Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB

ESB implementation options for maturing SOA

Enhance your knowledge of IBM ESB products

Learn how to enable your environment with ESB patterns

Rufus Credle
Jonathan Adams
Kim Clark
Yun Peng Ge
Hatcher Jeter
Joao Lopes
Samir Nasser
Kailash Peri

Redbooks

IBM

International Technical Support Organization

Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB

July 2007

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

First Edition (July 2007)

This edition applies to WebSphere Application Server 6.1, WebSphere DataPower, WebSphere Enterprise Service Bus 6.0.2, and WebSphere Message Broker 6.0.0.3.

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 about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

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

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements 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 illustrate 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.

Trademarks

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

Redbooks (logo) ®	DB2 Universal Database™	Rational®
developerWorks®	DB2®	Redbooks®
z/OS®	Everyplace®	RUP®
zSeries®	HACMP™	S/370™
AIX®	IBM®	SupportPac™
BladeCenter®	IMS™	System x™
Component Business Model™	Lotus®	System z™
CICS®	MQSeries®	Tivoli Enterprise™
DataPower®	MVS™	Tivoli®
Domino®	Rational Unified Process®	WebSphere®

The following terms are trademarks of other companies:

BAPI, SAP R/3, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

EJB, Java, JDBC, JRE, JVM, J2EE, J2SE, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

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

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

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

Preface

The 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® Redbooks® publication focuses on the use of the WebSphere Enterprise Service Bus and WebSphere Message Broker together to form an enterprise service bus (ESB) implemented in a service-oriented architecture (SOA).

This book discusses patterns for integrating WebSphere Enterprise Service Bus and WebSphere Message Broker and includes a scenario to help you design, develop, and deploy these products.

This book is designed to assist customers that are approaching the use of both advanced and basic ESB products from typically messaging and J2EE™ worlds, but are not quite sure when each is appropriate.

The team that wrote this book

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

Figure 1   The team from bottom left to right: Kim Clark, Ge Yun Peng, Rufus Credle, Kailash Peri, Samir Nasser, Joao Lopes, Hatcher Jeter (Jonathan Adams not pictured)

Rufus Credle is a Certified Consulting IT Specialist at the ITSO, Raleigh Center. In his role as Project Leader, he conducts residencies and develops IBM Redbooks about network operating systems, ERP solutions, voice technology, high availability and clustering solutions, Web application servers, pervasive computing, IBM and OEM e-business applications, IBM System x™, System x, and IBM BladeCenter®. Rufus' various positions during his IBM career have included assignments in administration and asset management, systems engineering, sales and marketing, and IT services. He holds a BS degree in business management from Saint Augustine's College. Rufus has been employed at IBM for 27 years.

Jonathan Adams is an IBM Distinguished Engineer. He has been an IT architect with IBM for 38 years. For the last ten years he has been focused on developing

a pattern language that allows an architect to describe a complex solution as a composite of coarse-grained patterns, to provide a powerful solution decomposition technique for enabling business transformation, and to provide the basis for a tooling approach for developing IT solutions based upon proven patterns and industry best practice. Since September 1998, he has been working in the SWG Technical Strategy organization leading the definition and development of the Patterns for e-business. These patterns have been built by teaming across all of the major IBM divisions. The resultant patterns are being used by IBM personnel, customers, and Business Partners to help reduce risk and increase speed to market on many e-business solution developments.

Kim Clark is an IT Specialist working in the UK and has been working in the IT industry for 13 years. He was a technical lead on some of the first implementations of the SOA Foundation products, and presents regularly on SOA design. He holds a degree in Physics from the University of London.

Yun Peng Ge is an IT Specialist with the technical sales support team supporting mainframe WebSphere® products to customers in China. He has a bachelor's degree in computer science from Fudan University in Shanghai. His area of expertise includes CICS® TS, WebSphere Message Broker, J2EE, and z/OS®. He is currently engaged in several SOA transformation projects in China.

Hatcher Jeter is a consulting IT professional with over 22 years of experience in a wide variety of computer disciplines, ranging from MVS™ on S/370™ Mainframes using SNA networks, to PCs on LAN/WAN networks, to the current Internet e-business environments of WebSphere MQ, WebSphere Message Broker on UNIX®, Intel®, and z/OS. He has had extensive training and experience on multiple operating systems such as MVS, z\OS, Intel, and UNIX and has excellent debugging and problem-solving skills. Since 1998, Hatcher has been focusing on the WebSphere MQ product, where he has developed a strong skill set. Recently, he has gained experience in WebSphere Message Broker. His present expertise is in the areas of z/OS and Distributed Platforms WebSphere MQ and WebSphere Message Broker installation, customization, problem determination, and connecting to applications (IMS™ and CICS) on z-Series mainframes.

Joao Lopes is an Application Architect working in the Service Innovation at IBM. He is part of the Architectural CoC for SOA in Brazil and has been working on some of the first implementations of SOA release products. As a Java™ developer, prior to J2EE, he continues to participate in Open Sources development in ASF and also SourgeForge.net initiatives regarding OSOA projects like SCA and SDO for Java and C++. He holds a degree in Physics.

Samir Nasser is a Certified IT Consultant in the USA. He has over 10 years of experience in the IT integration area. He holds a BS degree in Physics, a BS degree in Electronics Engineering from UNC Charlotte, and an MS degree in

Materials Science and Engineering from NC State University. His areas of expertise include SOA, J2EE, and the family of WebSphere products.

Kailash Peri is a Software Engineer in the AIM division of IBM. He is currently working in L2 support for WebSphere Message Broker and WebSphere Service Registry and Repository products. Kailash Peri has been working with IBM since 2002. He has 20 years of experience designing and developing large-scale software applications using C++ and Java. His current responsibilities include resolving customer issues related to Message Broker. Kailash holds a Bachelor's Degree in Science from Andhra University, India.

Scott Simmons, IBM Software Group, Worldwide Sales
IBM Boulder

Greg Flurry, IBM Software Group, Application and Integration Middleware
Software, SOA Advanced Technology
IBM Austin

Marc-Thomas Schmidt, IBM Software Group, Application and Integration
Middleware Software, Distinguished Engineer
IBM Somers

Rachel Reinitz, IBM Software Group, Application and Integration Middleware
Software, Distinguished Engineer
IBM Mountainview

IBM Redbooks team for *WebSphere Service Registry and
RepositoryWebSphere Service Registry and Repository Handbook*, SG24-7386,
who provided the WSRR scenarios and guidance on WSRR connectivity with
other WebSphere products

Become a published author

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

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 book form found at:

 `ibm.com`/redbooks

► Send your comments in an email to:

 redbooks@us.ibm.com

► Mail your comments to:

 IBM Corporation, International Technical Support Organization
 Dept. HYTD Mail Station P099
 2455 South Road
 Poughkeepsie, NY 12601-5400

1

Introduction

Service-oriented architecture (SOA) is a long-term goal. It takes time to mature the enterprise infrastructure into a state where its core business capabilities are exposed for re-use and yet remain flexible.

In the course of that maturing process, the enterprise service bus (ESB) at the core of the SOA is not static. It grows and changes in complexity and coverage over time. Although *enterprise service bus* is an industry standard term for an architectural pattern, the implementation will be different for every company. Their starting points will be different in terms of their existing back end systems, their current integration maturity, and the levels of autonomy between different domains of the organization. We need to be able to introduce new features to the pattern without compromising the existing implementation.

This book focuses on SOA design from the point of view of the enterprise service bus. Specifically, we look inside the ESB pattern to see what technologies are involved, and how they are combined.

We look at some key issues regarding ESBs in a maturing service-oriented architecture:

► How do you strengthen an existing ESB? How do you augment the pattern to cater for future demands? How do integration technologies combine?

