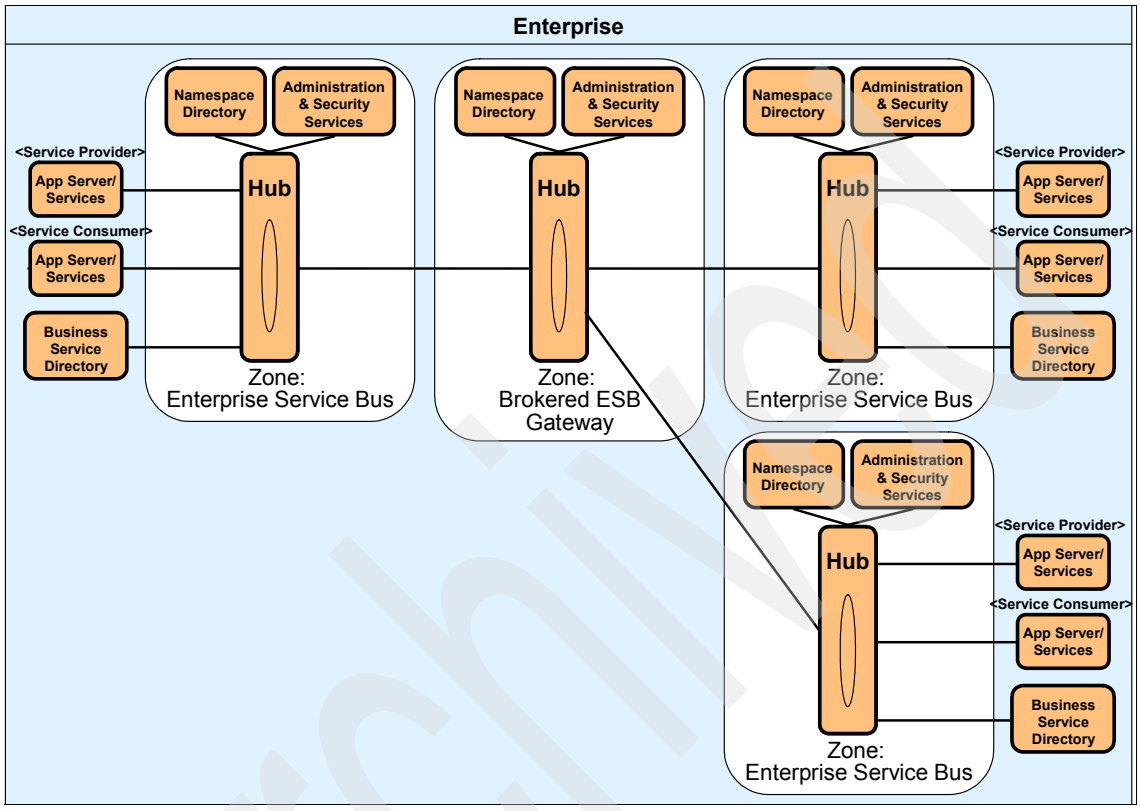

Figure 6-2 Brokered ESBs pattern

Business drivers

- Growth through mergers and acquisitions
- Distributed governance model
- Maximize speed and flexibility to change business processes
- Support cross-organizational processes

IT drivers

- Route requests between more than two ESBs
- Central management of ESB routing rules
- Frequent changes to the interactions between ESBs
- Minimize changes to existing ESBs
- Loose coupling between ESBs
- Provide service provider location transparency to service consumers

Pattern description

As with the Directly Connected ESBs pattern, the Brokered ESBs pattern is also based on an Application Integration pattern: this time the Broker application pattern. The Broker application pattern is described in the redbook *Patterns: Broker Interactions for Intra- and Inter-enterprise*, SG24-6075.

The Brokered ESBs pattern separates the ESB integration logic and related business rules from the ESBs. This pattern reduces the number of point-to-point connections, which is a benefit over the Directly Connected ESBs pattern.

The Brokered ESB Gateway is a component between the ESBs. This component contains all of the knowledge about how to connect the ESBs. This component may be referred to as a hub, a bus, or a gateway depending on how it is implemented and who is describing it. Regardless of what the component is called, it prevents the ESBs from linking directly with each other and centralizes control of the inter-ESB connections.

The Brokered ESB Gateway centralizes much of the connection behavior between the ESBs. The new centralized broker component may be located in a different governance zone than any of the ESBs it integrates. This component may fall under the governance of an Integration Center of Excellence (COE) or enterprise architecture group. In other cases it may be a marketplace under the control of a third party.

As is the case with an application integration broker, each ESB integrates with just the broker component rather than each individual ESB. This encapsulates ESB implementation details, such as routing rules, into the broker. Each ESB now has less implementation knowledge of the other ESBs. This creates a looser coupling so changes in one ESB will be less likely to affect the other ESBs.

The Brokered ESB Gateway will offer many of the same capabilities as a typical ESB but it is not required to offer all ESB capabilities. A Brokered ESB Gateway can offer nothing more than routing service. In this case it could not be considered a full-fledged ESB.

Another difference is that the Brokered ESB Gateway has no service providers or service consumers directly attached. It also may not have its own Business Service Directory because it has no service providers to publish.

The Brokered ESB Gateway often connects heterogeneous ESBs. When it does, it must be able to adapt to a wide range of standards and protocols. If different ESBs implement different levels of the same standard the Brokered ESB Gateway may have to mediate the potentially incompatible standards. For example, an ESB using SOAP V1.1 may not be able to consume a SOAP V1.2 message generated by another ESB. The Brokered ESB Gateway pattern allows centralization of this type of mediation rather than each ESB implementing it directly.

6.4.3 Federated ESBs pattern

Figure 6-3 shows the Federated ESBs runtime pattern.

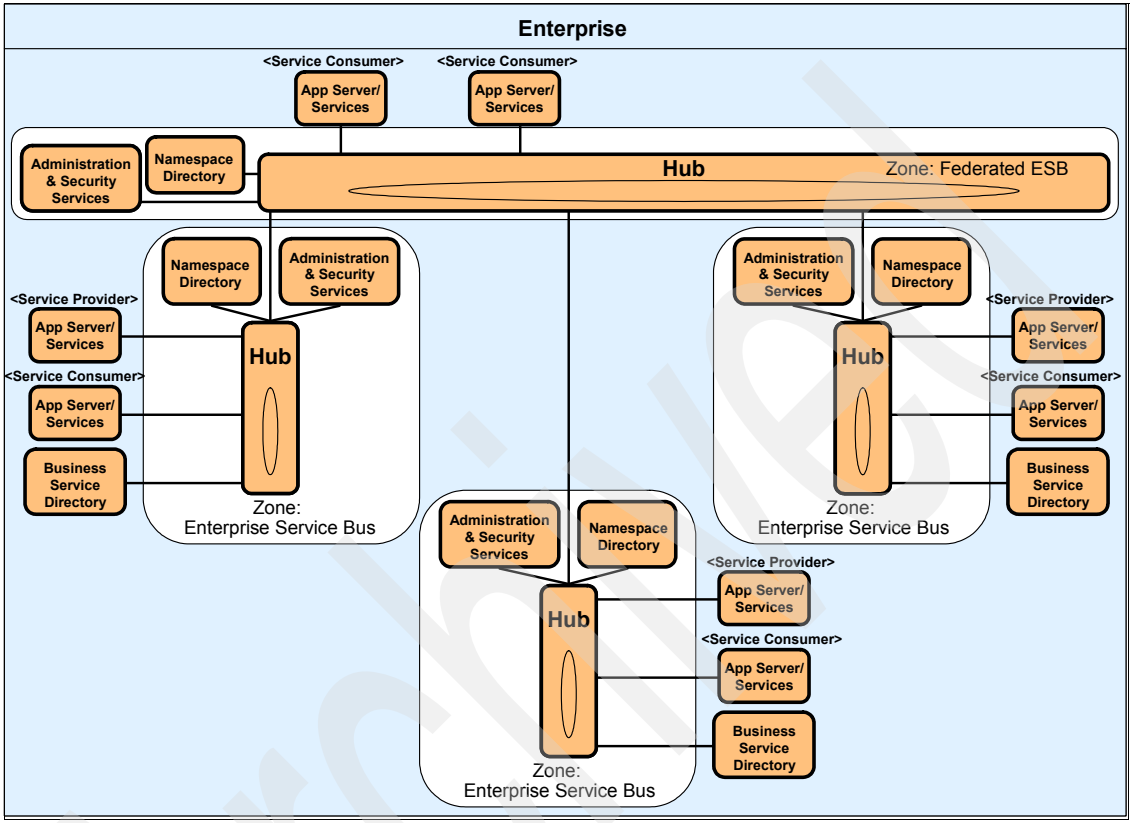

Figure 6-3 Federated ESBs pattern

Business drivers

- Growth through mergers and acquisitions
- Distributed governance model
- Maximize speed and flexibility to change business processes
- Support cross-organizational processes

IT drivers

- Route requests between more than two ESBs
- Service consumer request requires coordination of multiple service providers on multiple ESBs
- Central management of ESB routing rules

- Frequent changes to the interactions between ESBs
- Loose coupling between ESBs
- Provide service provider location transparency to service consumers

Pattern description

The Federated ESBs pattern adds orchestration of service consumer requests that span multiple ESBs, multiple service providers, or both. It also adds the ability to attach a service consumer directly to the hub component.

A service consumer can make requests that require resources from multiple service providers on multiple ESBs. One option is for the service consumer's ESB to coordinate the process. This could be a good option if there are only one or two occurrences of this requirement. If there are more than one or two occurrences this solution would lead to federation logic scattered across multiple ESBs. The resulting decentralized management would be similar to the decentralized management problem of the Directly Connected ESBs pattern. As with the Directly Connected ESBs pattern, there is a penalty in the form of higher costs to maintain and extend the implementation.

To avoid the decentralized management problem, the federation component (hub, bus, or gateway) will control the coordination of multiple service providers. This implements the same design pattern for integration as products such as WebSphere Information Integrator implement for data. The key characteristic is to span heterogeneous ESB implementations and aggregate responses from service providers on those ESBs. This will be transparent to the service consumers and service providers involved.

Under some scenarios there is an advantage to connecting those service consumers who require coordination of service providers on multiple ESBs directly to the federation component. The service consumer's ESB will not have to forward the request to the federation component nor will the federation component have to forward the results back. This results in a simplification of the architecture. It reduces the number of connections and the amount of traffic between the ESBs and the federation component.

The Federated ESBs pattern is not simply a Brokered ESB Gateway with service consumers directly attached. Its purpose is to coordinate a service consumer's request across multiple providers on multiple ESBs.

Introducing the coordination of service providers adds new Application Integration patterns: the Serial Process and Parallel Process patterns. These patterns introduce the need for state management and possibly compensating transactions. Ensure that there are enough requirements to support the additional cost and complexity.

6.5 ESB Governance patterns

A *governance zone* includes all of the components under the control of the same governance body. The *governance body* may be a business unit, an enterprise, or some other organizational construct. Implementation decisions about a component within a governance zone are made by the governance body. A governance body does not have much, if any, influence over components outside of its governance zone. *Governance patterns* define the relationships of components in different governance zones.

Governance patterns are not unique to ESB integration. They apply to many different disciplines in software architecture. The three governance patterns listed here are not all of the known governance patterns but they are the most commonly encountered in ESB integration. The criteria that will drive the choice of governance patterns in ESB integration are control and coupling.

Who has control over the component? As discussed in Chapter 5, “To ESB but not two ESB?” on page 81 one reason for more than one ESB in an enterprise is the existence of multiple governance bodies. A governance pattern determines which governance body controls the components.

What is the acceptable amount of coupling between components in different governance zones? Coupling between governance zones occurs when one governance zone has knowledge about how another governance zone implements components. Coupling between governance zones requires new levels of coordination and communication, which may inhibit or impede the ability to make changes. The more coupling, the more dramatic the impact.

The components needed to integrate two ESBs fall into three governance patterns:

- Local Governance

  The component is placed in the same governance as the ESB.

- Intermediary Governance

  The component is placed in a different governance zone than any of the ESBs.

- Federated Governance

  Parts of the component are placed in more than one governance zone.

6.5.1 Local Governance pattern

Figure 6-4 shows the Local Governance pattern.

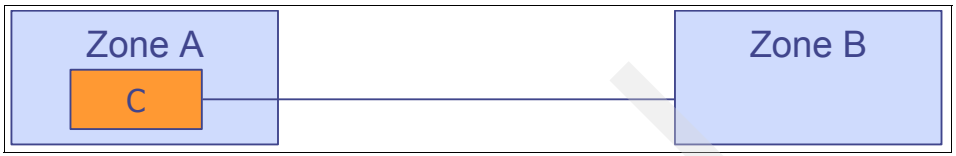

Figure 6-4 Local Governance pattern

Business drivers

- A highly regulated environment for business processes

IT drivers

- Implementation knowledge required by one or both ESBs
- Control within in a single governance zone
- Tight control over external connections

Pattern description

In the Local Governance pattern, component C in Figure 6-4 is placed in the existing governance Zone A. Zone B has no architectural or operation control over component C.

The only coupling is the interface exposed by component C. As long as this interface remains constant there is no need for communication or coordination between governance zones when changes are made. Control of component C is entirely within one governance zone.

Logging exemplifies this type of service context capability. Zone B communicates only what level of logging is needed by its service consumer. The implementation of the logging component is completely under the control of Zone A.

6.5.2 Intermediary Governance pattern

Figure 6-5 shows the Intermediary Governance pattern.

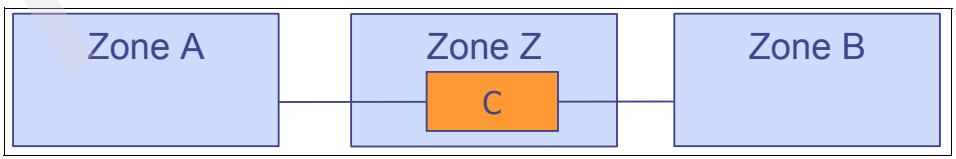

Figure 6-5 Intermediary Governance pattern

Business drivers

- Business processes cross governance boundaries

IT drivers

- No knowledge of other ESBs
- Third-party control of ESB integration components (either internal or external to the enterprise)
- Control within a single governance zone

Pattern description

In Figure 6-5 on page 102, component C is place in Zone Z, which is independent of ESB Zones A and B.

Similar to the Local Governance pattern, the only coupling of component C is the interfaces it exposes to Zone A and Zone B. Additional coupling has also been removed because Zone B no longer knows Zone A exists and the reverse is also true. The only zone visible to either of them is Zone Z. What is on the other side of Zone Z could be of immense or insignificant complexity and neither Zone A nor Zone B would be aware of it. This differs from the Local Governance pattern, in which Zone B knows Zone A exists.

In this case neither Zone A or Zone B controls component C. An independent Zone Z controls the component. Neither Zone A or Zone B can control the implementation since it is in the hands of a third party. This is a common pattern when there are multiple business units (Zone A and Zone B) and a corporate or headquarters unit (Zone Z). None of the business units have control over the component; it is controlled by the headquarters zone.

An example of this pattern would be routing and transformation rules for integrating ESBs. Zone Z would centralize the routing and transformation rules for maintenance and operations purposes. The ESBs in Zone A and Zone B delegate this capability to the Zone Z component, which is outside their governance zones.

6.5.3 Federated Governance pattern

Figure 6-6 shows the Federated Governance pattern.

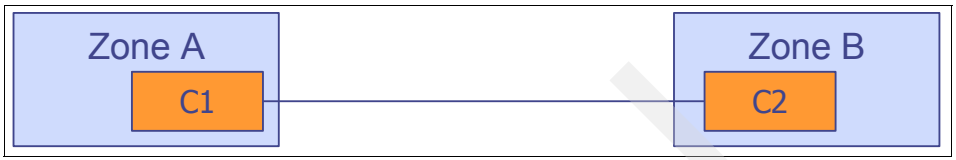

Figure 6-6 Federated Governance pattern

Business drivers

- Business processes cross governance boundaries.
- Business processes are in a highly regulated environment.

IT drivers

- One or both ESBs requires implementation knowledge.
- Information needed to coordinate behavior is in multiple governance zones.
- Coordination must be synchronous.

Pattern description

In Figure 6-6, component C has behavior partitioned into components C1 and C2. A portion of component C is placed under the governance of each zone. The subcomponents collaborate across the zone boundaries.

This has the potential for the tightest coupling of the three governance patterns examined so far. Architects use this governance pattern when components in two governance zones must closely cooperate to deliver some set of functionality. This close cooperation often results in the component parts sharing implementation knowledge. It is possible to hide implementation details using techniques such as XML and Web services but other constraints often prevent this.

The control over component C is jointly held by Zone A and Zone B. Firmly established communication and cooperation is needed between the governance bodies. Most often a trust relationship is also needed to use this pattern.

An example is a federated security manager. Component C1 would contain the implementation details to validate a user with component C2. This would prevent replication of user information between Zone A and Zone B.

Another common use of this pattern is as a gateway between the two governance zones. This isolates the technical implementation so only component C1 knows how component C2 is implemented. Any other component

in Zone A would use C1 whenever it needs to access Zone B. This provides some level of isolation. If C2 changes, then only C1 needs to change with it.

The interaction between C1 and C2 is often but not always a synchronous request/response handshake. In the security example, C1 sends a request containing the user information to C2 and waits for a response from C2 with the authentication results.

6.5.4 Multiple governance patterns

Multiple governance patterns will be used within the same ESB integration. Best practice is to use the pattern that best fits the solution and not to try to force everything into one governance pattern.

Figure 6-7 shows an example of all three governance patterns in use in the same ESB integration.

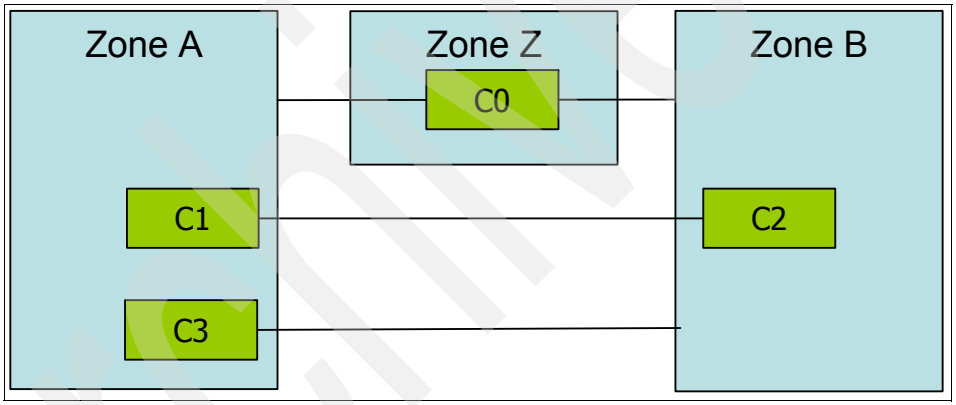

Figure 6-7 Multiple governance patterns co-exist

An example of the multiple types of governance patterns shown in Figure 6-7 is where:

- Routing uses a component that lies outside both ESB zones represented by C0.
- Security uses components to both ESB zones. This is shown using C1 and C2.
- Logging could use a component in Zone A only. This is shown using C3.

Each governance pattern solves a different type of integration problem. Pick the governance pattern that best fits the particular problem.

6.6 ESB Adapter Connector patterns

When integrating two ESBs, it is not difficult to pass the service call content. The difficult part is translating the service context from the semantics of one ESB to the semantics of another ESB. This is especially true when the ESBs are from different vendors.

The service context is made up of all of the behaviors the service collaboration expects the ESB to perform on its behalf.

The links between the ESBs can be one of two architectures:

- Adapter Connector pattern

  Service behaviors for the interaction determined at build time.

- Boundary Services Adapter Connector pattern

  Service behaviors for the integration determined at runtime.

6.6.1 Adapter Connector pattern

Figure 6-8 shows the Adapter Connector pattern.

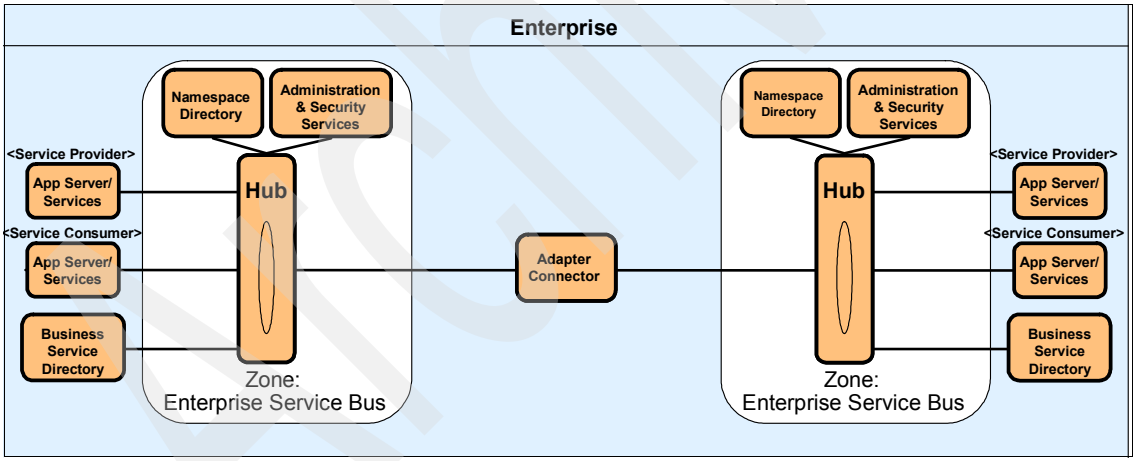

Figure 6-8 Adapter Connector pattern

Business drivers

- Low initial implementation cost desired over more agility integration
- Static interactions between LOBs and/or regions

IT drivers

- Binding of service request context to ESB capabilities: done at build time
- Limited combinations of ESB capabilities needed to support service requests
- Primary requirement: translate between otherwise incompatible technologies
- Lower initial implementation cost

Pattern description

The Adapter Connector pattern is an established pattern and is not unique to ESB integration. It is a common pattern when connecting heterogeneous applications. In the case of ESB integration, the Adapter Connector pattern supports the interaction between a service consumer on a local ESB and a service provider on a foreign ESB. The logic to translate ESB behaviors (for example, logging, assured delivery, and security) to support that interaction are built into the connector and then bound to the connector at design time.

Since the mediations a connector offers are fixed at build time, the only services the connector can perform are those that were known at build time. If a new or existing interaction needs a different ESB behavior mediation than the existing connector supports, it will require modification of an existing connector or building a new connector. As the number of interactions increase, the complexity and number of connectors increase at a similar or faster rate.

A connector can be used to mediate a technology mismatch between two ESBs. They each may support different standards or levels of standards. The connector would act as a specialized protocol switch to overcome the mismatch.

Encryption is an example of mediating a technology mismatch between two ESBs. If the service provider ESB encrypts the message, the service consumer ESB may not know how to decrypt the message. The connector component would know how to decrypt messages from the service provider and re-encrypt them in a way the service consumer understands. The specific decisions how and when to do the encryption translation are all made during the connector design.

Another example is if the local and foreign ESBs use different JMS implementations. Conformance with the JMS specification does not promise or provide for interoperability. The connector would mediate the incompatibility between different JMS implementations by acting as a compatible JMS client for each side of the interaction.

Figure 6-9 on page 108 shows two different service consumers who want to access service providers on a foreign ESB.

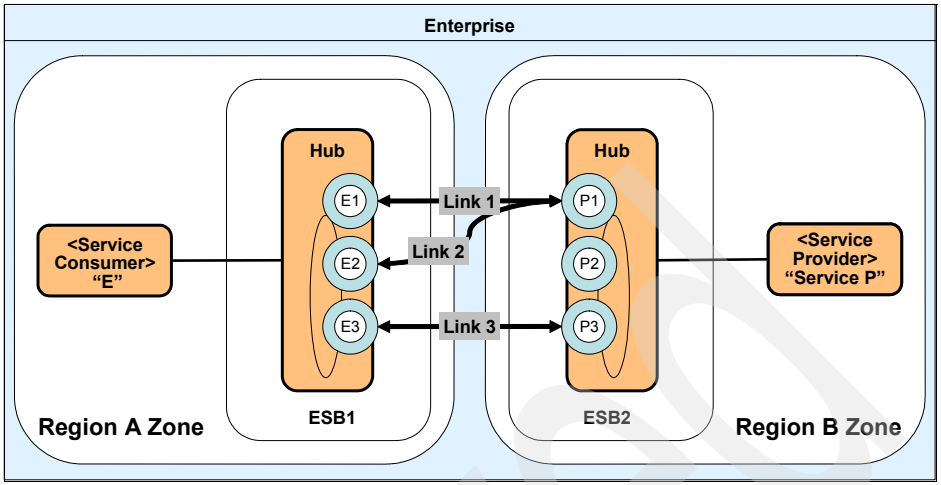

Figure 6-9 Example of connector architecture

The interaction shown in Figure 6-9 between E1 and P1 may require authentication and authorization, encryption of sensitive data, and extensive logging for regulatory purposes. The interaction between E3 and P3 requires logging to support a business process management dashboard. This is likely to require two unique connectors due to the radically different requirements of each interface. Now a business process change requires a new interaction between E2 and P2. This new interaction must support both regulatory logging and business process management logging. This is likely to result in a new connector specific to that interaction even though it may reuse code from the other two connectors.

Since the connector's behaviors are set at design time, this type of ESB connection can be fast and easy to build. There are potential penalties for this initial deployment speed. These penalties can offset the initial development speed and create an expensive and brittle infrastructure.

The Adapter Connector pattern is good for a small number of connections between ESBs. As you can see in Figure 6-9, the number of connector components will typically grow at the same rate as the number of service consumers using service providers on the foreign ESB. This is similar to the exponential growth of point-to-point application integration interfaces that plagues many large enterprises.

It is possible for a connector to support more than one interaction, but there are risks. Even a well-designed connector may become brittle as each layer of code is added to support new and changing requirements. Significant rework is common when a requirements change demands new or different services than

the connector currently supports. At a minimum, a new code branch is needed within the connector to bind an alternative service, or a completely new connector may be required. In either case there are code and configuration changes whenever the requirements change because the connector can only be used to support interactions that were known at design time.

Tip: If there are many links with different behaviors or a need to frequently change the links, it is advisable to look at a more robust and flexible linking method.

6.6.2 Boundary Services Adapter Connector pattern

Figure 6-10 shows the Boundary Services pattern.

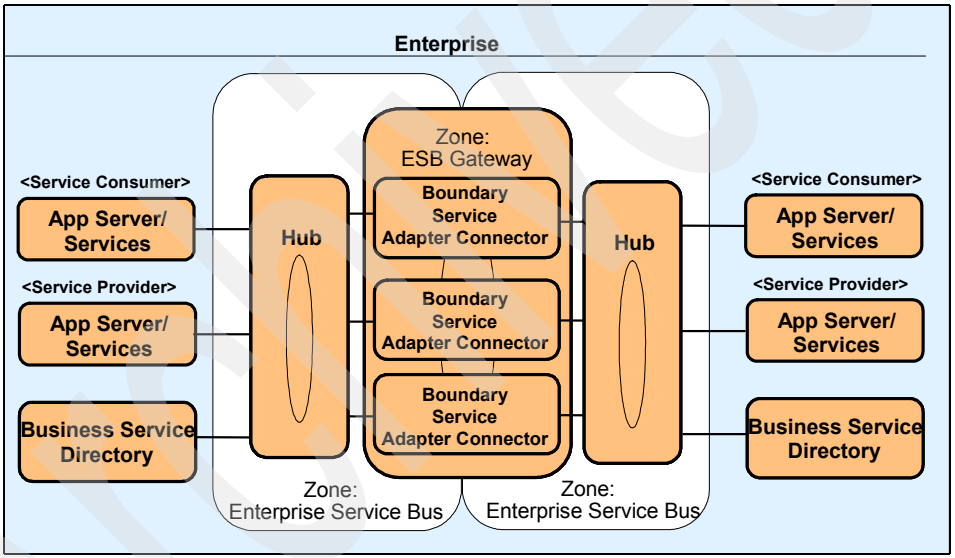

Figure 6-10 Boundary Services Adapter Connector pattern

Business drivers

- More agile integration between organizational (governance) silos.
- Agility desired over initial integration development costs.
- Interaction between LOBs and/or regions are constantly changing.
- Reduce impact of change.
- Provides lower total cost of ownership.
- Accelerate integration of mergers and acquisitions.

IT drivers

- Binding of service request context to ESB capabilities is done at run time.
- Needs wide variety of ESB capability combinations to support service requests.
- Limit number of custom components needed.
- Process changes require minimal component changes.
- Service-oriented approach.

Pattern description

Boundary Services are an emerging pattern to create runtime-configurable and composible interactions between applications. Boundary Services live at the edge of an application, take the context of an event outside of the boundary, and translate it to a context understood within the boundary. They also take the internal event context and prepare it for external consumption. The Boundary Services Adapter Connector pattern has roots in SOA techniques.

Boundary Services use metadata from the inbound service call (other than the service location) to determine what behaviors the interaction requires. The metadata could be part of the transport header (JMS or HTTP header), part of the message envelope (SOAP header), or part of the message body (XML name value pair).

The metadata will contain the previously agreed-to semantics that describe the service call context to the Boundary Services. This is the contract between the local (service consumer's) ESB and the foreign (service provider's) ESB. The semantics in the metadata describe the responsibilities the local ESB expects the foreign ESB to perform on its behalf.

Each Boundary Service or group of Boundary Services focuses on the delivery of one set of capabilities. These Boundary Services implement SOA best practices by hiding their technical implementation and making no assumptions about context and state.

A reliable delivery Boundary Service could offer three service levels:

- Best effort delivery
- At least once delivery
- Once and only once delivery

A data security Boundary Service could also offer two services:

- Sign message
- Encrypt message

A logging Boundary Service could offer four services:

- Process monitoring
- Non-repudiation

- Trace
- Debug

The service call context metadata may change from one invocation to the next between the same service consumer and service provider. The Boundary Services adapt to these changes at runtime. Compare this to the design time binding of service call context metadata required by a connector. This is an example of the flexibility and agility that is inherent to a service-oriented design. The Boundary Services can be combined in ways the original designers did not anticipate.

The Boundary Service knows how to map the service call metadata to an internal implementation of that service level. For example, when a logging Boundary Service receives a non-repudiation request, it knows that it is a request to record an event in a tamper-proof log. The actual mechanism to perform this function is internal to the ESB and is not part of the Boundary Service. The Boundary Service only has to know how to map the metadata request to the correct internal functions.

A Boundary Service can map more than one metadata definition to the same internal service. For example “trace” and “debug” could both map to the same level of logging. This is allowable as long as the foreign bus maps the service context metadata to a service level that exceeds the local bus’s expectation. Mapping “once and only once” assured delivery to a “best effort” service would not be acceptable, but mapping a “best effort” request to a “once and only once” service level would be acceptable (but more resource intensive).

Boundary Services can force delivery of a minimum service level. The governance body of the ESB can choose the minimum service levels it will deliver to a foreign service consumer regardless of the request. For example, a service call may request no message logging. This does not mean that the ESB *cannot* log the message, only that it is not violating the contract if it does not.

Boundary Services Adapter Connectors may not deliver less than the requested service level. If the service level specifies to encrypt the message using 256-bit encryption, then the message cannot be sent with only 128-bit encryption. The Boundary Services Adapter Connectors must honor the service level associated with the semantic definitions agreed upon between the governance bodies. If the service level is only “encrypt,” then either 128-bit or 256-bit encryption is acceptable.

In some cases the Boundary Service may be based on an industry standard such as WS-Security. In other cases an industry standard may not yet exist that would require a proprietary solution. Following design and development best practices enables the standards-based solution to replace the proprietary solution at some future point.

In some cases the invocation of a Boundary Service may be optional. An example of an optional Boundary Service could be error handling response processing. In some cases the service consumer may want a response in the event of an error and in other cases it is not interested in error conditions. The service call metadata could include metadata requesting an error response on one message, and the next service call may not include this metadata element.

A Boundary Service can have a default service level if a specific service level is not selected. A default service level may be set for a logging Boundary Service if the service call metadata does not request any other option. This ensures a minimum level of service that may be required by the governance organization of the ESB receiving the request.

6.6.3 Composite

As with the Governance patterns and Topology patterns it is possible to use more than one ESB Adapter Connector pattern. It is doubtful that any instance of integrated ESBs will be implemented using only the Adapter Connector or the Boundary Services Adapter Connectors pattern. The reasons to use both ESB Adapter patterns can be financial and technical, and it will be necessary to make trade-offs between implementing Boundary Services and Adapter Connector patterns.

The first reason is the “art of the possible” using the technology available today. The authors of this book are not aware of a working service-based ACID[1] transaction model for ESBs. Developing such a model may be possible but it is likely to be expensive. The WS-AtomicTransaction specification[2] is under development to address this level of integration. When this standard is in early adopter or mainstream use, then coordination of a transaction between two ESBs will be much simpler than it would be today. At that time it may be economical to expose transaction services as a Boundary Service. Meanwhile a Adapter Connector may be the only solution available to coordinate transactions.

In contrast, development of Boundary Services for logging has wide tool and framework support. The WebSphere Application Server Common Event Infrastructure[3] provides a foundation for an event-driven logging Boundary Service. There are also open source solutions for logging[4] frameworks that could be used as a foundation for building Boundary Services.

---

[1] atomicity, consistency, isolation, and durability

[2] http://www.ibm.com/developerworks/library/specification/ws-tx/

[3] http://www.ibm.com/developerworks/websphere/library/techarticles/0504_brodie/0504_brodie.html

[4] http://www.ibm.com/developerworks/websphere/library/techarticles/0207_barcia/barcia.html

Using these examples, it would not be hard to imagine integrating two ESBs using a Adapter Connector for the transaction coordination and Boundary Services for logging. The transaction Adapter Connector could be converted to a Boundary Service at a later date if there is adequate return on investment.

Postponing the use of the Boundary Services pattern until every aspect of ESB integration can be accommodated is unnecessary and probably unwise. There is business and IT benefit to implementing even one Boundary Service.

6.6.4 Comparing Adapter Connectors and Boundary Services

Table 6-7 lists the major differences between the two ESB Adapter Connector patterns.

Table 6-7 ESB Adapter Connector patterns selection decision table

Driver	Adapter Connector	Boundary Services
Service binding	Design time	Run time
Shared services	Few	Many
Collaborations	Static	Dynamic
Initial deployment effort	Less	More
Modification effort	More	Less

Part 2

Business scenario and guidelines

This part contains the following chapters:

7

The business scenario used in this book

The chapters in Part 3, “Scenario implementation” on page 147, use a common business scenario for all of the scenario implementations. This business scenario describes a supply chain management application where the supply chain applications are split between two organizations. The business scenario is based on the WS-I sample business scenario.

This chapter contains the following sections:

- A description of the WS-I sample business scenario
- The business context of the sample business scenario used in this book
- A definition of each application in the sample business scenario
- An example of using the sample business scenario

7.1 WS-I sample business scenario

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

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

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

http://www.ws-i.org

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

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

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

7.2 Sample business scenario used in this book

We use two fictional organizations to demonstrate design, development, and runtime considerations for integrating ESBs. The business scenario used is a modified version of the IBM implementation of the WS-I sample business scenario. This section describes the high-level business context of the sample business scenario, describes how the supply chain works, and shows screen shots from a runthrough of the sample application implementation.

7.2.1 Business context

Organization A provides a Web site on which users place orders for electronic goods such as televisions, DVD players, and video cameras. The supply chain management application handles the order of these goods through a Retailer, the delivery of these goods through a Warehouse, and the restocking of the

Warehouse through a number of Manufacturers. The Manufacturer applications belong to Organization B. The users of the Web site are all internal Organization A employees.

Figure 7-1 shows the application that makes up the supply chain management scenario.

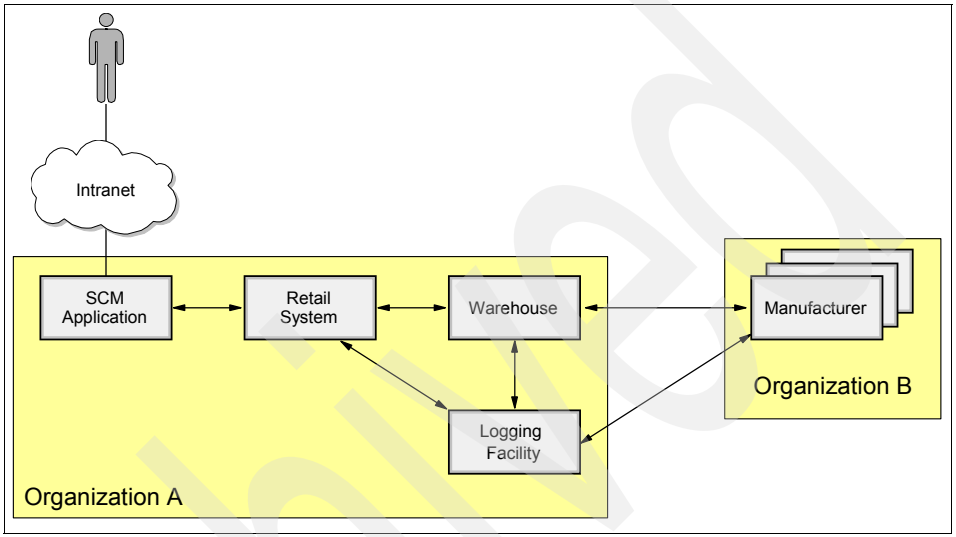

Figure 7-1 High-level business context

7.2.2 Applications in the supply chain management

The supply chain management scenario works as follows:

- The SCMSampleUI application provides the Web front end for users to access the supply chain management process.
- The Retailer application can be used to retrieve a list of products sold by the Retailer, and to place orders.
- The Warehouse application ships product orders if the Warehouse has sufficient stock.
- The Manufacturer applications (of which there are three) replenish the Warehouse of products if the Warehouse stock levels fall below a certain threshold. Each Manufacturer produces different products and belongs to a separate organization from the other applications.
- The Logging Facility application records the status of orders as they pass through the supply chain management scenario. This can be used to track the progress of a given order.

7.2.3 Example of using the sample application

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

1. Figure 7-2 shows the first window presented to the user. Click **Place New Order**.

Figure 7-2 SCM Sample application

2. This retrieves a list of all products that the Retailer application sells, which are displayed in the Shopping Cart window (Figure 7-3).

Figure 7-3 SCM Sample product listing

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

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

3. The Order Status window, as shown in Figure 7-4, shows which orders were placed and which orders were not placed due to insufficient stock. To track the progress of orders, click **Track Order**.

Figure 7-4 SCM Sample order status page

4. This retrieves information stored in the Logging Facility application. Figure 7-5 shows the results of an order in which products 605001 and 605002 were shipped and a reorder for 19 units of product 605002 was placed with Manufacturer B.

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

Figure 7-5 SCM Sample track order page

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

8

Technology options

This chapter describes technologies that are relevant to integrating Enterprise Service Bus (ESB) infrastructures.This chapter discusses:

- Web services
- Messaging
- J2EE
- Service Data Objects

8.1 Web services

Web services is a recent re-invention of concepts that have been around for some time. They introduce many new advantages and capabilities. In a sense, none of the function that Web services provide is new; CORBA has provided much of this function for many years. However, Web services builds on existing open Web technologies, such as XML, URL, and HTTP. Web services are defined in several different standards, such as SOAP and WSDL, which build on general Web and other Web services standards. These standards are defined by the World Wide Web Consortium (W3C), the Organization for the Advancement of Structured Information Standards (OASIS), and Web Services Interoperability Organization (WS-I).

Basic Web services support provides for three simple usage models. These are:

- One-way usage scenario

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

- Synchronous request/response usage scenario

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

- Basic callback usage scenario

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

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

Key features of Web services are:

- Based on open standards
- Location and platform independence
- Self described
- Loosely coupled

One of the main aims of Web services is to provide a loose coupling between service consumers and service providers. While this is limited to a certain extent by a requirement for the consumers and providers to agree on a WSDL interface definition, Web services have been created with significant flexibility with regard

to the location of these Web services. Figure 8-1 shows how the Web services interaction model has been designed with this form of loose coupling.

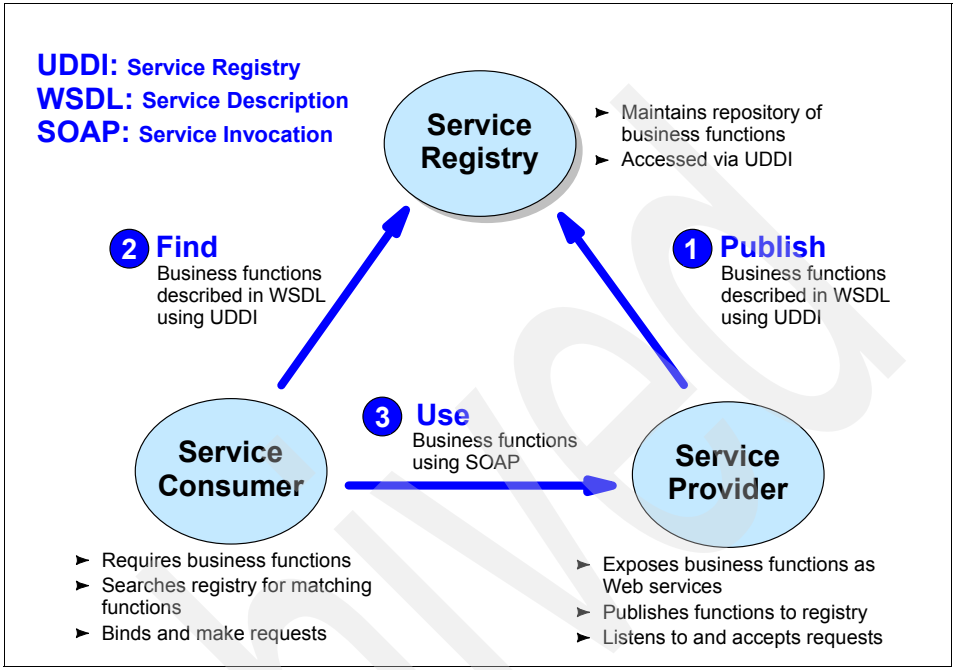

Figure 8-1 Basic Web service interaction model

The interactions work as follows:

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

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

8.1.1 SOAP

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

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

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

It is common to talk about SOAP in combination with the transport protocol used to communicate the SOAP message. For example, SOAP being transported using HTTP is referred to as *SOAP over HTTP* or *SOAP/HTTP*.

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

SOAP Version 1.1 and 1.2 specifications are available from W3C.

8.1.2 Web Services Description Language (WSDL)

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

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

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

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

8.1.3 Universal Description, Discovery, Integration (UDDI)

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

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

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

  http://www.ibm.com/developerworks/webservices/library/ws-rpu1.html
- "The role of private UDDI nodes, Part 2: Private nodes and operator nodes"

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

8.1.4 Web services interoperability

To facilitate the development of truly interoperable Web services, the Web Services Interoperability Organization (often referred to as the WS-I) was formed in February 2002. The WS-I aims to promote interoperability of Web services implementations by publishing *profiles*, which are descriptions of conventions and practices for the use of specific combinations of Web services standards through which systems can interact. Technology vendors can then produce compliant implementations and publicize that compliance, offering some level of assurance to technology customers as to the level of Web services interoperability that can be achieved with different implementations.

The WS-I published the first profile for interaction, the Basic Profile V1.0, in July 2003, and many technology vendors provide product implementations of Web

services that are compliant with this profile. This is described further in the next section.

In August 2004 the WS-I published the Basic Profile V1.1, splitting the original profile in two: the Basic Profile V1.1 and the Simple SOAP Binding Profile V1.0. The idea is that the combination is equivalent to Basic Profile V1.0. This has been done to aid in the incorporation of different binding mechanisms, such as SOAP with Attachments. This enables an implementation to make the claim that it is compliant with Basic Profile V1.1 and Attachments Profile V1.0 without needing to implement the Simple SOAP Binding Profile V1.0.

The Web Services Interoperability Organization Web site contains links to published, draft, and planned interoperability profiles and information about vendor compliance:

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

8.1.5 WS-I Basic Profile V1.0

The WS-I Basic Profile V1.0 specifies a set of usage scenarios and Web services standards that can be used to integrate systems. It focuses on the core foundation technologies upon which Web services are based. Basic Profile V1.0 was approved unanimously on July 22, 2003, by the WS-I board of directors and members.

The WS-I Basic Profile V1.0 - Profile Specification consists of the following non-proprietary Web services related specifications:

- SOAP V1.1
- WSDL V1.1
- UDDI V2.0
- XML V1.0 (Second Edition)
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes
- RFC2246: The Transport Layer Security Protocol Version V1.0
- RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile
- RFC2616: HyperText Transfer Protocol V1.1
- RFC2818: HTTP over TLS
- RFC2965: HTTP State Management Mechanism
- The Secure Sockets Layer Protocol Version V3.0

The WS-I Supply Chain Management sample application depicts an application for a fictitious consumer electronics retailer. This sample application is the basis of the scenarios in this book as described in Chapter 7, “The business scenario used in this book” on page 117.

See also the following IBM developerWorks® articles:

- First look at the WS-I Basic Profile 1.0

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

- First look at the WS-I Usage Scenarios

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

- Preview of WS-I sample application

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

8.1.6 WS-I Basic Profile V1.1

WS-I Basic Profile V1.1 removes the SOAP binding requirements and moves them to the Simple SOAP Binding Profile V1.0. This means that being WS-I Basic Profile V1.1 compliant by itself is not very interesting. Only when a binding is applied can the Web services interact. Two bindings have been created:

- The Simple SOAP Binding Profile V1.0, which together with the WS-I Basic Profile V1.1 allows equivalent function to the Basic Profile V1.0
- The Attachments Profile V1.0, which allows SOAP with attachments as a binding option

8.1.7 Advanced and future Web services standards

There are many successful implementations of the basic Web services standards, particularly SOAP and WSDL, but many aspects of service interaction and integration are not directly supported by those basic standards, such as security, transactionality, delivery assurance, and process modeling.