► What do ESBs look like in very large enterprises where there are multiple domains of control? How do you connect multiple ESBs? What is the difference between multiple ESBs and multiple technologies within an ESB?

1

► What are the latest advances in IBM technology in relation to the ESB pattern? How should these new features and products be used? What might I expect in future releases?

Although we build on a number of IBM Redbooks publications that have come before this one, we aim to bring the technology discussions up to date with the current product versions.

This book encompasses details on significant new releases of several of the key products. Many features have been introduced to the product range specifically with SOA in mind. This book explains how these features should be incorporated into ESB patterns and demonstrates some of the new features in the practical scenarios.

The principle products in the enterprise service bus review are:

► WebSphere Enterprise Service Bus V6.0.2
► WebSphere Message Broker V6.0.0.3

We also discuss products that are significant in the augmentation of an ESB:

► WebSphere DataPower SOA AppliancesV3.6
► WebSphere Enterprise Service Bus V6.0.2
► WebSphere Partner Gateway V6.0
► WebSphere Transformation Extender V8.1
► WebSphere Transformation Extender for Message Broker V8.1

1.1 Document structure

This book is organized into three parts, as shown in Figure 1-1 on page 4. Each part has its own introduction discussing what will be found in the individual chapters.

► Part 1, "Concepts, patterns, and products" on page 7, introduces the background concepts, such as service-oriented architecture (SOA) and enterprise service bus (ESB), and provides brief descriptions of the key products discussed in this book.

► Part 2, "Product capabilities in relation to SOA and ESB" on page 51, introduces the runtime patterns of an ESB initially as a technology agnostic pattern, then mapped to specific technologies. It then introduces the issues surrounding multiple ESBs within an organization. Each of the products is addressed individually, highlighting its specific contributions to a service-oriented architecture. Finally, there is a detailed chapter comparing, contrasting, and combining ESB and the related technologies.

► Part 3, "Physical scenarios" on page 241, offers practical scenarios with step-by-step instructions showing how to use some of the key SOA-oriented features of the main products.
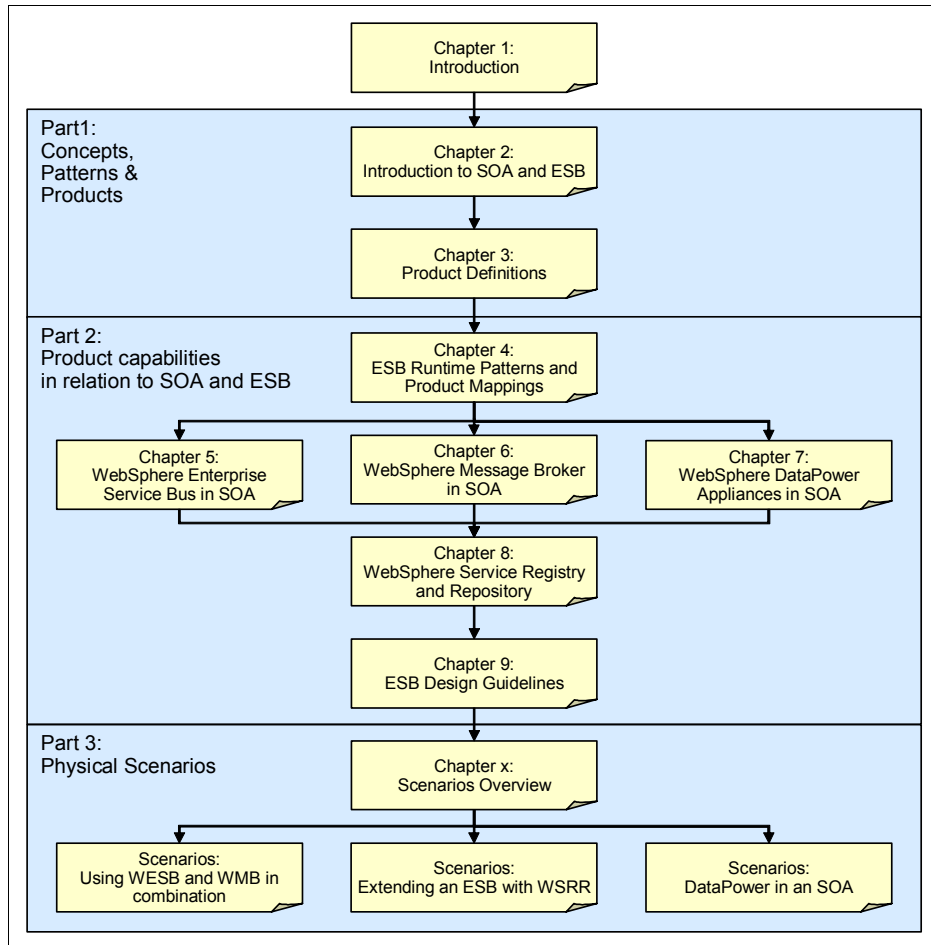
Figure 1-1 An overview of the document structure

1.2 Related IBM Redbooks publications

The following IBM Redbooks publications form the foundation on which this book is written:

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

Introduces the core patterns involved in combining enterprise service buses including scenarios with WebSphere Message Broker and WebSphere Application Server. WebSphere Enterprise Service Bus and DataPower were not available at the time.

- *Patterns: SOA Foundation Service Connectivity Scenario*, SG24-7228

  Prepares an ESB based scenario using WebSphere Enterprise Service Bus and WebSphere Message Broker, amongst other products.

- *Getting Started with WebSphere Enterprise Service Bus*, SG24-7212

  An introduction to the WebSphere Enterprise Service Bus product reviewing all of the key functions.

- *WebSphere Message BrokerWebSphere Message Broker Basics*, SG24-7137

- *Enabling SOA using WebSphere Messaging*, SG24-7163

  Comparative cover of both WebSphere Enterprise Service Bus and WebSphere Message Broker individually. Brief coverage of their combined use.

- *Patterns: Extended Enterprise SOA and Web Services*, SG24-71355

  Discusses SOA scenarios specifically in the context of Web services. WebSphere Enterprise Service Bus was not available at the time of writing.

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

  Covers the theoretical aspects of ESB in detail. Written before the emergence of many of the SOA Foundation products.

Part 1

Concepts, patterns, and products

In this part of the book we discuss concepts, patterns, and products.

2

Introduction to SOA and ESB

Since an understanding of service-oriented architecture is fundamental to this book, this chapter offers a brief explanation of the key concepts, with specific emphasis on the enterprise service bus pattern.

2.1 Service-oriented architecture overview

This section includes an overview for a service-oriented architecture (SOA). First, we define the key terms and components used to describe an SOA. Second, we review the key challenges and drivers for SOA. Third, we highlight the reasons why SOA is the right choice now. Lastly, we describe an example scenario for building a solution using an SOA approach.

2.1.1 Definition of a service-oriented architecture

Figure 2-1 highlights the key terms used to describe a service-oriented architecture.

... a service?

A **repeatable business task**. For example, check customer credit; open a new account.

... service orientation?

A way of integrating your **business as linked services** and the outcomes that they bring.

... service oriented architecture (SOA)?

An IT **architectural style** that supports service orientation.

... a composition application?

A set of **related and integrated** services that support a business process built on an SOA.

Figure 2-1   Definition of key terms for a service-oriented architecture

A *service* is representative of a repeatable business task. Services are used to encapsulate the functional units of an application by providing an interface that is well defined and implementation independent. Services can be invoked (consumed) by other services or client applications.

Service orientation defines a method of integrating business applications and processes as linked services.