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

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

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

8.1.8 Web services security

In theory, Web services can leverage any security model that is appropriate to the underlying communication technologies. (SOAP/HTTP can utilize basic

HTTP authentication or SSL authentication and encryption.) However, such simple point-to-point models are insufficient for the widespread integration needs of SOA. For example:

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

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

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

Further information

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

  http://www.ibm.com/developerworks/library/ws-secmap/
- "Web Services Security: Moving up the stack"

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

8.1.9 WS-ReliableMessaging and SOAP/JMS

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

The WS-ReliableMessaging specification defines a protocol for reliable communication (including SOAP messages) that use a variety of communication

technologies, which may themselves be less reliable. An updated specification was published in March 2004.

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

Further information

- Updated: “Web Services Reliable Messaging”

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

- “Implementation strategies for WS-ReliableMessaging”

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

Business Process Execution Language for Web Services

The encapsulation and exposure of business functions as services in an SOA enables the definition of processes consisting of those services. The Business Process Execution Language for Web Services (WS-BPEL) provides a standard, XML language for expressing business processes consisting of functions that are defined through WSDL interfaces. WS-BPEL supports both short-lived processes and long-lived processes (processes that must wait at certain points until some event occurs, such as the receipt of an event).

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

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

Further information

- WS-BPEL specification

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

- Business Process with WS-BPEL, a series of introductory articles and references

  http://www.ibm.com/developerworks/webservices/library/ws-bpelcol1/

- WS-BPEL support in WebSphere Business Integration Server Foundation

  http://www.ibm.com/software/integration/wbisf/features/

- WS-BPEL support in WebSphere Studio Application Developer Integration Edition

  http://www.ibm.com/software/integration/wsadie/features/

Web services transactions

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

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

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

Further information

- Web Services Transactions specifications

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

- “Transactions in the world of Web services,” part 1 and part 2

  http://www.ibm.com/developerworks/webservices/library/ws-wstx1/
  http://www.ibm.com/developerworks/webservices/library/ws-wstx2/

Web Services Policy Framework (WS-Policy)

The Web Services Policy Framework is intended to provide a set of languages by which service consumers and providers can express their requirements and capabilities concerning qualities of service of service interactions, such as

security, transactionality, and communication reliability. Eventually a framework of such languages, supported by Enterprise Service Bus middleware, will enable open-standard implementations of negotiated coupling between various aspects of service interactions.

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

Further information

- "Web Services Policy Framework"

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

- "Web Services Policy Framework: New specifications improve the WS-Security model"

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

8.2 Messaging

Messaging is a form of communication between two or more software applications or components. Messaging is commonly used for application integration where the application does not need an immediate answer to proceed.

In messaging, the requester sends a message to a destination. At some point the provider receives the message and does some processing. It does not require the two applications to be available at the same time. The power of messaging lies in this disconnect. Messaging is often referred to as loosely coupled, but to get the full advantage of this, advanced broker functionality is required. Without this broker functionality, the requester and provider must agree on a format and location for the messages. The addition of broker functionality allows for routing of messages between destinations and for code to be inserted into the messaging middleware in order to transform message formats.

Infrastructure enabling interaction by exchanging messages is called *message-oriented middleware*.

8.2.1 JMS

The Java Message Service (JMS) is a cross-platform Java API for accessing message-oriented middleware. The JMS specification defines a set of message types and APIs for sending and receiving the messages to and from destinations.

In JMS, messages have three distinct sections:

- Header

  The JMS header contains information about where the message was sent and where responses should be sent. These properties are typically for use by the JMS provider.

- Properties

  JMS properties are application-level properties. JMS properties can be strings, numbers, or booleans and are named. The JMS properties are intended as an extensible form of application-level header.

- Body

  The body of the message contains the data being transported. It is intended for the payload of the message.

JMS defines two different interaction styles for messaging:

- Point-to-point

  A single message sent to a destination is received by a single client.

- Publish subscribe

  A single message sent or published to a destination is received by all clients.

Messages may be persisted at the destinations. The intent is that persistent messages are guaranteed to be delivered and not duplicated. Messages also may be sent as a part of an externally coordinated two-phase commit transaction.

The J2EE V1.3 specification integrates support for JMS V1.0.2b and requires that J2EE V1.3 compliant application servers include an integral JMS provider. It also introduces the concept of message-driven beans (MDBs), which enables a message to be delivered to an EJB, allowing the asynchronous invocation of business logic.

With J2EE V1.4, the JMS specification is upgraded to the V1.1 level, which includes support for domain-neutral messaging. In JMS 1.0.2b, the application writer has to decide which of the two messaging models—point-to-point or publish subscribe—the program should use. In JMS V1.1, the programming model is the same for both interaction models. The type of destination that is used determines which model that maps to. Destinations in JMS are considered administrative objects that get bound into a JNDI namespace and looked up later. J2EE V1.4 also makes the concept of an MDB more generic, providing a framework for anyone wishing to trigger work asynchronously into an enterprise application.

Advantages of JMS

Some of the advantages of using JMS are:

- 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 enables you to extend existing message-based applications by adding new JMS clients that are integrated fully with their existing non-JMS clients.
- Developers have to learn only one common interface for accessing diverse messaging systems.

Disadvantages of JMS

Some disadvantages of JMS are:

- It is not a protocol, so all of your JMS applications have to access the same JMS provider.
- JMS resources require an extra level of administration.

8.2.2 WebSphere MQ messaging

WebSphere MQ messaging is an implementation of message-oriented middleware. The WebSphere MQ programming model enables communication of applications or components across a network. The WebSphere MQ programming model offers an API for widely used programming languages on major platforms. Applications designed and written to use the Websphere MQ API are known as *message queuing* applications. WebSphere MQ also supports a JMS interface.

Message queuing has been used for many years in e-mail communication. In WebSphere MQ it works similarly: A message producer sends messages to a persistent queue, and messages are taken out by message consumer. This style of communication does not require an immediate reaction from the message consumer.

This section describes some basic WebSphere MQ messaging terms.

Message

A message is collection of data sent by one message producer to be received by message consumer. WebSphere MQ defines four types of messages:

- Datagram

  A simple message. Reply is not expected.

- Request

  A message for which a reply is expected.

- Reply

  A message that represents a reply to a request message.

- Report

  A message that describes an event such as a system message or error.

WebSphere MQ messages consist of application data and control information. Message descriptors represents the structure for control information and contain information such as the type of message, message priority, and the message ID.

Message queue

A message queue is a destination that holds messages. Message producers put messages on message queues, and message consumers get messages from message queues. A message queue can be a volatile in-memory buffer or a persistent buffer on underlaying storage (such as a disk). Message queues are managed by a queue manager.

Queue manager

A queue manager is a system program that provides services for administering and connecting to queues. It provides an API so that applications can put messages in and read messages from a queue, and provides administration services so that applications can create and remove queues, and change queue properties.

Advantages of WebSphere MQ messaging

Some of the advantages of using WebSphere MQ are:

- Support for major programming languages
- JMS support
- Highly available and reliable messaging
- Tight integration with WebSphere Application Server

Further information

More information about WebSphere MQ can be found in this IBM publication: *WebSphere MQ Application Programming Guide*, SC34-6064-03.

8.2.3 Service integration bus

The service integration bus, which is part of WebSphere Application Server V6, provides advanced support for application integration. It combines support for applications connecting via native JMS, WebSphere MQ JMS, WebSphere MQ, and Web services. It supports the message-oriented middleware and request-response interaction models. As a part of this, the service integration bus supports multiple message distribution models, reliability options, and transactional messaging.

Concepts and architecture

The service integration bus technology introduces a number of new concepts, which we discuss in this section.

Bus

A service integration bus, or bus, provides a conceptual connection point and a namespace for destinations and services. The application integration capabilities of the service integration bus are provided by a number of connected messaging engines.

Messaging engine

A messaging engine provides the messaging capabilities of the service integration bus. Messaging engines provide two functions:

- Message management

  A messaging engine manages messages by routing them to the appropriate endpoint (via additional messaging engines if required). These messages can be persisted to a database and managed within a transactional scope.

- Connection management

  While the conceptual entity clients connect to the bus, the physical connection is to a messaging engine. Clients can connect to and send messages to any messaging engine in the bus. If the destination is assigned to a different messaging engine, the messaging engine will route it to the correct messaging engine.

A messaging engine is assigned to a bus member.

Bus member

A bus member is an application server, or cluster, that is a member of a bus and therefore is hosting a messaging engine.

Destination

A destination is an addressing point within a bus. A destination is assigned to one bus member and therefore one or more messaging engines. Clients send messages to a destination, and the bus ensures that it is routed to the correct localization on the bus. The service integration bus supports six destination types:

- Web service destinations

  Web service destinations are a representation of an outbound Web service in the bus. They are used as a placeholder for a port selection mediation.

- Port destinations

  Port destinations are a representation of an outbound Web service port. Sending a Web service request to a port destination results in the target Web service being invoked.

- Queue destinations

  Queue destinations are destinations that are configured for point-to-point messaging.

- Topic space destinations

  Topic space destinations are destinations that are configured for publish/subscribe messaging.

- Alias destinations

  Alias destinations are destinations that are configured to refer to another destination. They provide an extra level of indirection for messaging applications. An alias destination can also be used to override some of the values specified on the target destination, such as default reliability and maximum reliability. An alias destination can also refer to a destination on a foreign bus.

- Foreign destinations

  Foreign destinations are not actual destinations within a service integration bus, but they can be used to override the default reliability and maximum reliability properties of a destination that exists on a foreign bus.

Destinations can be mediated to provide advanced message formatting and routing function.

Inbound service

An inbound service is defined to enable Web service clients to connect to the bus. An inbound service converts an incoming Web service request into a message and places it on a destination, where it can then be routed, transformed, and processed. Inbound services can be invoked using SOAP over JMS or SOAP over HTTP by associating the service with the relevant endpoint listener.

Outbound service

An outbound service enables a Web service request in the bus to exit and invoke a Web service. The Web service can be invoked by SOAP over JMS or SOAP over HTTP. Creating an outbound service causes Web service and port type destinations to be created. Sending a Web service message to the Web service destination causes the Web service to be invoked. By routing a request from an inbound service to an outbound service, the service integration bus can be inserted in the Web service flow, providing some Enterprise Service Bus capabilities.

Endpoint listener

An endpoint listener listens for incoming Web service requests via HTTP or JMS and passes them onto the relevant inbound service. An endpoint listener can be thought of as a localization point for an inbound service.

Message point

When a destination is assigned to a bus member, a message point is created. The messages are stored on the message point.

Three types of message point can be contained with a messaging engine:

- Queue points

  A queue point is the message point for a queue destination.

- Publication points

  A publication point is the message point for a topic space. Creating a topic space destination automatically defines a publication point for each messaging engine within the bus.

- Mediation points

  A mediated destination also has mediation points. A mediation point is where messages are stored while they wait to be mediated.

Mediation

A mediation processes in-flight messages between the production of a message by one application and the consumption of a message by another application.

Mediations enable the messaging behavior of a bus to be customized. Examples of processing that can be performed by a mediation are:

- Transforming a message from one format into another.
- Dynamically routing messages to one or more target destinations that were not specified by the sending application.
- Augmenting messages by adding data from a data source.
- Disaggregation of a request into several requests and then aggregation of the responses.

A mediation is defined within a bus. This mediation can then be associated with a destination on the bus. A destination with which the mediation is associated is referred to as a mediated destination.

Foreign bus

A bus can be configured to connect to, and exchange messages with, other messaging networks. A foreign bus is how the service integration bus refers to one of these networks.

A foreign bus encapsulates information related to the remote messaging network, such as the type of foreign bus and whether messaging applications are allowed to send messages to the foreign bus.

When buses are interconnected, applications can send messages to destinations that are defined on other buses.

Foreign bus link

When a foreign bus is configured on a bus, it simply names a foreign bus but does not define a link between the two. In order for the two buses to be able to communicate with each other at runtime, links must be configured between a specific messaging engine within the local bus and a specific messaging engine, or queue manager, within the foreign bus. When configuring a direct service integration bus link, these links must be configured in both directions in order for the two buses to be able to communicate. At runtime, messages that are routed to a foreign bus will flow across the corresponding link.

Exception destinations

If a message cannot be delivered to the target destination, or client, the message will be placed on an exception destination. This ensures that messages are not lost in the event of delivery failure, and enables applications to continue in the event of a corrupt, or poisoned, message.

Further information

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

8.3 J2EE Connector Architecture

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

J2EE Connector Architecture V1.5 support is a requirement of the J2EE V1.4 specification. Resource adapters allow J2EE applications to connect to a particular EIS. The J2EE Connector Architecture specification defines two different types of resource adapters:

- Outbound adapters

  Outbound adapters are used by application-initiated requests to an EIS.

- Inbound adapters

  Inbound adapters are used by the EIS making calls to a message-driven bean.

The J2EE Connector Architecture provides a Common Client Interface API (CCI) with both common and resource adapter specific interfaces. Application programmers code to this single API rather than needing to use different interfaces for each proprietary system. However, it is common for a resource adapter to make use of its own or an existing API, such as JDBC or JMS.

The J2EE Connector Architecture specification provides support for transactions, security, and sharing of connections between different clients.

Advantages of the J2EE Connector Architecture

Some of the advantages of using the J2EE Connector Architecture are:

- Standard for integration legacy systems into a J2EE environment
- Leverage J2EE transactions, security, and resource model
- Leverage the J2EE connection pooling model
- Connectors are reusable J2EE components acting as resources

Disadvantages of the J2EE Connector Architecture

Among the disadvantages of using the J2EE Connector Architecture: Connectors are not available for all legacy systems.

8.4 Service Data Objects

The main aim of Service Data Objects (SDO) is to simplify and unify access, by using a Java API, to heterogeneous data sources including relational databases, XML data sources, Web services, and enterprise information systems. SDO is a generic data model that is self described and has a built-in validation and integrity-checking mechanism.

The Java specification request for SDO is JSR-235, which can be found here:

http://www.jcp.org/en/jsr/detail?id=235

SDO is designed to support:

- Relationship integrity
- Metadata
- Navigation through graphs of data
- Validation
- Static and dynamic APIs
- History tracking

SDO is not intended to replace other data access technologies, but rather to provide an alternate choice. It has the advantage of simplifying the application programming tasks required to access data stores.

The SDO data model consist of *Data Objects* and *Data Graphs*. Metadata is stored in Data Objects as named properties. Values can be primitive types or references to another Data Object. A Data Graph is an envelope for Data Objects, and it represents a normal unit of transport between components.

8.4.1 SDO architecture

Data Objects are the core of the SDO data model. A set of connected Data Objects forms Data Graphs, which keep track of the schema that describes Data Objects and maintains changes made on each node (Data Object) in the graph. Access to the Data Graph object is mediated by a *Data Mediator Service*, as shown in Figure 8-2 on page 145.

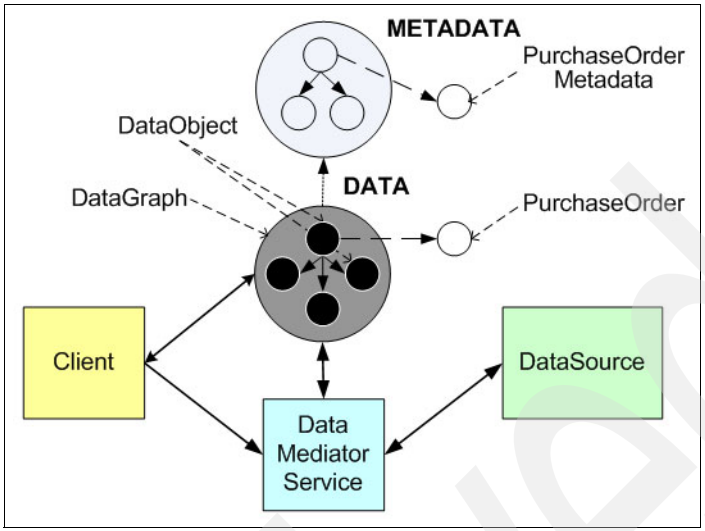

Figure 8-2 Components of the SDO solution

Note: For more information about the SDO technology, refer to the IBM and BEA white paper *Next-Generation Data Programming: Service Data Objects*, November 2003, which you can download from:

`http://ftpna2.bea.com/pub/downloads/commonj/Next-Gen-Data-Programming-Whitepaper.pdf`

Part 3

Scenario implementation

This part describes the following two scenarios from the viewpoint of design guidelines, development guidelines, and runtime guidelines:

- Chapter 9, “Directly Connected homogeneous ESBs” on page 149

  This chapter describes the integration between two ESBs when both are implemented in WebSphere Application Server V6.

- Chapter 10, “Directly Connected heterogeneous ESBs” on page 241

  This chapter describes the integration between an ESB implemented in WebSphere Application Server V6 and WebSphere Business Integration Message Broker V5.

9

Directly Connected homogeneous ESBs

In this chapter, the Enterprise Service Bus (ESB) integration patterns move from concept to practical implementation by applying the Directly Connected ESB Topology pattern between two homogeneous ESB implementations. Using the WS-I sample business scenario, we demonstrate how the Directly Connected ESB pattern can be used to integrate two separate ESBs, each implemented using WebSphere Application Server V6.

In this chapter, the following points are discussed:

- Design guidelines and business needs addressed by the sample scenario, and selection of the relevant ESB integration patterns
- Development guidelines to describe client binding retargeting options
- Runtime guidelines to create and integrate two ESBs, both implemented in WebSphere Application Server

The IBM Enterprise Service Bus strategy:

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

- WebSphere Enterprise Service Bus V6

  Delivers an ESB with Web services connectivity and data transformation.

- WebSphere Message Broker V6

  Delivers an advanced ESB with universal connectivity and data transformation.

When this book was written, WebSphere Enterprise Service Bus was not generally available. In lieu of this product, the service integration bus of WebSphere Application Server V6 is used in this chapter to build ESB solutions.

For more information about the IBM ESB strategy, see:

`http://www.ibm.com/software/info1/websphere/index.jsp?tab=landings/esb`

9.1 Design guidelines

This section discusses sample business needs for linking two ESBs that belong to different organizations. It maps these business requirements to the sample scenario and to the appropriate ESB integration patterns.

9.1.1 Business scenario

The business scenario implemented in this chapter represents a variation of the WS-I sample business scenario as defined in Chapter 7, “The business scenario used in this book” on page 117. It defines a supply chain management process that is split across two organizations, as seen in Figure 9-1.

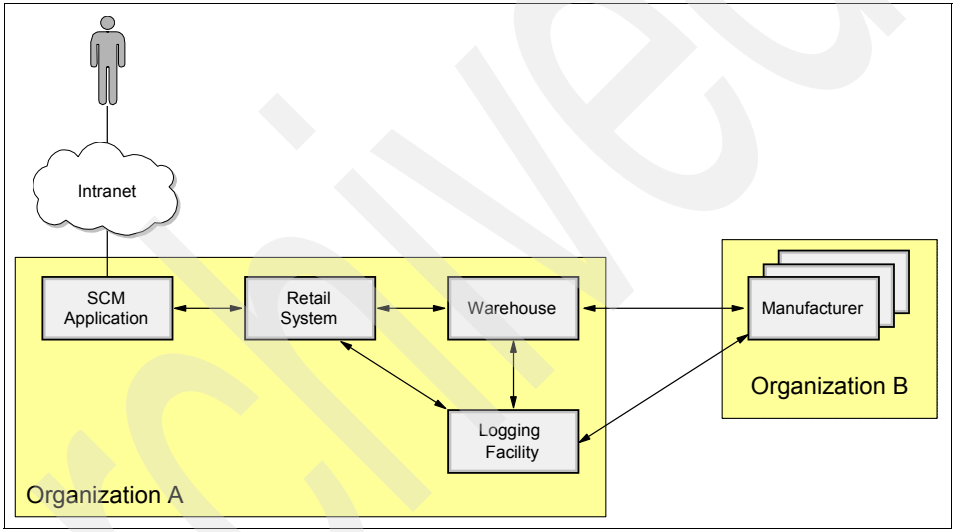

Figure 9-1 High-level business context showing the existing infrastructure

High-level business context

In this scenario, customers access an electronics retailer’s Web site, review a catalog of available products, and place orders for items such as televisions, DVD players, and video cameras. The retailer system requests fulfilment of a consumer’s order from the internal company warehouse, which responds as to whether line items from the order can be provided. If stock for any line item falls below a minimum threshold in the warehouse, the company needs to send a replenishment order to an external manufacturer.

The electronics retailer has shown steady growth for an extended period, and it is decided that diversification into manufacturing would provide a suitable method for successfully expanding the company further within their business model. After

a rigorous process of selection, and after due diligence has been performed, the acquisition of a manufacturing conglomerate goes ahead. The business drivers hoping to be achieved through the acquisition are as follows:

- Increased volumes of sales due to slicker integration between the IT infrastructures of the two Enterprise Service Buses, including reuse of existing code. There is no prospect of application code being rewritten.
- An expanded customer base for the manufacturer's goods.
- Faster provision of items to the Warehouse when items go out of stock.
- Improved customer satisfaction.

The acquisition of the Manufacturing company means that the electronics retailer organization now contains two separate enterprises, each of which has its own existing Enterprise Service Bus architecture in order to facilitate communications with their separate functional components.

Organizational overview

In this scenario we have two separate enterprises that have recently merged to provide joint marketing, sales, and manufacturing capabilities. The two, previously separate, enterprises have the following characteristics:

- Organization A
  - Internet-based e-commerce systems (SCM application)
  - Retail system
  - Warehouse and delivery
- Organization B
  - Manufacturing capability

Organization A has a number of *commercial off-the shelf* (COTS) packages, and these applications have been linked using the ESB technology in WebSphere Application Server V6.

Organization B has grown through acquisition and now provides the manufacturing capability for three brands of electronic goods that are manufactured by separate divisions. These divisional systems have been linked together using a simple home-grown integration layer written in J2EE, again using an ESB. In the scenario these divisional systems are known as ManufacturerA, ManufacturerB, and ManufacturerC.

Integration of organizations

Because of the new recent merger there is a requirement to link the two organizations to enable:

- Increased revenue through the direct e-commerce sale of the manufactured products.
- Faster fulfillment of out-of-stock items.
- Improved customer satisfaction by being able to meet customer demand.
- Reuse of existing application code, allowing for a fast integration between the two infrastructures.

Both of the organizations are keen to follow industry standards for their supporting IT infrastructure and have made a number of their application modules available as Web services. They are both committed WebSphere Application Server users, and the new, joint enterprise is keen to take advantage of new capabilities available in WebSphere Application Server V6.

The major requirement from the business is to integrate the two organizations' business processes quickly through a reasonably small set of static business services. Longer term it is thought that gaining flexibility in their business processes will become important but for now, getting the two organizations working together has top priority.

Therefore, the business is asking the IT department to deliver simple routing of requests initially between the two organizations, but using integration technology that will enable more dynamic routing in the future. The IT department recognizes this need, but is mindful of future development costs and wants to ensure that their new infrastructure can also be changed very quickly. On their wish list is a requirement for most service changes to be made through configuration changes rather than a lengthy code change process.

Because of the requirement to integrate their infrastructure quickly, the two separate IT organizations have decided to set up a joint implementation team to get their business services integrated together. However, because of the time that it will take to agree any new and invasive governance pattern, it has been decided that the two existing organizations will be responsible for architectural and operational control over the components within its domain.

Figure 9-2 on page 154 shows the two ESBs and their exposed services. The business requirement for this scenario is to integrate these two ESBs.

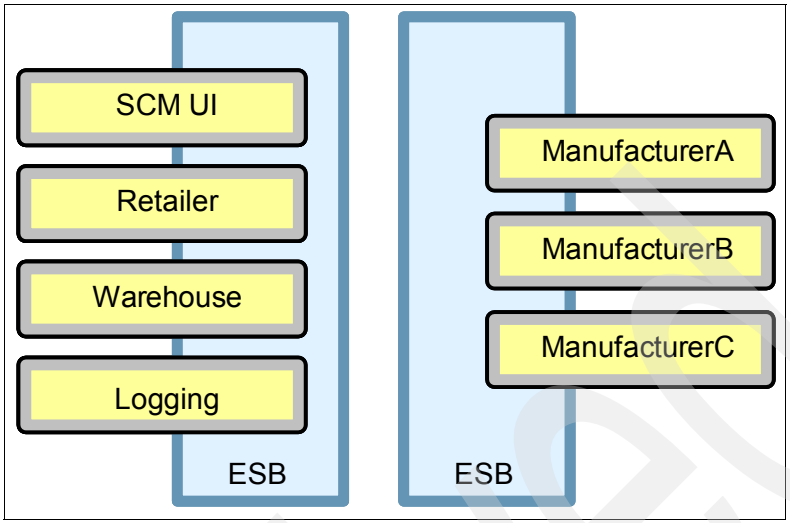

Figure 9-2 The multiple ESBs and their exposed services

9.1.2 Selecting ESB integration patterns

We can apply the ESB integration patterns to the business scenario. The ESB integration patterns are described in more detail in Chapter 6, “Integrating ESBs” on page 87.

Selecting an ESB Topology pattern

The ESB Topology patterns describes network relationships between ESBs. To help us select the appropriate ESB Topology pattern, we should select the business and IT drivers that apply to our scenario, as described in 6.3.1, “ESB Topology patterns overview” on page 90.

Our scenario requires the following business driver:

- Limited interaction between different enterprise governance zones

Our scenario requires the following IT drivers:

- Route requests between two ESBs
- Only basic interactions

Therefore, based on these requirements for a simple, routing-based connection we select the Directly Connected ESBs runtime pattern. This pattern, shown in Figure 9-3 on page 155, describes a simple point-to-point connection between the two ESBs.

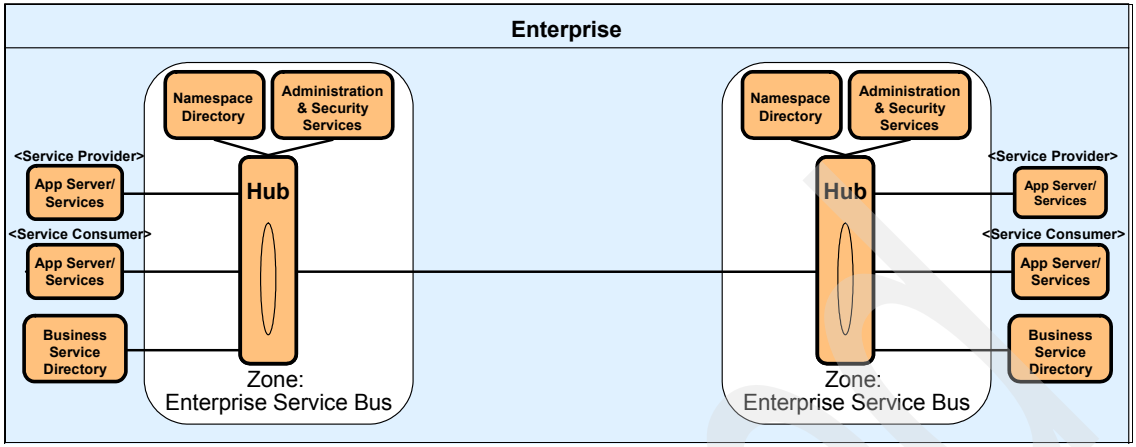

Figure 9-3 Directly Connected ESBs runtime pattern

This simple connection is easy to integrate, as the two ESBs are implemented in homogeneous products. Therefore the Directly Connected ESBs pattern meets the primary business requirement that the integration of organizations is achieved quickly.

Product mapping for the Directly Connected ESBs pattern

Figure 9-4 shows the Product mapping for the Directly Connected ESBs pattern.

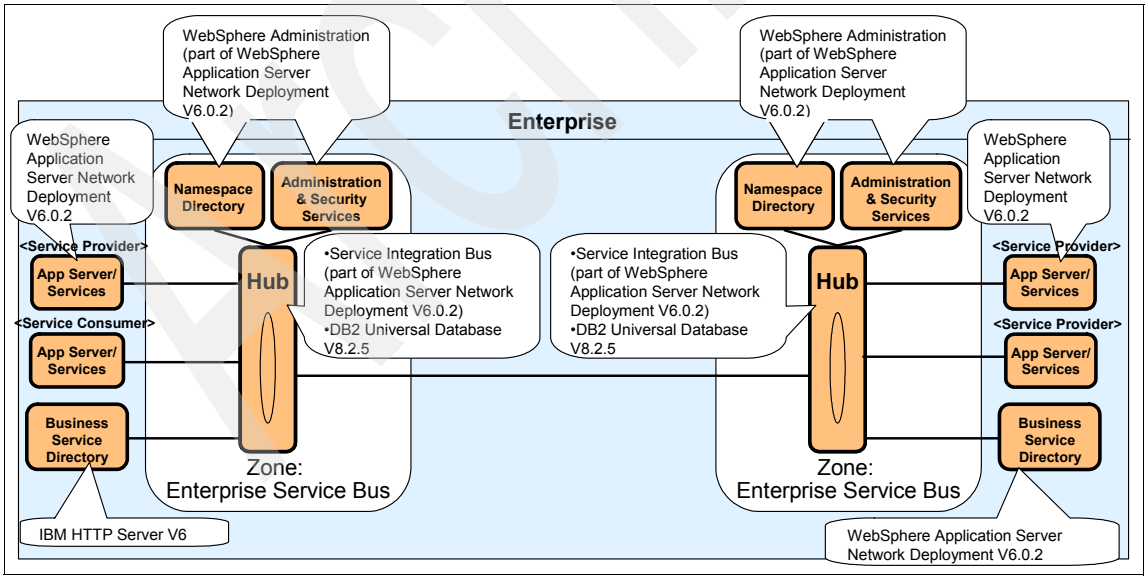

Figure 9-4 Product mapping for homogeneous Directly Connected ESBs

In this Product mapping, the WebSphere Application Server Network Deployment V6.0 product is used to implement most aspects of both ESBs. Two ESBs are deployed in the same cell using the service integration bus component. The service integration bus can be administered through the administration capability of WebSphere Application Server Network Deployment V6. Each service integration bus is hosted on its own WebSphere Application Server node.

Service consumers and providers connect to each ESB using SOAP over HTTP enabled through the service integration bus of WebSphere Application Server Network Deployment V6.0.

The Business Service Directory is supported by an IBM HTTP Server, which is used to host the WSDL descriptions of each Web service used with the ESBs.

The database used needs to support distributed access in a multiple ESB scenario. DB2 Universal Database V8.2 is used to store the Service Data Objects (SDO) repository as it provides robust distributed database capability.

Other ESB integration patterns

The other ESB integration patterns do not apply to this simple scenario. Each ESB in the Directly Connected ESBs pattern is governed separately.

The Direct Connection link between the ESBs uses HTTP. As both ESBs are implemented using the same product set, and both support HTTP connections, neither a Connector or Boundary Service is required to enable this connection.

9.2 Development guidelines

This section discusses the development guidelines for building an integrated ESB solution using two WebSphere Application Server V6 servers. This solution can be built without any changes to development code, therefore this section does not contain any step-by-step instructions.

This section describes:

- How the scenario is implemented as a set of J2EE enterprise applications
- The choices for retargeting Web service client bindings

9.2.1 Scenario implementation

The scenario that this book uses is based on the WS-I sample application. This scenario is described more fully in Chapter 7, “The business scenario used in this book” on page 117.

This book uses a J2EE implementation of this business scenario. Each Web service is implemented as a J2EE enterprise application. Figure 9-5 on page 157 shows these applications and the interactions made between them.

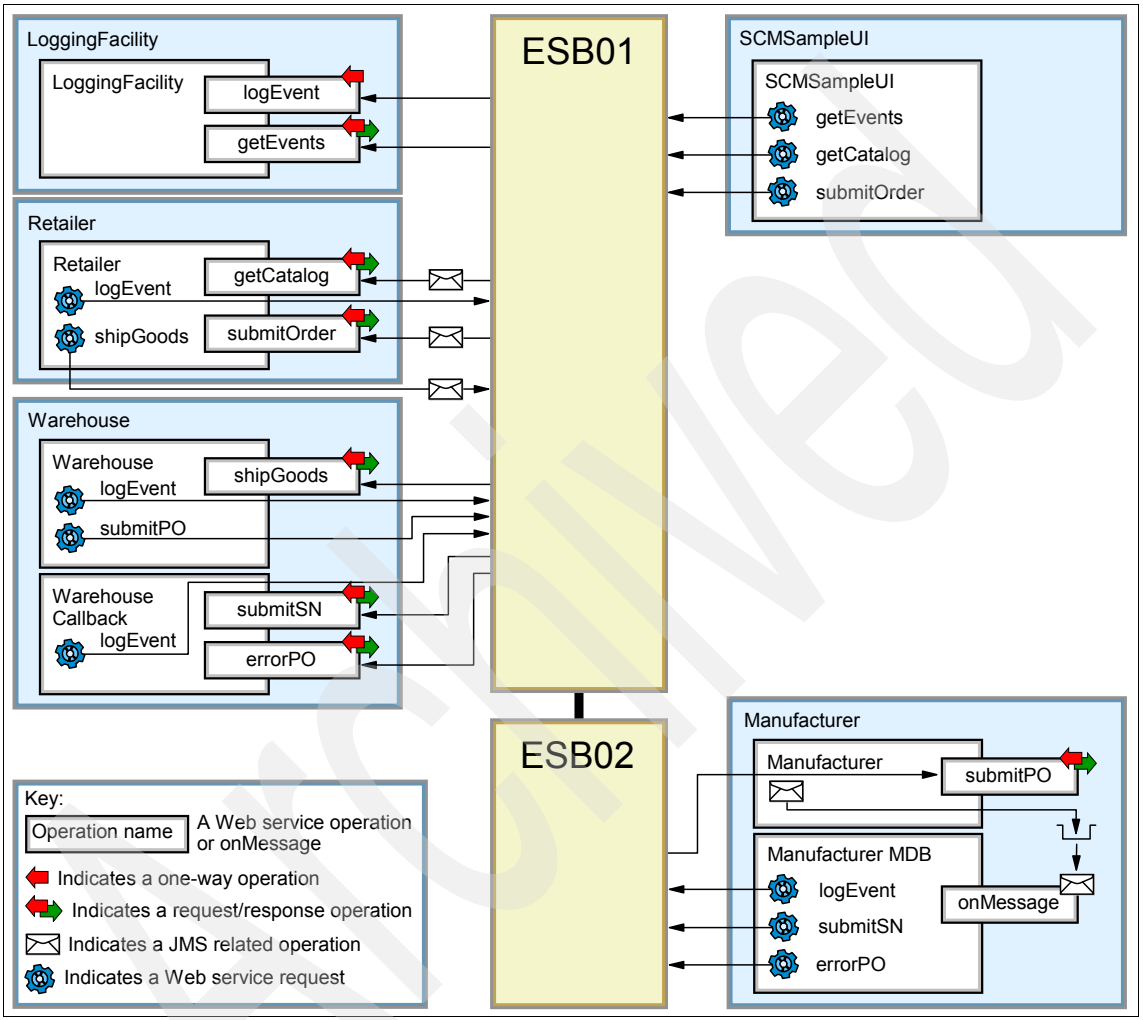

Figure 9-5 WS-I sample application architecture

Figure 9-5 shows how the application has been written and how it interacts with other components. This includes the enterprise applications (blue boxes in the first and third columns), the Web services (white boxes), and the operations (the small white boxes coming out of the larger white boxes). It also shows whether the operation in question is a one-way or request-response Web service interaction or a JMS operation. All arrows connecting cogs to operations indicate

a Web service invocation, and lines decorated by an envelope designate a SOAP over JMS invocation. All other invocations are SOAP over HTTP.

The application interacts as follows:

1. The SCMSampleUI application:
   a. Provides a Web user interface.
   b. Invokes the Retailer Web service to obtain a list of all items that can be purchased.
   c. Invokes the Retailer Web service to order an item.
   d. Invokes the LoggingFacility to track an order.
2. Retailer Web service when order is submitted:
   a. Invokes the LoggingFacility to log events that occur in the order.
   b. Invokes the Warehouse to find out whether the order can be shipped and, if so, have it shipped.
3. When a request to ship goods is made, the Warehouse Web service:
   a. Determines whether there is enough in stock to ship the order.
      - If there is not enough in stock, refuse to ship order
      - If there is enough in stock, ship the order
   b. Determines whether more should be ordered.
      - If more should be ordered, submit a purchase order to the relevant manufacturer.
      - If there is enough in stock, do nothing.
4. When a purchase order is submitted, the Manufacturer Web service sends a JMS message to a queue.
5. The Manufacturer message-driven bean, when triggered:
   a. If the purchase order can be filled
      - Invokes submitSN on the WarehouseCallback.
      - Invokes LoggingFacility to log that the item has been shipped.
   b. If the purchase order cannot be filled
      - Invokes errorPO on the WarehouseCallback.
      - Invokes LoggingFacility to indicate that the item cannot be shipped.
6. The WarehouseCallback Web service, when invoked, calls the LoggingFacility to indicate whether the purchase order was filled or not.

9.2.2 Retargeting Web service client bindings

A Web service client contains a reference to the location of a Web service endpoint. When an ESB is added, Web service clients are retargeting to point to the ESB rather than directly to the Web service endpoint.

There are two different approaches to achieve this Web service retargeting:

- Redevelop the client.

  The Web service client can be rebuilt in a development tool such as Rational Application Developer. This process is described in *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6,* SG24-6494.

  The regeneration of a Web service client will mean that the application using the Web service client will need to go through a testing process if it were to be deployed in a full production environment.

- Modify the endpoint URL.

  When Web service clients are deployed to WebSphere Application Server, the Web service client bindings can be overridden using the WebSphere Application Server administrative console.

  By specifying the name of an endpoint URI that is used to override the current endpoint, a client uses this endpoint instead of the endpoint specified in the WSDL file. If an assembled application contains a Web service client that is statically bound, the client is locked into using the implementation (service end point) identified in the WSDL file used during development. Overriding the endpoint is an alternative to configuring the deployed WSDL attribute. The overridden endpoint URI attribute is specified on a per-port basis. It does not require an alternative WSDL file within the module. The overridden endpoint URI takes precedence over the deployed WSDL attribute. The client uses this value for the service end point URI or SOAP address, instead of the value in the static client bindings.

  The process for overriding the endpoint URL is described in “Editing the client bindings to call inbound services” on page 226.

9.3 Runtime guidelines

The following section provides a high-level overview and summary of the steps required to build both ESB implementations and integrate them, including:

- Software requirements
- Steps to complete the scenario
- Building the WebSphere Application Server Network Deployment infrastructure

- Building the service integration bus infrastructure
- Deploying and building the WS-I scenario
- Testing the scenario

For additional information on the steps described in this section, consult the following documents:

- *Patterns: Implementing an SOA using an Enterprise Service Bus in WebSphere Application Server*, SG24-6494
- *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451
- "Configure a Service Integration Bus In a network deployment environment"

  http://www.ibm.com/developerworks/webservices/library/ws-sibus/
- WebSphere Application Server V6 information center (section on the service integration bus)

  http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.pmc.nd.doc/concepts/cjw_epl_port.html

9.3.1 Software requirements

This scenario requires the following levels of software:

- WebSphere Application Server Network Deployment V6
- WebSphere Application Server Version 6.0 Refresh Pack 2, also known as Version 6.0.2
- IBM HTTP Server powered by Apache V6
- DB2 Universal Database V8.2 with FixPack 5

Note: In our example, we installed WebSphere Application Server Network Deployment into c:\WAS\AppServer and *not* the default location of c:\Program Files\WebSphere\AppServer. This was because there is a limit on file names of around 260 characters, which prevents J2EE applications from installing.

This scenario is logically built on two host systems: one called ITSOESB01.itso.ral.ibm.com and one called ITSOESB02.itso.ral.ibm.com. These two host systems can be physically located on a single machine, or across two machines with a network connection. The instructions in this chapter assume that both systems will be located on the same machine, but the instructions can equally be applied to using two host systems.

This scenario assumes that although WebSphere Application Server Network Deployment is installed, neither the deployment manager profile or application server profiles have been created.

9.3.2 Steps to complete the scenario

This chapter describes how to build the ESB infrastructure described in the business scenario and how to integrate the ESBs using the Directly Connected ESB pattern.

This section summarizes the implementation steps required to implement this solution. The remainder of the chapter contains step-by-step instructions for implementing the scenario.

Building the WebSphere Application Server Network Deployment infrastructure

This section describes how to create deployment manager and application server profiles for WebSphere Application Server Network Deployment V6, as well as the required network host file changes, in these sections:

- Creating a hosts file
- Starting the profile creation wizard
- Creating the deployment manager profile
- Creating the application server profiles

Building the service integration bus infrastructure

This section describes the configuration necessary in both WebSphere Application Server instances relating to the service integration bus (used to perform the majority of the ESB functionality in this scenario). It describes how to create, configure, and link service integration buses:

- Setting up the messaging engine repositories
- Setting up the SDO repository
- Setting up the messaging engines and SDO repositories
- Installing the SDO repository application
- Installing service integration bus applications and resources
- Installing endpoint listener applications
- Creating a service integration bus
- Adding a bus member
- Creating a second service integration bus and bus member
- Creating a foreign bus
- Creating the mirror foreign bus
- Creating a service integration bus link
- Creating a mirror service integration bus link

Deploying and building the WS-I scenario

This section describes how to deploy the WS-I sample scenario into each of the WebSphere Application Server servers. It describes the messaging support required by the Manufacturer J2EE enterprise applications, the creation of inbound and outbound services in the service integration buses, and modifying Web service client bindings to use the ESB. It contains the following sections:

- Creating the destinations
- Creating a JMS connection factory
- Creating the JMS queues
- Creating the JMS activation specifications
- Hosting the WSDL files
- Modifying a virtual host for ITSOESB02.itso.ral.ibm.com
- Installing the applications
- Creating the endpoint listeners
- Creating the outbound services
- Routing service requests between buses
- Creating inbound services
- Editing the client bindings to call inbound services

Testing the scenario

This section contains step-by-step instructions for testing the application by running the SCMSampleUI Web application.

9.3.3 Building the WebSphere Application Server Network Deployment infrastructure

This scenario requires an infrastructure with two nodes, and a WebSphere Application Server Network Deployment server instance running in each node, as shown in Figure 9-6 on page 163.

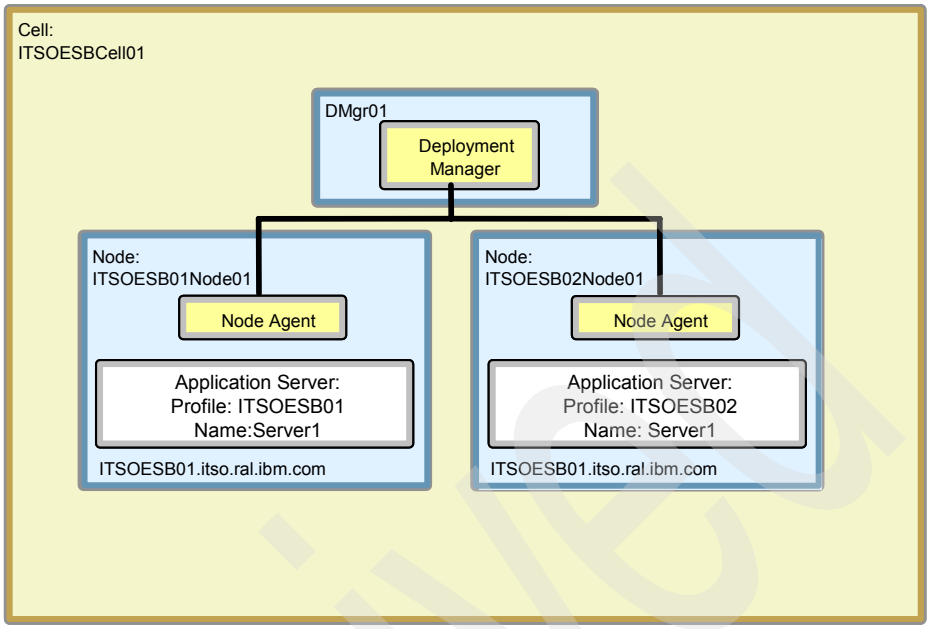

Figure 9-6 WebSphere Application Server Network Deployment infrastructure

Note the following characteristics shown in Figure 9-6:

- The cell for the entire infrastructure is called `ITSOESBCell01`.
- The cell contains a deployment manager with a profile name of `Dmgr01`.
- Two nodes are registered with the deployment manager: `ITSOESB01Node01` and `ITSOESB02Node01`.
- Each node contains a single application server. There are two application servers in total, with the profile names of `ITSOESB01` and `ITSOESB02`.
- Each node and application server pair run on their own logical machine which are assigned IP host names of `ITSOESB01.itso.ral.ibm.com` and `ITSOESB02.itso.ral.ibm.com`.
- The deployment manager also runs on `ITSOESB01.itso.ral.ibm.com`.

This section describes how to build this infrastructure. We provide step-by-step instructions for building this infrastructure on both a single machine and on two separate machines with their own IP addresses.

For more in-depth instructions about building WebSphere Application Server Network Deployment infrastructures, refer to *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

Creating a hosts file

The guidelines in this chapter use the host names ITSOESB01.itso.ral.ibm.com and ITSOESB02.itso.ral.ibm.com. You need to map these host names to the actual machine or machines you are using.

In a Windows environment this can be achieved by modifying the hosts file with a standard text editor:

1. Navigate to <Windows_home>\system32\drivers\etc and open the **hosts** file in a text editor.
2. Add the following entries to map the host names required by the scenario to your local machine:

```
127.0.0.1    ITSOESB01.itso.ral.ibm.com
127.0.0.1    ITSOESB02.itso.ral.ibm.com
127.0.0.1    appsrv1a.itso.ral.ibm.com
```

> **Note:** appsrv1a.itso.ral.ibm.com is required by the WS-I sample scenario to locate WSDL files from an HTTP server. You will configure the HTTP server in a later step.

3. Save the file. You should now be able to **ping** any of these host names and have them resolve to your local machine as shown in Example 9-1.

Example 9-1 Ping of ITSOESB01.itso.ral.ibm.com

```
C:\>ping ITSOESB01.itso.ral.ibm.com

Pinging ITSOESB01.itso.ral.ibm.com [127.0.0.1] with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
```

> **Note:** If you choose to use two separate machines to implement this scenario, you should define the following.
>
> In the hosts file of the first machine:
>
> ```
> 127.0.0.1    ITSOESB01.itso.ral.ibm.com
> 127.0.0.1    appsrv1a.itso.ral.ibm.com
> ```
>
> In the hosts file of the second machine:
>
> ```
> 127.0.0.1    ITSOESB02.itso.ral.ibm.com
> 127.0.0.1    appsrv1a.itso.ral.ibm.com
> ```

Starting the profile creation wizard

The first steps for creating a profile are the same regardless of the type of profile you will create. (This section specifically creates the deployment manager profile.) You can start the profile creation wizard in one of the following ways:

- From the Start menu in Windows only, select:

  Start → Programs → IBM WebSphere → Application Server Network Deployment v6 → Profile creation wizard

- Use the platform-specific command in the *<was_home>*/bin/ProfileCreator directory:
  - Windows: **`pctWindows.exe`**
  - AIX®: **`pctAIX.bin`**
- Check the box directly after installation from the install wizard to launch the profile creation wizard.

Creating the deployment manager profile

This section summarizes the steps required to create the deployment manager profile.

Attention: These steps must be completed on the ITSOESB01.itso.ral.ibm.com host system.

1. Start the profile creation wizard as described above. Click **Next** on the Welcome window to move to the Profile type selection screen.
2. Select **Create a deployment manager profile** and click **Next.**
3. Type the profile name `Dmgr01`. Click **Next** to continue**.**
4. Accept the default location and click **Next.**

5. For this scenario we simplify the node and cell names. Specify the following values (Figure 9-7 on page 166):

Node name	`ITSOESBCellManager01`
Host name	`ITSOESB01.itso.ral.ibm.com`
Cell name	`ITSOESBCell01`

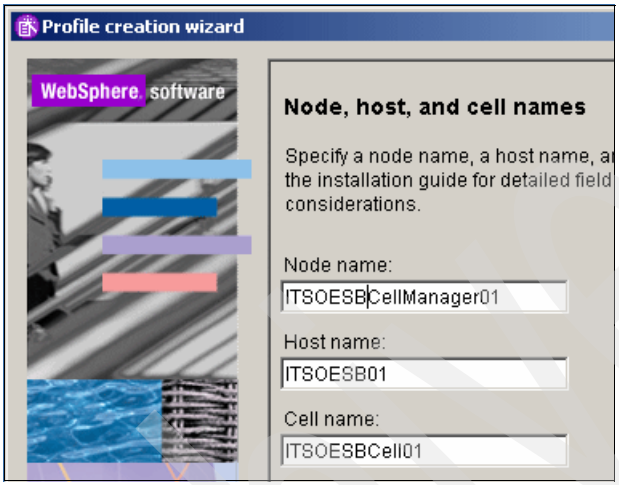

Figure 9-7 Profile create wizard: setting the node, host, and cell names

6. Accept the default port assignments. Click **Next.**

7. Choose whether to **Run the application server process as a Windows service** and click **Next.**

> **Tip:** As we had no requirement to start/stop the server through the Microsoft Management Console, we decided *not* to Run the application server process as a Windows service.

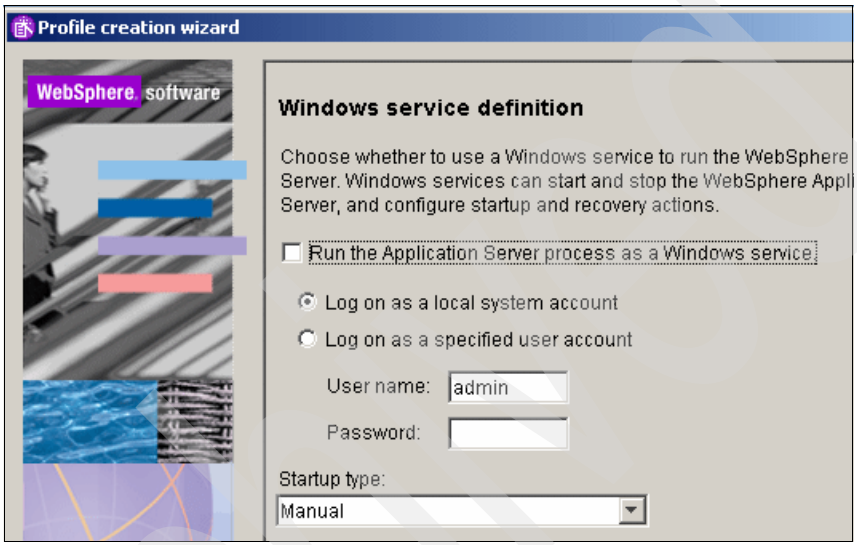

Figure 9-8 Profile create wizard: Windows service definition

8. Click **Next** on the Profile Summary window to create the deployment manager profile.
9. When the profile wizard informs you that the profile creation is complete, de-select the **Launch First steps console** check box and click **Finish.**
10. After the deployment manager profile has been created, start the manager. On the deployment manager machine:
    a. Change the directory to the *<profile_home>*/bin directory of the Network Deployment installation.
    b. Use the **`startManager`** command to start the deployment manager.

    If you are successful, you will see the process ID for the deployment manager process displayed in the command window, as shown in Example 9-2.

Example 9-2 Starting the deployment manager from the command line

```
C:\WAS\AppServer\profiles\Dmgr01\bin>startmanager
ADMU0116I: Tool information is being logged in file
           C:\WAS\AppServer\profiles\Dmgr01\logs\dmgr\startServer.log
ADMU0128I: Starting tool with the Dmgr01 profile
ADMU3100I: Reading configuration for server: dmgr
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server dmgr open for e-business; process id is 1544
```

Creating the application server profiles

Two WebSphere Application Server application servers are required: one for the ITSOESB01 host system and one for ITSOESB02. Create these application servers by generating profiles from the deployment manager, then federate their nodes to a cell:

1. Start the profile creation wizard, as described in “Starting the profile creation wizard” on page 165.
2. When you start the wizard, you first see the Welcome window. Click **Next** to move to the Profile type selection screen.
3. Select **Create an application server profile** and click **Next.**
4. Type the profile name `ITSOESB01`. You may choose to select the **Make this profile the default** check box, but this will depend on how you plan to manage the configuration after it is installed. Click **Next** to continue**.**

   Tip: The default profile is the default target for commands issued from the bin directory in the product installation root.

5. Accept the default location and click **Next.**
6. Accept the defaults for node and host names. Click **Next**.
7. Accept the default port assignments. Click **Next.**
8. Choose whether to **Run the application server process as a Windows service** and click **Next.**

   Tip: We decided *not* to run the application server process as a Windows service because we will not use the Microsoft MMC Services interface to start and stop our servers.

9. Click **Next** on the Profile Summary to create the application server profile.
10. When the profile wizard informs you that the profile creation is complete, de-select the **Launch First steps console** check box and click **Finish.**

After the application server profile has been created, you are ready to federate the node to the cell. In our scenario we federated the node using the command line. Perform the following steps:

1. Ensure that the deployment manager is started.
2. Open a command window on the system where you created the application server profile for this node. Switch to the *<profile_home>*\bin directory by typing:

   ```
   cd WAS\appserver\profiles\ITSOESB01\bin
   ```

3. Run the **addNode** command with the host name of the deployment manager and potentially the SOAP connector port number in this form:

   ```
   addNode <dmgrhost> <dmgr_soap_port>
   ```

   To add ITSOESB01 to the deployment manager, enter:

   ```
   addNode ITSOESB01.itso.ral.ibm.com
   ```

   Example 9-3 shows the sample output.

Example 9-3 Adding the node

```
C:\WAS\AppServer\profiles\ITSOESB01\bin>addNode ITSOESB01.itso.ral.ibm.com
ADMU0116I: Tool information is being logged in file
C:\WAS\AppServer\profiles\ITSOESB01\logs\addNode.log
ADMU0128I: Starting tool with the ITSOESB01 profile
ADMU0001I: Begin federation of node ITSOESB01Node01 with Deployment Manager at
ITSOESB01.itso.ral.ibm.com:8879.
ADMU0009I: Successfully connected to Deployment Manager Server:
ITSOESB01.itso.ral.ibm.com:8879
...
...
ADMU0003I: Node ITSOESB01Node01 has been successfully federated.
```

4. Repeat the previous steps to create a second host system called ITSOESB02. This host system could be created on a separate physical machine from the deployment manager and the ITSOESB01 profiles, or it can be hosted on the same host system.
   a. Use the profile creation wizard to create an application server profile, specifying the following values:

      Profile name ITSOESB02

      Node name ITSOESB02Node01

      Host name ITSOESB02.itso.ral.ibm.com

   b. After the profile is created, federate it by changing to the bin directory of the profile and running the **addNode** command:

      ```
      addNode ITSOESB02.itso.ral.ibm.com
      ```

9.3.4 Building the service integration bus infrastructure

Now that the WebSphere Application Server Network Deployment cell is configured you can start to build the service integration bus infrastructure.

The service integration bus installation and configuration is a post-install exercise that requires a certain amount of planning. Two data stores are required for service integration bus operation:

- The SDO repository, which holds the registered service WSDL information.
- The messaging engine repository/repositories. The underlying messaging layer requires persistence of the internal message data format.

These data stores are created, and their use is transparent to the user of a stand-alone server, as the server and databases are all resident on the same machine. For a network-deployed implementation, it is necessary for each of the servers within the cell that participate on a service integration bus to have access to the data stores, and as such this cannot be configured transparently by the service integration bus install process. The administrator needs to configure this prior to installation.

Figure 9-9 displays the architecture deployed for this scenario showing that two service integration buses are created and linked. Each service integration bus has a single bus member (application server) connected via its messaging engine.

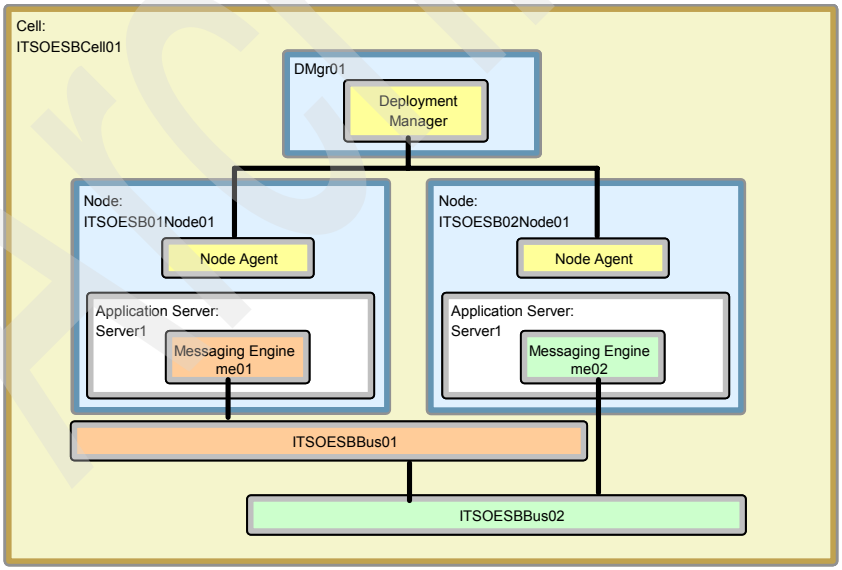

Figure 9-9 Infrastructure of buses and messaging engines

The following steps are required to set up the service integration bus infrastructure.

Set up the messaging engine repositories

An individual message store table is required for each of the servers shown in Figure 9-9 on page 170. WebSphere Application Server automatically handles the table and schema creation. Therefore, you can either set up separate databases for each server or set up a single database with separate schemas for each server. In our example, we set up separate databases named `me01` and `me02`.

1. From a DB2 Command Window on host system ITSOESB01.itso.ral.ibm.com enter the following commands:

```
db2 create database me01

db2 create database me02
```

Note: Individual schemas for each messaging engine and tables are created when the messaging engines are created.

Set up the SDO repository

1. The Service Data Objects (SDO) repository is used to store and serve WSDL definitions for the service integration bus. The SDO repository supports a wide variety of databases. By default the SDO repository uses embedded Cloudscape but in this scenario we use DB2 Universal Database.

   Create a database named `sdodb`:

   a. From the same DB2 Command Window, enter:

```
db2 create database sdodb

db2 connect to sdodb

db2 create schema sdorep

db2 create table sdorep.bytestore (name varchar(250) not null, bytes
blob(1G), timestamp1 bigint not null)

db2 disconnect sdodb
```

2. Now with the databases and tables created, configure connectivity to them from WebSphere Application Server. On each of the host systems a homogenous directory structure should be created into which the client JAR files for DB2 Universal Database should be placed. This is to ensure that WebSphere Application Server can connect to the database through the drivers supplied by the database. In our example, `db2jcc.jar`,

`db2jcc_license_cu.jar`, and `db2jcc_license_cisuz.jar` are placed in a new directory named `c:/WAS/AppServer/db2udb`.

a. From a Command Prompt or from Windows Explorer, navigate to **c:\WAS\AppServer** and create a subdirectory named `db2udb`.

b. Copy these three JAR files from C:\Program Files\IBM\SQLLIB\java to c:\WAS\AppServer\db2udb:

```
db2jcc.jar
db2jcc_license_cu.jar
db2jcc_license_cisuz.jar
```

Set up the messaging engines and SDO repositories

1. The data source used by the SDO repository requires a component-managed authentication alias, which is used to allow the same user ID and password combination to be used in many different places. In this case the DB2 database has security configured, so specify the same user ID and password as created during the DB2 install. Complete these steps to create an alias:

   a. Access the admin console at the following URL and log in.

   ```
   http://ITSOESBO1.itso.ral.ibm.com:9060/ibm/console
   ```

   b. In the navigation panel, click **Security** → **Global Security**

   c. Under **Authentication**, expand **JAAS Configuration** and click **J2C Authentication data** → **New**. (See Figure 9-10.)

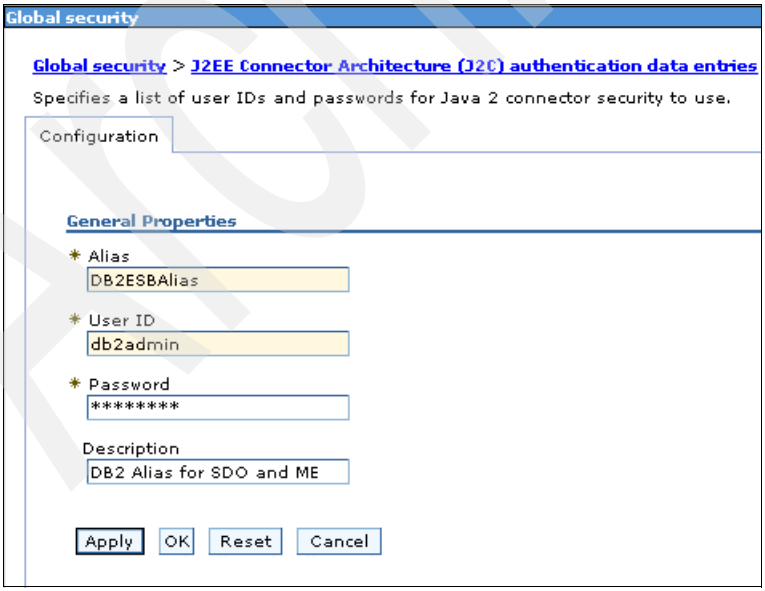

Figure 9-10 Setting the J2C authentication data

– **Alias**

The name by which this alias will be known in the admin console.

In this case we specify the name `DB2ESBAlias` but it can be anything you like.

– **User ID**

The user ID that will be used to log in. A value must be specified.

Specify the same value as the ID created when installing DB2. In this case we specify the name `db2admin`

– **Password**

The password associated with the user ID. A value must be specified.

Specify the same value as the password created when installing DB2.

d. Click **OK** and save the changes to the master configuration by clicking **Save**.

2. Create a JDBC provider for IBM DB2 Universal Database V8.2.

The next step is to configure the service integration buses to access the SDO repository database using DB2. To do this, you must define a new JDBC provider. Because the JDBC provider will be used by multiple (two) service integration buses running on separate nodes we decided to create the JDBC provider at the Cell Scope. In the administration console, complete the following steps to create a JDBC provider:

a. In the navigation panel click **Resources** → **JDBC Providers**.

b. Clear the Node entry field and click **Apply** (Figure 9-11).

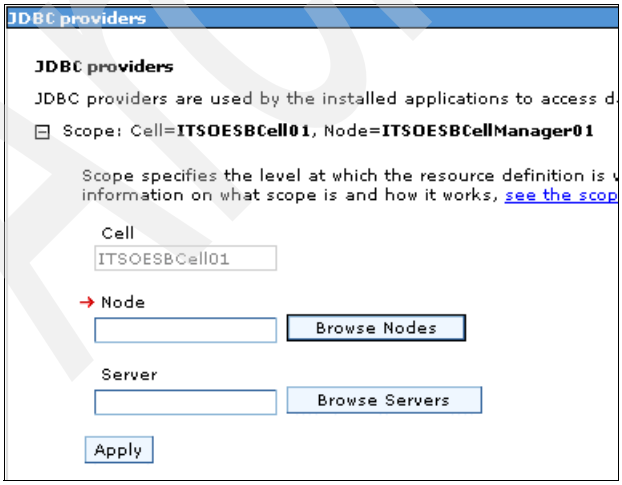

Figure 9-11 Setting the scope to Cell level

c. With Cell scope now applied, on the same window click **New** to create the JDBC provider.

d. Enter some general information about the type of database and the connection mechanism as shown in Figure 9-12. Note that the pull-down boxes will be disabled until the values in the preceding boxes have been filled in.

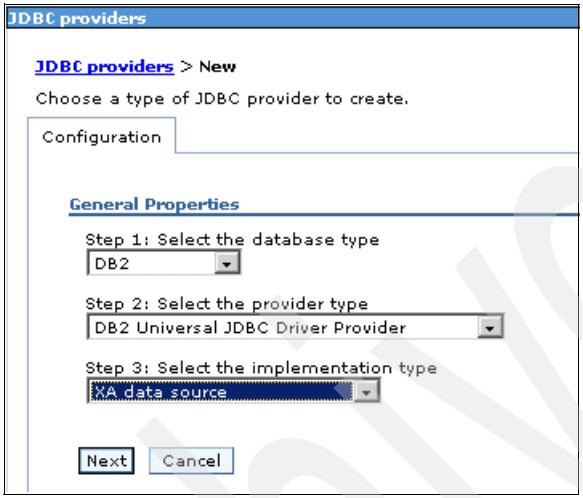

Figure 9-12 Specifying properties for DB2 JDBC provider

Select the database type: This is used to specify the type of database the JDBC provider will connect to. In this case choose **DB2**.

Select the provider type: This is used to specify how the database will be accessed. In this case choose **DB2 Universal JDBC Driver Provider**.

Select the implementation type: This is used to specify how the provider will be implemented. In this case choose **XA data source**.

e. Click **Next**.

f. Modify the following JDBC provider properties panel:

   i. Remove the existing Class path entries and add the three DB2 classes you copied to the c:\WAS\AppServer\db2udb directory in the previous section.

   ii. Clear the **Native library path** field.

   iii. The completed panel is shown in Figure 9-13 on page 175. Accept all other defaults and click **OK**.

Note: The screen capture below is restricted to showing two of the three DB2 classes. Ensure you enter all three.

Also, ensure that you enter the class path entries using the *forward* slash (/) to delimit directories.

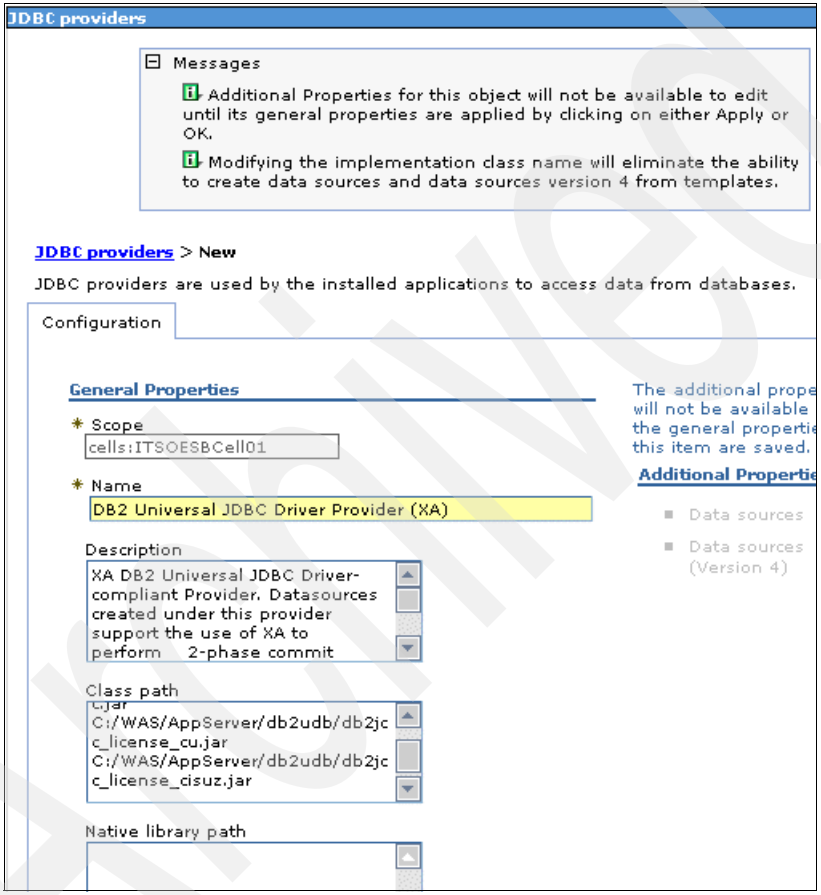

Figure 9-13 Specifying additional properties for DB2 JDBC provider

g. Save the changes to the master configuration by clicking **Save**.

3. Create the JDBC data sources.

   The next step is to create the JDBC data sources for accessing the SDO and messaging engine tables in DB2. Assuming that you are at the JDBC

providers panel from the preceding step, complete the following steps to create the JDBC data sources:

a. Select **DB2 Universal JDBC Driver Provider (XA)** → **Data sources** → **New**.

b. Enter the details for the new data source used for the SDO repository. The panel is a long, scrolling, browsing panel. The top half should look like Figure 9-14.

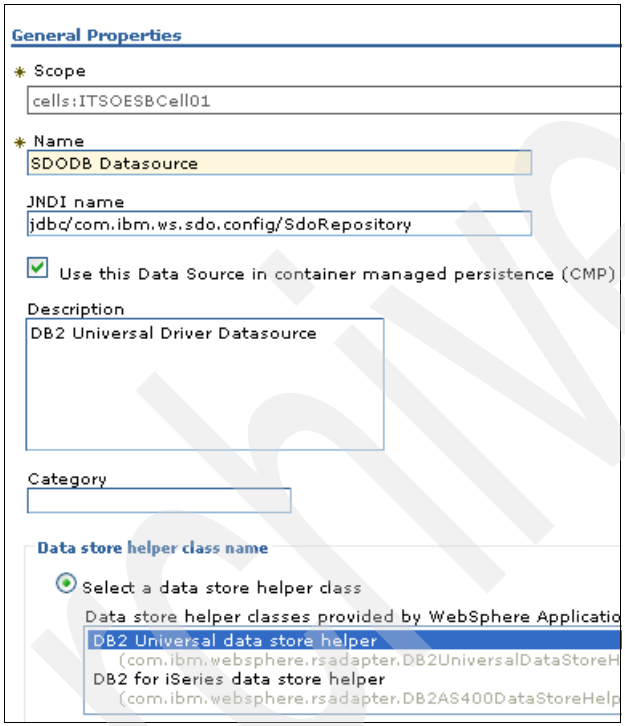

Figure 9-14 Specifying the SDO data source parameters: part 1

The bottom half of the same panel should look like Figure 9-15 on page 177.

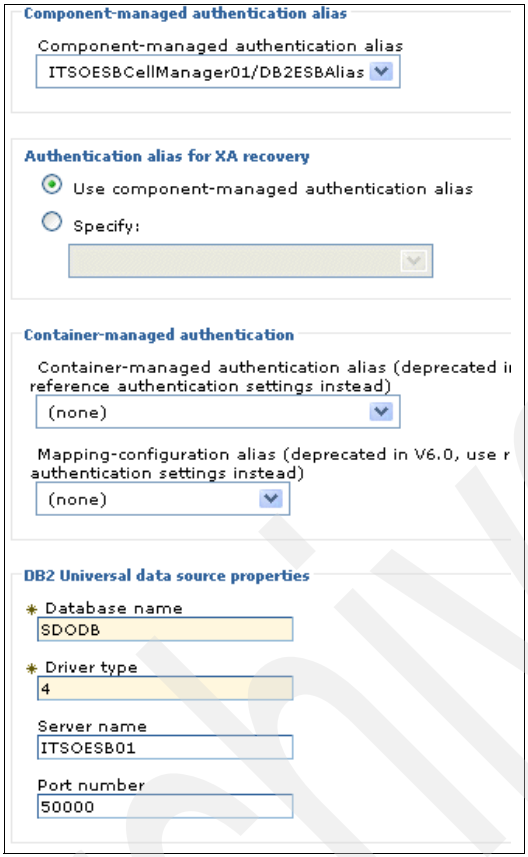

Figure 9-15 Specifying the SDO data source parameters: part 2

- **Name**

  The name is just an administrative entity that has meaning only within the administrative console.

  This can be specified as anything you like. In our example we used `SDODB Datasource`

- **JNDI Name**

  Applications pick up the data source from the JNDI name.

  Specify a value of `jdbc/com.ibm.ws.sdo.config/SdoRepository`

- **Component-managed authentication alias**

  The alias to be used when making connections to the database where the application-managed authentication is being used by the application.

  Select the value that ends in **DB2ESBAlias**.

– **DB2 Universal data source properties**

The path to the SDO DB2 database.

i. Database name

The name of the SDO database. Enter `SDODB`**.**

ii. Driver type

Change to driver type `4`**.**

> **Note:** Type 4 JDBC drivers are direct-to-database pure Java drivers (“thin” drivers).

iii. Server name

The host name where the DB2 server is running. Enter `ITSOESB01.itso.ral.ibm.com`

c. Leave all other values as default. Click **OK.**

d. Create the data sources for the two messaging engines following the same procedure, using the modifications below:

i. First messaging engine data source:

Name	me01 datasource
JNDI Name	jdbc/com.ibm.ws.sib/me01
Database Name	me01

ii. Second messaging engine data source:

Name	me02 datasource
JNDI Name	jdbc/com.ibm.ws.sib/me02
Database Name	me02

e. Save the changes to the master configuration by clicking **Save.**

f. To ensure that the data sources have been created successfully, check all three datasources and click **Test connection** from the **JDBC providers** → **DB2 Universal JDBC Driver Provider (XA)** → **Data sources** panel. You should see confirmation of three successful connections.

Installing the SDO repository application

The service integration bus Web services support stores the WSDL and schemas for the Web services in the SDO repository. When WebSphere Application Server is installed, it does not install the SDO repository. This must be installed manually and is completed using the installSdoRepository.jacl script.

The SDO repository is backed by a database and can use a wide variety of databases. In our scenarios we use DB2 Universal Database Enterprise Edition.

1. Open a Command Prompt window on your deployment manager host system, in this example on ITSOESB01. The installSdoRepository.jacl script is provided in the <*install_root*>/bin directory, where install_root is the root directory for the installation of WebSphere Application Server. In our example, navigate to c:\WAS\appserver\bin.

   Example 9-4 shows the format of the installSdoRepository script.

Example 9-4 Format of the installSdoRepository script

```
<install_root>/bin/wsadmin.sh
   -conntype SOAP
   -port 8879
   -f $<install_root>/bin/installSDORepository.jacl
   <nodename>
   <server>
```

   If you are running on the deployment manager host system with default port settings, no parameters are required. On a host system that does not host the deployment manager, the nodename and server parameters may be required:

   nodename ITSOESBCellManager01 (the deployment manager node name)

   server dmgr (the deployment manager server name)

2. Run the command (partial output shown in Example 9-5):

   ```
   wsadmin -f installSDORepository.jacl ITSOESBCellManager01 dmgr
   ```

Example 9-5 Installing the SDO repository application

```
C:\WAS\AppServer\bin>wsadmin -f c:/WAS/AppServer/bin/installSDORepository.jacl
ITSOESBCellManager01 dmgr
WASX7209I: Connected to process "dmgr" on node ITSOESBCellManager01 using SOAP
connector;  The type of process is: DeploymentManager
WASX7303I: The following unrecognized options are passed to the scripting
environment and are available as argument that is stored in the argv variable:
"[ITSOESBCellManager01, dmgr]"
CWSJO0016I: Installing SDO repository on node ITSOESBCellManager01, server
dmgr.
CWSJO0021I: Installing SDO repository EJB application.
CWSJO0052I: Using datasource backend ID: CLOUDSCAPE_V5
...
...
ADMA5013I: Application SDO Repository installed successfully.
CWSJO0022I: Saving configuration.
CWSJO0023I: SDO repository installation completed successfully.
```

Note: The output from the installSDORepository.jacl script indicates that the SDO will use Cloudscape as the backend data source, although we use DB2. Do not worry about this as the backend data source will be changed in the next step.

3. Change the SDO data source backend type. A list of backend database types can be found in directory <install_root>/util/SDORepository:

```
wsadmin -f installSDORepository.jacl -editBackendId DB2UDB_V82
```

4. Finally, the SDO repository application has to be installed on all application servers that will use a service integration bus. Run the following commands:

```
wsadmin -f installSdoRepository.jacl ITSOESB02Node01 server1
wsadmin -f installSdoRepository.jacl ITSOESB01Node01 server1
```

Installing service integration bus applications and resources

A service integration bus is a logical entity that has a physical manifestation in the form of several enterprise applications, a resource adapter, and an activation specification. As part of the WebSphere Application Server post-install process, these applications must be installed and started before any service integration bus creation and configuration activities can take place. The application installation is achieved by using a JACL script that is shipped with WebSphere Application Server in the <install_root>/util directory, and it is called sibwsInstall.jacl.

The sibwsInstall.jacl script must be run multiple times: once for every application and resource required by the service integration bus support.

1. Before you can install any resources, use the **startNode** tool to start the ITSOESB01 and ITSOESB02 profiles. In the <install_root>/bin directory, run the following two commands:

```
startNode -profileName ITSOESB01
startNode -profileName ITSOESB02
```

2. Install the resource adapter. This must be installed before the other resources and is required. To install the resource adapter, navigate to the <install_root>/bin directory and execute the following command:

```
wsadmin -f <install_root>/util/sibwsInstall.jacl INSTALL_RA -installRoot
<install_root> -nodeName NODE_NAME
```

<install_root> is the directory in which you installed WebSphere Application Server, and NODE_NAME is the name of the application server node.

Important: The second <install_root> must have elements in the path separated by a forward slash (/) even on Windows systems. So a path of c:\WAS\AppServer becomes c:/WAS/AppServer.

In our example the commands entered were as shown in Example 9-6.

Example 9-6 Install the service integration bus resource adapter

```
wsadmin -f c:/was/appserver/util/sibwsInstall.jacl INSTALL_RA -installRoot
"C:/WAS/AppServer" -nodeName ITSOESB01Node01

wsadmin -f c:/was/appserver/util/sibwsInstall.jacl INSTALL_RA -installRoot
"C:/WAS/AppServer" -nodeName ITSOESB02Node01
```

Note: The output from the commands above and similar JACL scripts will show an informational message, WASX7303I, suggesting unrecognized options. This is normal.

3. Install the Web services support application onto each server. This can be done from the <install_root>/bin directory by executing the following command:

   ```
   wsadmin -f <install_root>/util/sibwsInstall.jacl INSTALL -installRoot
   <install_root> -nodeName NODE_NAME -serverName server1
   ```

   <install_root> is the directory in which you installed WebSphere Application Server; the second <install_root> uses forward slashes rather than back slashes; and NODE_NAME is the name of the application server node.

   In our example the commands entered were as shown in Example 9-7.

Example 9-7 Install the service integration bus enterprise application

```
wsadmin -f c:/was/appserver/util/sibwsInstall.jacl INSTALL -installRoot
"C:/WAS/AppServer" -nodeName ITSOESB01Node01 -serverName server1

wsadmin -f c:/was/appserver/util/sibwsInstall.jacl INSTALL -installRoot
"C:/WAS/AppServer" -nodeName ITSOESB02Node01 -serverName server1
```

Installing endpoint listener applications

Install the endpoint listener applications for HTTP and JMS to all service integration bus servers.

While the Web services enterprise application support has now been installed it is not until an endpoint listener application is installed that it can be used. There are two different endpoint listener applications, one for SOAP over HTTP and

one for SOAP over JMS. For this scenario we install only the SOAP over HTTP application to the servers at this point. The HTTP endpoint listener is installed using the following command:

```
wsadmin -f <install_root>/util/sibwsInstall.jacl INSTALL_HTTP -installRoot
<install_root> -nodeName NODE_NAME -serverName server1
```

The JMS endpoint listener is installed using the following command:

```
wsadmin -f <install_root>/util/sibwsInstall.jacl INSTALL_JMS
-installRoot <install_root> -nodeName NODE_NAME -serverName server1
```

<install_root> is the directory in which you installed WebSphere Application Server; the second <install_root> uses forward slashes rather than back slashes; and NODE_NAME is the name of the application server node.

In our example the commands entered were as shown in Example 9-8.

Example 9-8 Install the SOAP over HTTP endpoint listener application

```
wsadmin -f c:/was/appserver/util/sibwsInstall.jacl INSTALL_HTTP -installRoot
"C:/WAS/AppServer" -nodeName ITSOESB01Node01 -serverName server1

wsadmin -f c:/was/appserver/util/sibwsInstall.jacl INSTALL_HTTP -installRoot
"C:/WAS/AppServer" -nodeName ITSOESB02Node01 -serverName server1
```

Creating a service integration bus

This scenario requires two service integration buses to be created and linked, forming one logical bus infrastructure. The first step is to create the first service integration bus to host the Retailer, Warehouse, and Logging Facility services.

Use the values shown in Table 9-1 to perform the following steps.

Table 9-1 Create service integration bus values

Item	Value
bus_name	ITSOESBBus01
bus_member	ITSOESB01Node01:server1
jndi_name	jdbc/com.ibm.ws.sib/me01

1. Access the WebSphere Application Server admin console at http://itsoesb01.itso.ral.ibm.com:9060/ibm/console and log in.
2. Expand **Service integration** and click **Buses**.
3. Click **New**.

4. In the field labeled Name enter the *bus_name* value, as shown in Figure 9-16. For the other values accept the defaults.

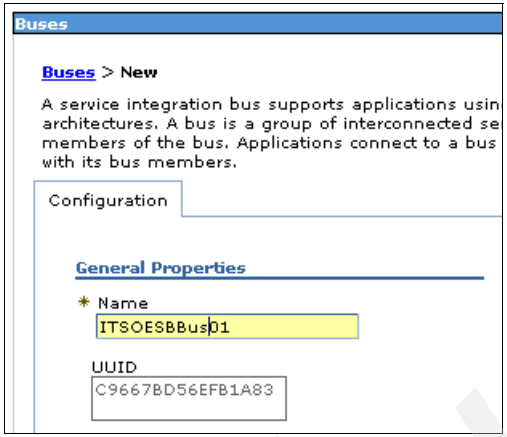

Figure 9-16 Create a service integration bus

5. Click **OK** and the bus will be created. **Save** your workspace changes to the master configuration.

Adding a bus member

Creating a bus just creates an administrative entity. It does not create any resources for messaging. In order to do this we need to add a bus member. This will have the effect of creating a messaging engine. Perform the following:

1. Click the *bus_name* value to show its properties. Under Topologies, click **Bus members**.
2. Click **Add**.
3. Complete the panel using the information below. On this page, you can choose which server or cluster to add.
   a. Select the *bus_member* server.
   b. Set the Data source JNDI name to the *jndi_name* value.
   c. Accept the remaining defaults. Click **Next**.
4. Click **Finish** on the Confirm page and the server will be added as a member of the bus and a messaging engine created as shown in Figure 9-17 on page 184.
5. Save your workspace changes to the master configuration.

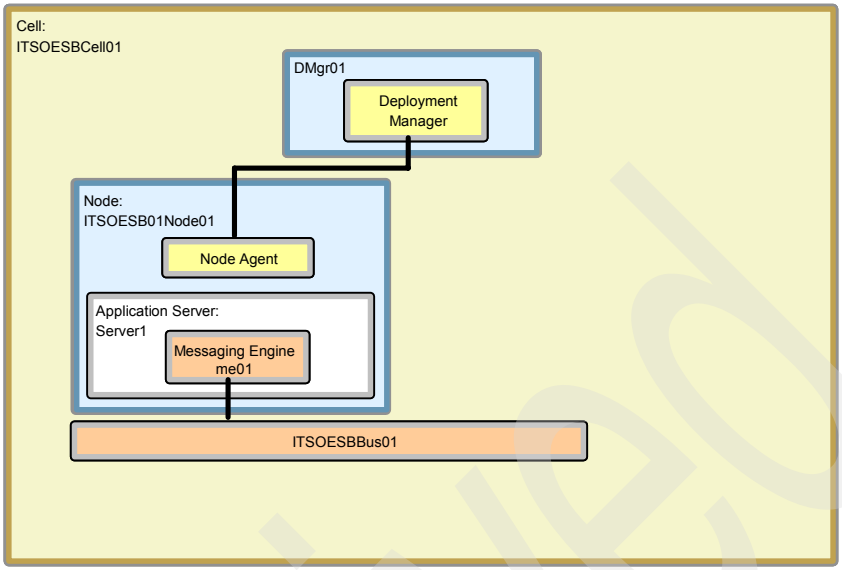

Figure 9-17 After the service integration bus and messaging engine have been created

Creating a second service integration bus and bus member

The preceding section created the first service integration bus that will host the Retailer, Warehouse, and Logging Facility services. A second service integration bus is required to host the Manufacturer services.

1. Repeat the steps in "Creating a service integration bus" on page 182 and "Adding a bus member" on page 183 using the values in Table 9-2.

Table 9-2 Create service integration bus values

Item	Value
bus_name	ITSOESBBus02
bus_member	ITSOESB02Node01:server1
jndi_name	jdbc/com.ibm.ws.sib/me02

When these steps are completed, the console at **Service integration** → **Buses** will appear as shown in Figure 9-18 on page 185 and you will have created two buses.

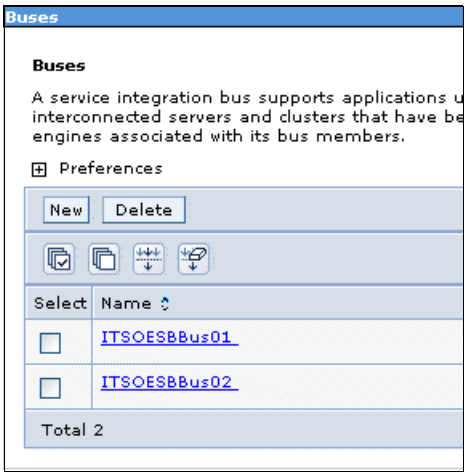

Figure 9-18 Service integration buses

Creating a foreign bus

In order for the buses to communicate, you must define them to each other. Use the replacement values in Table 9-3 in the following steps.

Table 9-3 Create service integration bus values

Item	Value
bus_name	ITSOESBBus01
foreign_bus_name	ITSOESBBus02

1. In the admin console navigate to **Service integration** → **Buses** and then select the *bus_name* value.
2. Select **[Topology] Foreign buses,** and click **New**.
3. Enter the *foreign_bus_name* value in Name, and click **Next.**
4. The next page allows the routing type to be selected. There are three options:
   - Direct, service integration bus link
   - Direct, WebSphere MQ link
   - Indirect

   Click the Routing Type pull-down menu and select the default **Direct, service integration bus link**, and click **Next** as shown in Figure 9-19 on page 186.

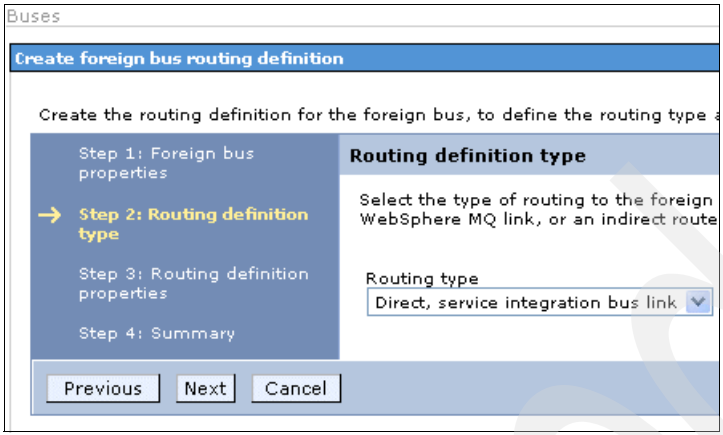

Figure 9-19 Foreign bus routing definition

5. On the next page you could specify the user ID to be used for inbound and outbound message authentication, but these can be ignored in our scenario so click **Next**.
6. The last page is a summary page so just click **Finish** and the foreign bus will be created.

Creating the mirror foreign bus

In order for the buses to communicate, define the first bus to the second bus:

1. Repeat the steps in “Creating a foreign bus” on page 185 using the replacement values in Table 9-4.

Table 9-4 Create service integration bus values

Item	Value
bus_name	ITSOESBBus02
foreign_bus_name	ITSOESBBus01

2. Save your workspace changes to the master configuration.

Creating a service integration bus link

The next step is to create the service integration bus link for each bus. A service integration bus link connects a messaging engine on one bus to a messaging engine on another.

In the following steps, use the replacement values in Table 9-5 on page 187.

Table 9-5 Create service integration bus link values

Item	Value
bus_name	ITSOESBBus01
me_name	ITSOESB01Node01.server1-ITSOESBBus01
foreign_node_name	ITSOESB02Node01
foreign_link_name	ITSOESBLink
foreign_bus_name	ITSOESBBus02
foreign_me_name	ITSOESB02Node01.server1-ITSOESBBus02
foreign_host_name	ITSOESB02.itso.ral.ibm.com

1. Determine the SIB_ENDPOINT_ADDRESS port value:
   a. In the admin console select **Servers** → **Application Servers**.
   b. In the Application servers list, click **server1** which corresponds to the node *foreign_node_name*.
   c. Under Communications, expand **Ports**. Note the value of SIB_ENDPOINT_ADDRESS. In our scenario, this port value is 7278.
2. Go to the admin console and navigate to the bus details panel for *bus_name*.
3. Under Topology, click **Messaging Engines**.
4. Click the *me_name*.
5. Under Additional Properties click **Service integration bus link**.
6. Click **New**.
7. You should now have navigated to the following console page:

   Buses → *bus_name* → Messaging engines → *me_name* → Service integration bus link → New

   Complete the window shown in Figure 9-20 on page 188, which defines the service integration bus link.

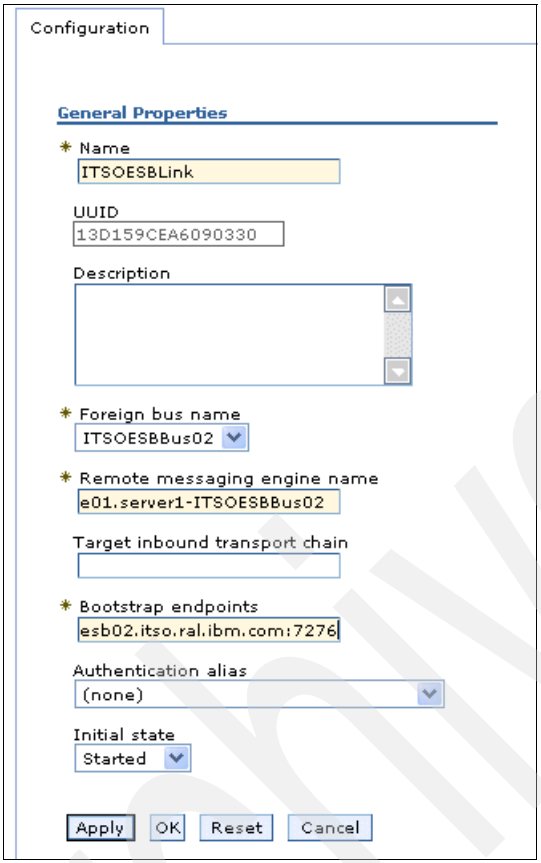

Figure 9-20 Creating a new service integration bus link

Only the mandatory information has to be filled in on this page.

– Name

  The name is an administrative entity. Enter the *foreign_link_name* value.

– Foreign bus

  The foreign bus that this messaging engine will be linked to. This pull-down list should contain a single entry. Select the *foreign_bus_name* value.

– Remote messaging engine name

  The name of the messaging engine on the foreign bus that this messaging engine will be connected to. Enter the name of the messaging engine on the foreign bus. We entered the *foreign_me_name* value.

– Bootstrap endpoints

The bootstrap endpoints specify where to find the messaging engine. It is a comma-separated list of entries. Each entry consists of up to three parts; if one part is missing it will assume a default value:

- Host name

  The name of the host.

- Port number

  The port number the messaging engine is listening on. Default: 7276.

- Protocol name

  The symbolic name of the messaging protocol being used. There are currently two: BootstrapBasicMessaging (the default) and BootstrapSecureMessaging.

Because we have used default values, simply entering the host name that the first ESB is hosted on is sufficient, and the port number. Use the port number assigned to SIB_ENDPOINT_ADDRESS, which you noted in step 1. We entered *foreign_host_name*:7278.

8. Click **OK**. The service integration bus link has been configured.
9. Save your workspace changes to the master configuration.

Creating a mirror service integration bus link

Repeat the process for the other ESB hosted on ITSOESB02.itso.ral.ibm.com.

1. Repeat the steps in “Creating a service integration bus link” on page 186, using the replacement values in Figure 9-6.