Service-oriented architecture (SOA) can be different things to different people depending on the person's role and context (business, architecture, implementation, operational). From a business perspective, SOA defines a set of business services composed to capture the business design that the enterprise wants to expose internally, as well as its customers and partners. From an

architecture perspective, SOA is an architectural style that supports service orientation. At an implementation level, SOA is fulfilled using a standards-based infrastructure, programming model, and technologies such as Web services. From an operational perspective, SOA includes a set of agreements between service consumers and providers that specify the quality of service, as well as reporting on the key business and IT metrics.

A *composite application* is a set of related and integrated services that support a business process built on an SOA.

Basic components of an SOA

At the most basic level, an SOA consists of the following three components:

► Service provider
► Service consumer
► Service Registry

Each component can also act as one of the two other components. For instance, if a service provider needs additional information that it can only acquire from another service, it acts as a service consumer. Figure 2-2 shows the operations each component can perform.
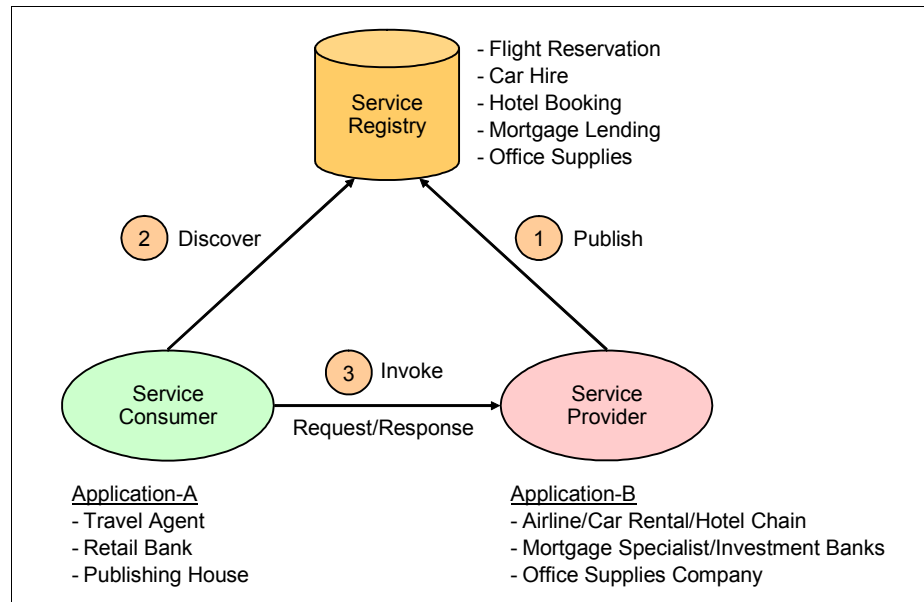
Figure 2-2   SOA components and operations

The *service provider* creates a service and in some cases publishes its interface and access information to a Service Registry.

Each provider must decide which services to expose, evaluate trade-offs between security and easy availability, and determine how to price the services or determine how to exploit the value of the services if they are free. The provider also has to decide in which category the service should be listed, and what sort of trading partner agreements are required to use the service.

The *Service Registry* is responsible for making the service interface and implementation access information available to service consumers.

The implementers of a Service Registry must consider the scope with which the registry will be implemented. For example, there are public service registries available over the Internet to an unrestricted audience, as well as private service registries that are only accessible to users within a company-wide intranet.

The *service consumer* locates (discovers) entries in the Service Registry and then binds to the service provider in order to invoke the defined service.

2.1.2 Challenges and drivers for SOA

In March 2006, IBM commissioned a Global CEO survey and found that 78% of CEOs surveyed believe that integrating business and technology is fundamental for innovation. Another key finding from this survey was that only one in ten CEOs believes his or her organization has the ability to be very responsive to changing market conditions.

As noted from the survey, businesses need the ability to integrate business and technology rapidly to achieve their business objectives. Businesses also have a strong desire to leverage the investment of existing business applications and systems without a complete and costly rewrite. There are many schemes that exist today to integrate systems within and between enterprises. In most cases these solutions are proprietary, not easily adaptable, and not responsive to rapid changes needed by the business.

There is a growing demand for an architecture and technologies that support the connection or sharing of resources and data in a very flexible and industry-standard manner. There is a need to further structure large applications into building blocks that can be reused and composed into business processes.

A shift towards a service-oriented approach standardizes the interaction with applications and business processes, and allows for more flexibility in the process. By adopting an SOA approach, existing application functionality can be turned into reusable services that can be consumed by a new set of client applications and users. SOA brings the flexibility vital to realizing innovation and desired outcomes of business.

Business requirements and drivers for SOA

Figure 2-3 highlights the common elements of a business that require flexible integration.
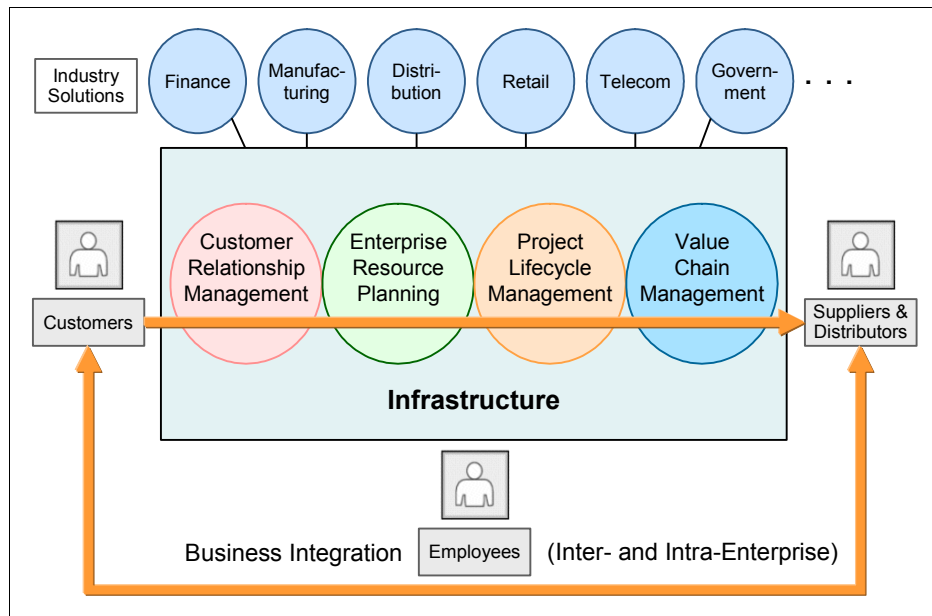
Figure 2-3   Business requirements

Here we summarize the common business drivers that require rapid and flexible integration of IT systems:

► Support an agile business model.

The marketplace can be very dynamic and competitive. There is a great need to have a business model and IT architecture that can rapidly change to support the business model and its objectives.

- Reduce cycle time and costs.

  Eliminate duplicate systems by reusing existing applications. This has the effect of reducing the time required to integrate systems, reducing cost, and simplifying the skill set required to implement the solution.

  When applying these concepts to external business processes, enterprises can move from costly manual transactions to automated transactions with suppliers.

- Simplify integration across the enterprise.

  Many existing IT systems can be inhibitors to change. They are too complex and, as a result, inflexible. Also, existing integration includes multiple technologies and point-to-point integration, which is often inflexible. The need to simplify integration is essential, especially considering the challenges raised from events such as business mergers and acquisitions.

- Achieve better IT use and return on investment.

  Return on investment (ROI) is a comparison of profit earned or lost for the investment with the amount invested. The investment in IT should facilitate the business objective and help the business achieve the targeted ROI.

Greater need for a flexible architecture

There are many possible reasons that a flexible business model is needed, such as business transformation, business process outsourcing, mergers, and acquisitions. SOA provides a flexible IT infrastructure and on demand operating environment to support the initiatives of a flexible business model.

For the purposes of comparison with SOA, we highlight the integration deficiencies of monolithic (silos) and component-based architectures. Next, we describe the flexibility gained by using an SOA approach.

Historically, business applications were built with a monolithic purpose (silos). While this kind of architecture can be effective, it is often very difficult to change and integrate with other applications within the enterprise and between enterprises (custom-coded connections required).

For example, a monolithic business application must periodically synchronize inventory information, as you can see in Figure 2-4. In this approach, pricing information for each Web order is inserted differently based on the application structure. Lastly, there is no common customer or inventory database to be shared across the enterprise, or flexibility in the business processes.
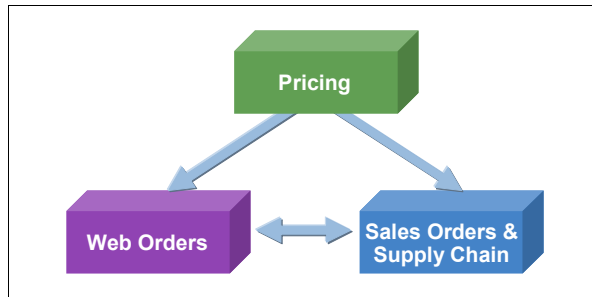
Figure 2-4   Monolithic business application (silos)

Although component-based application architecture does define services as units of business logic, there are some inherit problems with this approach. The flow of control is bound into the service logic. The transformation of data formats is also bound to the service logic. There is tight coupling between the services, as seen in Figure 2-5, thus making this application integration architecture.
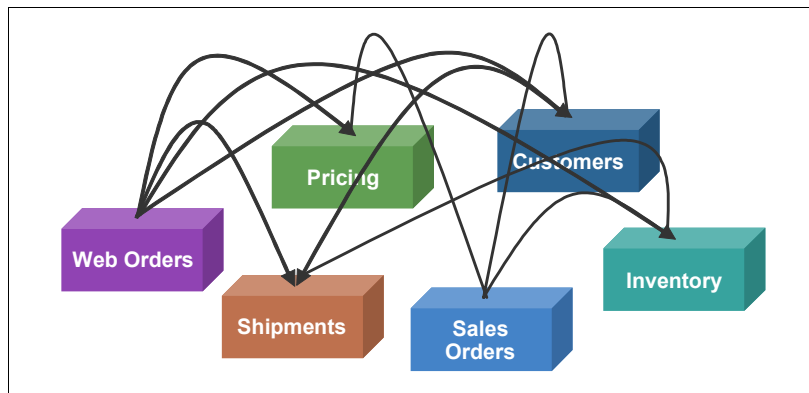
Figure 2-5   Component-based application

When using an SOA approach (see Figure 2-6), the services are defined as units of business logic separated from the flow of control and routing, and the data transformation and protocol transformation. This approach provides loose coupling, thus making this approach much more flexible for integration.
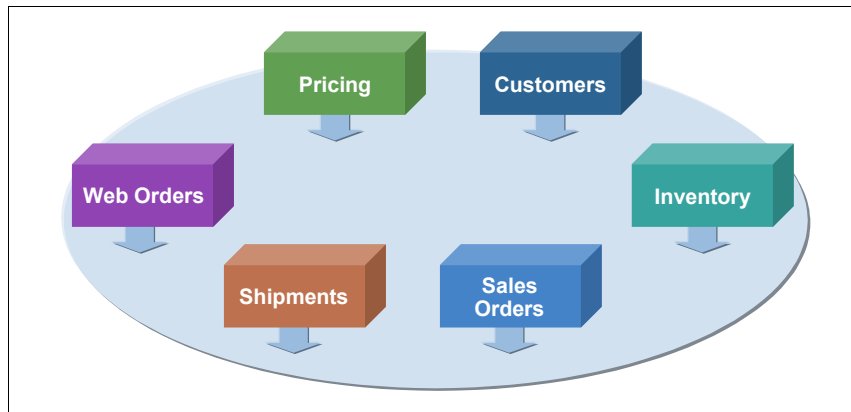
Figure 2-6   SOA-based application

2.1.3 Why SOA now

In the previous section we explained how a service-oriented architecture provides the flexibility to align you IT with your business goals. In this section we explain why SOA is the right choice now. When there is a shift in architecture, it is important to understand why a shift is needed, and evaluate the maturity level of the architecture that supports adoption.

We highlight the following key reasons why SOA is the right choice now:

► Business driving a shift in IT
► Enables flexibility of both IT and business
► Open standards and platforms
► Best practices
► Software for SOA

Business driving a shift in IT

Table 2-1 provides a summary of business needs that are driving a shift in IT from function-oriented to process-oriented and service-oriented to achieve flexibility.

Table 2-1   Shift in IT driven by business

From function-oriented	To process and service-oriented
Build for permanence.	Build to change.
One long development cycle.	Incremental development cycle.
Application silos.	Orchestrated solutions that work together.
Tightly coupled.	Loosely coupled.
Structure applications using components and objects.	Structure applications using services.
Known implementation.	Implementation abstraction.

Enables flexibility of both IT and business

SOA enables flexibility of both IT and business through flexible connectivity of business services:

► Represent applications or data as a service with a standardized interface.

► Enable applications as services to exchange structured information (messages, documents, and other business objects).

► Mediate the message exchange through an enterprise service bus (ESB).

► Provide on-ramps to the bus for existing applications and systems.

Open standards and platforms

Another key reason that SOA is the right choice for your enterprise is that it is based on open standards and platforms, as summarized in Figure 2-7. These open standards are widely adopted across the industry.

Open Development Integration Platform	**Eclipse**	IBM donated $40M of initial technology
Open Application Integration	**Web Services**	IBM led or co-led the creation of SOAP, WSDL, UDDI, WS-Security, BPEL4WS …
Open Application Server	**J2EE and Apache**	IBM contributed technology to J2EE & helped form the Apache Software Foundation
Open Operating System	**Choice Includes Linux**	IBM is the #1 commercial supporter

Figure 2-7   Summary of SOA open standards and platforms

IBM continues to be a leader in SOA-based technologies, products, and solutions. IBM is a key partner in helping define the specifications and technologies used to implement an SOA, such as Web services, Service Component Architecture (SCA), and Service Data Objects (SDO).

Best practices

Best practices are used to deliver a particular outcome by leveraging the knowledge learned from experience. Best practices include methodologies, techniques, guidelines, and patterns. By leveraging the knowledge captured in best practices listed here, your project can be run with less problems and be deployed more rapidly.

> **Note:** Many of the best practices listed in this section are described in greater detail in Chapter 4, "Best practices for SOA," in *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240.

Best practices in use today are:

► SOA adoption

   The SOA adoption process provides guidelines that assist in developing a road map towards a successful migration to an SOA.

► SOA Governance

SOA Governance helps clients extend the planned SOA across the enterprise in a controlled manner.

► Methodology

A well-established set of methodologies can help to break down complex problems into smaller and more manageable pieces that are easier to analyze and, therefore, develop solutions. Example methodologies include the Component Business Model™ (CBM), IBM Service Integration Maturity Model (SIMM), Rational® Unified Process® (RUP®), and Service-Oriented Modeling and Architecture (SOMA).

► Process modeling

Process modeling is used to define business processes. A processes flow is a sequence of tasks and decision elements with multiple branches, linked by connectors.

► Model-driven development

Model-driven development is a style of software development where the primary software artifacts are models from which code and other artifacts are generated. A model is a description of a system from a particular perspective, omitting irrelevant detail so that the characteristics of interest are more clear.

► Reference architecture

A reference architecture provides the underlying architecture components used to overcome the initial problems of finding an architecture with which to begin. The most notable reference architecture for SOA is the IBM SOA Foundation.