Table 9-6 Create service integration bus link mirror values

Item	Value
bus_name	ITSOESBBus02
me_name	ITSOESB02Node01.server1-ITSOESBBus02
foreign_node_name	ITSOESB01Node01
foreign_link_name	ITSOESBLink
foreign_bus_name	ITSOESBBus01
foreign_me_name	ITSOESB01Node01.server1-ITSOESBBus01
foreign_host_name	ITSOESB01.itso.ral.ibm.com

2. Save your workspace changes to the master configuration.

9.3.5 Deploying and building the WS-I scenario

This section describes how to set up the service integration bus resources for the WS-I application.

Creating the destinations

The Manufacturer enterprise application uses JMS to trigger some work to occur asynchronously as shown in Figure 9-5 on page 157. This requires some destinations to be created. These next steps describe how to create these destinations and will use the values in Table 9-7.

Table 9-7 Create the first manufacturer destinations values

Item	Value
bus_name	ITSOESBBus02
destination_identifier	`ManufacturerASIBQ`
bus_member_name	ITSOESB02Node01.server1

1. From the bus details page for *bus_name* under Destination resources, click **Destinations**.
2. Click **New**.
3. The next page is for creating or selecting the type of destination. Accept the default of **Queue** and click **Next**. This launches the Queue creation wizard.
4. The first page of the wizard, shown in Figure 9-21 on page 191, asks for an identifier and description to be entered. The identifier is the name by which the destination will be exposed to applications. Enter the value for *destination_identifier* and click **Next**.

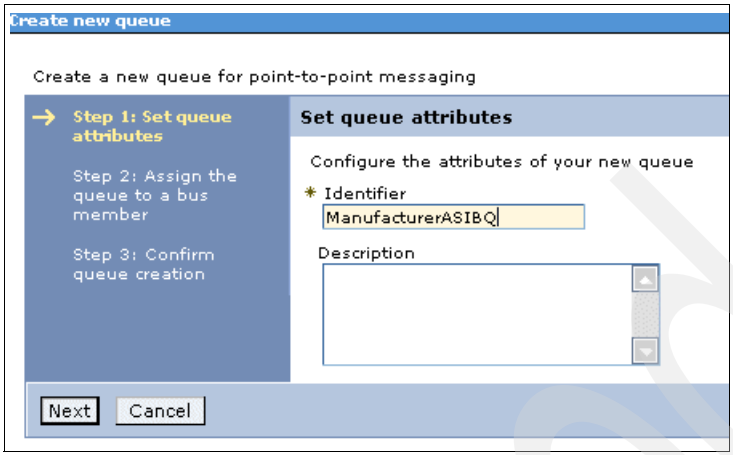

Figure 9-21 New destination wizard

5. Specify which bus member to assign the destination to. The Manufacturer will communicate via this queue through a bus member to the service integration bus, and if you have followed this scenario there should only be one bus member to choose. Select the *bus_member_name* as shown in Figure 9-22 and click **Next**.

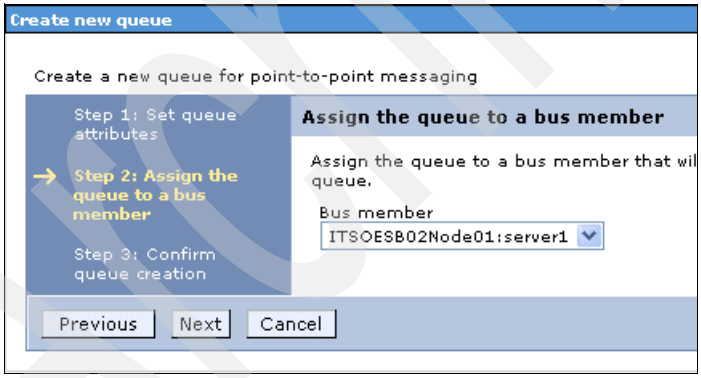

Figure 9-22 Assign the queue to a bus member

6. The final page is just a summary, so click **Finish** and the destination will be created.

7. We create two additional queue type destinations, one each for the other two manufacturers. Repeat steps 2 on page 190 through step 6, once each using the data in tables Table 9-8 on page 192 and Table 9-9 on page 192.

Table 9-8 Create second manufacturer destinations values

Item	Value
bus_name	ITSOESBBus02
destination_identifier	`ManufacturerBSIBQ`
bus_member_name	ITSOESB02Node01.server1

Table 9-9 Create third manufacturer destinations values

Item	Value
bus_name	ITSOESBBus02
destination_identifier	ManufacturerCSIBQ
bus_member_name	ITSOESB02Node01.server1

8. Save your workspace changes to the master configuration.

Creating a JMS connection factory

The next step is to create a JMS connection factory so the manufacturers can connect to the service integration bus to send JMS messages.

Table 9-10 Create a JMS connection factory values

Item	Value
node_scope_name	ITSOESB02Node01
CF_name	Manufacturer Connection Factory
CF_JNDI_name	Sample/WSI/CF
bus_name	ITSOESBBus02

1. From the admin console expand **Resources** → **JMS providers** and click **Default messaging**.
2. Because the JMS connection factory will be utilized only on the ITSOESB02 bus, we need to set the correct scope. Ensure that the Node entry field has the *node_scope_name* value entered as shown in Figure 9-23 on page 193.

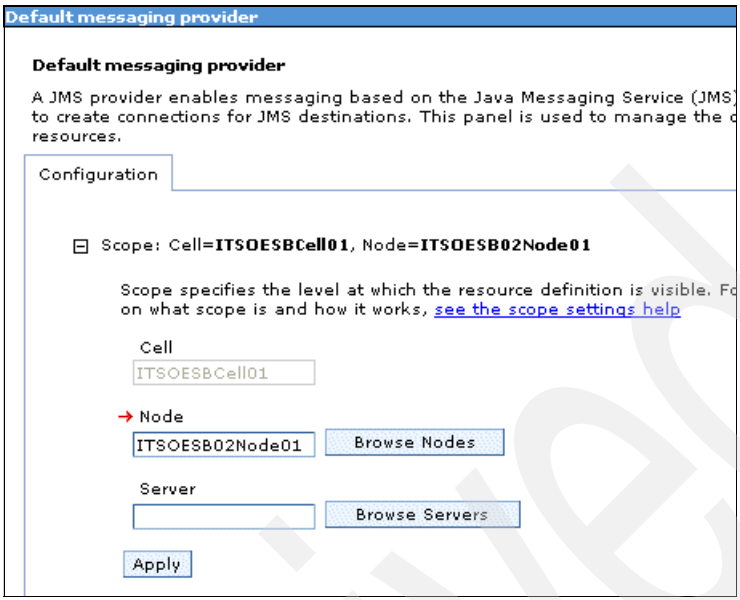

Figure 9-23 Set scope for JMS definitions

Note: If the scope is set incorrectly, click **Browse Nodes**, select the *node_scope_name* value from the list, and select **OK** to return to the original panel shown in Figure 9-23.

3. Under Connection Factories, click **JMS connection factory**.
4. Click **New**.
5. The next page, shown in Figure 9-24 on page 194, enables you to specify the properties for the JMS connection factory.

   Most of the values on this page can be left as the defaults, but these are the values that must be entered:

   – Name

     An administrative name used for locating the connection factory in the admin console. Enter the *CF_name* value.

   – JNDI Name

     Where in the JNDI namespace to bind the connection factory. The application resource reference will be bound to this. Enter the *CF_JNDI_name* value.

– Bus name

The name of the bus to which the connection factory should connect. This is specified by a pull-down of all of the buses in the cell. It also allows an arbitrary value to be entered by choosing other. Select the *bus_name* value from the pull-down menu.

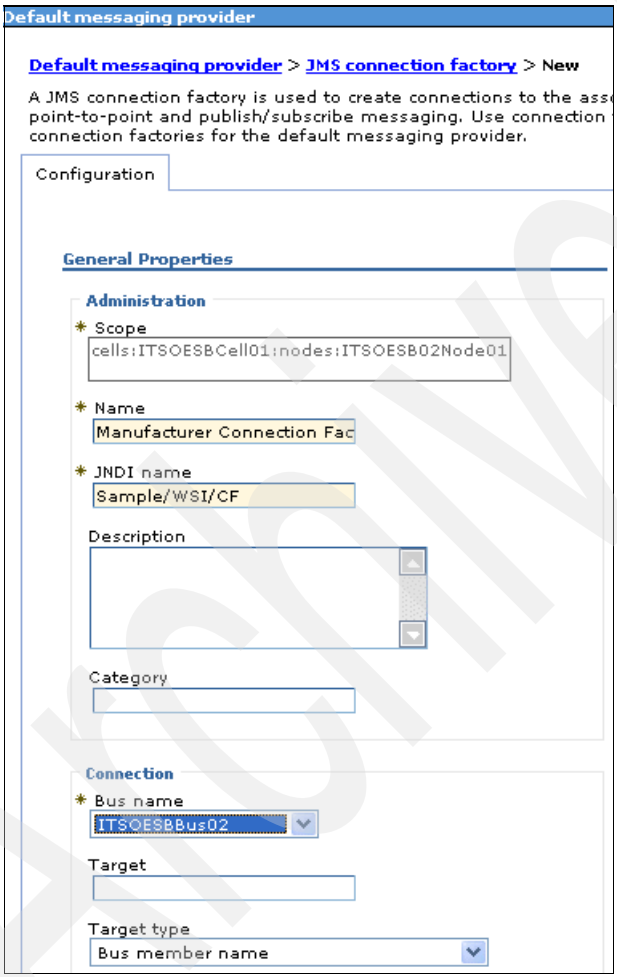

Figure 9-24 Create new JMS connection factory page

6. Click **OK** and save your workspace changes to the master configuration.

Creating the JMS queues

Now we create some JMS queues, one for each of the service integration bus queue type destinations we defined in “Creating the destinations” on page 190.

Table 9-11 Create a JMS queue values for Manufacturer A

Item	Value
node_scope_name	ITSOESB02Node01
jmsQ_name	ManufacturerJMSQ
jmsQ_JNDI_name	Sample/WSI/Manufacturer
bus_name	ITSOESBBus02
Q_name	ManufacturerASIBQ

1. From the admin console expand **Resources** → **JMS providers** and click **Default messaging**.
2. As before, ensure the scope is set to the *node_scope_name* value.
3. Under Destinations, click **JMS queue**.
4. Click **New**.
5. The next page, shown in Figure 9-25 on page 196, allows you to specify the values for the queue.
   - Name

     An administrative name used for locating the JMS queue in the admin console. Enter the value for *jmsQ_name*.
   - JNDI Name

     Where in the JNDI namespace to bind the JMS queue. The applications message reference will be bound to this. Enter the *jmsQ_JNDI_name* value.
   - Bus name

     The name of the bus you are connecting to. Select the value of *bus_name*. This causes the page to be reloaded with the Queue names list filled in.
   - Queue name

     This field specifies the service integration bus queue type destination that will be used to store the messages sent to this JMS queue. Select the value of *Q_name*.

   Click **OK**.

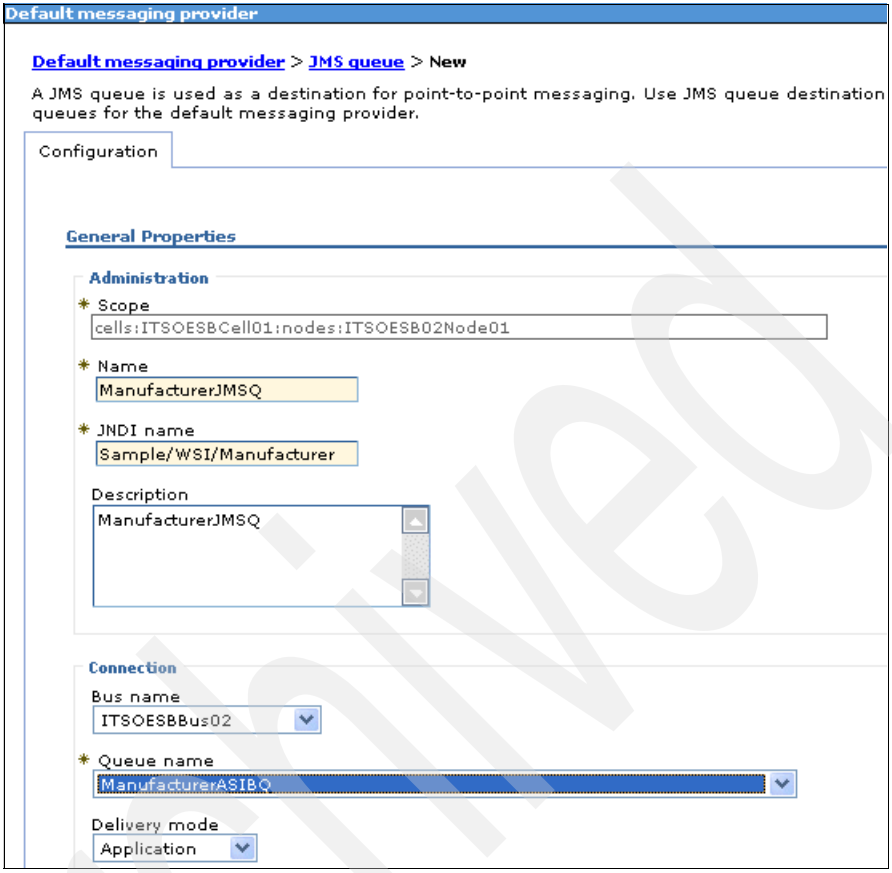

Figure 9-25 Create a new JMS queue page

6. Repeat steps 4 and 5 to create two other JMS queues, once each using the data in Table 9-12 and Table 9-13 on page 197.

Table 9-12 Create a JMS queue values for Manufacturer B

Item	Value
node_scope_name	ITSOESB02Node01
jmsQ_name	ManufacturerBJMSQ
jmsQ_JNDI_name	Sample/WSI/ManufacturerB
bus_name	ITSOESBBus02
Q_name	ManufacturerBSIBQ

Table 9-13 Create a JMS queue values for Manufacturer C

Item	Value
node_scope_name	ITSOESB02Node01
jmsQ_name	ManufacturerCJMSQ
jmsQ_JNDI_name	Sample/WSI/ManufacturerC
bus_name	ITSOESBBus02
Q_name	ManufacturerCSIBQ

You should now have three JMS queues defined as shown in Figure 9-26.

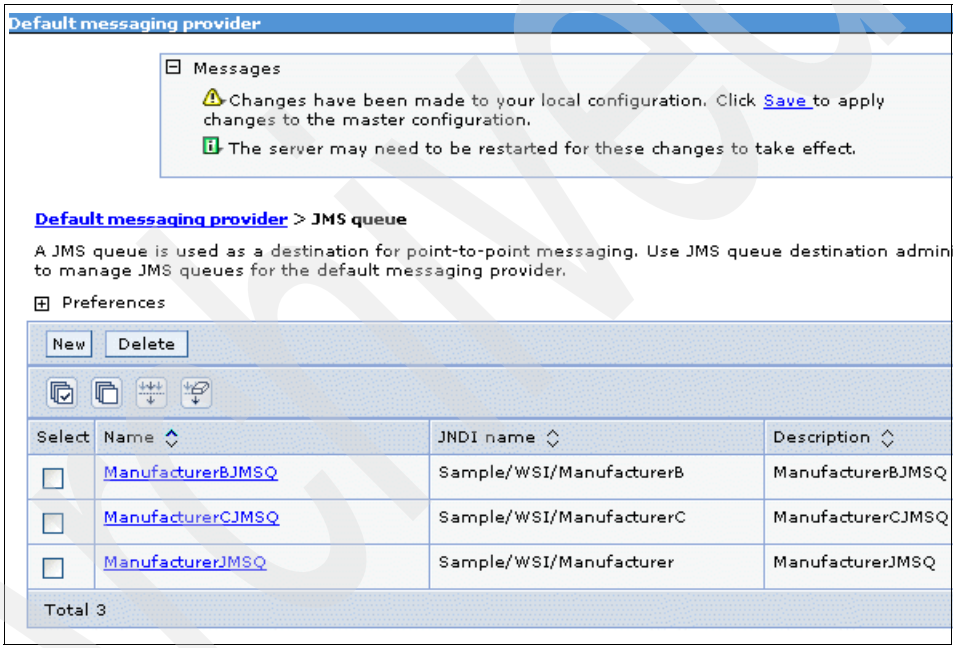

Figure 9-26 JMS queues defined for the scenario

7. Save your workspace changes to the master configuration.

Creating the JMS activation specifications

Now we create three activation specifications, one for each JMS queue we created in “Creating the JMS queues” on page 195. Use the values in Table 9-14 on page 198.

Table 9-14 Create a JMS activation specification for Manufacturer A

Item	Value
node_scope_name	ITSOESB02Node01
as_name	ManufacturerAS
as_JNDI_name	Sample/WSI/ManufacturerAS
jmsQ_JNDI_name	Sample/WSI/Manufacturer
bus_name	ITSOESBBus02

1. From the admin console, expand **Resources** → **JMS providers** and click **Default messaging**.
2. As before, ensure that the scope is set to the *node_scope_name* value.
3. Under Activation Specifications, click **JMS activation specification**.
4. Click **New**.
5. On the next page, shown in Figure 9-27, specify the values for the activation specification. Most of the values can keep their default values, but you must specify the following values.
   - Name

     An administrative name used for locating the JMS activation specification in the admin console. Enter a value of *as_name*.
   - JNDI name

     Where in the JNDI namespace to bind the JMS activation specification. The Manufacturer application's message-driven beans will be bound to this for message delivery. Enter a value of *as_JNDI_name*.
   - Destination type

     The type of JMS destination that will be used to deliver messages to the message-driven bean. Accept the default of Queue.
   - Destination JNDI name

     The location in JNDI of the JMS destination that should be used to receive messages from. Enter a value of *jmsQ_JNDI_name*.
   - Bus name

     The name of the bus from which the JMS destination will receive messages. This is not required, but for consistency select the value of *bus_name*.

   Click **OK**.

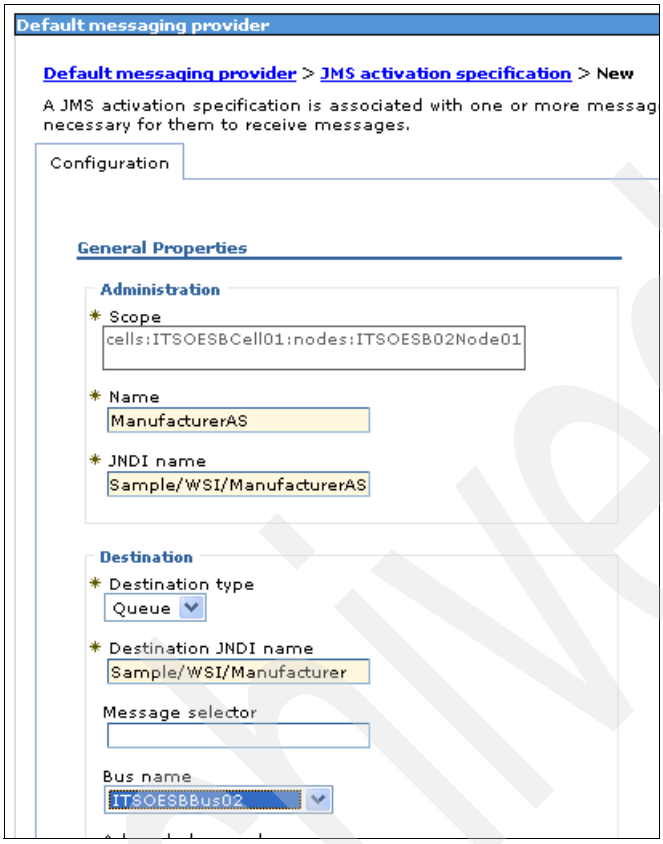

Figure 9-27 Create a new JMS activation specification

6. Repeat steps 4 and 5 to create two other activation specifications, once each using the data in Table 9-15 and Table 9-16.

Table 9-15 Create a JMS activation specification for Manufacturer B

Item	Value
node_scope_name	ITSOESB02Node01
as_name	ManufacturerBAS
as_JNDI_name	Sample/WSI/ManufacturerBAS
jmsQ_JNDI_name	Sample/WSI/ManufacturerB
bus_name	ITSOESBBus02

Table 9-16 Create a JMS activation specification for Manufacturer C

Item	Value
node_scope_name	ITSOESB02Node01
as_name	ManufacturerCAS
as_JNDI_name	Sample/WSI/ManufacturerCAS
jmsQ_JNDI_name	Sample/WSI/ManufacturerC
bus_name	ITSOESBBus02

7. Save your workspace changes to the master configuration.

Hosting the WSDL files

Each Web service client in the WS-I sample application references WSDL files containing port type and binding information. Import statements dictate the location of these files. For example, the SCMSampleUI enterprise application contains a WSDL file that references the Retailer port type and binding by using the following import:

```
<wsdl:import location="http://appsrv1a.itso.ral.ibm.com/wsdl/Retailer.wsdl"
namespace="http://www.ws-i.org/SampleApplications/SupplyChainManagement/
2002-08/Retailer.wsdl"/>
```

It is therefore necessary to host an HTTP server where these files can be retrieved. Furthermore, this HTTP server must be assigned the address of `appsrv1a.itso.ral.ibm.com`.

We installed IBM HTTP Server V6, which is shipped with the WebSphere Application Server V6 installation. The WSDL and XSD files to be hosted on this HTTP server are provided with the additional materials supplied with this book. For information about obtaining the additional materials, see Appendix A, “Additional material” on page 317.

1. From the additional material, copy the contents of the \Beth scenario\wsdl directory to the following directory in the HTTP server:

   ```
   <HTTP_Server_home>\htdocs\en_US\wsdl
   ```

2. Perform a similar process on the second host system (ITSOESB02.itso.ral.ibm.com) if you are not using the same physical machine.
3. Make sure the HTTP server is started, then test whether the WSDL is available by entering the following URL into a Web browser. It should show the WSDL for the Retailer Web service:

   ```
   http://appsrv1a.itso.ral.ibm.com/wsdl/Retailer.wsdl
   ```

Modifying a virtual host for ITSOESB02.itso.ral.ibm.com

If you have host names ITSOESB01.itso.ral.ibm.com and ITSOESB02.itso.ral.ibm.com running on the same host system, modify the default_host virtual host to associate port 9081 with ITSOESB02.itso.ral.ibm.com.

Note: Skip this section if ITSOESB01 and ITSOESB02 are running on separate host systems.

1. In the WebSphere Application Server admin console, expand **Environment** and click **Virtual Hosts**.
2. Click the *default_host* virtual host.
3. Under Additional Properties, click **Host Aliases**.
4. Click the host name associated with port 9081.
5. Change the host name to `ITSOESB02.itso.ral.ibm.com` (Figure 9-28).

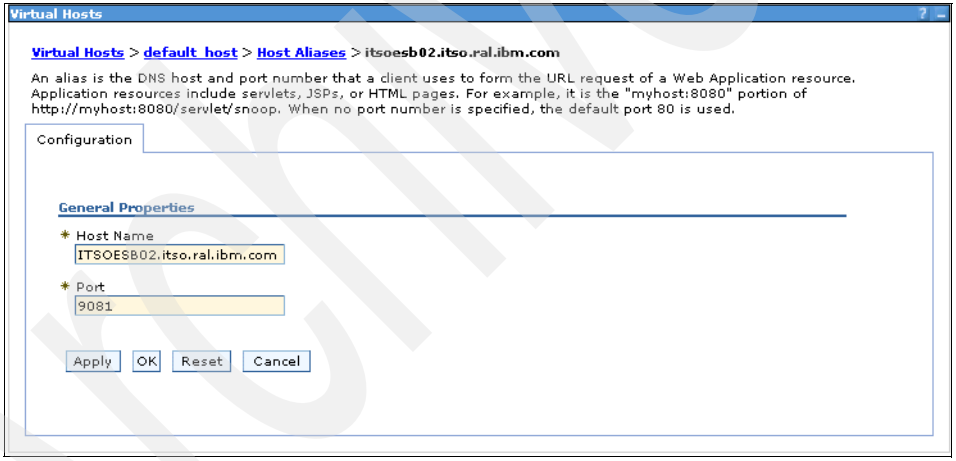

Figure 9-28 Modifying the host name for port 9081

6. Click **OK** then save the changes.

Installing the applications

The final step to getting a base version of the scenario up and running is to install the scenario enterprise applications. This scenario uses the enterprise applications provided specifically for this scenario. Use the values in Table 9-17.

Table 9-17 Install the SCMRetailerLogging application values

Item	Value
server_name	ITSOESB01Node01
app_name	UIRetailerLogging.ear

1. From the WebSphere Application Server admin console, expand **Applications** and click **Install New Application**.
2. The enterprise applications required for this scenario are located in the Scenario1\ears directory of the additional material supplied with this book. Enter the location of the *app_name* file on your local file system by either entering it directly or clicking **Browse** and navigating to the file in the open file dialog. Click **Next**.
3. On the next page, check the **Generate Default Bindings** check box, accept the remaining defaults, and click **Next**.
4. The next page to open is step 1 in the wizard. Click **Next.**
5. In installation step 2, on the Map modules to servers panel, specify the server on which the application will run. In this case select the node *server_name*, select all checkboxes for every module row and click **Apply.** The output should resemble Figure 9-29 on page 203**.** Click **Next**.

Figure 9-29 Select the node to install the SCMRetailerLogging application

6. The remainder of the application is fully configured and deployed, so the defaults on each page should be used. Click the last step, labeled **Summary**, then click **Finish**. Ignore any warnings.
7. When the application installation completes, save your changes.
8. Repeat steps 2 to 7 for the Manufacturer applications, once each using the data in Table 9-18, Table 9-19 on page 204, and Table 9-20 on page 204.

Table 9-18 Install the Manufacturer application values

Item	Value
server_name	ITSOESB02Node01
app_name	ManufacturerA.ear

Table 9-19 Install the ManufacturerB application values

Item	Value
server_name	ITSOESB02Node01
app_name	ManufacturerB.ear

Table 9-20 Install the ManufacturerC application values

Item	Value
server_name	ITSOESB02Node01
app_name	ManufacturerC.ear

9. Be sure all changes are saved, then start the two application servers. In a Command Prompt, change to the bin directory of each profile and run the **startserver** command:

```
c:\WAS\AppServer\profiles\ITSOESB01\bin\startserver server1
c:\WAS\AppServer\profiles\ITSOESB02\bin\startserver server1
```

Creating the endpoint listeners

The next task is to create some endpoint listeners, which listen for incoming Web service requests and forward them onto the relevant inbound services. Inbound services get bound to an endpoint listener when they are created. We create an HTTP endpoint listener on each server on each node. Use the values in Table 9-21.

Table 9-21 Create the endpoint listeners for the first service integration bus

Item	Value
node_name	ITSOESB01Node01
host_name	ITSOESB01.itso.ral.ibm.com
URL_name	http://ITSOESB01.itso.ral.ibm.com
bus_name	ITSOESBBus01

1. Access the admin console at http://ITSOESB01.itso.ral.ibm.com:9060/ibm/console and log in.
2. Expand **Servers** and click **Application Servers**.
3. Click **server1** for node *node_name*.
4. Under Communications, expand **Ports** and note the port value of WC_defaulthost. We refer to this value as *port_number*.

5. Under Additional Properties, click **Endpoint Listeners**.
6. Click **New**.
7. The next panel (Figure 9-30) enables the creation of an endpoint listener.
   - Name

     The name of the endpoint listener. It must be SOAPHTTPChannel1.
   - URL root

     The base URL for Web service requests into this endpoint listener. The URLs used for making Web service requests to the service integration bus have this at the beginning. Set this to:

     URL_name:port_name/wsgwsoaphttp1

     For example:

     http://ITSOESB01.itso.ral.ibm.com:9080/wsgwsoaphttp1
   - WSDL serving HTTP URL root

     This corresponds to the location where you access the servlet that hosts WSDL for this endpoint, and is used by the service integration bus when generating URLs to access the WSDL representing an inbound service (for example, when publishing services to UDDI). Set this to:

     URL_name:port_name/sibws/wsdl

     For example:

     http://ITSOESB01.itso.ral.ibm.com:9080/sibws/wsdl

   Click **Apply**.

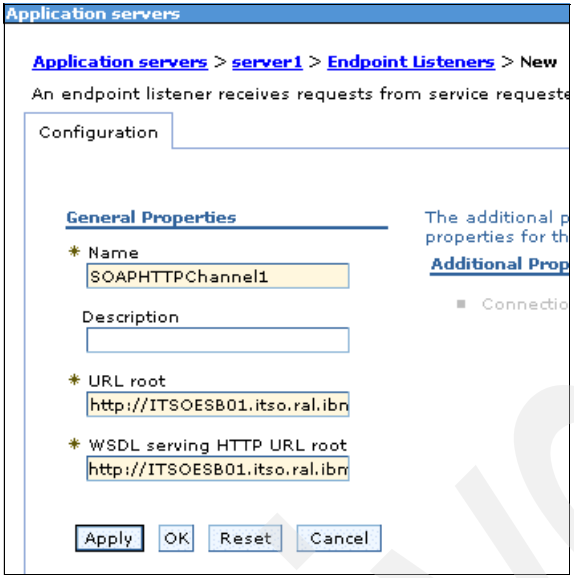

Figure 9-30 Create an endpoint listener

8. Under Additional Properties, click **Connection Properties**.
9. Click **New**.
10. In the Bus name pull-down menu select the *bus_name* value (Figure 9-31) and click **OK**.

Figure 9-31 Connection properties for an endpoint listener

11. Repeat steps 2 to 11 for the second node/server using the data in Table 9-22 and save your workspace changes to the master configuration.

Table 9-22 Create the endpoint listeners for the second service integration bus

Item	Value
node_name	ITSOESB02Node01
host_name	ITSOESB02.itso.ral.ibm.com
URL_name	http://ITSOESB02.itso.ral.ibm.com
bus_name	ITSOESBBus02

Creating the outbound services

Outbound services define Web service requests that leave the service integration bus and are received by a service provider.

Creating the LoggingFaclityService outbound service

We define an outbound service for the LoggingFacilityService Web service (on the first ESB because that is where the service actually resides) using the values in Table 9-23.

Table 9-23 Create the outbound service for LoggingFacilityService

Item	Value
bus_name	ITSOESBBus01
ent_app	UIRetailerLogging
app_name *port_name*	LoggingFacility
wsdl_root	http://ITSOESB01.itso.ral.ibm.com:9080/LoggingFacility/services/LoggingFacility?wsdl
host_name	ITSOESB01.itso.ral.ibm.com
wsdl_loc	http://ITSOESB01.itso.ral.ibm.com:9080/LoggingFacility/services/LoggingFacility/wsdl/LoggingFacility_Impl.wsdl
outbound_name *wsdl_service_name* *service_dest_name*	LoggingFacilityService
port_dest_name	LoggingFacilityService:LoggingFacility
member_name	ITSOESB01Node01:server1

Tip: Ensure that the UIRetailerLogging, Manufacturer, ManufacturerB, and ManufacturerC applications are started.

1. From the admin console, expand **Service integration** and click **Buses**.
2. Click *bus_name*.
3. Under Services, click **Outbound Services**.
4. Click **New**.
5. The first page of the wizard (Figure 9-32) requires you to specify a URL or UDDI repository where a WSDL definition of the service can be found. In our case we use a URL. The URL option enables you to specify an HTTP URL or a file system path. Enter the *wsdl_loc* value in the WSDL location field.

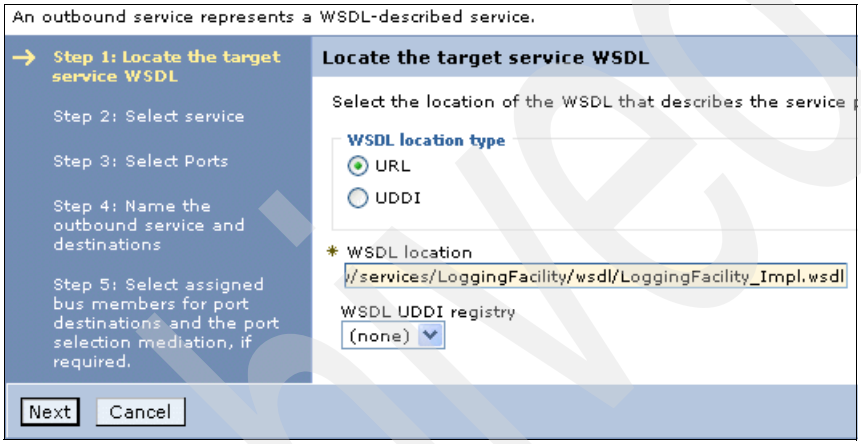

Figure 9-32 Locate the target WSDL

Tip: Where does the value of *wsdl_loc* come from? See the sub-bullets below starting at item a.

a. It is recommended that you open a separate browser window at http://ITSOESB01.itso.ral.ibm.com:9060/ibm/console/ (logging in as a different user) because you will need to cut and paste from two WebSphere Application Server admininstration console panels.

b. From the second admin console browser window, expand **Application** → **Enterprise Applications**

c. Select *ent_app***.**

d. Under Additional Properties, select **Publish WSDL files.**

e. Select **<ent_app> WSDLfiles.zip** as shown in Figure 9-33 and open the file with a zip file editor.

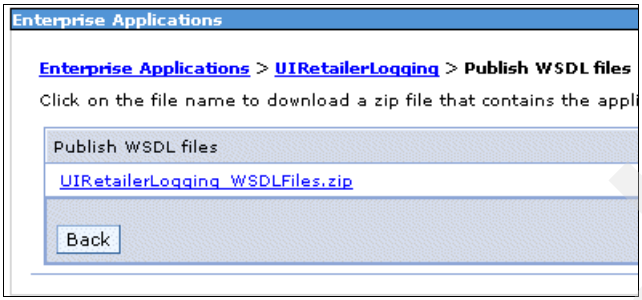

Figure 9-33 Publish WSDL files

f. Edit the *app_name*_Impl.wsdl file (LoggingFacility_Impl.wsdl, for example). If there is a choice of files, ensure that you choose the EJB module for the *app_name*.

g. Copy to the clipboard the location item for the <wsdl:service> definition as shown in Figure 9-34.

Figure 9-34 Select the WSDL location name

h. Paste the location into your browser's address field, appended with `?wsdl`. An example of this URL is defined as *wsdl_root* in Table 9-23 on page 207.

> **Tip:** Ensure that the deployment manager, nodes, and servers are started.

i. Copy the redirected address (Figure 9-35).

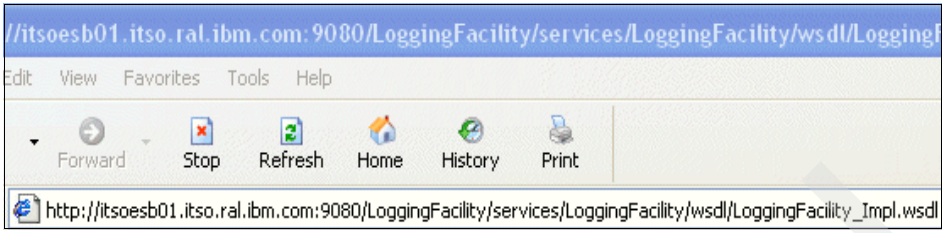

Figure 9-35 Copy the WSDL address location

j. Paste the address into the WSDL location field in the original browser window shown in Figure 9-36 on page 210 - where you are defining the outbound service.

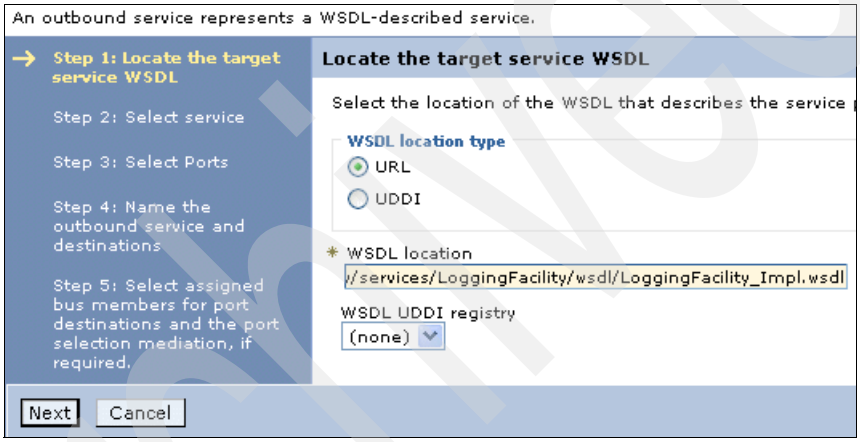

Figure 9-36 Paste the WSDL location

6. In the original browser window, click **Next** to continue.
7. The next panel displays the available services defined in the WSDL file. On this page, you select which service you wish to create an outbound service for. In our case there is only the *wsdl_service_name* service. Click **Next**.

8. The next panel displays the ports defined for the selected service. There is only one port in our service so check *port_name* and click **Next** as shown in Figure 9-37 on page 211.

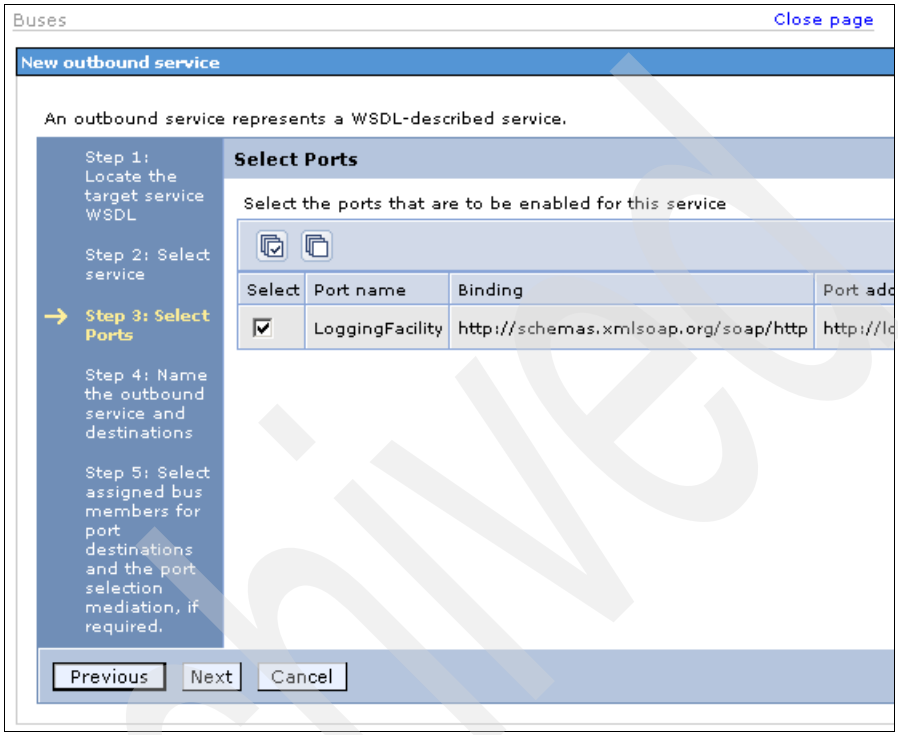

Figure 9-37 Port selection page

9. On the next page, you change the name of the outbound service, service destination name, and port destination name. We decided to modify some of these parameters to make it simpler.
   - Outbound service name

     outbound_name
   - Service destination name

     Shorten this to just the last part of the pre-filled entry field:

     service_dest_name
   - Port destination name

     Shorten this to just the last part of the pre-filled entry field:
     port_dest_name

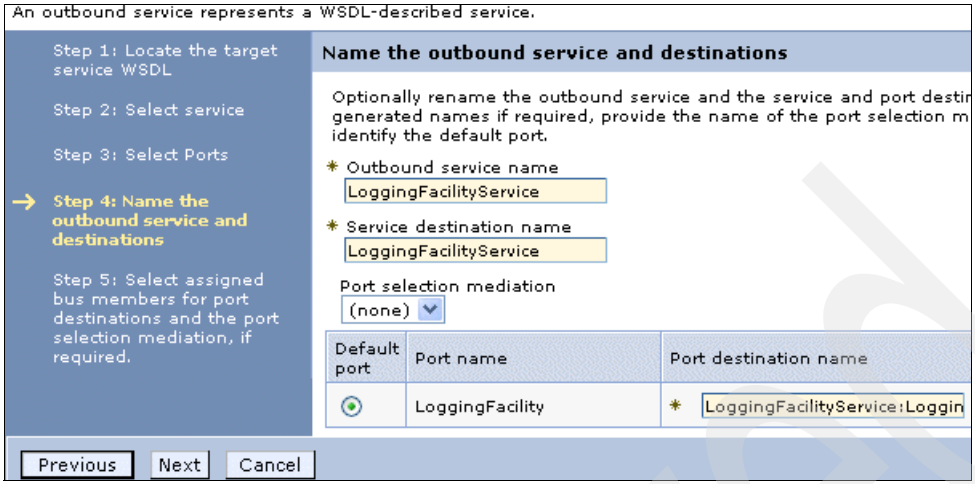

Figure 9-38 Name the outbound service and destinations

10. Click **Next**.
11. On the final page, select the bus member to assign the outbound service to. Select *member_name* and click **Finish**. The outbound service will be created.

Creating the Retailer and Warehouse outbound services

You also need to create outbound services on ITSOESB01.itso.ral.ibm.com for the Retailer, Warehouse, and WarehouseCallBack services. Assuming you are using the default port of 9080, complete the fast track instructions below to create an outbound service for the Retailer, using the values in Table 9-24.

Table 9-24 Create the outbound service for RetailerService

Item	Value
bus_name	ITSOESBBus01
wsdl_loc	http://itsoesb01.itso.ral.ibm.com:9080/Retailer/services/Retailer/wsdl/Retailer_Impl.wsdl
outbound_name *service_dest_name*	RetailerService
port_dest_name	RetailerService:Retailer

12. Click **Service Integration** → **Buses** and click *bus_name*.
13. Under Services, click **Outbound Services**, then click **New**.
14. In the WSDL location field enter *wsdl_loc* and click **Next**.

15. In the Select a service window, the correct service is selected, so click **Next**.
16. In the Select Ports window, the correct port is selected, so click **Next**.
17. Enter the following information:
    a. Set Outbound service name to *outbound_name*.
    b. Set Service destination name to *service_dest_name*.
    c. Set Port destination name to *port_dest_name*.
18. Click **Next** then click **Finish** to create the outbound service.
19. Repeat steps 12 to 18 to create outbound services for the Warehouse (using values in Table 9-25) and the WarehouseCallBack (using values in Table 9-26) services.

Table 9-25 Create the outbound service for WarehouseService

Item	Value
bus_name	ITSOESBBus01
wsdl_loc	http://itsoesb01.itso.ral.ibm.com:9080/Warehouse/services/Warehouse/wsdl/Warehouse_Impl.wsdl
outbound_name *service_dest_name*	WarehouseService
port_dest_name	WarehouseService:Warehouse

Table 9-26 Create the outbound service for WarehouseCallBackService

Item	Value
bus_name	ITSOESBBus01
wsdl_loc	http://itsoesb01.itso.ral.ibm.com:9080/Warehouse/services/WarehouseCallBack/wsdl/WarehouseCallBack_Impl.wsdl
outbound_name *service_dest_name*	WarehouseCallBackService
port_dest_name	WarehouseCallBackService:WarehouseCallBack

Creating the Manufacturer outbound services

Outbound services must be created for the three Manufacturer services. These services are located on ITSOESBBus02, so the outbound services must be defined on ITSOESBBus02.

Note: The following instructions assume that the server running on ITSOESB02.itso.ral.ibm.com uses port 9081 for WC_defaulthost. To determine the WC_defaulthost port being used, perform the following:

- Click **Servers** → **Application Servers** and click on the server assigned to the node ITSOESB02Node01.
- Under Communications, expand **Ports**. This shows the value of WC_defaulthost.

Attention: If the value of WC_defaulthost is different from 9081, use this value in place of 9081 in the instructions below.

20. To create an outbound service for ManufacturerA, repeat steps 12 to 19, using the values in Table 9-27.

Table 9-27 Create the outbound service for ManufacturerAService

Item	Value
bus_name	ITSOESBBus02
wsdl_loc	http://ITSOESB02.itso.ral.ibm.com:9081/Manufacturer/services/Manufacturer/wsdl/Manufacturer_Impl.wsdl
outbound_name service_dest_name	ManufacturerService
port_dest_name	ManufacturerService:Manufacturer

21. To create an outbound service for ManufacturerB, repeat steps 12 to 19, using the values in Table 9-28.

Table 9-28 Create the outbound service for ManufacturerBService

Item	Value
bus_name	ITSOESBBus02
wsdl_loc	http://ITSOESB02.itso.ral.ibm.com:9081/ManufacturerB/services/ManufacturerB/wsdl/ManufacturerB_Impl.wsdl
outbound_name service_dest_name	ManufacturerBService
port_dest_name	ManufacturerBService:ManufacturerB

22. To create an outbound service for ManufacturerC, repeat steps 12 to 19, using the values in Table 9-29 on page 215.

Table 9-29 Create the outbound service for ManufacturerCService

Item	Value
bus_name	ITSOESBBus02
wsdl_loc	http://ITSOESB02.itso.ral.ibm.com:9081/ManufacturerC/services/ManufacturerC/wsdl/ManufacturerC_Impl.wsdl
outbound_name *service_dest_name*	ManufacturerCService
port_dest_name	ManufacturerCService:ManufacturerC

23. Save your changes to the master configuration.

Routing service requests between buses

Next we route service requests between the two service integration buses. For this, we need to have destinations on the local bus that route requests to the target (or intermediary) bus.

These destinations will be used to route messages between inbound services on one bus and outbound services on another bus, as shown in Figure 9-39.