► Patterns

As a general principle, starting from the beginning, each time should be avoided. The use of *patterns* is one specific form of capturing and reusing reoccurring design elements. For example, the Patterns for e-business include reusable architecture and implementation assets used to accelerate the creation of a solution design and implementation.

Software for SOA

The marketplace offers many software choices for SOA. IBM is the market leader in providing mature software and solutions for SOA. For detailed information about the SOA software and solutions provided by IBM, refer to "SOA Foundation scenarios" in *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240.

2.1.4 SOA approach for building a solution

This section includes an example SOA approach for building a solution. In this example, the company wants to implement a new business process to support customers who are placing orders from an Internet Web site.

The company has existing retail, warehouse, and billing systems, as seen in Figure 2-8. The company would like to build new business processes by reusing the functionality provided by the existing systems rather than having to write new applications or new proprietary interfaces to the existing systems.
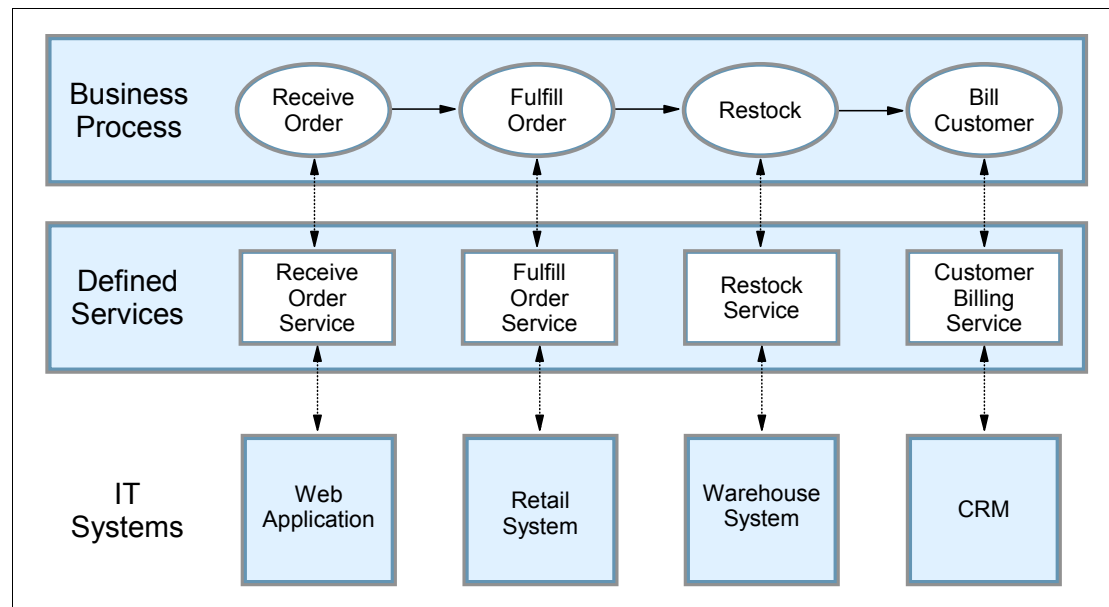
Figure 2-8   Service-oriented approach to building systems

If the company has already adopted an SOA approach, it will have defined the interfaces to its existing systems in terms of the functions or services that they offer in support of building business processes. The defined interfaces make building the new system Web front end very simple. The company simply needs to develop an application that invokes (consumes) services to complete the new business process.

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

2.2 Getting started with SOA

In this section we explore the question of how to get started with SOA from both a business and an architectural perspective.

2.2.1 SOA adoption

SOA adoption provides an iterative and incremental process, and guidelines that assist in developing a road map towards a successful migration to SOA. As seen in Figure 2-9, the SOA adoption process begins by defining the scope of possible projects that fit the criteria for being a good fit for a service-oriented architecture.
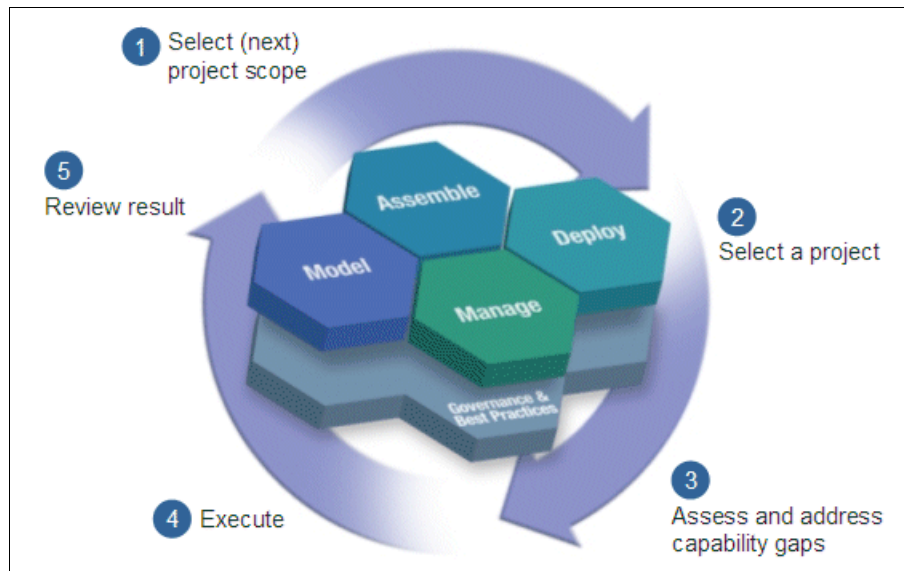
Figure 2-9   SOA adoption process

There are two primary perspectives, strategic vision and project plan. The *strategic vision* perspective describes the business and IT statement of direction, which can be used as a guideline for decision making, organizational buy-in, and standards adoption. The *project plan* perspective (or *tactical* perspective) refers to implementation projects to meet immediate needs of the current business drivers.

Defining the strategic vision starts with assessing the current business maturity across multiple dimensions including business, methodology, and technical. The IBM Service Integration Maturity Model (SIMM) can be used to help in this

assessment. If you are more comfortable with starting with a self assessment, you can use the IBM online SOA Assessment Tool:

http://www.ibm.com/software/solutions/soa/soaassessment/index.html

After the assessment has been performed, the business must establish targets for where they want to be. This includes documenting important goals and metrics for transition across the maturity dimensions. In addition, it is important to have regular checkpoints to reassess the vision.

> **Note:** Refer to "SOA Adoption" in *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240, for more detailed information.

2.2.2 IBM SOA entry points

As seen in Figure 2-10, SOA connects people, processes, and information. To help customers get started with SOA, IBM has defined three core business-centric starting points (people, processes, and information), and two IT-centric starting points (connectivity and reuse). These are known as the SOA entry points.
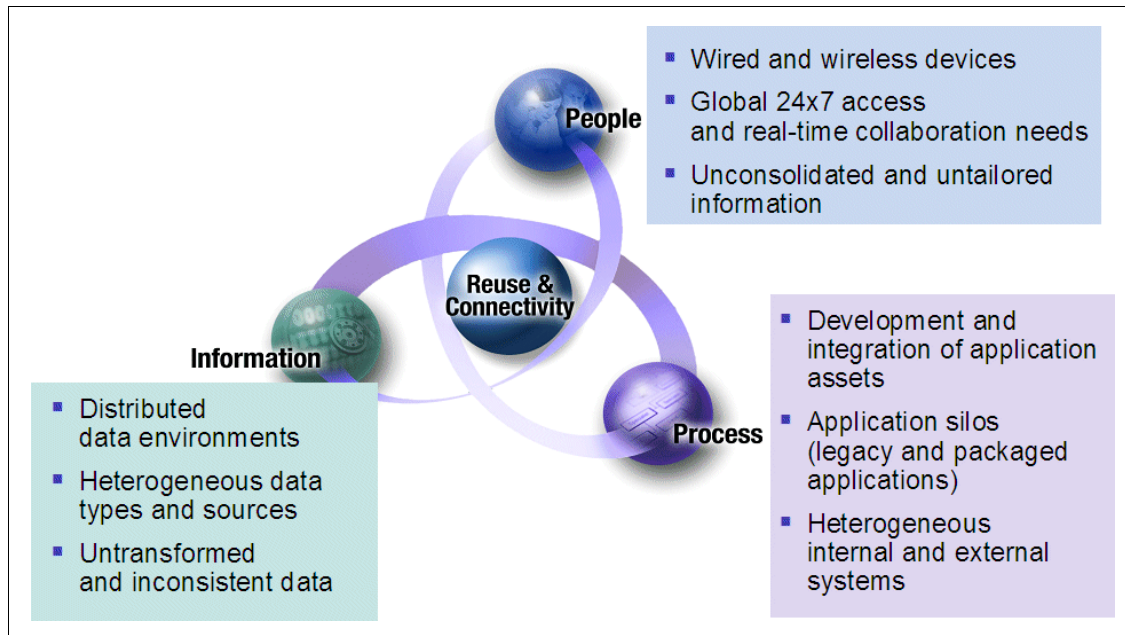
Figure 2-10   SOA connects people, processes, and information

Through business-centric SOA, companies can tie IT projects to their business needs directly by addressing the companies' immediate pain points.

► People: productivity through collaboration

Improve people productivity by aggregating views that deliver information and interaction in the context of a business process. This enables human and process interaction with consistent levels of service.

Start by building a view of a key business process by aggregating information to help people make better decisions. The next steps are tighter management of performance with alert-driven dashboards that link to more processes.

► Process: business process management for continuos innovation

Deploy innovative business models quickly with reusable and optimized processes to adapt the enterprise to changing opportunities and threats.

Start by modeling an under-performing process, remove bottlenecks, then simulate and deploy the optimized process. Next create flexible linkages between multiple processes across the enterprise and outside the firewall to suppliers and partners. Then monitor the process to measure and track performance.

► Information: delivering information as a service

Improve business insight and reduce risk with trusted information services delivered in-line and in context.

Start by discovering and understanding information sources, relationships, and the business context. The next steps are to expand the volume and scope of the information delivered as a service across internal and external processes.

The IT-centric entry points to help the enterprise integrate the business-centric SOA entry points are as follows:

► Connectivity: underlying connectivity to enable business-centric SOA

Connectivity has always been a key requirement. SOA brings new levels of flexibility. As well as acting as a building block for additional SOA initiatives, connectivity provided through SOA has distinct, standalone value.

► Reuse: creating flexible, service-based business applications

Cut costs, reduce cycle times, and expand access to core applications through reuse. Analysts estimate that it is up to five times less expensive to reuse existing applications than to write new applications.

Use portfolio management to consider which assets you need to run your company. Identify high-value existing IT assets and service-enable them for reuse. Satisfy remaining business needs by creating new services. Finally,

create a Service Registry and repository to provide centralized access and control of these reusable services.

2.2.3 IBM SOA Foundation

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

The SOA Foundation scenarios (or simply SOA scenarios) are representative of common scenarios of use of IBM products and solutions for SOA engagements. The SOA scenarios quickly communicate the business value, architecture, and IBM open standards-based software used within the SOA scenario. The concept of realizations are used to provide more specific solution patterns and IBM product mappings within the SOA scenarios.

The SOA scenarios can be used as a reference architecture implementation (starting point) to accelerate the SOA architecture and implementation of your scenario. The SOA scenarios can be implemented using an incremental SOA adoption approach, whereby a customer can incrementally add elements of other SOA scenarios to the environment to achieve their business objectives.

> **Note:** Refer to Chapter 2, "IBM SOA Foundation," in *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240, for more detailed information.

2.2.4 IBM SOA Foundation and Patterns for e-business

The IBM SOA Foundation is a reference architecture used to build new (or extend existing) applications and business processes. The IBM SOA Foundation includes an integration architecture, best practices, patterns, and SOA scenarios to help simplify the packaging and use of IBM open standards-based software.

The IBM Patterns for e-business and are a group of proven, reusable assets that can be used to increase the speed of developing and deploying On Demand business applications. 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.

Using a combined SOA process identified by IBM, both the SOA Foundation and Patterns for e-business can be used to help select the appropriate architecture

and products to build ESB solutions. WebSphere Enterprise Service Bus and WebSphere Message Broker both fit into the Service Connectivity SOA scenario.

Consult the following resources for more information:

► IBM SOA Foundation and the Service Connectivity SOA scenario

 – *Patterns: SOA Foundation Service Creation Scenario*, SG24-7240

 – *Patterns: SOA Foundation Service Connectivity Scenario*, SG24-7228

► IBM Patterns for e-business

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

2.3 Web services and SOA

This section describes the core technologies of Web services, as well as how Web services are used to implement an SOA.

> **Note:** For more detailed information about Web services, we recommend that you read *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461.

2.3.1 Web services technologies

Web services technology is a collection of standards (or emerging standards) that can be used to implement an SOA. Web services technology is vendor-neutral and platform-neutral, interoperable, and supported by many vendors today.

Web services are self-contained, modular applications, that can be described, published, located, and invoked over networks. Web services encapsulate business functions ranging from a simple request-reply to full business process interactions. The services can be new or wrap around existing applications.

Core elements

The following are the core technologies used for Web services:

► Extensible Markup Language (XML)

XML is the markup language that underlies most of the specifications used for Web services. XML is a generic language that can be used to describe the content in a structured way, separated from its presentation to a specific device.

► Simple Object Access Protocol (SOAP)

SOAP is a specification for the exchange of structured XML-based messages between the service provider, service consumer, and Service Registry, consisting of three parts:

– The format of a SOAP message is an envelope containing zero or more headers and exactly one body. The *envelope* is the top element of the XML document, providing a container for control information, the addressee of a message, and the message itself. *Headers* contain control information such as quality-of-service attributes. The *body* contains the message identification and its parameters.

– Encoding rules are used for expressing instances of application-defined data types. SOAP defines a programming language independent data type schema based on an XML Schema Descriptor (XSD), plus encoding rules for all data types defined to this model.

– RPC representation is the convention for representing remote procedure calls (RPC) and responses.

SOAP, in principle, is a protocol-independent transport. Consequently, it can potentially be used in combination with a variety of protocols such as HTTP, JMS, SMTP, or FTP. Currently, the most common way of exchanging SOAP messages is through HTTP, which is also the only protocol supported by WS-I Basic Profile 1.0.

► Web services Description Language (WSDL)

WSDL is an XML-based interface and implementation description language. A WSDL document contains the following main elements:

– *Types* is the container for data type definitions using a type system such as an XML Schema.

– An abstract, typed definition of the data being communicated, a *message* can have one or more typed parts.

– A *port type* is an abstract set of one or more operations supported by one or more ports.

– An *operation* is an abstract description of an action supported by the service that defines the input and output message and optional fault message.

– The *binding* is a concrete protocol and data format specification for a particular port type. The binding information contains the protocol name, the invocation style, a service ID, and the encoding for each operation.

– The *service* is a collection of related ports.

– The *port* is a single endpoint, an aggregation of a binding and a network address.

► Universal Description, Discovery, and Integration (UDDI)

UDDI is both a client-side API and a SOAP-based server implementation that can be used to store and retrieve information about service providers and Web services.