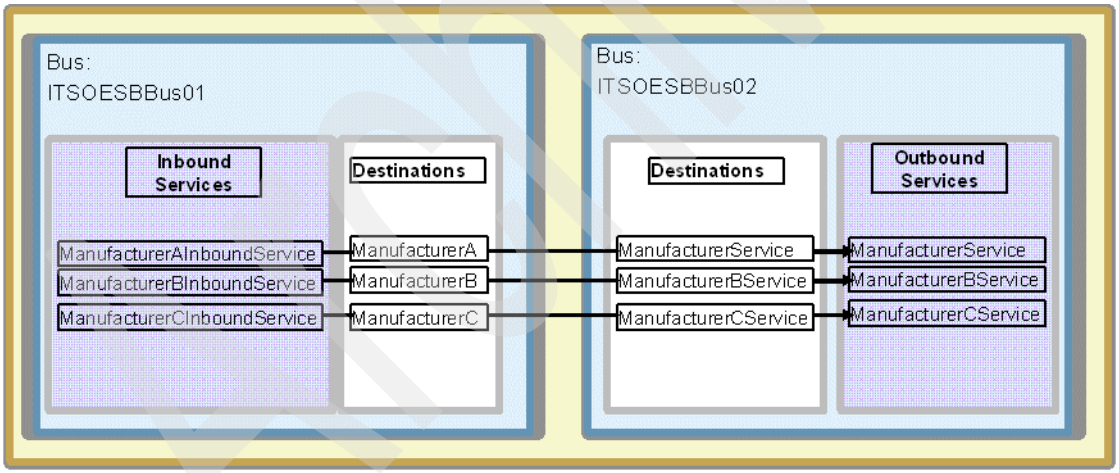

Figure 9-39 Service invocations crossing the buses

Routing paths

A routing path defines a sequential list of intermediary bus destinations that messages must pass through to reach a target bus destination. A routing path is used to apply optional mediations configured on several destinations to messages sent along the path.

A forward routing path identifies a list of bus destinations that a message should be sent to from the producer to the last destination from which consumers retrieve messages. The reverse routing path is constructed automatically for request/reply messages and identifies the list of destinations that any reply message should be sent to from the consumer back to the producer. Use of reverse routing path enables a reply message to optionally take a different route back to the producer, and therefore have more mediations applied.

When a message arrives at a destination in the path, mediations can manipulate the entries in the forward routing path, to change the sequence of destinations through which messages pass. If a mediation manipulates the forward routing path, and the reverse routing path has been set (for a request message that expects a reply), then the mediation is responsible for making any corresponding changes to the reverse routing path.

Configuring forward routing destinations

1. Create three queue type destinations. Initially create a destination for ManufacturerA using the values in Table 9-30.

Table 9-30 Create destination and routing path for ManufacturerA

Item	Value
local_bus_name	ITSOESBBus01
dest_bus_name	ITSOESBBus02
destination_identifier	ManufacturerA
bus_member_name	ITSOESB01Node01:server1
forward_routing_path	ITSOESBBus02:ManufacturerService

2. From the admin console expand **Service integration** and click **Buses**.
3. Click *local_bus_name*.
4. From the bus details page for *local_bus_name* under Destination resources, click **Destinations**.
5. Click **New**.

6. The next page enables creation and selection of the type of destination. Accept the default **Queue** and click **Next** to launch the Queue creation wizard.
7. The first page of the wizard, shown in Figure 9-40, asks for an identifier and description to be entered. The identifier is the name by which the destination will be exposed to applications. Enter *destination_identifier* and click **Next**.

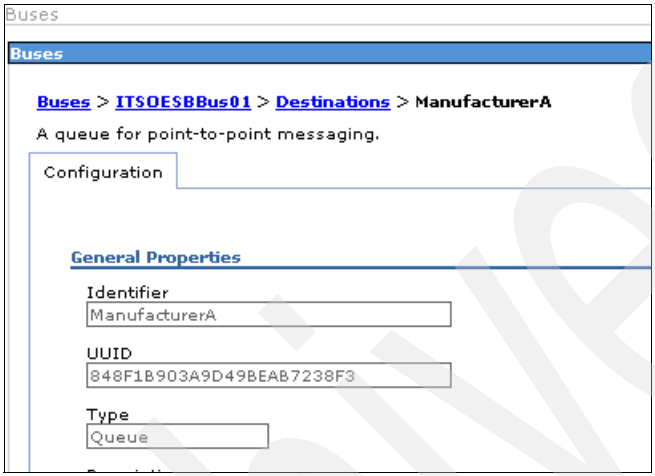

Figure 9-40 New destination for ManufacturerA on local bus

8. On the next page, specify which bus member to assign the destination to. The Manufacturer will communicate via this queue through a bus member to the service integration bus. Select *bus_member_name* and click **Next** as shown in Figure 9-41.

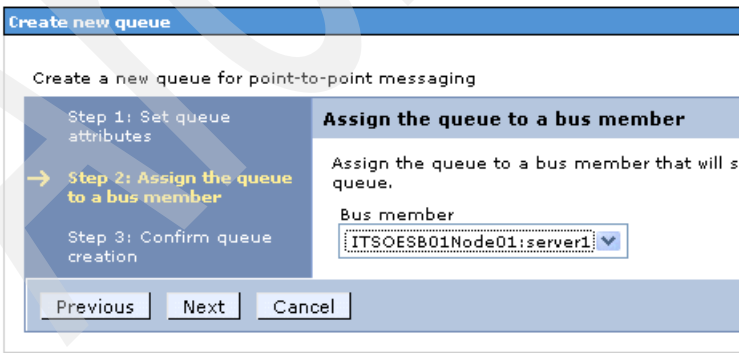

Figure 9-41 Select bus member to store and process messages for the queue

9. On the final page, a summary, click **Finish** and the destination will be created.

With the destination created, route the inbound service requests to the destination (or intermediate) bus.

10. In the list of destinations, click *destination_identifier*.
11. On this panel, shown in Figure 9-42, we are going to set up a default forward routing path.

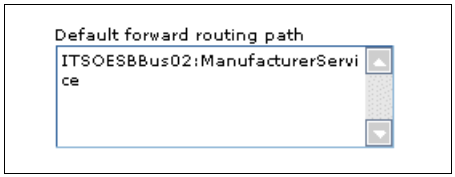

Figure 9-42 Configuring the forward routing path

A default forward routing path is applied to messages sent to a destination if the forward routing path of that message is available. This means that messages sent to the *destination_identifier* (ManufacturerA) destination on *local_bus_name* (ITSOESBBus01) can be routed elsewhere, in our case to the second ESB at *dest_bus_name* (ITSOESBBus02). The format of this box is a comma-separated list of qualified destination names (a bus name and a destination name separated by a colon). The bus name is optional if the destination is on the current bus.

Enter the *forward_routing_path* value and click **OK.** Requests now sent to the ManufacturerService will be forwarded to the second ESB.

12. Repeat steps 2 to 11 for the ManufacturerB destination using the values in Table 9-31.

Table 9-31 Create destination and routing path for ManufacturerB

Item	Value
local_bus_name	ITSOESBBus01
dest_bus_name	ITSOESBBus02
destination_identifier	ManufacturerB
bus_member_name	ITSOESB01Node01:server1
forward_routing_path	ITSOESBBus02:ManufacturerBService

13. Repeat steps 2 to 11 for the ManufacturerC destination using the values in Table 9-32 on page 219.

Table 9-32 Create destination and routing path for ManufacturerB

Item	Value
local_bus_name	ITSOESBBus01
dest_bus_name	ITSOESBBus02
destination_identifier	ManufacturerC
bus_member_name	ITSOESB01Node01:server1
forward_routing_path	ITSOESBBus02:ManufacturerCService

14. Repeat steps 2 to 11 for the WarehouseCallBack destination using the values in Table 9-33 on ITSOESBBus02.

Note: The WarehouseCallBack (and LoggingFacility) destinations must be created on the second ESB (ITSOESBBus02). This is because the applications that call these services are running on a server that attaches only to this bus. The destination then forwards the requests to the remote bus.

Table 9-33 Create destination and routing path for WarehouseCallBackService

Item	Value
local_bus_name	ITSOESBBus02
dest_bus_name	ITSOESBBus01
destination_identifier	WarehouseCallBackService
bus_member_name	ITSOESB02Node01:server1
forward_routing_path	ITSOESBBus01:WarehouseCallBackService

15. Repeat steps 2 to 11 for the WarehouseCallBack destination using the values in Table 9-34 on ITSOESBBus02, and save all of your changes.

Table 9-34 Create destination and routing path for LoggingFacilityService

Item	Value
local_bus_name	ITSOESBBus02
dest_bus_name	ITSOESBBus01
destination_identifier	LoggingFacilityService
bus_member_name	ITSOESB02Node01:server1
forward_routing_path	ITSOESBBus01:LoggingFacilityService

Creating the inbound services

An inbound service is defined to allow Web service clients to connect into the bus. An inbound service converts an incoming Web service request into a message and places it on a destination. The message can then be routed, transformed, and processed. Inbound services can be invoked using SOAP over JMS or SOAP over HTTP by associating the service with the relevant endpoint listener. In this scenario all inbound services requests are performed using SOAP over HTTP.

Inbound services define how Web service consumers communicate with the service integration bus. Outbound services define how the service integration bus communicates with the Web service providers. An inbound and outbound service must be defined for each Web service routed through the service integration bus.

Therefore, the next task is to create the inbound services so that Web service requests can be made into the bus.

Creating inbound services on the first ESB

The first inbound service to be defined on the first ESB (ITSOESBBus01) will be for the LoggingFacilityService using the values defined in Table 9-35.

Table 9-35 Create the inbound service for RetailerService

Item	Value
bus_name	ITSOESBBus01
service_dest_name	RetailerService
wsdl_loc	http://ITSOESB01.itso.ral.ibm.com:9080/Retailer/services/Retailer/wsdl/Retailer_Impl.wsdl
inbound_name	RetailerServiceInboundService
endpoint_name	ITSOESB01Node01:server1:SOAPHTTPChannel1

1. From the admin console expand **Service integration** and click **Buses**.
2. Click *bus_name*.
3. From the bus details page under Services, click **Inbound Services**.
4. Click **New**.

5. In the first window (Figure 9-43), select the Service destination name and supply the template WSDL service definition, then click **Next**:
   - Service destination name

     This field is for specifying the destination that inbound service requests should be placed on. In our case we enter the service destination that was designated while creating the outbound service definition described in “Creating the outbound services” on page 207.

     Select the value of *service_dest_name*.
   - Template WSDL location

     This field is for specifying the WSDL definition for the Web service to be invoked. Enter the value of *wsdl_loc*.

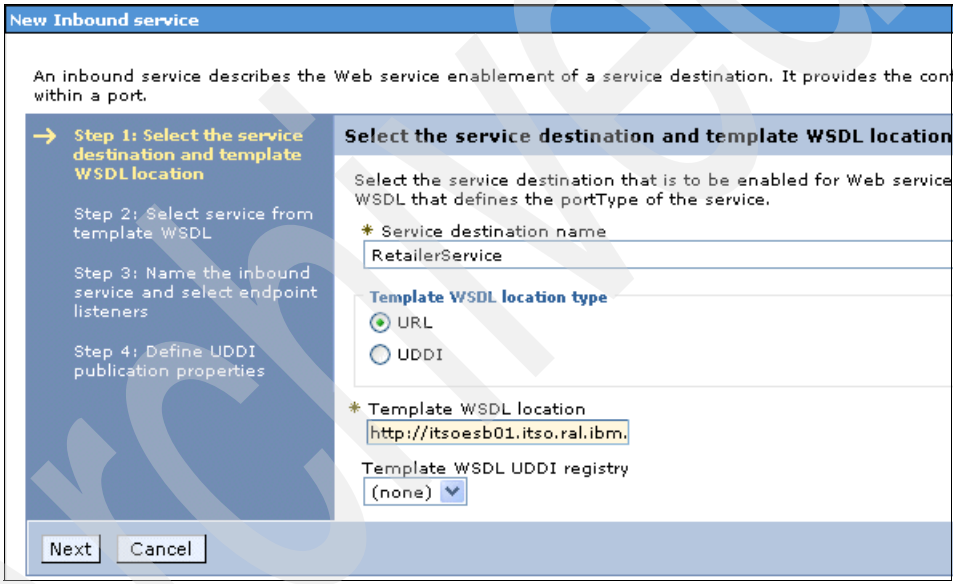

Figure 9-43 Service destination and template WSDL settings page

6. Select the service from the WSDL. Our WSDL has only one entry so accept the defaults and click **Next**.
7. The page shown in Figure 9-44 on page 222, requires you to enter the inbound service name and endpoint listener.
   - Inbound service name

     The name of the inbound service. This will be the name of the service in the WSDL and affects the code that is generated by the tooling. By default it is based on the service destination name with *InboundService* at the end. We will accept the default of *inbound_name*.

– Endpoint listeners

The endpoint listener defines what mechanism will be used to get Web service requests into the inbound service. There should only be one endpoint listener available for selection: *endpoint_name*.

Click **Next**.

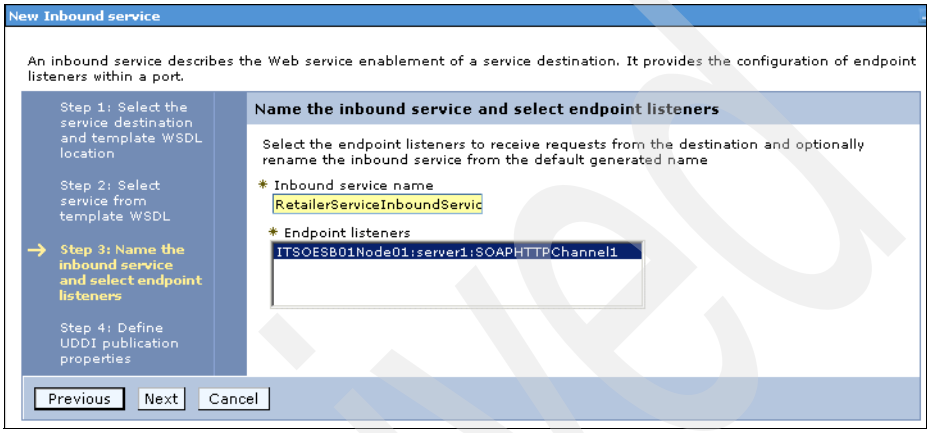

Figure 9-44 Specify inbound service name and endpoint listener

8. The final page enables UDDI-specific properties to be specified. As we are not using UDDI, accept the defaults and click **Finish**.
9. Repeat steps 1 to 8 for the LoggingFacility inbound service using the values in Table 9-36.

Note: The LoggingFacility service will also be defined later as an inbound service on ITSOESBBus02 as it is called by applications that are hosted on that host system.

Table 9-36 Create the inbound service for LoggingFacility

Item	Value
bus_name	ITSOESBBus01
service_dest_name	LoggingFacilityService
wsdl_loc	http://ITSOESB01.itso.ral.ibm.com:9080/LoggingFacility/services/LoggingFacility/wsdl/LoggingFacility_Impl.wsdl
inbound_name	LoggingFacilityServiceInboundService
endpoint_name	ITSOESB01Node01:server1:SOAPHTTPChannel1

10. Repeat steps 1 to 8 for the Warehouse inbound service using the values in Table 9-37.

Table 9-37 Create the inbound service for Warehouse

Item	Value
bus_name	ITSOESBBus01
service_dest_name	WarehouseService
wsdl_loc	http://ITSOESB01.itso.ral.ibm.com:9080/Warehouse/services/Warehouse/wsdl/Warehouse_Impl.wsdl
inbound_name	WarehouseServiceInboundService
endpoint_name	ITSOESB01Node01:server1:SOAPHTTPChannel1

11. Repeat steps 1 to 8 for the ManufacturerA inbound service using the values in Table 9-38.

Note: The Manufacturer services are hosted as outbound services on the second service integration bus (ITSOESBBus02). However, they are called by the Warehouse application that will make their service requests to the first service integration bus (ITSOESBBus01). Therefore, inbound services are required on ITSOESBBus01.

Also note, these instructions assume that ITSOESBBus02 is using a WC_defaulthost port of 9081.

Table 9-38 Create the inbound service for ManufacturerA

Item	Value
bus_name	ITSOESBBus01
service_dest_name	ManufacturerA
wsdl_loc	http://ITSOESB02.itso.ral.ibm.com:9081/Manufacturer/services/Manufacturer/wsdl/Manufacturer_Impl.wsdl
inbound_name	ManufacturerAInboundService
endpoint_name	ITSOESB01Node01:server1:SOAPHTTPChannel1

12. Repeat steps 1 to 8 for the ManufacturerB inbound service using the values in Table 9-39.

Table 9-39 Create the inbound service for ManufacturerB

Item	Value
bus_name	ITSOESBBus01
service_dest_name	ManufacturerB
wsdl_loc	http://ITSOESB02.itso.ral.ibm.com:9081/Manufacturer/services/Manufacturer/wsdl/Manufacturer_Impl.wsdl
inbound_name	ManufacturerBInboundService
endpoint_name	ITSOESB01Node01:server1:SOAPHTTPChannel1

13. Repeat steps 1 to 8 for the ManufacturerC inbound service using the values in Table 9-40.

Table 9-40 Create the inbound service for ManufacturerC

Item	Value
bus_name	ITSOESBBus01
service_dest_name	ManufacturerC
wsdl_loc	http://ITSOESB02.itso.ral.ibm.com:9081/Manufacturer/services/Manufacturer/wsdl/Manufacturer_Impl.wsdl
inbound_name	ManufacturerCInboundService
endpoint_name	ITSOESB01Node01:server1:SOAPHTTPChannel1

14. That completes the inbound service creation steps for the first ESB. Save your changes to the master configuration.

Creating inbound services on the second ESB

Follow the procedures in the previous section, using steps 1 to 8 from “Creating the inbound services” on page 220.

1. The first inbound service to be defined on the second ESB (ITSOESBBus02) is the LoggingFacilityService using the values defined in Table 9-41 on page 225.

Table 9-41 Create the inbound service for LoggingFacility

Item	Value
bus_name	ITSOESBBus02
service_dest_name	LoggingFacilityService
wsdl_loc	http://ITSOESB01.itso.ral.ibm.com:9080/LoggingFacility/services/LoggingFacility/wsdl/LoggingFacility_Impl.wsdl
inbound_name	LoggingFacilityServiceInboundService
endpoint_name	ITSOESB02Node01:server1:SOAPHTTPChannel1

Note: The LoggingFacility service is hosted as an outbound service on the first service integration bus (ITSOESBBus01). However, it is called by the Manufacturer MDB application that will make its service request to the second service integration bus (ITSOESBBus02). Therefore, inbound services are required on ITSOESBBus02. Also, the LoggingFacility service is also defined as an inbound service on ITSOESBBus01 as it is called by applications that are hosted on that host system as well.

2. Repeat step 1 on page 220 to step 8 on page 222 for the WarehouseCallBack inbound service using the values defined in Table 9-42.

Table 9-42 Create the inbound service for WarehouseCallBack

Item	Value
bus_name	ITSOESBBus02
service_dest_name	WarehouseCallBackService
wsdl_loc	http://ITSOESB01.itso.ral.ibm.com:9080/Warehouse/services/WarehouseCallBack/wsdl/WarehouseCallBack_Impl.wsdl
inbound_name	WarehouseCallBackServiceInboundService
endpoint_name	ITSOESB02Node01:server1:SOAPHTTPChannel1

Note: The WarehouseCallBack service is hosted as an outbound service on the first service integration bus (ITSOESBBus01). However, it is called by the Manufacturer MDB application that will make its service request to the second service integration bus (ITSOESBBus02). Therefore, inbound services are required on ITSOESBBus02.

3. Save your changes to the master configuration.

Editing the client bindings to call inbound services

We have now defined outbound services that point to underlying Web services, and we have defined inbound services that map to the outbound services. The final task is to modify the WS-I applications so they are retargeted to invoke services through the service integration bus inbound services. This is done by overriding the target endpoint in the Web service client bindings of the WS-I application EJB and Web modules.

The client bindings define the WSDL file name and preferred ports. The relative path of a Web service in a module is specified within a WSDL file that contains the actual URL to be used for requests. The address is needed only if the original WSDL file did not contain a URL, or when a different address is needed.

By specifying the name of an endpoint URI that is used to override the current endpoint, a client uses this endpoint instead of the endpoint specified in the WSDL file. If an assembled application contains a Web service client that is statically bound, the client is locked into using the implementation (service end point) identified in the WSDL file used during development. Overriding the endpoint is an alternative to configuring the deployed WSDL attribute. The overridden endpoint URI attribute is specified on a per-port basis. It does not require an alternative WSDL file within the module. The overridden endpoint URI takes precedence over the deployed WSDL attribute. The client uses this value for the service endpoint URI or SOAP address instead of the value in the static client bindings.

Editing client bindings for the second ESB

Three enterprise applications are installed in the server running on ITSOESB02.itso.ral.ibm.com: ManufacturerA, ManufacturerB, and ManufacturerC. Each of these enterprise applications contain two Web service clients: a Web service client for LoggingFacilityService and a Web service client for WarehouseCallBack. All of these Web service clients must be redirected to point to the relevant inbound service.

For example, ManufacturerA needs both Web service clients redirected as shown in Figure 9-45 on page 227.

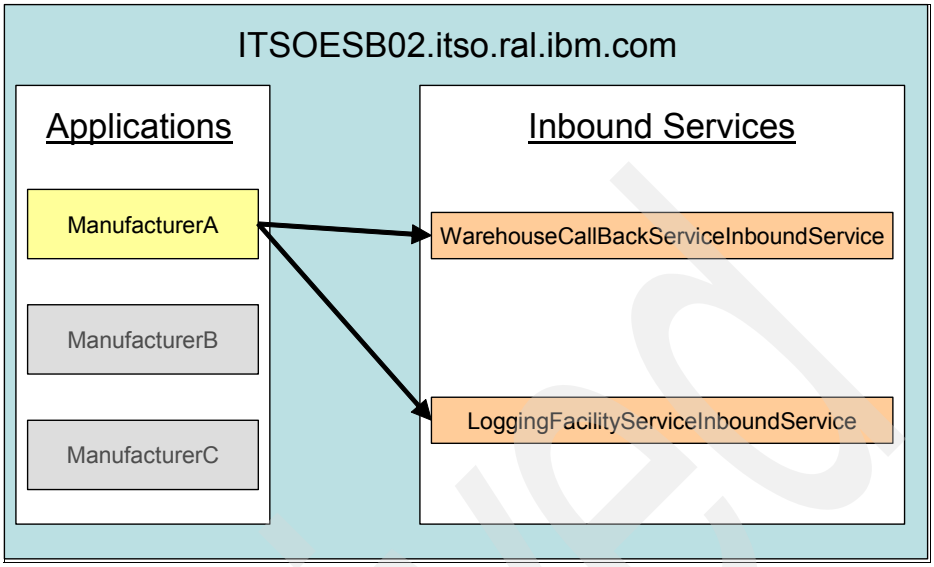

Figure 9-45 Web service client bindings to modify for ManufacturerA

Initially we retarget the WarehouseCallBackService Web service client in ManufacturerA using the values in Table 9-43.

Note: The following instructions assume that the server running on ITSOESB02.itso.ral.ibm.com uses port 9081 for WC_defaulthost. To determine the WC_defaulthost port being used, perform the following:

- Click **Servers** → **Application Servers** and click on the server assigned to the node ITSOESB02Node01.
- Under Communications, expand **Ports**. This shows the value of WC_defaulthost.

If the value of WC_defaulthost is other than 9081, you should use this value in place of 9081 in the instructions that follow.

Table 9-43 Edit bindings for WarehouseCallBackService from ManufacturerA

Item	Value
app1_name	Manufacturer
EJB1_name	ManufacturerEJB.jar
web_service_name	WarehouseCallBackService
port_name	WarehouseCallBack

Item	Value
bus_name	ITSOESBBus02
inbound_name	WarehouseCallBackServiceInboundService
publish_WSDL_name	WarehouseCallBackServiceInboundService.zip
WSDL_file_name	ITSOESBBus02.WarehouseCallBackServiceInboundServiceService.wsdl
location_name	http://ITSOESB02.itso.ral.ibm.com:9081/wsgwsoaphttp1/soaphttpengine/ITSOESBBus02/WarehouseCallBackServiceInboundService/ITSOESB02Node01_server1_SOAPHTTPChannel1_InboundPort

1. Open 2 admin console windows.
2. In the first admin console window, expand **Applications** and click **Enterprise Applications**.
3. Select *app1_name*.
4. From the Enterprise Application configuration panel under **Related Items,** click **EJB Modules**.
5. Select *EJB1_name*.
6. Under **Additional Properties,** select **Web services client bindings.**
7. In the *web_service_name* row, select **Edit** in the Port Information column as shown in Figure 9-46.

Web Service	EJB	WSDL Filename	Preferred Port Mappings	Port Information
WarehouseCallBackService	CallbackMessage	Use default (null)	Edit...	Edit...
LoggingFacilityService	CallbackMessage	Use default (null)	Edit...	Edit...

Figure 9-46 Select port information

8. In the second admin console window, expand **Service Integration** and click **Buses**.
9. Select *bus_name*.
10. Under **Additional properties,** click I**nbound Services.**
11. Select *inbound_name*.
12. Under **Additional Properties,** select **Publish WSDL files to zip file.**

13. Select *publish_WSDL_name* (Figure 9-47) and open it with a zip file editor.

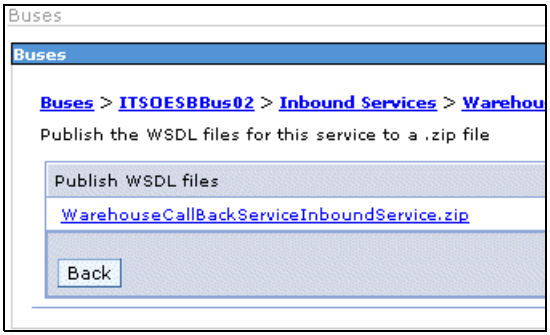

Figure 9-47 Publish WSDL file

14. Open the *wsdl_file_name* file.

15. Copy the *location_name* from the address location tag in the WSDL file as shown in Figure 9-48.

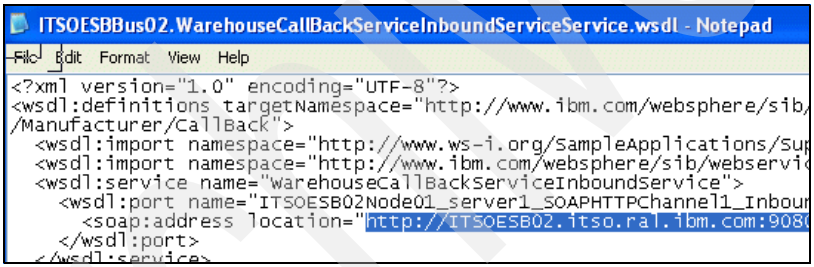

Figure 9-48 Copy the service location

16. Back in the first admin console window, paste the *location_name* into the Overridden Endpoint URL column entry of the *port_name* row, and click **OK**.

Figure 9-49 Override the endpoint URL

The LoggingFacilityService Web client also must be updated. You can determine the URL for the LoggingFacilityService using the method described above, but for brevity, we fast-track this process by providing you with this URL. Complete the following steps to edit the LoggingFacilityService Web service client bindings, using the values defined in Figure 9-44.

Table 9-44 Edit client bindings for LoggingFacilityService from ManufacturerA

Item	Value
app_name	Manufacturer
module_type	EJB Modules
module_name	ManufacturerEJB.jar
web_service_name	LoggingFacilityService
location_name	http://ITSOESB02.itso.ral.ibm.com:9081/wsgwsoaphttp1/soaphttpengine/ITSOESBBus02/LoggingFacilityServiceInboundService/ITSOESB02Node01_server1_SOAPHTTPChannel1_InboundPort

17. Expand **Applications** and click **Enterprise Applications**. Click *app_name*.
18. Click *module_type*, then click *module_name*.
19. Under Additional Properties, click **Web services client bindings**.
20. Click **Edit** under the Port Information column for the Web service *web_service_name*.
21. In the Overridden Endpoint URL field enter *location_name* and click **OK**.

The other two Manufacturer enterprise applications must be updated similarly, as shown in Figure 9-50.

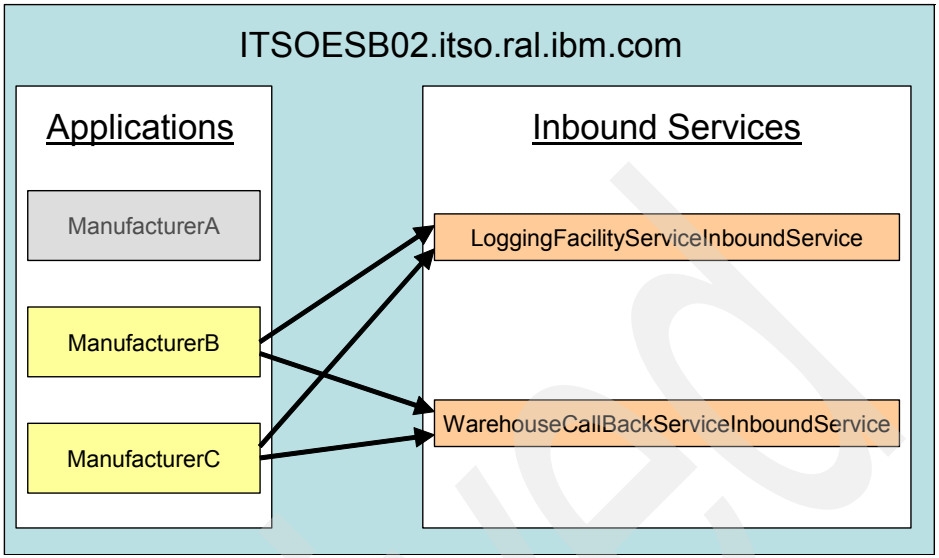

Figure 9-50 Web service client bindings to modify for Manufacturers B and C

22. Edit the Web service bindings for ManufacturerB by repeating steps 17 to 21. There are two bindings to change; use the values in Table 9-45 and Table 9-46.

Table 9-45 Edit client bindings for WarehouseCallBackService from ManufacturerB

Item	Value
app_name	ManufacturerB
module_type	EJB Modules
module_name	ManufacturerEJB.jar
web_service_name	WarehouseCallBackService
location_name	http://ITSOESB02.itso.ral.ibm.com:9081/wsgwsoaphttp1/soaphttpengine/ITSOESBBus02/WarehouseCallBackServiceInboundService/ITSOESB02Node01_server1_SOAPHTTPChannel1_InboundPort

Table 9-46 Edit client bindings for LoggingFacilityService from ManufacturerC

Item	Value
app_name	ManufacturerC
module_type	EJB Modules

Item	Value
module_name	ManufacturerEJB.jar
web_service_name	LoggingFacilityService
location_name	http://ITSOESB02.itso.ral.ibm.com:9081/wsgwsoaphttp1/soaphttpengine/ITSOESBBus02/LoggingFacilityServiceInboundService/ITSOESB02Node01_server1_SOAPHTTPChannel1_InboundPort

Editing client bindings for the first ESB

All Web service clients hosted on the first ESB (ITSOESB01) also have to be edited to point to the relevant inbound services. All of these Web service clients are hosted in a single enterprise application: UIRetailerLogging. This enterprise application contains the SCMSampleUI, Retailer, Warehouse, and LoggingFacility applications.

Figure 9-51 shows the client bindings to be edited for these applications.

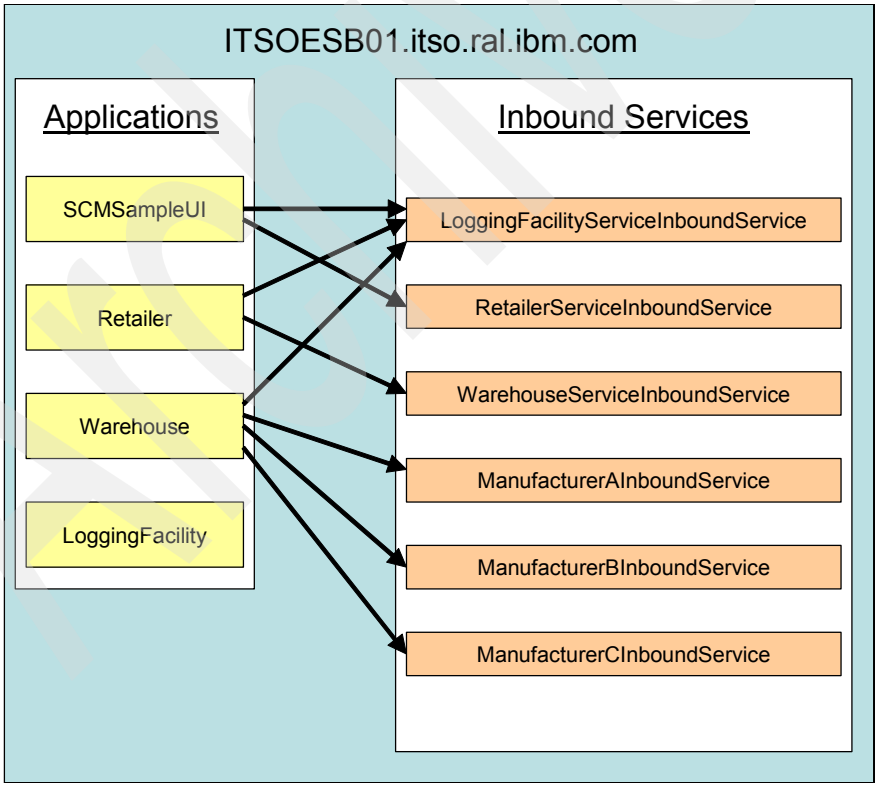

Figure 9-51 Web service client bindings to modify for UIRetailerLogging

23. Update each Web service client by repeating steps 17 to 21 in the previous section. There are eight Web service clients to update in total. Use the values in tables Table 9-47 through Table 9-54 on page 235, then save your changes to the master configuration.

Table 9-47 Edit client bindings for RetailerService from SCMSampleUIWeb

Item	Value
app_name	UIRetailerLogging
module_type	Web module
module_name	SCMSampleUIWeb.war
web_service_name	RetailerService
location_name	http://ITSOESB01.itso.ral.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/ITSOESBBus01/RetailerServiceInboundService/ITSOESB01Node01_server1_SOAPHTTPChannel1_InboundPort

Table 9-48 Edit client bindings for LoggingFacilityService from SCMSampleUIWeb

Item	Value
app_name	UIRetailerLogging
module_type	Web module
module_name	SCMSampleUIWeb.war
web_service_name	LoggingFacilityService
location_name	http://ITSOESB01.itso.ral.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/ITSOESBBus01/LoggingFacilityServiceInboundService/ITSOESB01Node01_server1_SOAPHTTPChannel1_InboundPort

Table 9-49 Edit client bindings for LoggingFacilityService from RetailerWeb

Item	Value
app_name	UIRetailerLogging
module_type	Web module
module_name	RetailerWeb.war
web_service_name	LoggingFacilityService
location_name	http://ITSOESB01.itso.ral.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/ITSOESBBus01/LoggingFacilityServiceInboundService/ITSOESB01Node01_server1_SOAPHTTPChannel1_InboundPort

Table 9-50 Edit client bindings for WarehouseService from RetailerWeb

Item	Value
app_name	UIRetailerLogging
module_type	Web module
module_name	RetailerWeb.war
web_service_name	WarehouseService
location_name	http://ITSOESB01.itso.ral.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/ITSOESBBus01/WarehouseServiceInboundService/ITSOESB01Node01_server1_SOAPHTTPChannel1_InboundPort

Table 9-51 Edit client bindings for LoggingFacilityService from WarehouseService

Item	Value
app_name	UIRetailerLogging
module_type	EJB Modules
module_name	WarehouseEJB.jar
web_service_name	LoggingFacilityService
location_name	http://ITSOESB01.itso.ral.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/ITSOESBBus01/LoggingFacilityServiceInboundService/ITSOESB01Node01_server1_SOAPHTTPChannel1_InboundPort

Table 9-52 Edit client bindings for ManufacturerService from WarehouseService

Item	Value
app_name	UIRetailerLogging
module_type	EJB Modules
module_name	WarehouseEJB.jar
web_service_name	ManufacturerService
location_name	http://ITSOESB01.itso.ral.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/ITSOESBBus01/ManufacturerAInboundService/ITSOESB01Node01_server1_SOAPHTTPChannel1_InboundPort

Table 9-53 Edit client bindings for ManufacturerBService from WarehouseService

Item	Value
app_name	UIRetailerLogging
module_type	EJB Modules
module_name	WarehouseEJB.jar
web_service_name	ManufacturerBService
location_name	http://ITSOESB01.itso.ral.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/ITSOESBBus01/ManufacturerBInboundService/ITSOESB01Node01_server1_SOAPHTTPChannel1_InboundPort

Table 9-54 Edit client bindings for ManufacturerCService from WarehouseService

Item	Value
app_name	UIRetailerLogging
module_type	EJB Modules
module_name	WarehouseEJB.jar
web_service_name	ManufacturerCService
location_name	http://ITSOESB01.itso.ral.ibm.com:9080/wsgwsoaphttp1/soaphttpengine/ITSOESBBus01/ManufacturerCInboundService/ITSOESB01Node01_server1_SOAPHTTPChannel1_InboundPort

9.3.6 Testing the scenario

The scenario can now be tested. The WS-I sample application can be tested by entering the following URL in a Web browser on the machine that is hosting the application server:

```
http://ITSOESB01.itso.ral.ibm.com:9080/SCMSampleUI/
```

This should start the Supply Chain Management Sample Application (Figure 9-52).

Figure 9-52 SCM Sample Application

The following steps describe how to use the application:

1. To retrieve a list of products, click **Place New Order**. This displays a list of 10 products, as shown in Figure 9-53.

Figure 9-53 SCM Sample product listing

2. You can order multiple quantities of each product. If the warehouse has sufficient stock for the product, an order will be placed. If the placement of the order causes the warehouse's stock level of that product to drop below a certain threshold, then a reorder request is sent to the manufacturer of the product.

   The warehouse stock level is stored in the org.ws_i.www.Impl.WarehouseImpl class in the WarehouseEJB project. For example, Table 9-55 shows the stock level for the first three products.

Table 9-55 Warehouse stock levels

Product number	Current level	Minimum level	Maximum level
605001	10	5	25
605002	7	4	20
605003	15	10	50

If the current stock level falls below the minimum stock level, the stock is reordered so that, after the reorder has arrived, the stock will be at maximum stock level. For example, you order six items of 605001. This reduces the current stock level to four (10 - 6). A reorder will be made for 21 new items.

Each manufacturer only manufactures certain products. For example, ManufacturerA manufacturers products 605001, 605004, and 605007.

3. Place orders for multiple products in the sample application by entering quantities and pressing **Submit Order**. For example, order three items of product 605001 and six items of product 605002. This triggers a reorder of product 605002 with ManufacturerB.
4. The order status window (Figure 9-54) shows which orders were placed and which orders were not placed due to insufficient stock.

Figure 9-54 SCM Sample order status page

5. Click **Track Order** to see the entries that were written to the LoggingFacility. As new entries are added to the Logging Facility, you must refresh this screen by clicking **Order Status** and then clicking **Track Order** again. Figure 9-55 on page 239 shows the results of an order in which products 605001 and 605002 were shipped and a reorder for 19 units of product 605002 was placed with ManufacturerB.

Figure 9-55 SCM Sample track order page

6. To start a new order, click **Configure**. At this point, all state is lost, and the warehouse stock levels return to their default values.
7. To test calls to all manufacturers, enter a quantity of 6 against products 605001, 605002, and 605003 as shown in Figure 9-56.

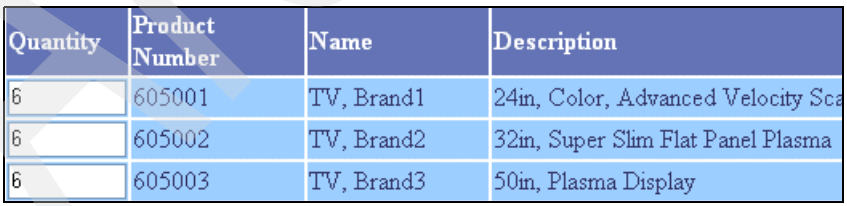

Figure 9-56 Enter quantity values

By entering these values, a replenishment request will be sent to each of the three manufacturers between the two ESBs, with logging and callback requests flowing in reverse. Figure 9-57 on page 240 shows the output from this request.

Service ID	Event ID	Description
Retailer.submitOrder	UC1-5	Order placed by A12345-9876543-xyz for 605001, 605002, 605003
WarehouseA.ShipGoods	UC2-2-1	WarehouseA will determine its ability to ship product(s) 605001, 605002, 6050
ManufacturerA.submitPO	UC3-3	ManufacturerA is replenishing stock for 605001.
ManufacturerB.submitPO	UC3-3	ManufacturerB is replenishing stock for 605002.
WarehouseA.ShipGoods	UC2-2-2	WarehouseA is able to ship 605001, 605002, 605003.
Retailer.submitOrder	UC1-9	Processing of the order from A12345-9876543-xyz has finished normally
ManufacturerC.submitPO	UC3-3	ManufacturerC is replenishing stock for 605003.
ManufacturerA.submitPO	UC5-5	ManufacturerA has produced additional units of 605001 and is shipping 21 units.
ManufacturerB.submitPO	UC5-5	ManufacturerB has produced additional units of 605002 and is shipping 19 units.
WarehouseA.submitSN	UC3-7-1	WarehouseA has received notice that product 605001 has been shipped by Mar
ManufacturerC.submitPO	UC5-5	ManufacturerC has produced additional units of 605003 and is shipping 41 units.
WarehouseA.submitSN	UC3-7-2	WarehouseA has replenished stock for product 605001
WarehouseA.submitSN	UC3-7-1	WarehouseA has received notice that product 605002 has been shipped by Mar
WarehouseA.submitSN	UC3-7-2	WarehouseA has replenished stock for product 605002
WarehouseA.submitSN	UC3-7-1	WarehouseA has received notice that product 605003 has been shipped by Mar
WarehouseA.submitSN	UC3-7-2	WarehouseA has replenished stock for product 605003

Figure 9-57 Output from an order causing replenishment from all manufacturers

10

Directly Connected heterogeneous ESBs

In this chapter, we expand the Directly Connected ESB pattern to show how to integrate two heterogeneous ESB implementations. On a technology and product level, this chapter describes the use of the WebSphere MQ Link to federate a WebSphere Application Server V6 hosted ESB with a WebSphere Business Integration Message Broker V5 ESB.

In this chapter, the following points are discussed:

- Design guidelines and business needs addressed by the sample scenario, and selection of the relevant ESB integration patterns.
- Development guidelines that describe how to create a mediation in Rational Application Developer to run in WebSphere Application Server, and how to create message flows and message sets for WebSphere Business Integration Message Broker.
- Runtime guidelines to create and integrate two ESBs, one implemented in WebSphere Application Server and one implemented in WebSphere Business Integration Message Broker.

The IBM Enterprise Service Bus strategy:

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

- WebSphere Enterprise Service Bus V6

  Delivers an ESB with Web services connectivity and data transformation.

- WebSphere Message Broker V6

  Delivers an advanced ESB with universal connectivity and data transformation.

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

For more information about the IBM ESB strategy see:

http://www.ibm.com/software/info1/websphere/index.jsp?tab=landings/esb

10.1 Design guidelines

This section discusses sample business needs for linking two heterogeneous ESBs that belong to different organizations. It maps these business requirements to the sample scenario and to the appropriate ESB integration patterns.

10.1.1 Business scenario

The business scenario implemented in this chapter represents a variation of the WS-I sample business scenario as defined in Chapter 7, “The business scenario used in this book” on page 117. It defines a supply chain management process that is split across two organizations, as seen in Figure 10-1.

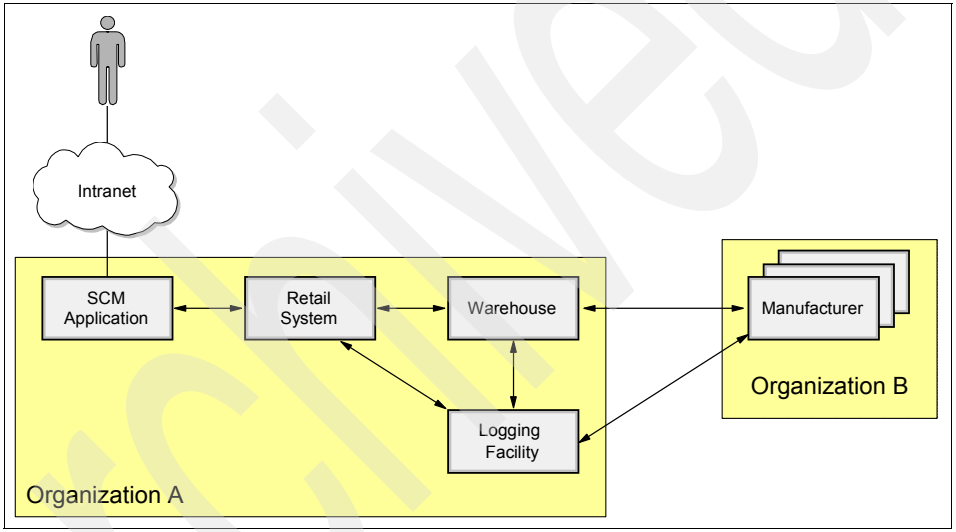

Figure 10-1 High-level business context showing the existing infrastructure

High-level business context

The business context for this scenario is the same as the business context described in Chapter 9, “Directly Connected homogeneous ESBs” on page 149. It describes a supply chain management scenario in which customers access an electonics retailer’s Web site, review a catalog of available products, and place orders at a warehouse for items such as televisions, DVD players, and video cameras. The stock of the warehouse is replenished by placing orders with the relevant manufacturer. The manufacturing systems reside in a different organization from the other components in the supply chain management scenario.

Organizational overview

As a result of the acquisition of the manufacturers, the company has two separate organizations, each containing its own functional components:

- Organization A
  - Internet based e-commerce systems (SCM application)
  - Retail system
  - Warehouse and delivery
- Organization B
  - Manufacturing capability

Organization A contains bespoke software applications linked together using the Enterprise Service Bus technology in WebSphere Application Server V6.

Organization B contains three stand-alone divisional systems known as ManufacturerA, ManufacturerB, and ManufacturerC.