Standards

Figure 2-11 displays a stacked view of Web services technologies. Most of the technologies displayed have been standardized. Since interoperability is a key goal of Web services, an open industry organization known as the *Web services Interoperability Organization* (WS-I) has been created to allow interested parties such as IBM and Microsoft® to work together to maximize interoperability between Web services implementations. For more information about WS-I, visit their Web site:

http://ws-i.org

Note: Web services standards are evolving at a rapid pace. For the most current information, we recommend that you reference the Web services standards information online at sites such as IBM developerWorks®:
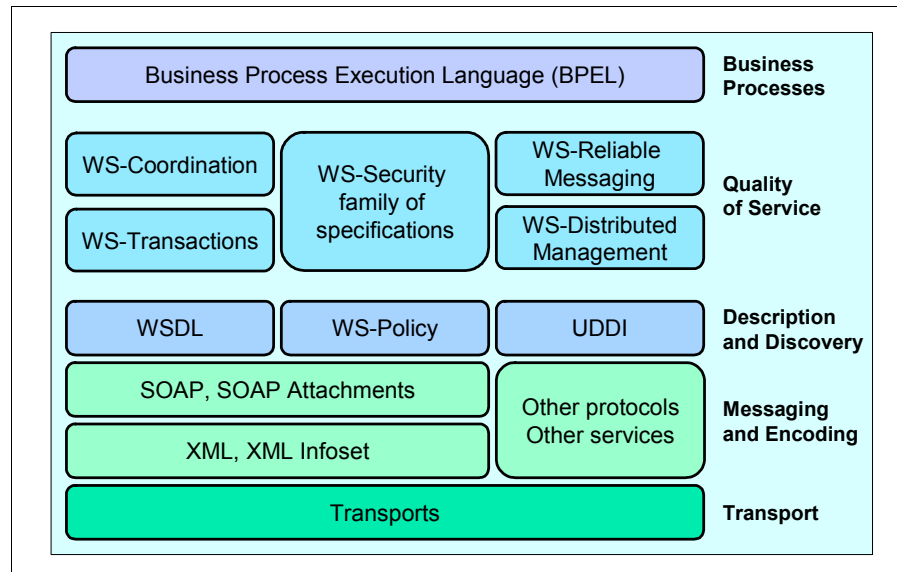
http://www.ibm.com/developerworks/webservices/standards/

Figure 2-11   Stack view of Web services technology

Web services for J2EE

Web services for J2EE V1.1 (WSEE) support is included in the J2EE V1.4 specification, which is used by WebSphere Application Server V6. The Java API for XML-based RPC (JAX-RPC) provides the programming model for SOAP-based applications by abstracting the runtime details and providing mapping services between Java and WSDL.

Exposing Web services

The port component is a fundamental part of a Web service used to define the programming model artifacts that make the Web service portable. The programming model includes:

- A *WSDL definition* provides the description of a Web service.
- The *service endpoint interface* (SEI) defines the operations of the Web service.
- A *service implementation bean* implements the SEI methods to provide the business logic of the Web service.
- The *security role references* provide instance-level security checks across different modules.

From a server programming model perspective, there are primarily two types of J2EE application artifacts exposed as Web services (service provider):

- Stateless session EJB™ (EJB container)
- JAX-RPC servlet-based service that invokes a Java Bean, known as a service endpoint (Web container)

There are three principal approaches to generating a Web service, depending on the elements that are used to start the creation of the service:

- An existing application is used to generate the Web service, which includes a service wrapper used to expose application functionality. This is known as the *bottom-up* approach.
- An existing service definition WSDL is used to generate a new application for a specific programming language and model. This is known as the *top-down* approach.
- An existing group of already generated Web services provides a new combination of functionality (multiple services). Composing a new Web service in this way might include the use of workflow technologies.

Invoking Web services

The J2EE client container provides the WSEE run time used by a Web services client application, to access and invoke Web service methods. The J2EE client container is responsible for the JNDI name to service implementation mapping.

From a client application programming perspective, there are three mechanisms used to invoke a Web service (service consumer) from the Web service client application:

► A *static stub* is created before being deployed to the runtime environment. This requires complete knowledge of the WSDL.

► A *dynamic proxy* class is created during run time. Only a partial WSDL definition is required (port type and bindings).

► A *dynamic invocation* interface does not require WSDL knowledge. The signature or service name is unknown until run time.

The task to build or generate a Web service client (service consumer) depends on the methods of how the client is binding to a Web service server. The client uses a local service stub or proxy to access the remote server and service. The WSDL document is used to generate or set up the particular stub or proxy. The stub or proxy knows at request time how to invoke the Web service based on the binding information.

2.3.2 Web services and SOA

Web services technology is a collection of standards (or emerging standards) that can be used to implement a service-oriented architecture (SOA). That is not to say that Web services and SOA are intrinsically linked, because they can be implemented separately.

In fact, many significant SOAs are proprietary or customized implementations that are based on reliable messaging and Enterprise Application Integration middleware (for example, IBM WebSphere MQ and IBM WebSphere Business Integration Message Broker) do not use Web services technologies. Also, most existing Web services implementations consist of point-to-point integrations that address a limited set of business functions between a defined set of cooperating partners.

The logical links between Web services and SOA are:

► Web services provide an open-standard model for creating explicit, implementation-independent descriptions of service interfaces.

► Web services provide communication mechanisms that are location-transparent and interoperable.

Web services are evolving, through Business Process Execution Language for Web services (WS-BPEL), document-style SOAP, and Web services Definition Language (WSDL), to support the implementation of well-designed services that encapsulate and model reusable function in a flexible manner.

2.4 The enterprise service bus

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

An ESB runtime pattern has been identified and detailed in 4.1.1, "ESB runtime pattern" on page 54.

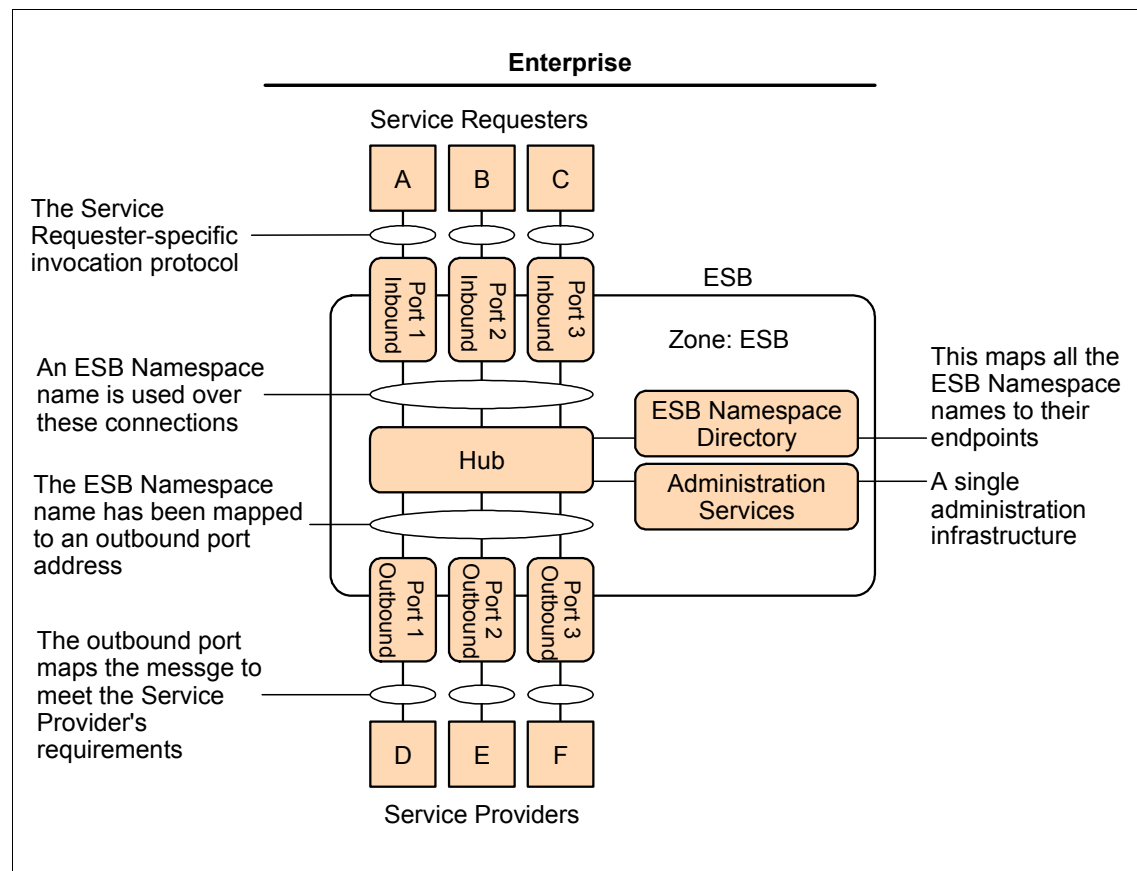Figure 2-12 illustrates the IBM Patterns for e-business ESB Pattern.