Integration of organizations

The manufacturing conglomerate itself has grown over time and actually currently encompasses three separate divisions of systems, known as ManufacturerA, ManufacturerB, and ManufacturerC. The Manufacturers have not yet embraced the J2EE paradigm and respond to fulfillment orders using messages that have an XML physical format and which are transported using WebSphere MQ as a JMS provider. In order to conduct business with external buyers (such as the warehouse of Organization A) the manufacturing company has previously linked the manufacturers using IBM WebSphere Business Integration Message Broker. This product provides a SOAP over JMS (with WebSphere MQ as the JMS provider) interface for enterprises wishing to get orders fulfilled by the Manufacturer.

The ESB built-in WebSphere Application Server will use the SOAP over JMS capabilities of WebSphere Business Integration Message Broker to integrate the two ESBs.

10.1.2 Selecting ESB integration patterns

We can apply the ESB integration patterns to the business scenario. The ESB integration patterns are described in more detail in Chapter 6, “Integrating ESBs” on page 87.

Selecting an ESB Topology pattern

The ESB Topology patterns describe network relationships between ESBs. To help us select the appropriate ESB Topology pattern, we should select the business and IT drivers that apply to our scenario, as described in 6.3.1, “ESB Topology patterns overview” on page 90.

Our scenario requires the following business drivers:

- Limited interaction between different enterprise governance zones

Our scenario requires the following IT drivers:

- Route requests between two ESBs
- Only requires basic interactions

Therefore, based on these requirements for a simple, routing-based connection, we select the Directly Connected ESBs runtime pattern (Figure 10-2). This pattern describes a simple point-to-point connection between the two ESBs.

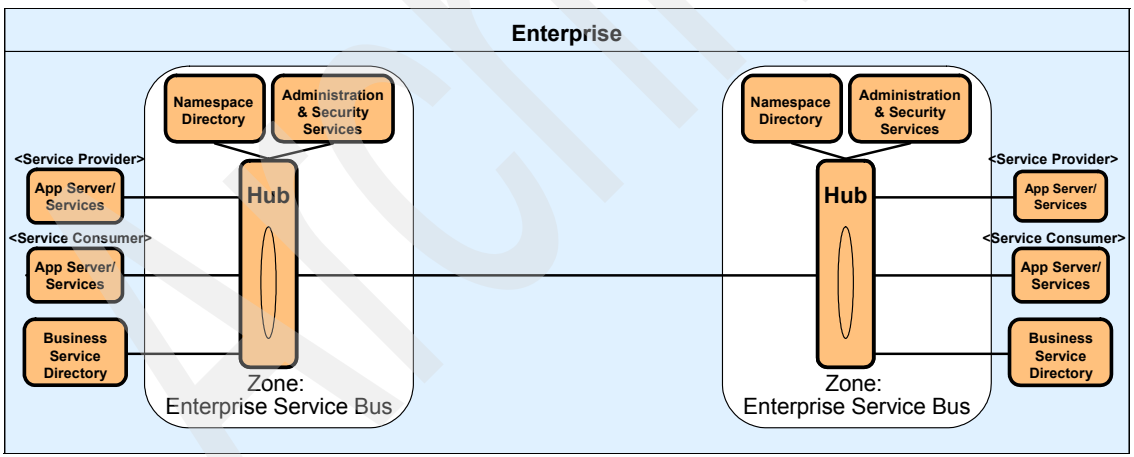

Figure 10-2 Directly Connected ESBs runtime pattern

This connection between the two ESBs will require an ESB Adapter Connector to connect the two ESBs together. The selection of the relevant ESB Adapter Connector is described later in this chapter.

Product mapping for the Directly Connected ESBs pattern

Figure 10-3 shows the Product mapping for the Directly Connected ESBs pattern.

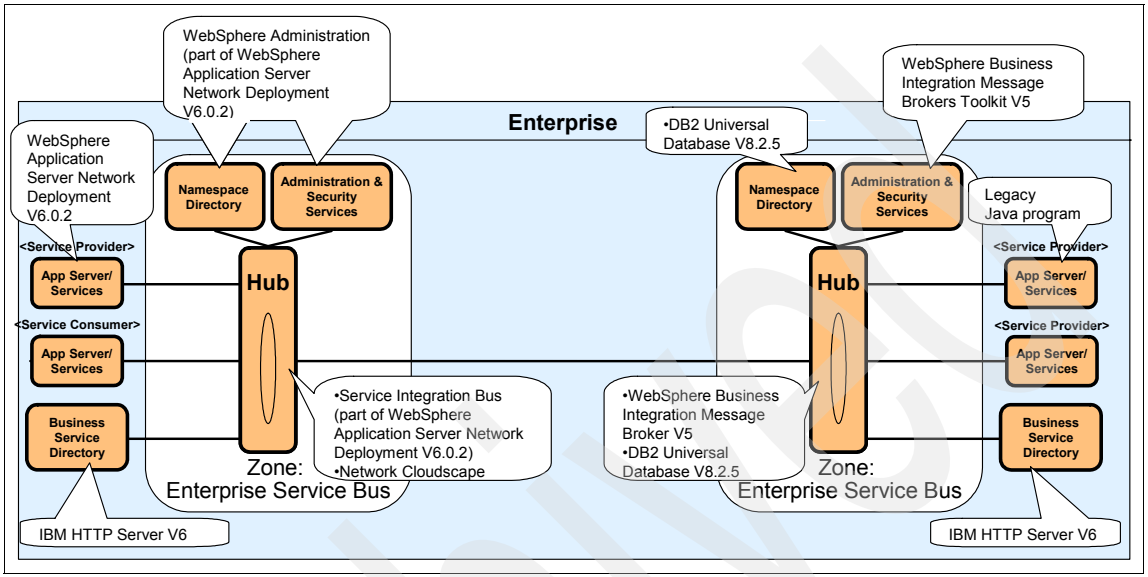

Figure 10-3 Product mapping for heterogeneous Directly Connected ESBs

In this Product mapping, one ESB is implemented in WebSphere Application Server Network Deployment V6, and the other is implemented in WebSphere Business Integration Message Broker V5.

ESB based on WebSphere Application Server

WebSphere Application Server Network Deployment V6 is used in single-server mode in this scenario to create a single WebSphere Application Server profile. The use of Network Deployment enables this solution to be customized later to support clustering and failover support.

The Hub of the ESB is implemented using the service integration bus feature of WebSphere Application Server Network Deployment. The service integration bus can be administered through the administration capability of WebSphere Application Server Network Deployment V6.

Service consumers and providers connect to the service integration bus using SOAP over HTTP and SOAP over JMS.

The Business Service Directory is supported by an IBM HTTP Server, which is used to host the WSDL descriptions of each Web service used with the ESB4.

Network Cloudscape is used to store the Service Data Objects (SDO) repository. In a production environment, a more powerful and reliable data store should be used to host the SDO repository such as DB2 Universal Database.

ESB based on WebSphere Business Integration Message Broker ESB

The Hub node for the second ESB is implemented as a combination of WebSphere Business Integration Message Broker V5, DB2 Universal Database V8.2.5, and WebSphere MQ.

The Namespace Directory containing the endpoint locations of the Manufacturers is stored in a table in DB2 Universal Database. This enables the endpoint locations to be changed without having to change the message flows in WebSphere Business Integration Message Broker that contain the ESB routing functionality.

The Hub receives SOAP over JMS messages, modifies the message format, and sends non-SOAP formatted JMS messages to the legacy manufacturers. These manufacturers are implemented as legacy Java applications.

The ESB is centrally administered using the WebSphere Business Integration Message Brokers Toolkit. The Business Service Directory is supported by an IBM HTTP Server, which is used to host the WSDL descriptions of each Web service used with the ESB.

Selecting an ESB Adapter Connector pattern

The ESB Adapter Connector patterns can be applied to the ESB Topology patterns to describe how calls between ESBs are implemented. The two Adapter patterns are:

- Adapter Connector pattern

  Service behaviors for the interaction determined at build time.

- Boundary Services Adapter Connector pattern

  Service behaviors for the integration determined at runtime.

Using the business and IT drivers for the ESB Adapter Connector pattern (as described in 6.3.3, “ESB Adapter Connector patterns overview” on page 93), we can determine whether a Adapter Connector or Boundary Services Adapter Connector pattern, or a combination of the two, is required in our scenario.

In our scenario, we have the following IT driver requirements:

- Primary requirement: Translate between otherwise incompatible technologies.
- Binding of service request context to ESB capabilities is done at build time.

Specifically, we need to mediate a technology mismatch between the SOAP/JMS messages produced by WebSphere Application Server and the SOAP/JMS

messages consumed by WebSphere Business Integration Message Broker. These products use different JMS implementations. Conformance with the JMS specification does not promise or provide for interoperability. We need an intermediary to mediate the incompatibility between different JMS implementations by acting as a compatible JMS client for each side of the interaction. The ESB Adapter Connector pattern, shown in Figure 10-4, addresses this requirement.

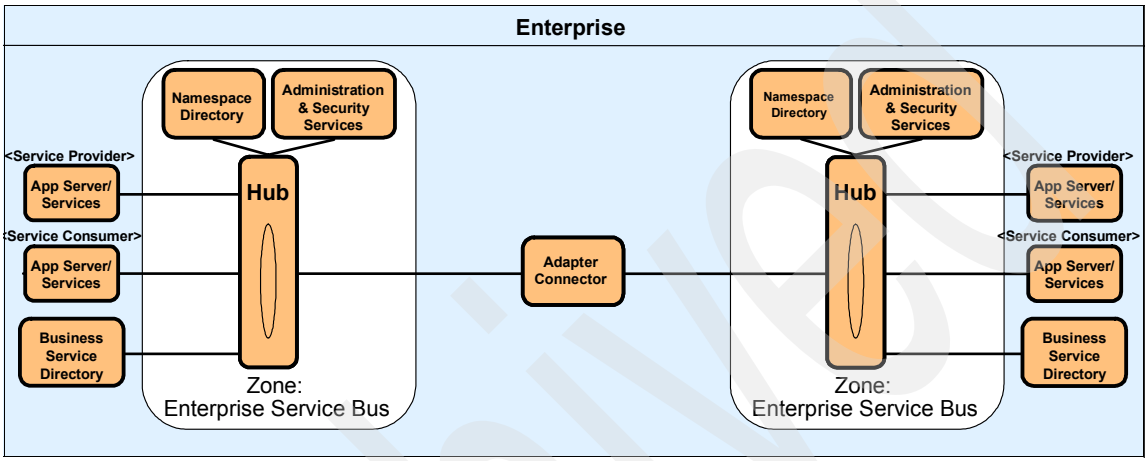

Figure 10-4 ESB Adapter Connector pattern

We use an ESB Adapter Connector pattern to connect the two ESB Hub nodes. When a service consumer on the first ESB tries to invoke a service provider on the second ESB, this connection is sent through the Connector node. The Connector node transforms the SOAP/JMS message from WebSphere Application Server to WebSphere Business Integration Message Broker.

10.2 Development guidelines

The development guidelines for this scenario describe the development steps required to build an ESB in WebSphere Application Server, an ESB in WebSphere Business Integration Message Broker, and all attached service consumers and providers.

10.2.1 ESB based on WebSphere Application Server

This section describes the development of an ESB based on WebSphere Application Server and its associated service consumers and providers.

Message transformation mediation

This mediation converts a SOAP request originating from WebSphere Business Integration Message Broker into one that can be processed using WebSphere Application Server. This conversion adds JMS headers to the message, and forwards the message on to the correct JMS queue.

Implementing the mediation

1. Create a new Enterprise Application Project in Rational Application Developer:
   a. Start the New Enterprise Application Project Wizard (**File** → **New** → **Project** → **J2EE** → **Enterprise Application Project**).
   b. Specify the name as `BrokerJMSProcessor`.
   c. Add a new module:
      i. Deselect **Create default module projects**.
      ii. Select **Enterprise Java Bean**.
      iii. Specify the name as `BrokerJMSProcessorEJB`.
   d. Complete the New Enterprise Application Project Wizard.

2. Create a new Java class called `BrokerJMSProcessor` in the `com.ibm.itso.ral` package with an interface of `com.ibm.websphere.sib.mediation.handler.MediationHandler` within the source folder BrokerJMSProcessorEJB/ejbModule. See Figure 10-5.

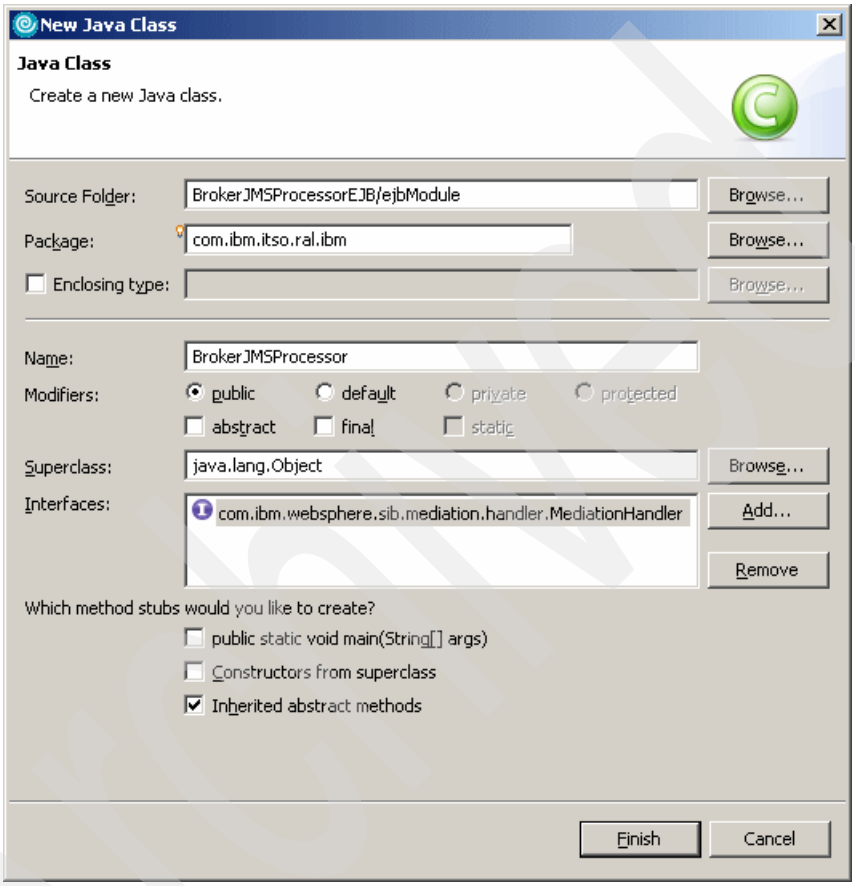

Figure 10-5 Creating a new Java class

3. Edit the Deployment Descriptor for the EJB BrokerJMSProcessorEJB. In the J2EE Perspective, select the correct deployment descriptor (Figure 10-6 on page 251).
   a. Select the Mediation Handlers pane.

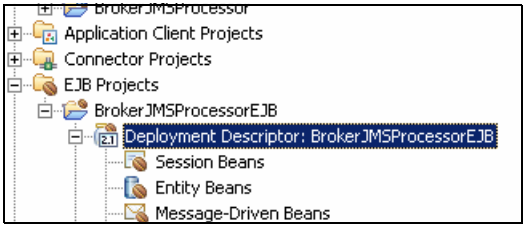

Figure 10-6 Edit the Deployment Descriptor for the EJB BrokerJMSProcessorEJB

b. Add a new mediation handler (Figure 10-7 on page 251):

- The Name of the Mediation Handler: `BrokerJMSProcessor`
- The Handler class: `com.ibm.itso.ral.BrokerJMSProcessor`

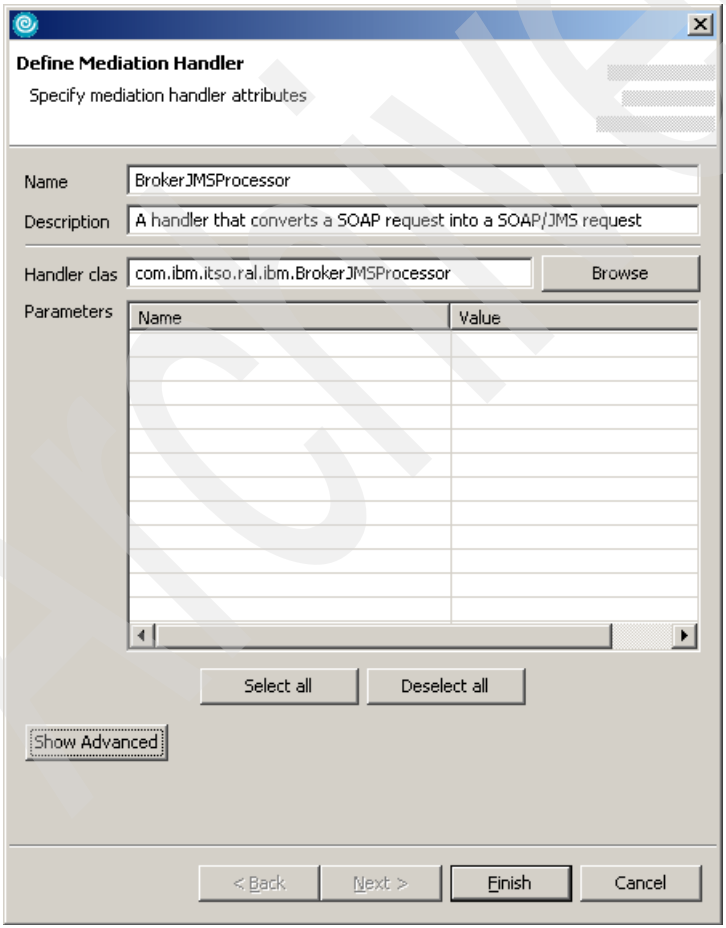

Figure 10-7 Defining a new Mediation Handler

4. Complete the Define Mediation Handler property page, then save the changes to the deployment descriptor. Figure 10-8 on page 252 shows the completed property page.

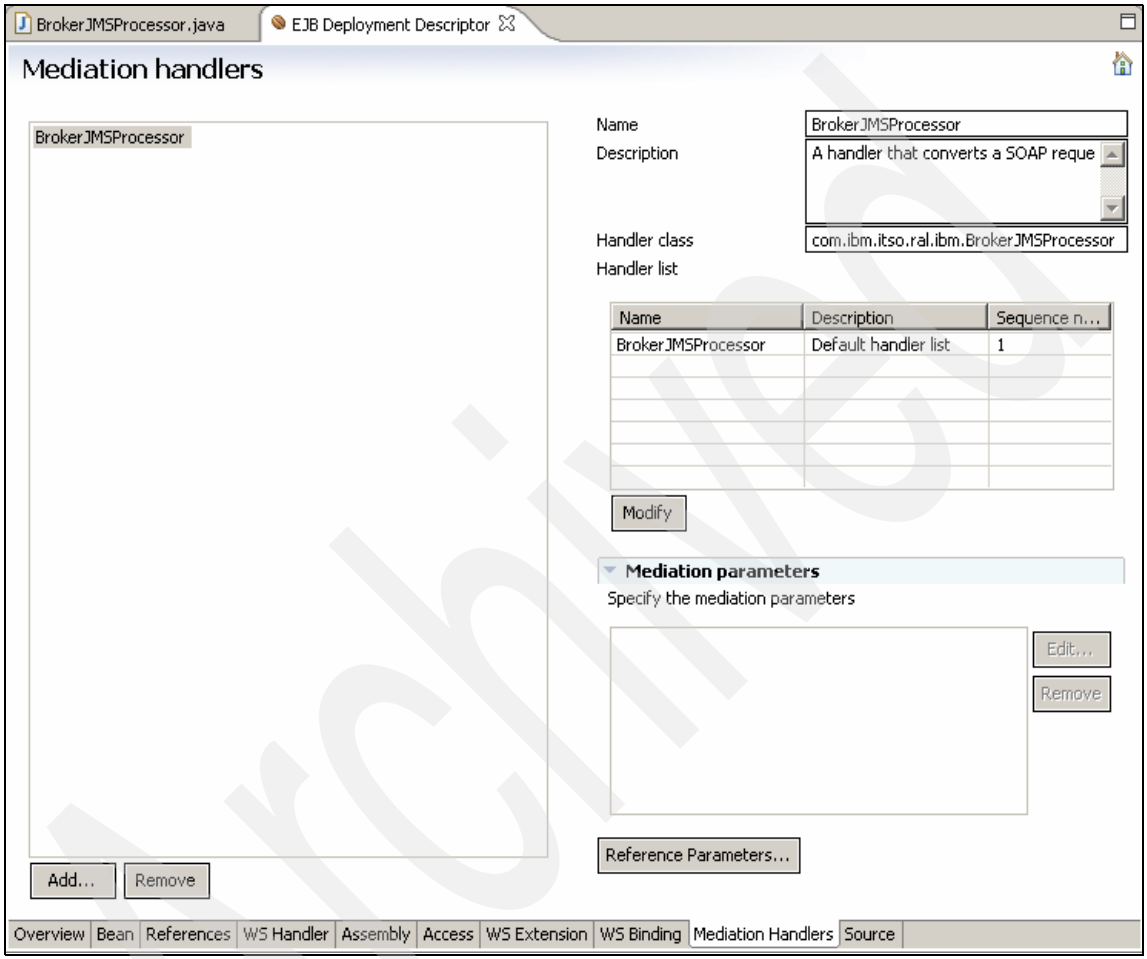

Figure 10-8 The defined mediation handler

5. Write the mediation code.

   a. There are two utility methods in the handler. One parses the endpointURL into its constituent properties, the other retrieves the definition of a JMS queue from JNDI (Example 10-1).

Example 10-1 Utility methods to parse the endpointURL and retrieve a JMS Queue from JNDI

```
final private static Properties getProperties(URL endpoint)
          throws UnsupportedEncodingException {
       Properties result = new Properties();

       StringTokenizer tokenizer = new StringTokenizer(endpoint.getQuery(),
             "|&");

       while (tokenizer.hasMoreTokens()) {
          String property = tokenizer.nextToken();

          int index = property.indexOf('=');
          String key, value;
          if (index != -1) {
             key = property.substring(0, index);
             value = property.substring(index + 1);
          } else {
             key = property;
             value = "";
          }

          key = URLDecoder.decode(key, "UTF-8");
          value = URLDecoder.decode(value, "UTF-8");
          result.put(key, value);

       }
       System.out.println("getProperties" + result);
       return result;

    }
final private static JmsQueue getDestination(String jndiReference) {
       JmsQueue result;
       try {

          InitialContext initContext = new InitialContext();

          Object objResult = initContext.lookup(jndiReference);

          if (objResult == null) {
             result = null;
          } else {
             if (objResult instanceof JmsQueue) {
                result = (JmsQueue) objResult;

             } else {
                result = null;
             }
```

```
        }

      } catch (Exception ex) {
         ex.printStackTrace();
         result = null;
      }
      return result;
   }
```

Note: To resolve the correct packages for the classes used in Example 10-1, use the **Source** → **Organize Imports** feature of Rational Application Developer. Where multiple packages are offered, choose:

- java.util.Properties
- java.net.URL

b. Next, write the handle method, which will be called each time a message is handled by this mediation.

   To retrieve the message that is being handled, the `getSIMessage()` method is called in the handle method (Example 10-2).

Example 10-2 Obtaining a message object inside a mediation

```
SIMessageContext siMsgContext = ((SIMessageContext) arg0);
SIMessage siMessage = siMsgContext.getSIMessage();
```

c. Next we reference the user property `endpointURL`. This will be set by WebSphere Business Integration Message Broker and contains all of the properties that are necessary to invoke a SOAP over JMS message. We complete this and following calls within a try-catch block (Example 10-3).

Example 10-3 Getting the JMS user property `endpointURL`

```
try {
   String endpointURL = (String) siMessage.getUserProperty("endpointURL");
```

d. The endpointURL is then parsed into its properties and the `targetService` JMS user property set (Example 10-4).

Example 10-4 Setting the targetService JMS user property

```
URL endpoint = new URL(endpointURL);
Properties properties = getProperties(endpoint);

String targetService = (String) properties.getProperty(
   "targetService", null);

if (targetService != null && targetService.length() != 0) {
```

```
   siMessage.setUserProperty("targetService", targetService);
}
```

e. The destination property from the endpointURL is found, and this is used to do a JNDI lookup for the JMS Queue of the service provider. This JMS Queue is then used to generate a `SIDestinationAddress`, which can be set on the Forward Routing Path of a message (Example 10-5).

Example 10-5 Finding the destination and create a SIDestinationAddress

```
String destinationName = (String) properties.getProperty(
      "destination", null);

JmsQueue dest = getDestination(destinationName);
SIDestinationAddress address = null;
if (dest != null) {
   address = SIDestinationAddressFactory.getInstance()
         .createSIDestinationAddress(dest.getDestName(),
      dest.getBusName());
}
```

Note: To resolve the correct packages for the classes used in Example 10-5, use the **Source** → **Organize Imports** feature of Rational Application Developer. Where multiple packages are offered, choose:

- java.util.List

The destination of the service provider is then added to the Forward Routing Path of the message. We will also catch any exceptions here (Example 10-6).

Example 10-6 Updating the Forward Routing Path

```
   List frp = siMessage.getForwardRoutingPath();
   frp.add(address);
   siMessage.setForwardRoutingPath(frp);
} catch (Exception ex) {
   ex.printStackTrace();
}
```

Mediation handlers should return `true` if the message is to continue processing or `false` if the message should be delivered to the exception destination,

6. The Enterprise Application Project `BrokerJMSProcessor` can now be exported to an EAR file.

10.2.2 ESB based on WebSphere Business Integration Message Broker

This section describes the development of an ESB based on WebSphere Business Integration Message Broker and its associated service consumers and providers.

All messages exchanged between the two ESBs use a SOAP over JMS protocol. Messages sent between the WebSphere Business Integration Message Broker ESB and the legacy manufacturers use native WebSphere MQ messages (MQ message body data is not SOAP, and the application does not use the WebSphere MQ JMS API). Five message flows are provided in order to mediate messages from the ESB built in WebSphere Application Server and the legacy manufacturer service providers attached to the WebSphere Business Integration Message Broker ESB. The message flows are named:

- PurchaseOrderRequest
- PurchaseOrderResponse
- WarehouseCallbackRequest
- WarehouseCallbackResponse
- LogEvent

In order to mediate messages, WebSphere Business Integration Message Broker receives data in a physical wire format and parses the information into what is commonly referred to as the *logical tree*. Users of WebSphere Business Integration Message Broker may select this parser using a message domain. Available message domains include BLOB (Binary Large OBject), MRM (Message Repository Manager), XML, and XMLNS (XML NameSpace). Of these alternatives, the MRM domain is the only method that implements a parser to check on the wire messages against a pre-defined format. In such circumstances the message is said to be validated against the provided message dictionary. When users select the message domain to be used, the decision should take into account:

- Whether the message data itself will be manipulated by a mediation.
- The requirements of validation.
- The level of expected performance (when conducting any kind of tree-walk or validation, a certain degree of performance overhead should be expected).

For the purpose of this book, the scenarios were implemented to demonstrate message flows that implemented parsing using the MRM domain. Parts of the ESQL code provided in the following chapter are based on techniques used in the following SupportPac™:

- IA81: WebSphere Business Integration Message Broker and Web Services

Version 3 of this SupportPac only provides code that creates a logical tree compatible with the XMLNS domain, so the ESQL has changed significantly.

The main functional purpose of this ESB in the scenario is to add and remove SOAP Envelopes from the data passing between service consumers and the legacy manufacturer service providers. Figure 10-9 shows the main message flows and the sequence in which they are typically called in the scenario.

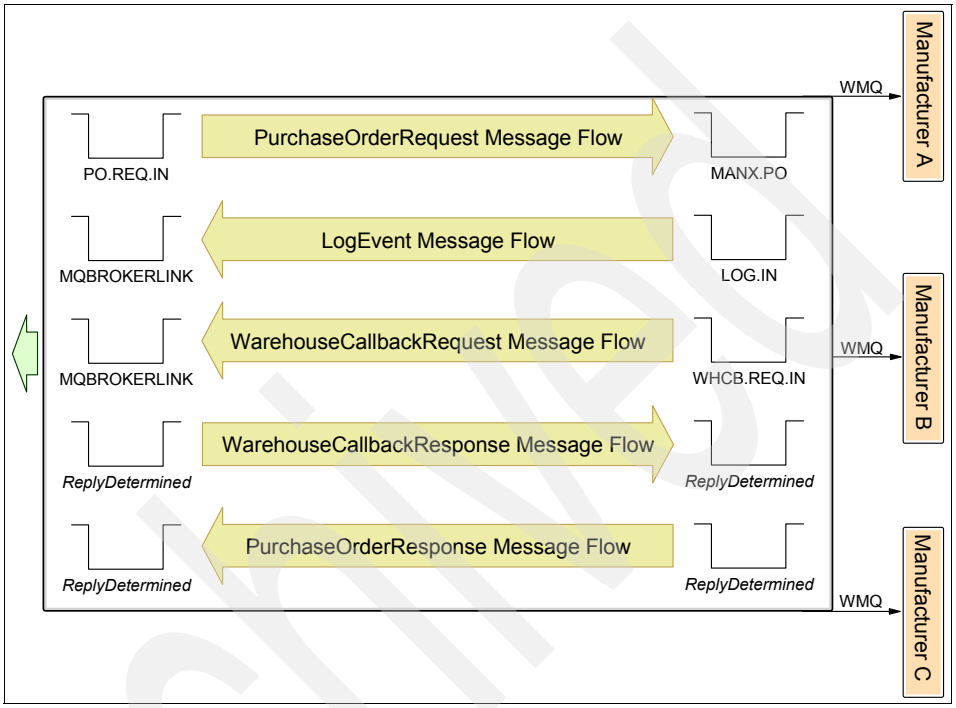

Figure 10-9 WebSphere Business Integration Message Broker Message Flows

When the Retailer application requests products from the Warehouse through the WebSphere Application Server ESB, such that the remaining stock in the Warehouse falls below a minimum threshold, the scenario requires the Warehouse to re-order stock from the Manufacturer. This is the first time in the business scenario when cross-bus messaging is required. A PurchaseOrder request message is sent using a SOAP over JMS message using the WebSphere MQ Link to the WebSphere Business Integration Message Broker ESB. The WebSphere MQ Link presents a facade to both the WebSphere Application Server service integration bus messaging engine and the WebSphere MQ queue manager. The service integration bus equivalent of an WebSphere MQ queue manager is a *messaging engine*.

Messaging engines can send and receive WebSphere MQ messages to and from queue managers using defined sender and receiver channels. In order to be bi-directional, a channel must have two parts, each of which has a definition at

each end of the channel. So to create a bi-directional WebSphere MQ Link between a queue manager and a messaging engine, a sender and receiver channel must be defined on the messaging engine and on the queue Manager. Each end of the channel is given the same name. In this scenario, the channel definitions were made as shown in Figure 10-10.

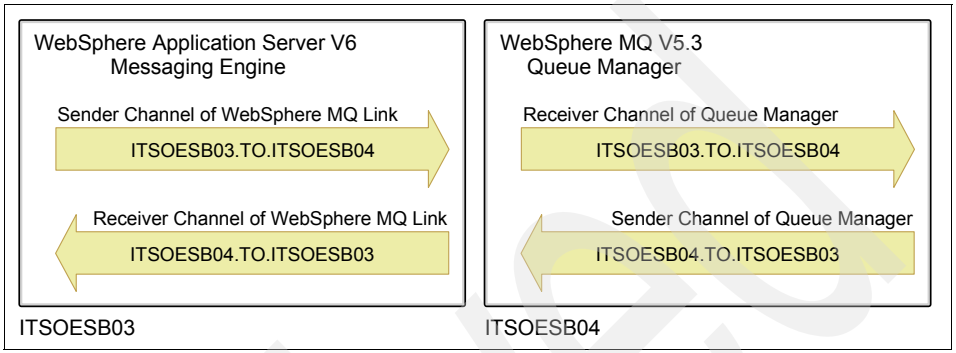

Figure 10-10 Naming convention employed for WebSphere MQ Link

Creation and import of Buildtime artifacts

In order to fully implement the WebSphere Business Integration Message Broker runtime ESB, development of the required broker artifacts must be carried out within the WebSphere Business Integration Message Broker Buildtime tooling component. The message broker requires two different kinds of artifact: Message Flows and Message Sets. Each of these kind of resources has an associated project type within the Message Broker Toolkit. The rest of this section provides instructions for the creation of all required Buildtime resources in order to run the scenario.

The required message flows are supplied in a pre-built message flow project as additional materials with this book. The instructions in “Import the message flow project” on page 259 describe how to import these message flows into the toolkit.

Full functional descriptions of these flows and the ESQL code that they contain are then provided in the sections that follow.

After this, full instructions are provided for the creation of the Message Set components. These instructions are provided in a step-by-step format. If you prefer, a complete version of the message set project is also provided in the additional materials accompanying this book.

All additional materials for this chapter can be downloaded as described in Appendix A, “Additional material” on page 317.

Import the message flow project

1. Download the samples provided with this book, and copy the mfp_Manufacturer directory.
2. Paste the mfp_Manufacturer folder into your message broker workspace. If you installed the WebSphere Business Integration Message Broker Toolkit in the default location, this will be C:\Program Files\IBM\WebSphere Business Integration Message Brokers\eclipse\workspace.
3. Repeat these two steps for the soaplib project.
4. Start the WebSphere Business Integration Message Broker Toolkit.
5. Import the message flow project into the eclipse environment:
   a. Click **File** → **Import** → **Message Set Project**.
   b. Select **Existing Project into Workspace** and click **Next.**
   c. Click **Browse** and locate the mfp_Manufacturer directory under your workspace directory, and click **Finish.**
6. Import the soaplib project in the same way.

All message flows used in this scenario are located within this message flow project. You may seem some warnings after these two projects have been imported. These can be ignored, and will be addressed when the message sets are built.

PurchaseOrderRequest message flow

The PurchaseOrderRequest Message Flow routes messages from the ESB built in WebSphere Application Server to the legacy manufacturer component within the overall architecture. Its primary purpose is to translate SOAP over JMS (provided by WebSphere MQ) messages into legacy XML over WebSphere MQ messages.

The logic in the PurchaseOrderRequest message flow could all have been contained in a single Compute node (for better performance), but to more clearly separate the logical units of code, three functional nodes have been used:

Compute node:	RemoveSOAP
Compute node:	Transform
Reset Content Descriptor node:	RCD

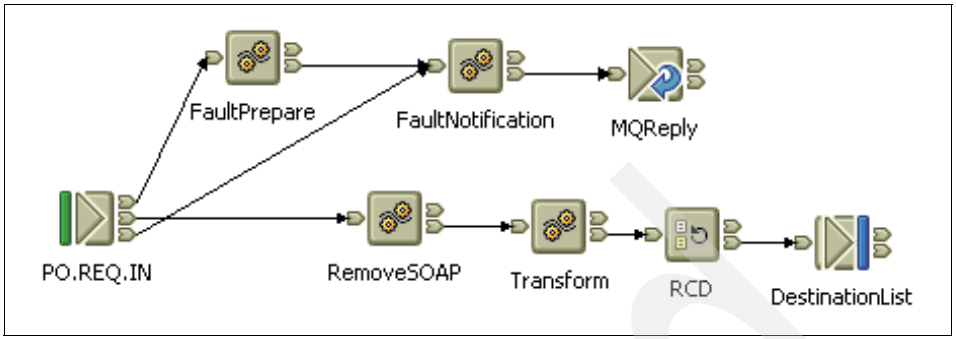

Figure 10-11 PurchaseOrderRequest message flow

Compute node: RemoveSOAP

The primary purpose of this node is to remove the SOAP Envelope from around the message. Before calling any of the library routines to carry out this function, the Reply information from the MQMD header is saved into the MQRFH2 header.

Example 10-7 RemoveSOAP: saving reply information

```
      CALL CopyMessageHeaders();
      -- Save Reply destination information so it can be reinstated by the
      -- PurchaseOrderResponse message flow once the Legacy Manufacturer
      -- has responded to the WBIMB ESB.
      SET OutputRoot.MQRFH2.usr.FinalReplyToQ = InputRoot.MQMD.ReplyToQ;
      SET OutputRoot.MQRFH2.usr.FinalReplyToQMgr = InputRoot.MQMD.ReplyToQMgr;
```

The MQMD Reply fields are given new values later in the flow in order to make the Legacy Manufacturer application direct its response back to the correct message flow (PurchaseOrderResponse). When the response message comes back to the Broker, this response flow reinstates this information as the destination that the WebSphere Business Integration Message Broker ESB must direct the message to on the message bus of the ESB built in WebSphere Application Server.

The next stage of the ESQL calls three library ESQL procedures (Example 10-8).

Example 10-8 RemoveSOAP: setting valid operations and headers

```
      -- Initialise Service Configuration Data
      CALL soaplib_service_init(Environment,'Intermediary','
www.itso.ral.ibm.com','PurchaseOrderRequest');
      -- Declare valid operations and headers in the Environment tree
      CALL soaplib_set_validop(Environment,'PurchaseOrder');
      CALL soaplib_set_validhdr(Environment,'Configuration');
      CALL soaplib_set_validhdr(Environment,'StartHeader');
```

The message flow project named soaplib contains the broker schema soapbrschema, in which the ESQL file soaplib.esql contains all of the library procedures. Note that ESQL modules are granted access to library procedures located in different files using PATH statements at the top of their files. The role of the soaplib_service_init procedure is to declare the SOAP ServiceName, SOAP ServiceRole, and SOAP ServiceActor in the Environment tree. If a SOAP fault is thrown by later code, these values will be used to construct the fault and therefore identify where in the service provider and consumer interaction the failure occurred.

The role of the soaplib_set_validop procedure is to declare the allowed operations for the SOAP message that has been received by the message flow. These valid operations are written into the Environment tree so that values from the message may be compared against them.

The role of the soaplib_set_validhdr procedure is to declare the allowed headers for the SOAP message which has been received by the message flow. These valid headers are written into the Environment tree so that values from the message may be compared against them.

Example 10-9 shows the code to decode the SOAP and save the headers.

Example 10-9 RemoveSOAP: Decoding the SOAP and saving the SOAP headers

```
      DECLARE Status CHAR;
      -- Remove the SOAP Envelope from the message
      CALL soaplib_decode(InputRoot,OutputRoot,Environment,Status);
      -- Save away SOAP Headers for use when constructing Legacy body later
      DECLARE soap11 NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
      DECLARE InRef REFERENCE TO InputRoot.MRM.soap11:Header;
      MOVE InRef FIRSTCHILD;
      CREATE FIELD Environment.SOAP.Headers;
      DECLARE Headers REFERENCE TO Environment.SOAP.Headers;
      WHILE LASTMOVE(InRef) DO
         CREATE LASTCHILD OF Headers FROM InRef;
         MOVE InRef NEXTSIBLING;
      END WHILE;
      -- Check for SOAP errors, and if necessary encode the fault from data in
      -- Environment and then throw the fault
      IF Status = 'Error' THEN
         CALL soaplib_encode_fault(OutputRoot,Environment,InputExceptionList);
         throw user exception VALUES ('soaplib_decode_request returned Error
status');
      END IF;
```

The role of the soaplib_decode procedure is to check the format of the incoming logical tree and then remove the SOAP Envelope. The ESQL has been coded

specifically with correlation names that refer to a tree constructed in the MRM domain. Branched code could be implemented to deal with both MRM and XMLNS input domain scenarios. In the interests of describing coding approaches not demonstrated in previous redbook publications, the code samples provided deal exclusively with the MRM domain. The soaplib_decode procedure ensures that the incoming logical tree contains a maximum of one SOAP Header, which must precede the SOAP Body element. It also checks that only a single SOAP Body exists. The SOAP V1.1 standard allows for the existence of additional children of the SOAP Envelope, but the Basic Profile does not. In line with the Basic Profile, the ESQL provided rejects the Logical Tree if more than one direct child of the Envelope exists.

If any of the error conditions discussed above are triggered by the library procedures, the character variable named `Status` returns an error value to the parent routine.

Compute node: Transform

The second compute node in the message flow, Transform, receives the logical tree (which no longer includes SOAP) and adjusts the outgoing logical tree to the layout expected by the Legacy Manufacturers. The first part of the ESQL for this is shown in Example 10-10.

Example 10-10 Transform: creating the output body from the ServiceData

```
-- The SOAP has been removed, now transform message into the format
-- expected by the Legacy Manufacturer
CREATE LASTCHILD OF OutputRoot DOMAIN 'MRM';
CREATE FIRSTCHILD OF OutputRoot.MRM NAMESPACE man NAME 'PurchaseOrder';
DECLARE Ref REFERENCE TO InputRoot.MRM.ServiceData;
MOVE Ref FIRSTCHILD;
WHILE LASTMOVE(Ref) DO
   CREATE LASTCHILD OF OutputRoot.MRM.man:PurchaseOrder FROM Ref;
   MOVE Ref NEXTSIBLING;
END WHILE;
-- The data originally included in SOAP Headers is expected in the main
-- message body by the Manufacturer
DECLARE HdrRef REFERENCE TO Environment.SOAP.Headers;
MOVE HdrRef FIRSTCHILD;
WHILE LASTMOVE(HdrRef) DO
   CREATE LASTCHILD OF OutputRoot.MRM FROM HdrRef;
   MOVE HdrRef NEXTSIBLING;
END WHILE;
```

The values of the fields in the message remain unchanged in the scenario but several elements, although given the same names, reside in different namespaces to the original message. The ESQL of Example 10-11 on page 263

alters the namespace property of several name-value pairs in the logical tree without re-creating the entire element.

Example 10-11 Transform: changing the namespace of elements in output message

```
      -- Many elements in the message expected by the Legacy Manufacturer
      -- reside in a different namespace.
      -- This ESQL changes the namespace of elements in the logical tree
      SET OutputRoot.MRM.man:PurchaseOrder.*:orderNum NAMESPACE = po;
      SET OutputRoot.MRM.man:PurchaseOrder.*:customerRef NAMESPACE = po;
      SET OutputRoot.MRM.man:PurchaseOrder.*:items NAMESPACE = po;
      SET OutputRoot.MRM.man:PurchaseOrder.po:items.*:Item NAMESPACE = po;
      DECLARE I INTEGER 1;
      WHILE I <=
CARDINALITY(OutputRoot.MRM.man:PurchaseOrder.po:items.po:Item[]) DO
         SET OutputRoot.MRM.man:PurchaseOrder.po:items.po:Item[I].*:ID
NAMESPACE = po;
         SET OutputRoot.MRM.man:PurchaseOrder.po:items.po:Item[I].*:qty
NAMESPACE = po;
         SET OutputRoot.MRM.man:PurchaseOrder.po:items.po:Item[I].*:price
NAMESPACE = po;
         SET I = I + 1;
      END WHILE;
      SET OutputRoot.MRM.man:PurchaseOrder.*:total NAMESPACE = po;
      SET OutputRoot.MRM.*:Configuration NAMESPACE = man;
      SET OutputRoot.MRM.*:StartHeader NAMESPACE = man;
      SET OutputRoot.MRM.*:StartHeader.*:conversationID NAMESPACE = man;
      SET OutputRoot.MRM.*:StartHeader.*:callbackLocation NAMESPACE = man;
```

The ESQL of Example 10-12 shows how to reset the MessageType field of the Properties folder of the logical tree. This automatically updates the equivalent property of the mcd folder in the MQRFH2 header. This property is examined by the broker when the logical tree is converted to a bitstream (commonly written to a WebSphere MQ queue by an MQOutput node). The code extract also shows how the broker consults its routing database over ODBC to discover the queue and queue manager to which outbound messages should be sent. These values are set dynamically using a destination list, which is later consulted by the message flow's MQOutput node.

Example 10-12 Transform: setting message name and consult Routing Directory

```
      -- Reset the message name now that SOAP Envelope has been removed
      SET OutputRoot.Properties.MessageType = 'Manufacturer';
      -- Examine the Routing Directory, to locate the destination for
      -- the which Manufacturer and set the destination list accordingly.
      -- The role of a naming directory for the WBIMB ESB is assumed by a DB2
data source named SVCDIR
      -- Values are taken from this database using ODBC calls from this ESQL
```

```
      SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueName =
RTRIM ( THE (SELECT ITEM A.QUEUE FROM Database.ROUTE AS A WHERE A.SERVICE =
InputRoot.MQRFH2.usr.targetService ));
      SET
OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueManagerName =
RTRIM ( THE (SELECT ITEM A.QUEUEMGR FROM Database.ROUTE AS A WHERE A.SERVICE =
InputRoot.MQRFH2.usr.targetService));
      -- Set up the reply information so that the Legacy Manufacturer service
      -- returns the acknowledgment message (ready for soapification) to the
      -- correct response message flow
      SET OutputRoot.MQMD.ReplyToQ = 'PO.RES.IN';
      SET OutputRoot.MQMD.ReplyToQMgr = 'QM1';
```

Reset Content Descriptor node: RCD

The final part of the message flow utilizes a ResetContentDescriptor node. This node is present in order to avoid having to make the Legacy Manufacturers alter the MQRFH2 headers of the messages they receive. The Manufacturer simply copies the MQRFH2 header portion of the messages they receive into the output messages, which are sent back to the ESB. This means that the mcd folder, which is used by the broker to decide on which message domain (and therefore parser and writer) should be used when interpreting the message on the wire, is simply copied as well. If the MRM domain message were written directly to the wire, then when received by the MQInput node of the PurchaseOrderResponse message flow, the values held in the mcd folder would not match with the (now altered) message body data output by the Manufacturer. This would cause the messages to fail to be parsed.

The ResetContentDescriptor node in the PurchaseOrderRequest message flow converts the output message to the BLOB domain before writing it to the wire. This does not change the rendering of the message in any way, but means that when the PurchaseOrderResponse message flow receives the reply message, it can be parsed using the BLOB domain before an equivalent ResetContentDescriptor node switches the message domain back to the MRM domain. This changing of message domain would be avoided in most real-world scenarios because the back-end application would actively change values in the MQRFH2 header. When messages are received by an MQInput node, the mcd folder of the MQRFH2 header takes precedence in determining which parser to invoke, over any hard-coded properties on the Default properties tab of the node.

PurchaseOrderResponse Message Flow

The PurchaseOrderResponse Message flow's primary purpose is to route responses from the Manufacturer back to the ESB built in WebSphere Application Server in the correct format (essentially converting non-SOAP, XML WebSphere MQ messages into SOAP over JMS WebSphere MQ messages). The message flow contains similar error-handling nodes, named FaultPrepare

and FaultNotification, as discussed in detail in “PurchaseOrderRequest message flow” on page 259. The main functional nodes are:

Rest Content Descriptor node: RCD
Compute node: AddSOAP
Reset Content Descriptor node: RCD1

The individual function of each node is discussed following the diagram of the PurchaseOrderResponse message flow depicted in Figure 10-12.

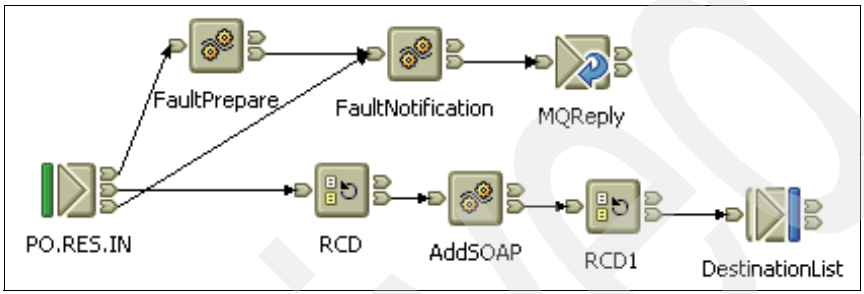

Figure 10-12 PurchaseOrderResponse message flow

Reset Content Descriptor node: RCD

When input messages arrive on the PurchaseOrderResponse message flow’s input queue, they are parsed by the BLOB domain. Effectively, this choice of message domain ignores the message body’s content and treats the entire structure as a single object. The RCD node that follows instantiates the MRM domain using the XML wire format parser. This creates a logical tree that the AddSOAP compute node can more easily manipulate. The response message generated by the legacy Manufacturer application contains an MQRFH2 header, which is identical to the original inbound message to the Manufacturer. This header describes the message body as BLOB, which overrides any setting made on the MQInput node. With a more sophisticated application, it would be possible to alter this header and use settings on the Default properties sheet of the MQInput node in order to dictate the method of parsing to WebSphere Business Integration Message Broker.

Compute node: AddSOAP

The primary purpose of this node is to add the SOAP Envelope around the message body, ready to be dispatched to the ESB built in WebSphere Application Server. Example 10-13 shows the first part of the ESQL code.

Example 10-13 AddSOAP: Setting the Destination for the message

```
-- Declare the data types of the variables which will be used
DECLARE endpointURL,destination,jndiDirectoryLocation CHAR;
DECLARE LocaterString,QueueName,QueueManagerName CHAR;
```

```
      DECLARE SeparatorPoint,destinationstartpos,destinationendpos INT;
      -- Set the destination on the WAS ESB using the reply information which
      -- was saved in the input message in the usr folder of the MQRFH2 header
      SET LocaterString = InputRoot.MQRFH2.usr.FinalReplyToQMgr || ':' ||
InputRoot.MQRFH2.usr.FinalReplyToQ;
      -- Set the WAS required SOAP over JMS Headers
      CALL
JMSPropsSetter(endpointURL,LocaterString,0,OutputRoot,QueueName,QueueManagerNam
e);
      -- Set Destination List which will be referred to by the MQOutput node
      SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueName =
QueueName;
      SET
OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueManagerName =
QueueManagerName;
```

The JMSPropsSetter procedure that is called demonstrates how to set the JMS Header properties (which are placed in the MQRFH2 header of the outbound WebSphere MQ message) such that the WebSphere Application Server SOAP over JMS client code can interpret the reply successfully. This routine takes the variables for endpointURL and LocaterString as input values. Typically these values could be taken by a remote look-up in an ESB's JNDI naming directory. In the example code supplied here, the LocaterString variable is constructed from within the ESQL code. Example 10-14 shows the code to add the SOAP envelope.

Example 10-14 AddSOAP: Adding the SOAP Envelope, and restructuring output

```
      DECLARE Status CHAR;
      -- Clear the service parameters from any previous requests
      CALL soaplib_reset_service_params(Environment);
      -- Set the operation for RPC style service, which will be used by the
      -- soaplib_encode_response
      CALL soaplib_set_out_op(Environment,'Response','PurchaseOrderResponse');
      -- soaplin_encode_response takes references to InputRoot and constructs
      an OutputRoot with a SOAP envelope.
      CALL soaplib_encode(InputRoot,OutputRoot,Environment,Status);
      -- Set the contents of the message using the input message
      -- This slightly different technique is required by the use of a
      -- message on input which has composition "Empty"
      -- Please see the Redbook text for further details regarding
      SET OutputRoot.MRM.*:Body.*:ackPO1 VALUE = FIELDVALUE(InputRoot.MRM);
      -- Reset the output message name, now that a SOAP Envelope
      -- has been constructed in the logical tree
      SET OutputRoot.Properties.MessageType = 'Envelope';
      SET OutputRoot.MQMD.CorrelId = OutputRoot.MQMD.MsgId;
      SET OutputRoot.MQRFH2.mcd.Msd = 'jms_bytes';
```

WarehouseCallbackRequest Message Flow

The WarehouseCallbackRequest Message Flow mediates messages from the Legacy Manufacturer to the ESB built in WebSphere Application Server. It contains two main functional components: compute nodes named Transform and AddSOAP. This function could have been placed within a single node, but was separated to provide a logical distinction in the function of each node.
Figure 10-13 shows the message flow.

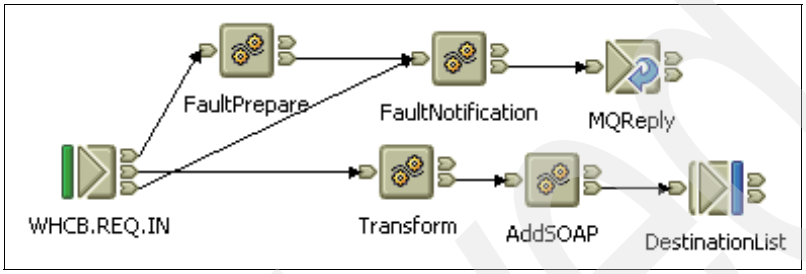

Figure 10-13 WarehouseCallbackRequest message flow

Compute node: Transform

The information that arrives inside the sections of the message named Configuration and CallbackHeader must be moved to a SOAP Header when the ShipmentNotice message is constructed for dispatch to the ESB built in WebSphere Application Server. In preparation for this, the data is moved to the Environment, as shown in Example 10-15.

Example 10-15 Transform ESQL

```
      CALL CopyMessageHeaders();
      -- Before adding the SOAP Envelope, the message needs transforming into
      -- the format expected by the WAS ESB. Information contained in the body
      -- of the message from the Manufacturer must eventually be placed in a
      -- SOAP Header.
      -- These statements move the information into the Environment, ready to
      -- be accessed by generic SOAP Header creation routines in next node
      SET Environment.SOAP.Out.Hdrs.*[1] = InputRoot.MRM.*:CallbackHeader;
      SET Environment.SOAP.Out.Hdrs.*[1] NAMESPACE =
'http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufactu
rer/CallBack';
      SET Environment.SOAP.Out.Hdrs.*[1].conversationID NAMESPACE =
'http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufactu
rer/CallBack';
      SET Environment.SOAP.Out.Hdrs.*[2] = InputRoot.MRM.*:Configuration;
      SET Environment.SOAP.Out.Hdrs.*[2] NAMESPACE =
'http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Configura
tion.xsd';
```

```
        -- The overall wrapper for the main SOAP Body takes a different name
        -- to the message outputted by Legacy Manufacturer
        SET OutputRoot.MRM = InputRoot.MRM.*:ShipmentNotice;
        -- Reset the name of the message so that the routines in the next node
        -- recognise it
        SET OutputRoot.Properties.MessageType =
FIELDNAMESPACE(InputRoot.MRM.*:ShipmentNotice) || ':' || 'ShipmentNotice';
```

Note that the namespace of the elements in the output message are different to the incoming message, hence the lines of code in the latter half of the procedure.

Compute node: AddSOAP

The AddSOAP compute node and ESQL are directly equivalent to the node of the same name contained in the PurchaseOrderResponse message flow. Refer to “PurchaseOrderResponse Message Flow” on page 264 for further details.

WarehouseCallbackResponse Message Flow

The WarehouseCallbackResponse Message Flow routes messages that arrive from the WarehouseCallback service hosted on the ESB build in WebSphere Application Server to the legacy Manufacturers. For normal execution, the flow’s critical path contains a single functional node, named AddSOAP. Figure 10-14 shows the message flow.

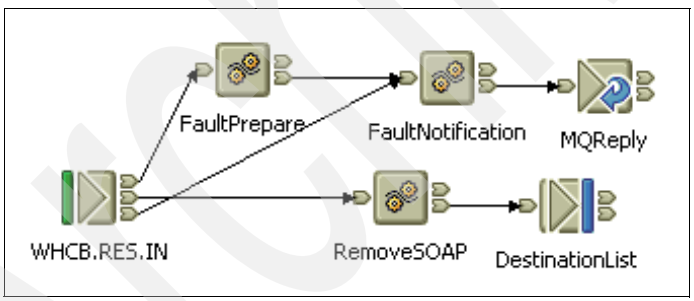

Figure 10-14 WarehouseCallbackResponse message flow

LogEvent Message Flow

The LogEvent Message Flow routes messages from the WebSphere Business Integration Message Broker ESB to the ESB built in WebSphere Application Server. No response message is expected from the LoggingFacility. For this reason, the LogEvent Message Flow is the only message flow that does not have a partner response flow. For normal execution, the flow’s critical path contains a single functional node, named AddSOAP. Figure 10-15 shows the message flow.

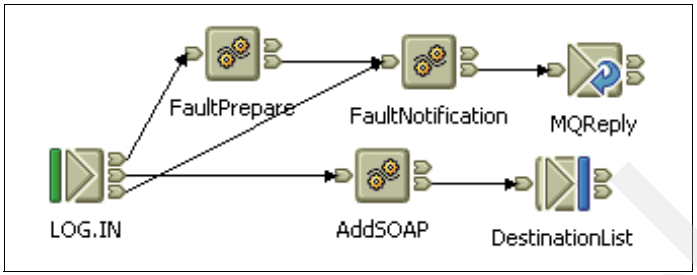

Figure 10-15 LogEvent message flow

Compute node: AddSOAP

The logic in this message flow is the simplest of all the required message flows. The only transformation that is required on messages received from the Legacy Manufacturer is the addition of a SOAP Envelope. “PurchaseOrderResponse Message Flow” on page 264 has further details about the node and ESQL, extracts of which appear in Example 10-13 on page 265 and Example 10-14 on page 266.

Create the Manufacturer message set

The message flows that carry out the routing and transformation of messages flowing through the ESB rely on metadata to parse data over the wire into a logical tree structure. A message set provides a holder for a collection of message definition files. Message definition files are valid XML schema that describe the structural content of XML messages. In addition to standard schema vocabulary, they also contain annotations (comments) that provide wire formatting information specific to WebSphere Business Integration Message Broker. This additional data means that message definition files can also describe a wide array of fixed-length and tagged delimited wire formats. Message definition files can make use of definitions contained in other message definitions files (using the schema import and include functions). This means that a single message set can contain definitions for multiple messages. XML schema does not have a concept of a message (the top-level field in a hierarchy describing an XML document), but uses global elements for this purpose. WebSphere Business Integration Message Broker places restrictions on message artefacts beyond those implied by the status of being a global element in an XML schema. One such restriction is that the name of the message must be unique (regardless of namespace) within the message set.

The legacy manufacturer and other service requesters and providers exchange messages that have the same name but different structures. The uniqueness of messages within a message set requirement necessitated the creation of two message sets.

Creating the message sets and importing the schema definitions

The following instructions create the message sets and import the schema definitions:

1. Start the WebSphere Business Integration Message Broker Toolkit.
2. Ensure that you are in the Broker Application Development perspective. Select **Window** → **Open Perspective** → **Broker Application Development** from the menu.
3. Create a new Message Set Project.
   a. Click **File** → **New** → **Message Set Project**.
   b. Enter `msp_Manufacturer` in Project name and click **Next.**
   c. Enter `ms_Manufacturer` in Message Set Name, check **Use namespaces**, and click **Next.**
   d. Check **XML Wire Format Name**, set the wire format name to **XML** (from XML1), and click **Finish.**

   This creates a message set within a message set project. It also creates a file called **messageset.mset**, which opens in the main editor of the toolkit.
4. In the main editor, customize the wire format (Figure 10-16 on page 271):
   a. In the Properties Hierarchy section, select **Physical Properties** → **XML**.
   b. Check **Suppress DOCTYPE** under XML document type settings. DTD declarations are not needed, as the system is schema-based rather than DTD-based.
   c. Set **Root Tag Name** to blank by deleting the default value MRM. Web services use SOAP messages, where the root tag name must be Envelope.
   d. Close the messageset.mset window and save the contents.

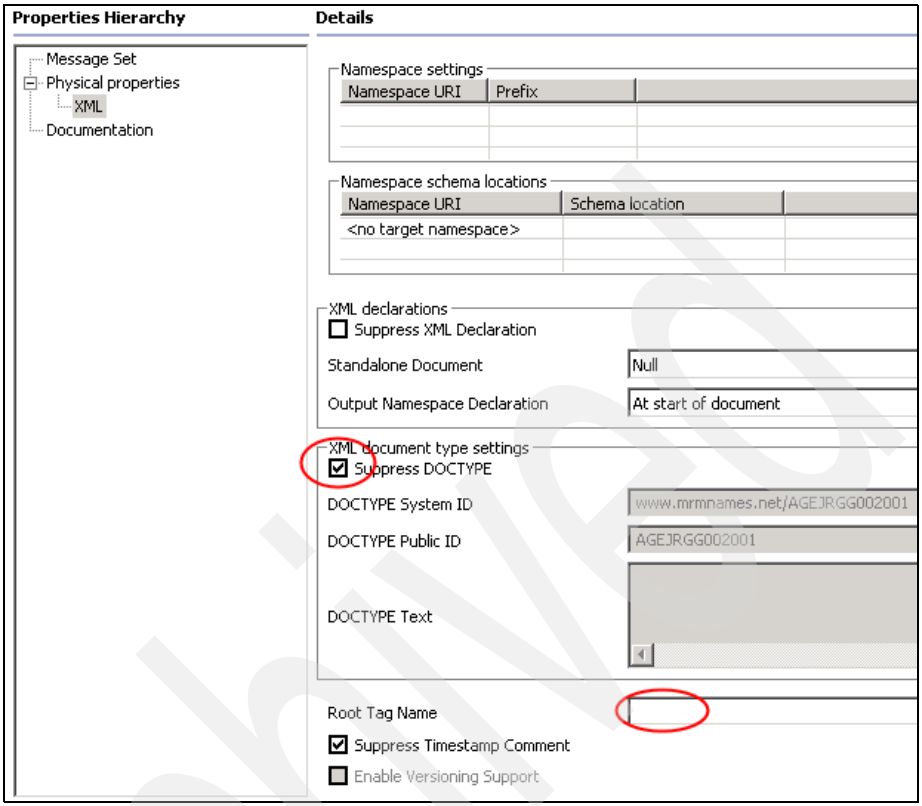

Figure 10-16 messageSet.mset settings for ms_Manufacturer

The SOAP V1.1 XML schema must be imported into the message set so that it can be used to model SOAP messages. This schema can be found at:

`http://www.w3.org/TR/2000/NOTE-SOAP-20000508/envelope-2000-04-18.xml`

You also must import the XML schemas that define the SOAP Header and SOAP Bodies of the messages exchanged by the WebSphere Business Integration Message Broker ESB. These are the files:

- soap11.xsd
- ManufacturerCallbackMessage.xsd
- Manufacturer.xsd
- ManufacturerPO_Legacy.xsd
- ManufacturerSN.xsd
- WarehouseCallbackMessage.xsd
- Configuration.xsd

Import these files into message set project msp_Manufacturer (see the additional material supplied with this redbook for these files):

1. Select the **msp_Manufacturer** project in the Resource Navigator.
2. Select **File** → **Import** from the menu.
3. Select **File system** and click **Next**.
4. Click **Browse** to the right of the Directory field and navigate to the directory to which you downloaded the additional material supplied with this book. Click **OK**.
5. Check **soap11.xsd, Manufacturer.xsd, ManufacturerCallbackMessage.xsd, ManufacturerPO_Legacy.xsd, ManufacturerSN.xsd, WarehouseCallbackMessage.xsd** and **Configuration.xsd**. Click **Finish**.

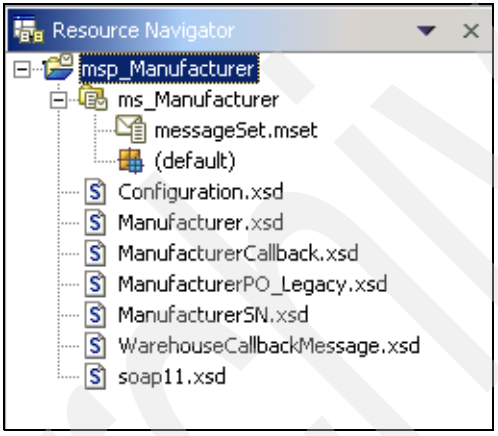

Figure 10-17 Resource Navigator pane after import of XML schema files

Creating Message Definition Files

Next, create a Message Definition File based on the soap11.xsd file you just imported:

1. Select **File** → **New** → **Message Definition File**. This starts the new message definition file wizard.
2. Click **XML schema file** and click **Next**.
3. Select file **soap11.xsd** from project msp_Manufacturer and click **Next.**
4. Select **ms_Manufacturer** in project msp_Manufacturer and click **Next.**
5. Click **Envelope** and **Fault,** and click **Finish.**

During this import, the global element Envelope is defined as a message. This means that message flows can nominate this message as the one to be

processed. The global element Fault is also defined as a message. This is because Fault is a child of the element Body. In the following customization, the complex type of Body will be amended to be of composition message. This enables children of Body to be defined as messages.

Several warnings are generated in the task list as a result of importing this schema. These can be eliminated by measures described later in this chapter.

Before any further customization of the SOAP message definition file occurs, you must create new message definitions based on the other XML schemas you have imported. Repeat the instructions above, creating a message definition file from each of the XML schema files that you imported into the msp_Manufacturer project:

- Configuration.xsd

  When asked about creation of messages from global elements, do not select any.

- ManufacturerPOLegacy.xsd

  When asked about creation of messages from global elements, select **PurchaseOrderLegacy**.

- ManufacturerCallback.xsd

  When asked about creation of messages from global elements, do not select any.

- Manufacturer.xsd

  When asked about creation of messages from global elements, select **Manufacturer**.

- ManufacturerSN.xsd

  When asked about creation of messages from global elements, select **ShipmentNotice** and **submitSNFault**.

- WarehouseCallbackMessage.xsd

  When asked about creation of messages from global elements, select **WarehouseCallbackMessage**.

After all of the message definition files have been created, the Resource Navigator should look similar to Figure 10-18 on page 274.

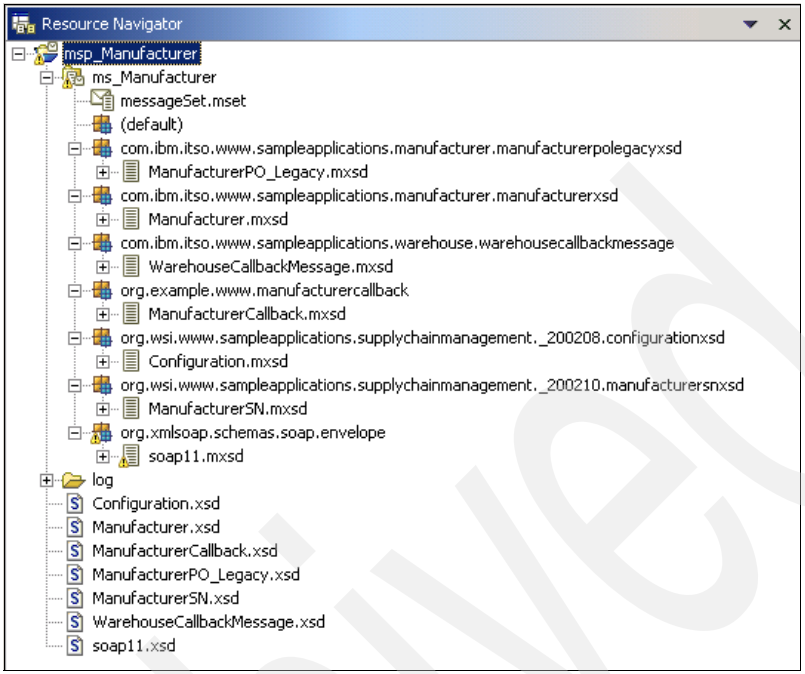

Figure 10-18 Resource Navigator Pane after message definition file creation

Removing warnings

At this stage, several additional task list warnings will be in your toolkit (unless you have suppressed these warnings using the Filter function). To get rid of these warnings:

1. Remove Wildcard Elements from soap11.mxsd. Doing this removes the Wildcard Element warnings generated by the Message Broker Toolkit.
   a. In the Resource Navigator, expand **msp_Manuacturer** → **org.xmlsoap.schemas.soap.envelope** and double-click **soap11.mxsd**.
   b. In the Outline view, expand **Elements and Attribute**.
   c. Expand **Envelope**, select **Wildcard Element**, and press **Delete**.
   d. Also delete the Wildcard Element for the elements **Header**, **Body**, and Fault **detail** (detail is a child of the Fault element).
2. Set the Content Validation of the Envelope complex type to **Open**. This means that any elements and attributes are allowed as children of Envelope. Envelope still has explicitly defined children named **Header** and **Body**. This

closely matches the business requirement for the SOAP Envelope while still satisfying the wildcard element removal.

a. Select **Envelope** under Elements and Attributes in the Outline view.

b. Click the **Properties** tab in the main editing window.

c. Click the **Goto Type Definition** link at the top.

d. Change the Content Validation from Closed to **Open** using the pull-down menu.

3. Similarly set the Content Validation of the **Header** complex type and the **detail** complex type (note that a detail *element* is also a child of the Fault element) to **Open**. This means that any elements and attributes are allowed as children of Header and detail. This closely matches the business requirement for the SOAP Header and Fault detail while still satisfying the wildcard element removal.

4. For each of the elements **Envelope**, **Header**, **Body,** and Fault **detail** under Elements and Attributes in the Outline view, remove the namespaces from Wildcard Attributes. This removes warnings generated by the Message Broker Toolkit. To do this, for each element:

a. Select **Wildcard Attribute** in the Outline view.

b. Remove the entry in the Namespace field (##other) in the main editing view (Figure 10-19).

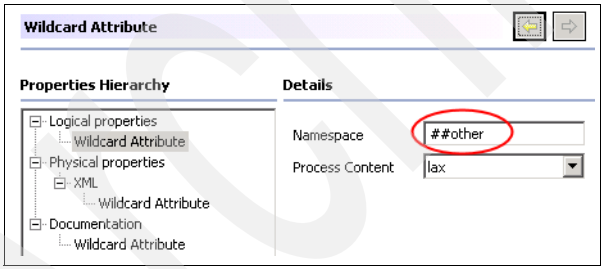

Figure 10-19 Wildcard Attribute properties before delete of Namespace entry

5. Remove the pattern facet of the **mustUnderstand** global attribute as follows. It is not required and its deletion removes a warning generated by the Message Broker Toolkit.

a. Select **mustUnderstand** in Outline view.

b. Click the **Goto Type Definition** link in the main editing window and select **Value Constraints** in Property Hierarchies.

c. Select **0/1** in Patterns and click **Delete** (Figure 10-20).

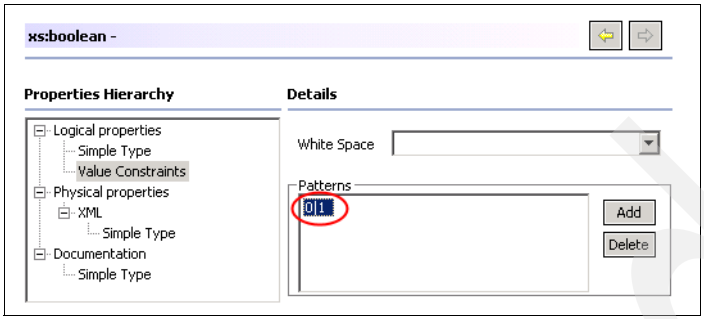

Figure 10-20 Value Constraint removal

6. Save your changes to soap11.mxsd by pressing Ctrl+S.

Creating a new message

This newly created message set (msp_Manufacturer) provides all of the metadata required to describe the messages that are exchanged between the WebSphere Business Integration Message Broker and the Legacy Manufacturer application. Note that the Legacy Manufacturer application that is supplied with this redbook's additional materials communicates using XML messages that do not have a SOAP envelope. This is designed to demonstrate the use of WebSphere Business Integration Message Broker as providing a SOAP facade for legacy applications; performing the addition and removal of SOAP envelopes.

You may have noticed that despite the Legacy Manufacturer not using SOAP in its messages, the soap11 schema file was imported into the ms_Manufacturer message set. The reason is so the schema file named Configuration.xsd (whose definitions are reused in both the Manufacturer.xsd and WarehouseCallback.xsd schemas) has a dependency on the soap11.xsd schema. Because the Manufacturer application does not use SOAP messages, there is no need to add any further messages as children of the Body element. In the next message set you create, you need to add five possible children to the Body element that relate to the five possible embedded messages that will be used by the message flows when communicating with the ESB built in WebSphere Application Server.

The schema file named ManufacturerPO.xsd contained a definition for a global element to describe the acknowledgement message sent back to the Warehouse from the Manufacturer after the order has been processed. This global element is of Boolean type, so on the wire it can take the value true or false. WebSphere Business Integration Message Broker can only create messages from global elements that are based on complex types. This means that when importing the schema as a message definition file, there was no option presented to you to create a message from this particular global element. You must create a

message to define this acknowledgement so that it can be used by the broker to parse the acknowledgement received from the Manufacturer application.

We model this message using a complex type that has a base type of Boolean:

1. Open the message definition file **ManufacturerPO_Legacy.mxsd** (to see it, expand **msp_Manufacturer** → **com.ibm.itso.www.sampleapplications.manufacturer.manufacturerpolegacyxsd**).
2. In the Outline view, expand the **Elements and Attributes** folder to locate the global element named ackPO. Select **ackPO** and press the Delete key.
3. In the Outline view, right-click **Types** → **Add Complex Type**.
4. Rename the type you have just created from **complexType1** to **ackPOType**.
5. In the main editor view, click the **Properties** tab.
6. Click the Base Type pull-down menu and select **xs:boolean**.
7. Save your changes by pressing Ctrl+S.
8. In the Outline view, right-click **Elements and Attributes** → **Add Global Element**.
9. In the main editor view, click the **Properties** tab.
10. Enter `ackPO` in Name, replacing the default globalElement1.
11. Click the Type pull-down menu and select **(More...)**.
12. In the Type Selection pop-up window, select **ackPOType** and click **OK** (Figure 10-21 on page 278).

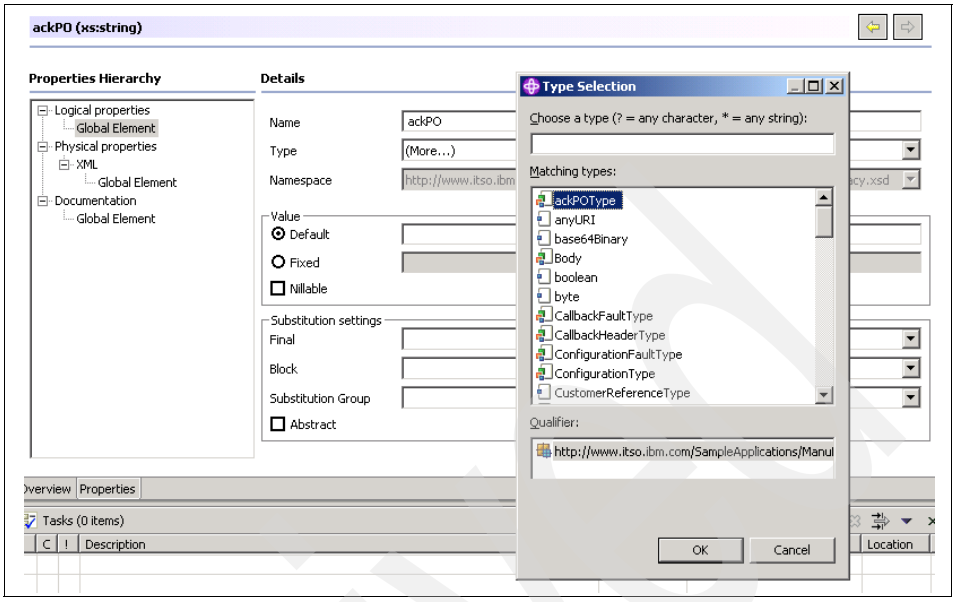

Figure 10-21 Type Selection pop-up window for new global element ackPO

13. In the Outline view, right-click **Messages** → **Add Message From Global Element**.
14. Select **ackPO**.
15. Save your changes by pressing Ctrl+S.

The message you just created, named ackPO, resides in a different namespace (http://www.itso.ibm.com/SampleApplications/Manufacturer/ManufacturerPO_Legacy.xsd) from the message that is also called ackPO in namespace http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/ManufacturerPO.xsd, which will be created in the second message set for the scenario. The existence of two separate messages of the same name (even though they have different namespaces) requires us to use two message sets for the scenario.

Create the other Message Set

Now create the second message set for the scenario. This message set holds definitions that will be used to parse messages coming over the WebSphere MQ Link to the WebSphere Business Integration Message Broker ESB from the ESB built in WebSphere Application Server. It will also be used for writing messages that will be sent in the opposite direction. The procedure to follow is basically the

same as has previously been described, so the following instructions have been kept intentionally brief:

1. Ensure that you are in the Broker Application Development perspective. Select **Window** → **Open Perspective** → **Broker Application Development** from the menu.
2. Create a new Message Set Project named `msp_Other` to contain a message set named `ms_Other`. The message set should be namespace-enabled and have an XML wire format layer named `XML`. As before, you should check **Suppress DOCTYPE** and remove the Root Tag Name.
3. Import the following XML schema files into the `ms_Other` message set project: `soap11.xsd`, `LoggingFacility.xsd`, `ManufacturerPO.xsd`, `ManufacturerSN.xsd`, `Callback.xsd`, and `Configuration.xsd`. Create a message definition file from each XML schema:
   - **LoggingFacility.xsd** (When asked about creation of messages from global elements, select all.)
   - **ManufacturerPO.xsd** (When asked about creation of messages from global elements, select all.)
   - **ManufacturerSN.xsd** (When asked about creation of messages from global elements, select all.)
   - **Configuration.xsd** (When asked about creation of messages from global elements, select all.)
   - **Callback.xsd** (When asked about creation of messages from global elements, select all.)
   - **soap11.xsd** (When asked about creation of messages from global elements, select **Envelope** and **Fault**.)

After creating the message definition files, remove the warnings that have been generated in the Task List. Follow the instructions from the previous message set, for **soap11.xsd**:

1. From LoggingFacility.mxsd, delete the Wildcard Element that is a child of the complex type named logEventRequestType. Select the Wildcard Element and press the Delete key. In place of the Wildcard element, amend the complex type Content validation to be **Open**. Doing this removes the Wildcard Element warnings generated by the Message Broker Toolkit, but maintains the allowed messages, should you choose to take advantage of validation.
2. Expand the **getEventsResponseType** and find the ANONYMOUS complex type that is defined as a child of the element LogEntry. Delete the Wildcard Element that is a child of the ANONYMOUS complex type. Amend the complex type's Content validation to be **Open**.

3. Save the changes to LoggingFacility.mxsd. This should remove the remaining warnings for LoggingFacility.mxsd.

The schema file named ManufacturerPO.xsd contains a definition for a global element to describe the acknowledgement message sent back to the Warehouse from the Manufacturer after the order has been processed. WebSphere Business Integration Message Broker can only create messages from global elements that are based on complex types. As in the previous message set, we must create a definition for the acknowledgement.

We model this message using a complex type that has a base type of Boolean:

1. Open the ManufacturerPO.mxsd message definition file.
2. In the Outline view, expand the **Elements and Attributes** folder to locate the global element named ackPO. Select **ackPO** and press the Delete key.
3. As before, create a complexType named **`ackPOType`** with a base type of `xs:boolean`.
4. Using this complexType, add a global element named `ackPO`.
5. Add a message from this global element, which will also be named `ackPO`.
6. Save your changes by pressing Ctrl+S.

Unlike the message created in the ms_Manufacturer message set, the message that you have just created in the ms_Other message set has a namespace of http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/ManufacturerPO.xsd.

In a similar manner to the instructions above, open the message definition file named **ManufacturerSN.mxsd** and delete the global element named **ackSN** (Figure 10-22 on page 280).

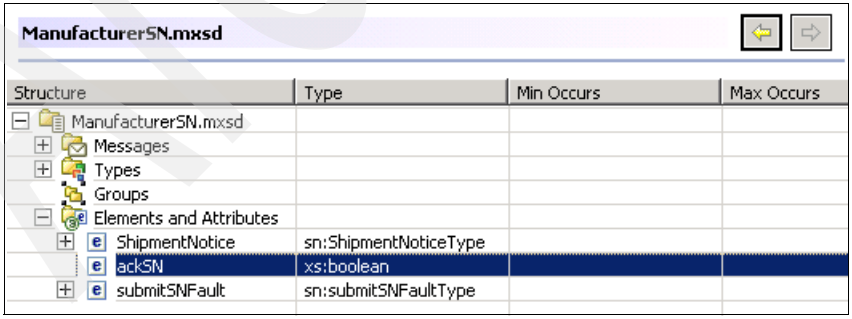

Figure 10-22 ManufacturerSN.mxsd with ackSN highlighted before deletion

1. Create a complextype named `ackSNType`, with a base type of `xs:boolean`.

2. Use this complex type to add a new global element named `ackSN`.
3. Add a message from this global element, which will also be named ackSN**.**
4. Save your changes by pressing Ctrl+S.

Having created a set of stand-alone schema, you must now customize the SOAP message definition file. Import the message definition files representing the Web services into the SOAP message definition file. This is the initial step in building a message hierarchy. At the top of this hierarchy is SOAP, and the Web services are subordinate.

1. Open the SOAP message definition file soap11.mxsd.
2. In the Outline view, click **soap11.mxsd**.
3. In the main editor pane, click the **Properties** tab.
4. In the main editor, right-click **Imports** → **Add import** in Properties Hierarchy. A wizard to select a message definition file appears.
5. Select **LoggingFacility.mxsd** from the hierarchy (**msp_Other** → **ms_Other** → **org** → **wsi** → **www** → **sampleapplications** → **supplychainmanagement** → **_200208** → **loggingfacilityxsd**) and click **Finish** (Figure 10-23 on page 281).

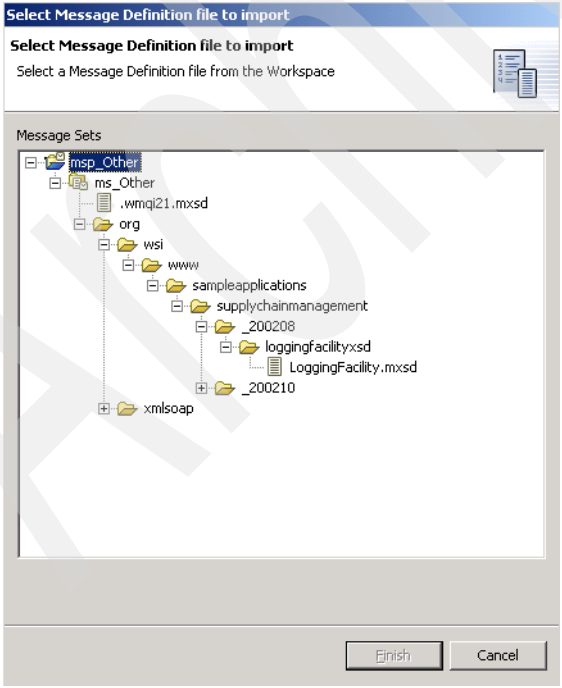

Figure 10-23 Select Message Definition file to import

6. Repeat this process for importing **ManufacturerSN.mxsd** in the soap11.mxsd message definition file.
7. Repeat this process for importing **ManufacturerPO.mxsd** in the soap11.mxsd message definition file.
8. Repeat this process for importing **Configuration.mxsd** in the soap11.mxsd message definition file.
9. Repeat this process for importing **Callback.mxsd** in the soap11.mxsd message definition file.
10. Save soap11.mxsd by pressing Ctrl+S.
11. The Broker will process the content of the SOAP Envelope as a message in its own right, so you must change the Body complex type to a Content composition of Message. This means that one and only one of Body's child elements must be present on the wire, and that each possible child must be defined as a message:
    a. In the Outline view, expand **Elements and Attributes** and select **Body**.
    b. In the main editor view, click the **Properties** tab.
    c. In the main editor view, click the **Goto Type Definition** link at the top. You are presented with details of Body's complex type.
    d. Click the Composition pull-down menu and select **message** (changing it from sequence).
    e. Note that the Content validation setting is Closed. This will affect later customization.
    f. Delete the Wildcard Attribute and Wildcard Element children of the Body complex type.
    g. Save soap11.mxsd by pressing Ctrl+S.

In our scenario, the SOAP Body may contain (at various stages in the message sequences) one of the following messages: `PurchaseOrder`, `ackPO`, `ShipmentNotice`, `ackSN`, `logEventRequestElement`, or a SOAP Fault message. These must be added as children to the SOAP Body element because:

- The Content validation setting of Body's type is set to Closed. This means that only children explicitly defined to Body are allowed.
- Having explicitly defined children enables the message bodies to be used in the ESQL editor's Content Assist when developing message flows.

To add the elements as children to the SOAP Body element:

1. If it is still open, close the soap11.mxsd message definition file.

2. Re-open **soap11.mxsd** (making sure you open it from within the correct message set, **ms_Other**) to ensure that all imported schema are recognized.
3. In the **Outline** view, expand **Elements and Attributes** and select **Body**.
4. In the main editor view, click the **Overview** tab.
5. In the main editor, right-click **Body** and select **Add Element Reference**.
6. From the pull-down menu, select **log:logEventRequestElement** (Figure 10-24 on page 283).

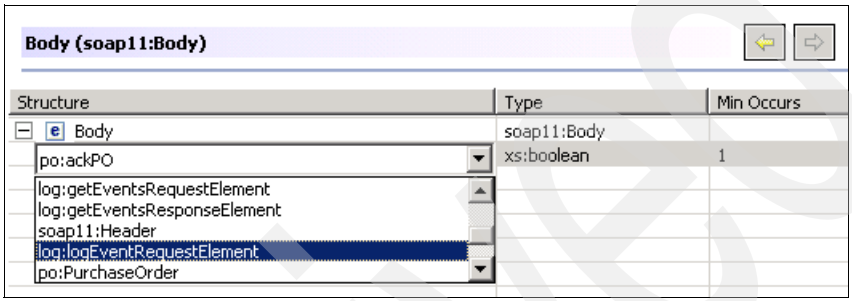

Figure 10-24 Adding Element References to the SOAP Body

7. Repeat the two previous steps, selecting **po:PurchaseOrder, po:ackPO, sn:ShipmentNotice,** and **soap11:Fault**

Also in this system, the SOAP Header may contain a Configuration element and a StartHeader element. These must be added as children of the SOAP Header element. To do this:

1. In the Outline view, expand **Elements and Attributes** and select **Header**.
2. In the main editor view, click the **Overview** tab.
3. In the main editor, right-click **Header** and select **Add Element Reference**.
4. From the pull-down menu, select **tns:Configuration**.
5. Repeat the last two steps, this time selecting **cb:StartHeader**.
6. Save soap11.mxsd.

To remove the remaining errors from soap11.mxsd, perform the following:

1. Ensure soap11.mxsd is still open. In the Outline view expand **Messages** → **Envelope** and click **Wildcard Attribute**. In the Properties view in the main menu, clear the entry in the Namespace field (it will be set to ##other).
2. Repeat this for the Wildcard Attributes under the **Header** and **detail** types.
3. Delete the **Wildcard Element** for the **Envelope** message and the **Header** and **detail** types.

4. Under Elements and Attributes, expand **mustUnderstand** and click **xs:boolean**. In the Properties view, expand **Logical properties** and click **Value constraints**. In the Patterns box, select **0|1** and click **Delete**.
5. Save your changes using Ctrl+S.

Add the message set projects to the message flow

References must be created between the message set projects and the message flow. Complete the following steps:

1. In the Resource Navigator, right-click **mfp_Manufacturer** and select **Properties**.
2. Select **Project References**.
3. Check **msp_Manufacturer** and **msp_Other**.
4. Click **OK**.

10.2.3 Legacy manufacturer application

The manufacturer application is a stand-alone Java application. The application communicates with external systems as a WebSphere MQ messaging client. The main aim of the application is to simulate a legacy system and show the integration of legacy systems into a Web service scenario. The legacy system does not leverage any open standard, such as JMS. Therefore, simple WebSphere MQ XML messages are used for communication with external systems. XML schemas provide a common method of describing the messages that are exchanged.

The application's architecture is depicted in the Figure 10-25 on page 285. The application is message driven and its business logic is depicted as a light green box on the right of the figure. The implementation of the business logic was adopted from the Manufacturer service implementation shipped with the IBM Redbook *Patterns: Implementing an SOA using an Enterprise Service Bus in WebSphere Application Server V6*, SG24-6495.

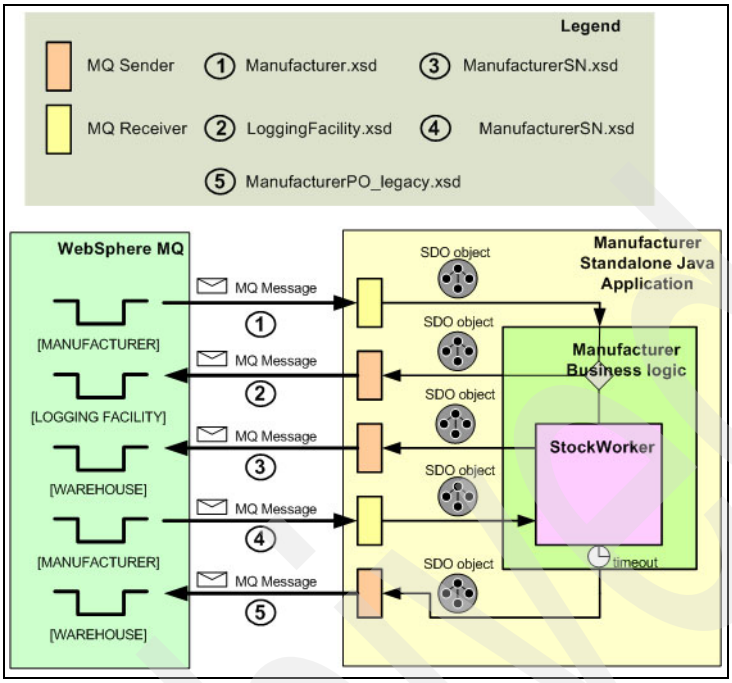

Figure 10-25 High-level design of Manufacturer application

The Warehouse client triggers the Manufacturer's execution by sending a message into the input queue [MANUFACTURER]. The message is represented by ManufacturerSchema.xsd and is received by the MQ receiver component of the legacy application. The MQ receiver component receives incoming messages and transforms them into an internal object format. The SDO data graph was chosen as the internal object format. Hereafter, the application's internal components work only with the internal object representation and are unaware of the WebSphere MQ message format.

During data processing, the business logic must communicate with an external application, and it sends messages to achieve this. Business logic remains independent of the messaging transport. Objects are serialized as WebSphere MQ messages by the MQ Sender component.

All communication from the Manufacturer service is conducted with the WebSphere Business Integration Message Broker ESB, and not with the services themselves, which reside on the separate ESB built in WebSphere Application Server. The first call-out from the application is carried out by the MQ Sender component sending a one-way message to the log service [LOGGING FACILITY]. The application code then sends a message back to the warehouse [WAREHOUSE] and waits for a reply [MANUFACTURER]. Following the

transmittal of a further log message, the transaction completes by sending a final acknowledgement message back to the warehouse [WAREHOUSE]. The delivery address for the message is taken from the original request. If no destination address is specified, the application throws an exception.

Generally, there are two types of communication:

- One-way calls

  No response address is sent in messages that use the one-way call communication style. No response message is expected.

- Request-response calls

  Sending of request-response messages requires knowledge of a response address. Response addresses can be obtained in two ways:

  - Static response address

    A static response address is taken from the configuration file and is not changed during the lifetime of the application's execution.

  - Dynamic response address

    A dynamic response address is set using the request message's header. Request messages must contain reply address information.

Detailed communication sequence

From a messaging point of view, the communication sequence in the Manufacturer application is as follows:

1. The Manufacturer application receives a message from the Warehouse service.

   The message format is defined by the schema Manufacturer.xsd and the root element is named Manufacturer. The MQMD message header contains information about the reply location (ReplyToQ and ReplyToQMgr).

2. The Manufacturer application sends a message to the LoggingFacility service.

   The message format is defined by the schema LoggingFacility.xsd, and the root element is named logEventRequestElement. The message is a one-way message and has contains no reply location information.

3. The Manufacturer application sends a message to the Warehouse service to submit a shipment notice.

   The message format is defined by the schema ManufacturerCallbackMessage.xsd, and the root element is named WareHouseCallbackMessage. The MQMD message header has ReplyToQ value set to the input queue name of the other receiver component of the Manufacturer application.

4. The Manufacturer application sends a message to the LoggingFacility service.
5. The Manufacturer application receives a response message from the Warehouse service.

   The message format is defined by the schema ManufacturerSN.xsd and the root element is named ackSN.
6. The Manufacturer application sends a message to the LoggingFacility service.
7. The Manufacturer application sends an acknowledgement message back to the original Warehouse service.

The message format is defined by the schema ManufacturerPO_Legacy.xsd and the root element is named ackPO.

10.3 Runtime guidelines for ESB based on WebSphere Application Server

This section describes the runtime administration of the ESB built in WebSphere Application Server and its associated service consumers and providers.

10.3.1 Building the WebSphere Application Server infrastructure

The ESB built in WebSphere Application Server hosts the following applications:

- LoggingFacility
- Retailer
- Warehouse
- SCMSampleUI

This section describes the runtime configuration steps that are required to produce an architectural topology as depicted in Figure 10-26 on page 288.

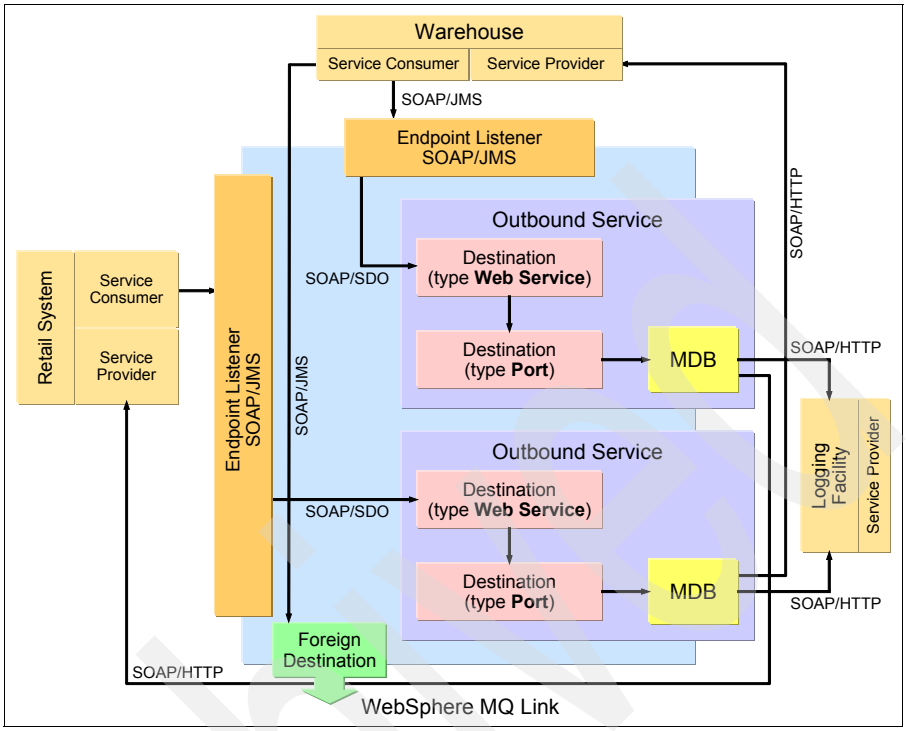

Figure 10-26 WebSphere Application Server scenario architecture

To produce the WebSphere Application Server infrastructure:

1. Install WebSphere Application Server Version 6.
2. Create an Application Server profile.
3. Create a bus.
4. Create a bus member and associated messaging engine.
5. Create a foreign bus.
6. Set the foreign bus destination defaults.
7. Create receiver queue destinations.
8. Attach a mediation to the receiver queue destination.
9. Create JMS resources.

The following instructions assume that the installation of the product has already been performed. It is also assumed that the following software requirements have already been installed on the system that is to host the ESB built in WebSphere Application Server:

- WebSphere Application Server Version 6.0.2
- IBM HTTP Server powered by Apache V6

Create a profile

Note that the scenario could be implemented using WebSphere Application Server Version 6 or, alternatively, the Network Deployment edition. Network Deployment function is used in many of the scenarios in this book, so for the sake of simplicity, the following instructions reference a Network Deployment Manager profile.

Note that this scenario does not explicitly require any function beyond the base Application Server version of the product. There is no functional reason why either product could not be used in this circumstance.

It is assumed that the full software stack listed above has been installed, but an application server profile has not yet been created. If you have completed the other scenarios in this book, it is possible that you may already have a server profile that can be reused, and some of the WebSphere Application Server infrastructure specified in the following sections may already exist. Instructions for the full installation of the topology are provided in case they are required.

The rest of this section describes the steps that are required to create a deployment manager profile. For a fuller description of profiles, see *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451.

Attention: In this scenario the discussion assumes that you are running commands, scripts, and GUI applications on a host system named ITSOESB03.itso.ral.ibm.com.

Start the profile creation wizard:

1. From the Start menu (in Windows only), select:

   Start → Programs → IBM WebSphere → Application Server Network Deployment v6 → Profile creation wizard

2. When you start the wizard, you see the Welcome window. Click **Next** to move to the Profile type selection window.
3. Select **Create an application server profile** and click **Next**.
4. Type the profile name `ITSOESB03`. You may chose to select the **Make this profile the default** checkbox, but this will depend on how you plan to manage the configuration after it is installed. Click **Next** to continue.
5. Accept the default location and click **Next.**
6. Set the node name to `ITSOESB03Node01and` the host name to `ITSOESB03.itso.ral.ibm.com` then click **Next.**
7. Accept the default port assignments. Click **Next.**

8. Choose whether to **Run the application server process as a Windows service** and click **Next.**

> **Tip:** We decided *not* to Run the application server process as a Windows service. This was done to ensure that Windows booted correctly in the unlikely event that there was a problem with this service.

9. Click **Next** on the Profile Summary to create the application server profile.

Now, map ITSOESB03.itso.ral.ibm.com to point to your local computer. To do this on a Windows system, perform the following:

1. Navigate to <Windows_home>\system32\drivers\etc and open the **hosts** file in a text editor.
2. Add the following entries to map the host names required by the scenario to your local machine:

```
127.0.0.1    ITSOESB03.itso.ral.ibm.com
127.0.0.1    appsrv1a.itso.ral.ibm.com
```

> **Note:** appsrv1a.itso.ral.ibm.com is required by the WS-I sample scenario to locate WSDL files from an HTTP server. You will configure the HTTP server in a later step.

3. Add an additional entry for ITSOESB04 that points to the IP address where the WebSphere Business Integration Message Broker ESB is running; for example:

```
1.2.3.4      ITSOESB04.itso.ral.ibm.com
```

4. Save the file. You should now be able to ping any of these host names and have them resolve to your local machine.

Create a service integration bus and bus member

Create a new service integration bus called ITSOESBBus03 as follows:

1. Access and log on to the WebSphere Application Server administrative console (the server must be running) at:

```
http://localhost:9060/ibm/console
```

2. Expand **Service integration** and click **Buses**.
3. Click **New**.
4. In the field labeled Name, enter `ITSOESBBus03`. For the other values, accept the defaults.
5. Click **Apply**, and the bus is created. Save the changes.

Creating a bus just creates an administrative entity. It does not create any resources for messaging. To create a resource, we need to add a bus member, which has the effect of creating a messaging engine. To add a bus member:

1. Click **ITSOESBBus03** to show its properties. Under Topology, click **Bus members**.
2. Click **Add**. Accept the defaults, and click **Next**.
3. Click **Finish**. The server is added as a member of the bus, and a messaging engine is created.
4. Save the changes.

Define messaging resources for the JMS endpoint listener

To support inbound Web service requests via SOAP over JMS, some JMS resources have to be created. These resources are used by the JMS endpoint listener, which you will create later. To create the JMS resources:

1. Create two service integration bus queue type destinations. Use the administrative console to create the queues (**Service integration** → **Buses** → **ITSOESBBus03** → **Destinations**).

   The queue type destinations should be called `wsqmjmsQ1` and `wsqmjmsQ2`.
2. Create two JMS queue connection factories (**Resources** → **JMS Providers** → **Default messaging** → **JMS queue connection factory)**. Use the settings that are specified in Table 10-1 and Table 10-2.

Attention: Be sure to select **JMS queue connection factory**, not JMS connection factory.

Table 10-1 JMS queue connection factory settings

Field	Value
Name	SOAPJMSConnFac1
JNDI name	jms/SOAPJMSFactory1
Bus name	ITSOESBBus03

Table 10-2 JMS queue connection factory settings

Field	Value
Name	SOAPJMSConnFac2
JNDI name	jms/SOAPJMSFactory2
Bus name	ITSOESBBus03

3. Create two JMS queues that point to the service integration bus queue type destinations that you created in step 1 (**Resources** → **JMS Providers** → **Default messaging** → **JMS queue)**. Use the settings that are specified in Table 10-3 and Table 10-4.

Table 10-3 JMS queue settings

Field	Value
Name	SOAPJMSQueue1
JNDI Name	jms/SOAPJMSQueue1
Bus name	ITSOESBBus03
Queue name	wsqmjmsQ1

Table 10-4 JMS queue settings

Field	Value
Name	SOAPJMSQueue2
JNDI Name	jms/SOAPJMSQueue2
Bus name	ITSOESBBus03
Queue name	wsqmjmsQ2

4. Create two activation specifications (**Resources** → **JMS Providers** → **Default messaging** → **JMS activation specification**). Use the settings that are specified in Table 10-5 and Table 10-6 on page 293.

Table 10-5 JMS activation specification settings

Field	Value
Name	SOAPJMSChannel1
JNDI name	eis/SOAPJMSChannel1
Destination type	Queue
Destination JNDI name	jms/SOAPJMSQueue1
Bus name	ITSOESBBus03

Table 10-6 JMS activation specification settings

Field	Value
Name	SOAPJMSChannel2
JNDI name	eis/SOAPJMSChannel2
Destination type	Queue
Destination JNDI name	jms/SOAPJMSQueue2
Bus name	ITSOESBBus03

5. Save the changes.

Prepare the Web services support

To use Web services with the service integration bus of WebSphere Application Server V6, you have to complete the following tasks:

- Install an SDO repository.
- Install service integration bus applications and resources.
- Define JMS resources for the JMS endpoint listener.
- Install the endpoint listener applications.

Install an SDO repository

The service integration bus Web services support stores the WSDL and schemas for the Web services in the SDO repository. When WebSphere Application Server is installed it does not install the SDO repository, so this step must be performed manually.

The SDO repository uses a database to store its information. The SDO repository supports a wide variety of databases. In this scenario, we use embedded Cloudscape database. The SDO repository installation script automatically sets up the relevant resources for the database configuration.

To install the SDO repository:

1. In a command prompt window, navigate to the <install_root>/bin directory, where <install_root> is the directory where you installed WebSphere Application Server.
2. Run the following command:

```
wsadmin -f installSdoRepository.jacl -createDb
```

Install service integration bus applications and resources

A service integration bus is a logical entity that has a physical manifestation in the form of several enterprise applications, a resource adapter, and an activation

specification. As part of the WebSphere Application Server post-install process, these applications must be installed and started before any service integration bus creation and configuration activities can take place. The application installation is achieved by using a JACL script that is shipped with WebSphere Application Server in the <install_root>/util directory, and it is called sibwsInstall.jacl.

The sibwsInstall.jacl script must be run multiple times, once for every application and resource required by the service integration bus support.

1. Install the resource adapter. This must be installed before the other resources and is required. To install the resource adapter, navigate to <install_root>/bin directory and execute the following command:

   ```
   wsadmin -f <install_root>/util/sibwsInstall.jacl INSTALL_RA -installRoot
   <install_root> -nodeName ITSOESB03Node01
   ```

   install_root is the directory where you installed WebSphere Application Server.

   Important: The second <install_root> must have elements in the path separated by a forward slash (/) even on a Windows system, so a path of c:\WAS\AppServer becomes c:/WAS/AppServer.

2. The next task is to install the Web services support application. This can be done from the <install_root>/bin directory by executing the following command:

   ```
   wsadmin -f <install_root>/util/sibwsInstall.jacl INSTALL -installRoot
   <install_root> -nodeName ITSOESB03Node01 -serverName server1
   ```

Install the endpoint listener applications

Install the endpoint listener applications for HTTP and JMS to the service integration bus server.

Although the Web services enterprise application support has now been installed it cannot be used until an endpoint listener application is installed. There are two different endpoint listener applications: one for SOAP over HTTP and one for SOAP over JMS. For this scenario, we install both endpoint listeners.

1. Install the HTTP endpoint listener using the following command:

   ```
   wsadmin -f <install_root>/util/sibwsInstall.jacl INSTALL_HTTP
   -installRoot <install_root> -nodeName ITSOESB03Node01 -serverName
   server1
   ```

2. Install the JMS endpoint listener using the following command:

```
wsadmin -f <install_root>/util/sibwsInstall.jacl INSTALL_JMS
-installRoot <install_root> -nodeName ITSOESB03Node01 -serverName
server1
```

Create endpoint listeners

Endpoint listeners listen for incoming Web service requests and forward them onto the relevant inbound services. Inbound services get bound to an endpoint listener when they are created. We need to create an HTTP and JMS endpoint listener. These are the steps:

1. Expand **Servers** and click **Application Servers**.
2. Click **server1**.
3. Under Additional Properties click **Endpoint Listeners**.
4. Click **New**.
5. Enter the following details:
   - Name: `SOAPHTTPChannel1`
   - URL root: `http://ITSOESB03.itso.ral.ibm.com:9080/wsgwsoaphttp1`
   - WSDL serving HTTP URL: `http://ITSOESB03.itso.ral.ibm.com:9080/sibws/wsdl`
6. Click **Apply**.
7. Under Additional Properties click **Connection Properties**.
8. Click **New**.
9. In the Bus name pull-down menu, select **ITSOESBBus03** and click **OK**.
10. Repeat these steps to create an endpoint listener for JMS using the following settings:
    - Name: `SOAPJMSChannel1`
    - URL root: `jms:/queue?destination=jms/SOAPJMSQueue1&connectionFactory=jms/SOAPJMSFactory1&`
    - WSDL serving HTTP URL: `http://ITSOESB03.itso.ral.ibm.com:9080/sibws/wsdl`
11. Save your changes.

Hosting WSDL files

Each Web service client in the WS-I sample application references WSDL files containing port type and binding information. Import statements dictate the

location of these files. Configure an HTTP server as described in “Hosting the WSDL files” on page 200.

Create a foreign bus

In order for the ESB built in WebSphere Application Server to exchange messages with the WebSphere Business Integration Message Broker ESB, it is necessary to establish a communication link between the messaging engine on the first ESB with the queue manager on the second ESB. The definition of this communication link requires the creation of a foreign bus. A foreign bus is a property of a service integration bus. It is used to represent another service integration bus, with which it can exchange messages. This representation may be of another service integration bus (as in the previous scenario) or, as in this case, a WebSphere MQ network. You need to create a foreign bus to represent the queue manager, which will be connected to using the WebSphere MQ Link.

1. In the navigation pane, click **Service integration** → **Buses** → **ITSOESBBus03** → **Foreign buses,** and click **New**.
2. Enter `QM1` in Name, and click **Next.**
3. Click the Routing Type pull-down menu and select **Direct, WebSphere MQ Link**, and click **Next.**
4. Accept the defaults and click **Next.**
5. Click **Finish,** and save your workspace changes to the master.

Set the foreign bus destination defaults

When messages are sent across the WebSphere MQ Link from the ESB built in WebSphere Application Server to the WebSphere Business Integration Message Broker ESB, it is necessary for an MQRFH2 header to be included with the messages. This header contains the JMS properties used by the bus, and it enables the routing of messages to the correct eventual service provider. On the foreign bus definition, set a property to make sure that MQRFH2 headers are always copied by default when messages are sent to the WebSphere Business Integration Message Broker ESB:

1. In the navigation pane, click **Service integration** → **Buses** → **ITSOESBBus03** → **Foreign buses** → **QM1** → **Destination defaults** → **Context properties**, and click **New**.
2. Enter `_MQRFH2Allowed` in Name
3. Leave the Context type pull-down menu as the default `Boolean`.
4. Enter `true` in Context value, and click **OK**.
5. Save your workspace changes to the master configuration.

Create receiver queue destinations

All request messages received over the WebSphere MQ Link from the WebSphere Business Integration Message Broker ESB are interpreted as Web service requests destined for one of the service providers attached to the ESB built in WebSphere Application Server. A destination of type queue must be created to receive these messages; we name it `WEBSERVICE.BROKER.REQUEST`. This destination has no service provider attached to it. Instead, a mediation processes messages placed on the destination queue and forwards them to the correct service provider. See “Attach a mediation to the receiver queue destination” on page 297 for further details about the implementation of this mediation.

Create the queue destination named `WEBSERVICE.BROKER.REQUEST`:

1. In the navigation pane, click **Service integration** → **Buses** → **ITSOESBBus03** → **Destinations**, and click **New**.
2. Select the destination type of **Queue**, and click **Next**.
3. Enter `WEBSERVICE.BROKER.REQUEST` in Name, and click **Next**..
4. Leave the Bus member pull-down menu setting as default and click **Next**.
5. Click **Finish,** and save your workspace changes to the master configuration.

Repeat this process to create a queue destination called `SOAPJMSQueue1`.

Attach a mediation to the receiver queue destination

A mediation is attached to a destination and processes messages arriving there. We must create a mediation that is attached to the queue destination named `WEBSERVICE.BROKER.REQUEST`. The purpose of the mediation is to use the information within the endpointURL property supplied in the JMS user-defined property of the MQRFH2 header, to determine the service provider for which the message is intended and route it there. The endpointURL is parsed into a targetService and a JMS destination reference. The JMS destination reference is used to consult the local WebSphere Application Server JNDI directory to locate a destination name and bus name (an SIDestinationAddress object). This destination address is added to the forward routing path and when the mediation completes its processing, the message is forwarded there.

Create the mediation:

1. In the navigation pane, click **Service integration** → **Buses** → **ITSOESBBus03** → **Mediations**, and click **New**.
2. Enter `BROKERJMSPROCESSOR` in Name. The Mediation Handler List specifies the Java classes that make up the mediation, and the order in which they should be applied to the message. Every mediation has a single Mediation Handler

List (which is deployed inside the deployment descriptor for the mediation EJB). Enter `BrokerJMSProcessor` in Handler List name, and click **OK**.

Attach the mediation you have just created to the destination named WEBSERVICE.BROKER.REQUEST:

1. In the navigation pane, click **Service integration** → **Buses** → **ITSOESBBus03** → **Destinations**, select **WEBSERVICE.BROKER.REQUEST**, and click **Mediate**.
2. Select **BROKERJMSPROCESSOR** in Name, and click **Next**.
3. Leave The mediation to apply to this destination the pull-down menu as the mediation that you just created, **BROKERJMSPROCESSOR**, and click **Next**.
4. Leave the bus member where the mediation point is assigned as the default setting and click **Next**.
5. Click **Finish,** and save your workspace changes to the master configuration.

Deploy the mediation as an application to server1:

1. In the navigation pane, click **Applications** → **Enterprise Applications**, and click **Install**.
2. Browse to the location of the enterprise archive file for the mediation named BrokerJmsProcessor.ear, which you created earlier in the chapter (a copy is also supplied with this book's additional materials), and click **Next**.
3. Select **Generate Default Bindings**, and click **Next**.
4. There is no need to change any further settings from their default values.
5. Select the final step and click **Finish**.
6. When installation is complete, save your changes to the master configuration.
7. In the navigation pane, click **Applications** → **Enterprise Applications**, select **BrokerJmsProcessor**, and click **Start**.
8. Repeat this process to install the UIRetailerLogging.ear file, which contains the Retailer, LoggingFacility, SCMSampleUI, and Warehouse applications.
9. Start the mediation point.

   In the navigation pane, click **Service integration** → **Buses** → **ITSOESBBus03** → **Destinations** → **WEBSERVICE.BROKER.REQUEST** → **Mediation Points.** Select the mediation and click **Start.**

Create JMS resources

JMS queues are used as a destination for point-to-point messaging. Inbound messages that are initiated from the WebSphere Business Integration Message Broker ESB (which are not responses to previous messages) and sent to the

WebSphere Application Server service integration bus are placed on a queue destination named WEBSERVICE.BROKER.REQUEST. The mediation that was discussed in the previous section processes these messages and uses the JMS queue information, which is held in the endpointURL property of the message, to locate the physical local queue referenced by the JMS queue definition, named SOAPJMSQueue1. Service providers attached to the ESB, for whom these messages are destined, also require the JMS queue definitions in order to locate the physical queue.

All JMS resources are defined within the WebSphere Application Server administrative console with a scope. The scope of a definition specifies the level at which the resource is visible. The scope of the resources created in this scenario is chosen to be Node level. This means that all executing code defined on the node (physical machine) can access the JMS resources defined.

Create a JMS queue with Node scope:

1. In the navigation pane, click **Resources** → **JMS Providers** → **Default messaging**, and select **Node:ITSOESB03Node01**, then click **Apply**.
2. Select **JMS queue**, and click **New**.
3. Enter `LOCALQ1JMS` in Name.
4. Enter `jms/localq1jms` in JNDI name.
5. Click the Bus name pull-down menu and select **ITSOESBBus03.**
6. Click the Queue name pull-down menu and select **SOAPJMSQueue1.**
7. Click **OK,** and save your workspace changes to the master configuration.

10.3.2 Linking the bus using the WebSphere MQ Link

Communication between the ESB built in WebSphere Application Server and the WebSphere Business Integration Message Broker ESB is achieved by flowing point-to-point messages across the architectural component known as the WebSphere MQ Link. A WebSphere MQ Link is defined on a messaging engine within a service integration bus. Configuration steps are also required on the WebSphere MQ queue manager to which messaging engine is connected. This scenario requires the flow of messages in both directions between the two ESBs, which means that the link must be configured with a sender and a receiver channel.

Create an alias destination

Before we configure the WebSphere MQ Link, we must define an alias destination in the service integration bus. This alias destination will point to the

queue PO.REQ.IN running on the remote WebSphere MQ manager. Perform the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → **ITSOESBBus03** → **Destinations** and click **New**.
2. Select a destination type of **Alias** and click **Next**.
3. Enter the following information then click **Next**:
   - Set the Identifier field to `PO.REQ.IN`
   - Select the Bus field as **ITSOESBBus03**.
   - Select the Target Bus (the bus where the real queue lives) as **QM1**.
   - Select the Target Identifier to **other, please specify** and specify a value of `PO.REQ.IN@QM1` (this represents the name of the target queue, and the name of the WebSphere MQ queue manager on which it is running).
4. In the next window, click **Finish**.

Create a JMS queue for the alias destination

With the alias defined, we can create a JMS queue that points to this destination. Perform the following steps:

1. In the navigation pane, click **Resources** → **JMS Providers** → **Default messaging**.
2. Select **JMS queue**, and click **New.**
3. Enter `poreqin` in Name.
4. Enter `jms/poreqin` in JNDI name.
5. Click the Bus name pull-down menu and select **ITSOESBBus03.**
6. Click the Queue name pull-down menu and select **PO.REQ.IN**.
7. Click **OK,** and save your workspace changes to the master configuration.

Create a WebSphere MQ Link

Perform the following to create a WebSphere MQ Link:

1. In the navigation pane, click **Service integration** → **Buses** → **ITSOESBBus03** → **Messaging engines** → **ITSOESBBus03Node01.server1-ITSOESBBus03** → **WebSphere MQ links**, and click **New**.
2. Enter `QM1LINK` in Name.
3. Click the Foreign bus name pull-down menu and select **QM1**.

4. Enter `ITSOESBBus03` in Queue Manager Name, and click **Next**.

   This setting determines the queue manager name that will be used to populate the ReplyToQMgr and JMS ReplyTo fields in the MQMD and MQRFH2 headers of messages sent across the link to WebSphere MQ.

5. Enter `ITSOESBBus03.TO.QM1` in Sender MQ channel name. The name of this channel must be the same as the name of the corresponding receiver channel on the Queue Manager.

6. Enter `ITSOESB04.itso.ral.ibm.com` in Host Name. This is the host name of the Queue Manager to which the sender link connects.

7. Check that the wizard has `1414` in Port Number. This is the port used by the WebSphere MQ Listener of the Queue Manager to which the sender link connects.

8. Click the Transport chain pull-down menu, select **OutboundBasicMQLink**, and click **Next**.

9. Enter `QM1.TO.ITSOESBBus03` in Receiver MQ channel name, and click **Next**.. The name of this channel must be the same as the name of the corresponding sender channel on the Queue Manager.

10. Click **Finish,** and save your workspace changes to the master configuration.

11. Restart the server so the WebSphere MQ link can be started.

10.3.3 Adding services to the bus

Optionally you could add inbound and outbound services to the service integration bus for each Web service call. This is recommended for building a full ESB solution. To do this, follow the instructions in “Creating the outbound services” on page 207 and “Creating the inbound services” on page 220. In terms of building a working sample application, this is optional.

At a minimum, however, you must override the Web service client bindings for the Warehouse enterprise application. In order for the Warehouse Service (acting as a consumer) to send a message to the Manufacturer service providers located on the WebSphere Business Integration Message Broker ESB, over the WebSphere MQ Link, it is necessary to alter the Web Services Client bindings for those particular services in the deployment descriptor of the Warehouse’s EJB.

Alter the Web Services client bindings Port Information:

1. In the navigation pane, click **Applications** → **Enterprise Applications** → **UIRetailerLogging** → **EJB Modules** → **WarehouseEJB.jar** → **Web**

services client bindings. For ManufacturerService, click the **Edit** link under Port information and enter the following in the Overridden Endpoint URL field:

```
jms:/queue?destination=jms/poreqin&connectionFactory=jms/SOAPJMSF
actory1|targetService=ITSOESBO4/MANUFACTURERA
```

2. Click **OK**.

10.4 Runtime guidelines for ESB based on WebSphere Business Integration Message Broker

This section describes how to deploy the WebSphere Business Integration Message Broker artefacts necessary to run this scenario. The five message flows that contribute to form the WebSphere Business Integration Message Broker ESB could all be deployed to a single (e.g. default) execution group. Production systems often choose to divide message flows between several different execution groups for performance reasons or organizational concerns. In order to demonstrate how this can be achieved, the following instructions choose to deploy the message flows between three execution groups. These runtime guidelines will carry out the following operations:

- Configuring WebSphere MQ queues and channels
- Connect the toolkit to the configuration manager
- Create execution groups
- Create and deploy Broker archive files
- Create database resources

10.4.1 Configuring WebSphere MQ queues and channels

You must define several resources in WebSphere MQ for the default queue manager QM1, which we use as a single queue manager for supporting the configuration manager and runtime broker:

1. Create a new WebSphere MQ Listener, listening on port 1414 using the TCP protocol, then start this listener.
2. Create a WebSphere MQ sender channel with the following attributes:
   - Channel Name: `QM1.TO.ITSOESBBus03`
   - Transmission Protocol: `TCP/IP`
   - Connection Name: `ITSOESBO3.itso.ral.ibm.com(5558)`

Note: This Connection Name setting assumes that ITSOESB03.itso.ral.ibm.com can be resolved to the host system where the WebSphere Application Server ESB is running.

   - Transmission Queue: `ITSOESBBus03`
3. Create a WebSphere MQ receiver channel called `ITSOESBBus03.TO.QM1` with a Transmission Protocol of TCP/IP.
4. Create a transmission queue called `ITSOESBBus03`.
5. Create the following local queues:
   - LOG.IN
   - PO.REQ.IN
   - PO.RES.IN
   - WHCB.REQ.IN
   - WHCB.RES.IN
   - MANA.PO
   - MANB.PO
   - MANC.PO
   - MANA.WHCB
   - MANB.WHCB
   - MANC.WHCB

10.4.2 Connect the toolkit to the configuration manager

In order to deploy build time artefacts (message sets and message flows) to the runtime broker, the WebSphere Business Integration Message Broker Toolkit must communicate with the runtime Configuration Manager. The toolkit uses a WebSphere MQ client connection to do this. The following instructions assume that you have already created a runtime configuration manager, and a runtime broker named BK1.

If you have not already done so, open the WebSphere Business Integration Message Broker Toolkit that was used to create the resources in the Development section of this scenario.

1. Open the Broker Administration perspective by selecting **Window** → **Open Perspective** → **Broker Administration**.
2. Right-click the **Domain Connections** folder in the Broker Administration Navigator and select **New** → **Domain**.

3. Enter `QM1` in Queue Manager Name, `localhost` in Host and `1414` in Port, and click **Next**. These settings assume that you installed the WebSphere Business Integration Message Broker Toolkit on the same physical machine as the runtime components.
4. Enter `Servers` in Server Project and `Domain1` in Connection Name, and click **Finish**. This creates a Server Project named Servers and stores the configuration manager connection information you specified within the project inside a file named Domain1.configmgr.
5. Having connected to the configuration manager, create a representation of your runtime broker within the Toolkit topology that is shown in the Domains view. This adds the runtime broker to the topology that is controlled by the configuration manager, and enables you to deploy to the runtime broker from the Toolkit. Right-click the **Broker Topology** level of the hierarchy displayed in the Domains view and select **New** → **Broker**.
6. Enter `BK1` in Broker name and `QM1` in Queue Manager Name, and click **Finish**.

When the action successfully completes, you will see a broker named BK1 appear in the hierarchy of the Domains view, beneath the Broker Topology level. The BK1 broker will have a single execution group as a child, named `default`.

10.4.3 Create execution groups

The following instructions describe the creation of separate execution groups, which will be used to organize the deployed message sets and message flows:

1. Right-click the broker **BK1** in the Domains view and select **New** → **Execution Group**.
2. Enter `PurchaseOrder` in the Execution Group name, and click **Finish**.
3. The new execution group should appear beneath the broker **BK1**.
4. Right-click the broker **BK1** in the Domains view and select **New** → **Execution Group**.
5. Enter `Log` in the Execution Group name, and click **Finish**.
6. The new execution group should appear beneath the broker **BK1**.
7. Right-click the broker **BK1** in the Domains view and select **New** → **Execution Group**.
8. Enter `WarehouseCallback` in the Execution Group name, and click **Finish**.
9. The new execution group should appear beneath the broker **BK1**.

10.4.4 Create and deploy Broker archive files

Three Broker Archive Files are used (one for each execution group) to deploy the message flows and message sets. Note that message dictionaries (the runtime term for a deployed message set) are not shared between separate execution groups, so they must be deployed separately to each execution group that needs access to this metadata.

1. Right-click the **Broker Archives** folder in the Broker Administration Navigator and select **New** → **Message Broker Archive**.
2. Select the **Servers** project and enter `PurchaseOrder` in File Name, then click **Finish**.
3. Click the **Add** icon in the main editing window.
4. Check the **msp_Manufacturer** message set project, check the **msp_Other** message set project, and click **OK** to add it to the PurchaseOrder.bar file. Click **OK** in the response dialog window.
5. Save the PurchaseOrder.bar file by pressing Ctrl+S.
6. Click the **Add** icon in the main editing window.
7. Click on the message flow project name **mfp_Manufacturer**, and the available message flows will appear in the panel on the right of the dialog. Select the message flows named **PurchaseOrderRequest.msgflow** and **PurchaseOrderResponse.msgflow** and click **OK** to add it to the PurchaseOrder.bar file. Click **OK** in the response dialog window.
8. Save the **PurchaseOrder.bar** file by pressing Ctrl+S.
9. If the domain is not already connected, connect it now. In the Domains view, connect to the Broker by right-clicking and selecting **Connect**.
10. Deploy the **PurchaseOrder.bar** file by dragging it from the Broker Administration Navigator view to the execution group named **PurchaseOrder** under the broker BK1 in the Domains view.
11. Click **OK** to acknowledge the response message from the Configuration Manager.
12. Double-click the **Event log** in the Domains view, and watch for the two successful messages with a current timestamp.

The message sets and message flows for the PurchaseOrder are now deployed and ready for use. Now deploy the rest of the WebSphere Business Integration Message Broker message flows and sets:

Deploy to the Log execution group:

1. Create a Broker Archive File named `Log.bar`.

2. Add message flow **LogEvent.msgflow** and message set **msp_Other**. (Check the message set project named **msp_Other**.)
3. Deploy **Log.bar** to the execution group named **Log** under broker BK1.

Deploy to the WarehouseCallback execution group:

1. Create a Broker Archive File named **WarehouseCallback.bar**.
2. Add message flows **WarehouseCallbackRequest.msgflow** and **WarehouseCallbackResponse.msgflow** and message sets **msp_Other** (check the message set project named **msp_Other**) and **msp_Manufacturer** (check the message set project named **msp_Manufacturer**).
3. Deploy WarehouseCallback.bar to the execution group named **WarehouseCallback** in broker BK1.

10.4.5 Create database resources

The PurchaseOrderRequest flow uses a database lookup to find out the queue and queue manager name of the location it should send its messages to. It finds these values by issuing an SQL select statement that uses the name of the target service from the MQRFH2 header of the input message sent from WebSphere Application Server.

In this section you will create this database in DB2 Universal Database, and define an ODBC data source for it.

Create the SVCDIR database

We will create a database and table in DB2 Universal Database to link the Manufacturer's service queues to the target service property in the input message from the ITSOESB03 (WebSphere Application Server) ESB.

The following steps create a table called SVCDIR and a table called ROUTE:

1. Start the DB2 Command Window.
2. Enter the following commands in the DB2 Command Window (these instructions assume you are logged in as user admin):

```
db2 create database SVCDIR

db2 connect to SVCDIR

db2 create table ROUTE( SERVICE varchar(100), QUEUE varchar(100),
QUEUEMGR varchar(100))

db2 insert into ROUTE
values('ITSOESB04/MANUFACTURERA','MANA.PO','QM1')
```

```
db2 insert into ROUTE
values('ITSOESB04/MANUFACTURERB','MANB.PO','QM1')

db2 insert into ROUTE
values('ITSOESB04/MANUFACTURERC','MANC.PO','QM1')
```

Create the ODBC data source

Perform the following to create an ODBC data source for this database:

1. In Windows select **Control Panel** → **Administrative Tools** → **Data Sources (ODBC)**.
2. Click the **System DSN** tab.
3. Click **Add** to create a new system data source.
4. In the Create a New Data Source window, select **IBM DB2 ODBC DRIVER** and click **Finish**.
5. In the ODBC IBM DB2 Driver window enter a Data source name of `SVCDIR` and set the Database alias to **SVCDIR**. Click **OK** to create the data source.

10.5 Runtime guidelines for legacy manufacturer application

The Manufacturer application is a stand-alone Java application that uses the WebSphere MQ API to communicate with the outer world. One application is provided that can be run as a Manufacturer A or B or C.

The same application code is designed to be run as three separate instances at the same time. Therefore the Manufacturer application has five parameters to control its operation. The Manufacturer application can be run by opening a command prompt and typing:

```
java -jar ManufacturerApplication.jar config\SubmitPO_receiver.properties
config\SubmitSN_receiver.properties config\SubmitSN_sender.properties
config\LogEvent_sender.properties A
```

The Table 10-7 on page 308 describes the meaning of the runtime parameters.

Table 10-7 ManufacturerApplication runtime parameters

Parameter position	Parameter name	Description
1	SubmitPO_receiver.properties	Properties for purchase order receiver. Queue name and associated properties specify the input queue, where request messages (defined by Manufacturer.xsd) arrive.
2	SubmitSN_receiver.properties	Properties for shipment notice acknowledgement receiver. Queue name and associated properties specify the input queue where acknowledgment messages arrive.
3	SubmitSN_sender.properties	Properties for shipment notice sender. Queue name and associated properties specify the output queue where shipment notice messages are sent.
4	LogEvent_sender.properties	Properties for log events sender. Queue name and associated properties specify the output queue where log messages are sent.
5	Manufacturer Type	Type of manufacturer application. Manufacturer type has three possible values: A, B, or C.

The Table 10-8 describes the format of the property files.

Table 10-8 Description of properties used in configuration files

Property name	Description
QUEUE_MANAGER_ NAME	Name of the WebSphere MQ Queue Manager
QUEUE_NAME	Name of WebSphere MQ Queue
PORT	Number of the port used by the WebSphere MQ Listener
HOSTNAME	Hostname or IP address of the computer hosting the WebSphere MQ Queue Manager
CHANNEL	Name of WebSphere MQ channel being use for client connection

Property name	Description
CCSID	CCSID of WebSphere MQ Queue Manager

Table 10-9, Table 10-10 on page 309, and Table 10-11 on page 310 describe the default configuration of the property files shipped with the sample applications.

Table 10-9 Application A property settings

Property	Value
QUEUE_MANAGER_NAME	QM1
QUEUE_NAME	MANA.PO
PORT	1414
HOSTNAME	ITSOESB04
CHANNEL	SYSTEM.DEF.SVRCONN
CCSID	1208

Default properties for the ManufacturerA application are stored in the /config directory and are named:

- LogEvent_sender.properties
- SubmitPO_receiver.properties
- SubmitSN_receiver.properties
- SubmitSN_sender.properties

Table 10-10 Application B property settings

Property	Value
QUEUE_MANAGER_NAME	QM1
QUEUE_NAME	MANB.PO
PORT	1414
HOSTNAME	ITSOESB04
CHANNEL	SYSTEM.DEF.SVRCONN
CCSID	1208

Default properties for the ManufacturerB application are stored in the /config directory and are named:

- LogEvent_senderB.properties
- SubmitPO_receiverB.properties

- SubmitSN_receiverB.properties
- SubmitSN_senderB.properties

Table 10-11 Application C property settings

Property	Value
QUEUE_MANAGER_NAME	QM1
QUEUE_NAME	MANC.PO
PORT	1414
HOSTNAME	ITSOESB04
CHANNEL	SYSTEM.DEF.SVRCONN
CCSID	1208

Default properties for the ManufacturerC application are stored in the /config directory and are named:

- LogEvent_senderC.properties
- SubmitPO_receiverC.properties
- SubmitSN_receiverC.properties
- SubmitSN_senderC.properties

To simplify execution of the applications, three batch files that already reference the predefined parameters discussed above, are stored in the /config directory. You can run the batch files by opening a command prompt and typing:

```
ManufacturerAApplication.bat
ManufacturerBApplication.bat
ManufacturerCApplication.bat
```

10.6 Testing the application

To test the WS-I application, ensure that the following things are running:

- The WebSphere Application Server server instance with the ESB configured as described in 10.3, “Runtime guidelines for ESB based on WebSphere Application Server” on page 287.
- The WebSphere Business Integration Message Broker server with the ESB configured as described in 10.4, “Runtime guidelines for ESB based on WebSphere Business Integration Message Broker” on page 302.
- The three legacy manufacturer Java applications are started as described in 10.5, “Runtime guidelines for legacy manufacturer application” on page 307.

To test the complete scenario, open a Web browser on the machine hosting the WebSphere Application Server ESB, and enter the following URL:

```
http://ITSOESB03.itso.ral.ibm.com:9080/SCMSampleUI
```

This invokes the WS-I sample application. Test the connection to the manufacturers by placing orders for the first three products, ensuring a quantity of at least 6 for each product. This triggers a call to the WebSphere Business Integration Message Broker ESB and to each manufacturer. For more information on how to test the WS-I sample application, see 9.3.6, “Testing the scenario” on page 236.

Part 4

Appendixes

Abbreviations and acronyms

ACID	Automicity, Consistency, Isolation, Durability
API	Application Programming Interface
BLOB	Binary Large Object
CCI	Common Client Interface
CICS	Customer Information Control System
COBOL	Common Business-Oriented Language
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-The-Shelf
DBMS	Database Management System
DMZ	Demilitarized Zone
DTD	Document Type Definition
DVD	Digital Video Disc
EAI	Enterprise Application Integration
EAR	Enterprise Archive
EDI	Electronic Data Interchange
EIS	Enterprise Information System
EJB	Enterprise JavaBean
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTP/S	Hypertext Transfer Protocol over Secure Sockets Layer
IDE	Integrated Development Environment
IIOP	Internet Interoperable ORB Protocol
IMS	Information Management System
ISBN	International Standard Book Number
ITSO	International Technical Support Organization
JAAS	Java Authentication and Authorization Service
JAR	Java Archive
JAX-RPC	Java API for XML-based Remote Procedure Call
JDBC	Java Database Connectivity
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JSP	Java Server Page
MDB	Message Driven Bean
OASIS	Organization for the Advancement of Structured Information Standards
ODBC	Open Database Connectivity
RMI	Remote Messaging Interface
RPC	Remote Procedure Call
SCM	Supply Chain Management
SDK	Software Development Kit
SDO	Service Data Object
SOA	Service Oriented Architecture
SQL	Structured Query Language
SSL	Secure Socket Layer
TCP/IP	Transmission Control Protocol / Internet Protocol

UDDI	Universal Description, Discovery, and Integration
UML	Unified Modeling Language
URI	Universal Resource Identifier
URL	Universal Resource Locator
WS-BPEL	Web Services Business Process Execution Language
WS-I	Web Services Interoperability
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

A

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246773

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, SG246773.

Using the Web material

The additional Web material that accompanies this book includes the following file:

File name	*Description*
SG246773.zip	Zipped Code Samples

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 10 MB for source material
Operating System: Windows operating system
Memory: 1.5 GB to run source material

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 322. Note that some of the documents referenced here may be available in softcopy only.

- *Patterns: Broker Interactions for Intra- and Inter-enterprise*, SG24-6075
- *Patterns: Direct Connections for Intra- and Inter-enterprise*, SG24-6933
- *Patterns: Serial and Parallel Processes for Process Choreography and Workflow*, SG24-6306
- *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6,* SG24-6494
- *Patterns: Using Business Service Choreography In Conjunction With An Enterprise Service Bus*, REDP-3908
- *WebSphere Application Server V6 System Management and Configuration Handbook*, SG24-6451

Other publications

These publications are also relevant as further information sources:

- Jonathan Adams, Srinivas Koushik, Guru Vasudeva, and George Galambos, *Patterns for e-business: A Strategy for Reuse,* IBM Press, 2001, ISBN 1931182027
- *WebSphere MQ Application Programming Guide*, SC34-6064-03.

Online resources

These Web sites and URLs are also relevant as further information sources:

- Patterns for e-business Web site
  http://www.ibm.com/developerWorks/patterns/
- IBM Enterprise Service Bus stategy
  http://www.ibm.com/software/info1/websphere/index.jsp?tab=landings/esb
- WebSphere Application Server
  http://www.ibm.com/software/webservers/appserv/was/
- WebSphere Business Integration Message Broker
  http://www.ibm.com/software/integration/wbimessagebroker
- WebSphere MQ
  http://www.ibm.com/software/ts/mqseries
- WebSphere Enterprise Service Bus
  http://www.ibm.com/software/integration/wsesb/
- Rational Application Developer
  http://www.ibm.com/software/awdtools/developer/application
- DB2 Universal Database
  http://www.ibm.com/software/data/db2/udb
- WebSphere Application Server benchmarks
  http://www.spec.org/jAppServer2004/results/jAppServer2004.html
- Web Services Interoperability Organization
  http://www.ws-i.org
- The role of private UDDI nodes in Web services, Part 1: Six species of UDDI
  http://www.ibm.com/developerworks/webservices/library/ws-rpu1.html
- The role of private UDDI nodes, Part 2: Private nodes and operator nodes
  http://www.ibm.com/developerworks/webservices/library/ws-rpu2.html
- First look at the WS-I Basic Profile V1.0
  http://www.ibm.com/developerworks/webservices/library/ws-basicprof.html
- First look at the WS-I Usage Scenarios
  http://www.ibm.com/developerworks/webservices/library/ws-iuse/
- Preview of WS-I sample application
  http://www.ibm.com/developerworks/webservices/library/ws-wsisamp/

- IBM Emerging Technologies Toolkit

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

- Security in a Web Services World: a Proposed Architecture and Road map

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

- Web Services Security: Moving up the stack

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

- Updated: Web Services Reliable Messaging: A new protocol for reliable delivery between distributed applications

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

- Implementation Strategies for WS-ReliableMessaging: How WS-ReliableMessaging can interact with other middleware communication systems

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

- WS-BPEL specification

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

- Business Processes with WS-BPEL, a series of introductory articles and references

  http://www.ibm.com/developerworks/webservices/library/ws-bpelcol1/

- WS-BPEL support in WebSphere Business Integration Server Foundation

  http://www.ibm.com/software/integration/wbisf/features/

- WS-BPEL support in WebSphere Studio Application Developer Integration Edition

  http://www.ibm.com/software/integration/wsadie/features/

- WS-AtomicTransaction specification

  http://www.ibm.com/developerworks/library/ws-atomtran/

- WS-BusinessActivity specification

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

- Transactions in the world of Web Services, part 1 and part 2

  http://www.ibm.com/developerworks/webservices/library/ws-wstx1/
  http://www.ibm.com/developerworks/webservices/library/ws-wstx2/

- WS-Coordination specification

  http://www.ibm.com/developerworks/library/ws-coor/

- WS-Policy framework specification

  http://www.ibm.com/developerworks/library/ws-polfram/

- Web Services Policy Framework: New specifications improve WS-Security

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

- Java specification for SDO

  http://www.jcp.org/en/jsr/detail?id=235

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

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Redbooks

Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture

Integrate ESBs in WebSphere V6 and Message Broker V5

Patterns for integrating ESBs

Learn by example with practical scenarios

The Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying e-business applications. This IBM Redbook focuses on how you can integrate Enterprise Service Bus (ESB) implementations in a service-oriented architecture (SOA). The book discusses patterns for integrating ESBs and includes step-by-step instructions for integrating ESBs implemented in WebSphere Business Integration Message Broker V5 and WebSphere Application Server V6. However, the ESB integration patterns and concepts apply to ESBs implemented with any product.

Part 1 introduces SOA and ESB concepts, and discusses the ESB capabilities of WebSphere Business Integration Message Broker V5 and WebSphere Application Server V6. It describes guidelines for determining when integration of ESBs is necessary, and describes patterns for integrating ESBs.

Part 2 describes the business scenario used in this book and explains key technologies relevant to SOA and ESB.

Part 3 guides you through the process of integrating ESBs. Two scenarios are described: integration of homogeneous ESBs and of heterogeneous ESBs. The homogeneous ESB scenario describes the integration of two ESBs implemented in WebSphere Application Server V6. The heterogeneous ESB scenario describes integration between an ESB implemented in WebSphere Application Server V6 and an ESB implemented in WebSphere Business Integration Message Broker V5.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6773-00 ISBN 0738492930