Figure 2-12   ESB Pattern

There are a few key differences between this design pattern and previous design patterns (for example, hub-and-spoke). The ESB design pattern has specific components that are not a part of previous integration design patterns. For example, most EAI design patterns did not show a namespace directory. A namespace could be simply described as a prefix on an object's name to make it unique to a particular domain of use. Two objects can have the same name as long as their name spaces are different. It is also important to note that an ESB is under the control of a single administrative services infrastructure.

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

In Figure 2-13 the ESB is depicted as a logical component in a service-oriented architecture. It acts as the mediator between the service consumers and service providers. The service providers and service consumers never interact directly. They always use the ESB as a mediator. The ESB provides services to resolve differences in protocol and format, and decouples the service consumer from the service provider.
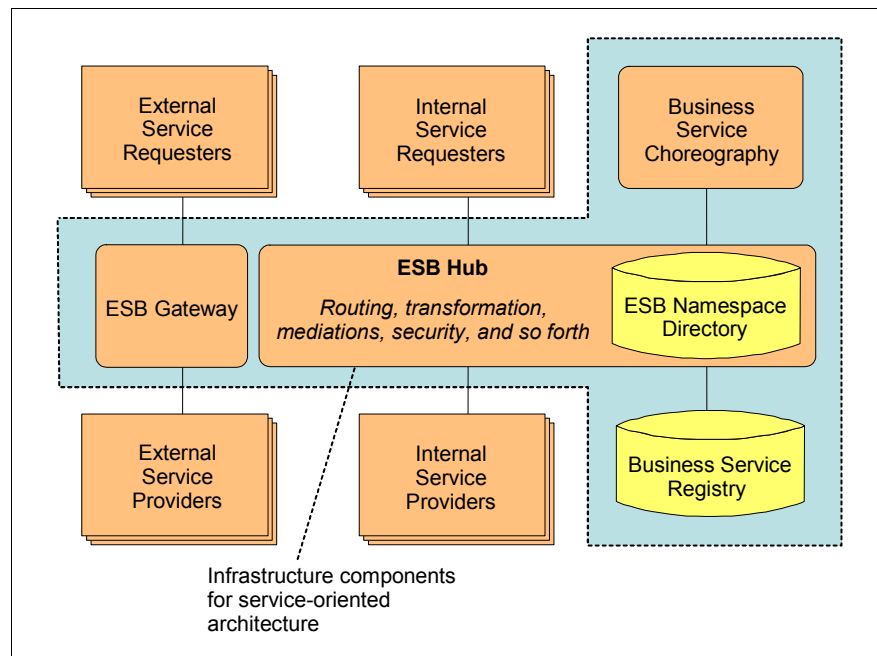

Figure 2-13   ESB and SOA

The ESB Hub is a software package. IBM currently offers two ESB products that serve two different markets. WebSphere ESB is built on proven messaging and Web services technologies, and it provides standards-based Web services connectivity and XML data transformation. WebSphere Message Broker is an Exposed ESB Gateway product that provides universal connectivity (including Web services) and any-to-any data transformation. See Figure 2-14.
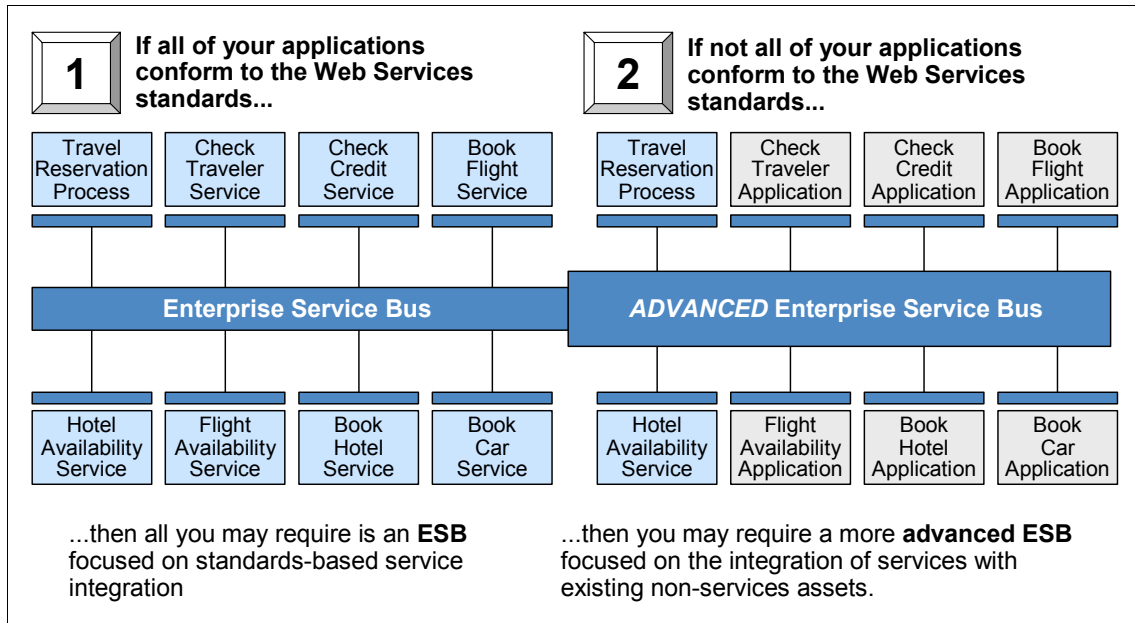

1 If all of your applications conform to the Web Services standards...

| Travel Reservation Process | Check Traveler Service | Check Credit Service | Book Flight Service |

Enterprise Service Bus

| Hotel Availability Service | Flight Availability Service | Book Hotel Service | Book Car Service |

...then all you may require is an **ESB** focused on standards-based service integration

2 If not all of your applications conform to the Web Services standards...

| Travel Reservation Process | Check Traveler Application | Check Credit Application | Book Flight Application |

ADVANCED **Enterprise Service Bus**

| Hotel Availability Service | Flight Availability Application | Book Hotel Application | Book Car Application |

...then you may require a more **advanced ESB** focused on the integration of services with existing non-services assets.

Figure 2-14   IBM ESB products

> **Tip:** More information about IBM ESB can be found here:
>
> http://www-306.ibm.com/software/info/middleware/applications/index.jsp

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

> **Tip:** Additional reading material on ESB patterns and products, and how they fit into a service-oriented architecture can be found at:
>
> http://www-128.ibm.com/developerworks/architecture/application.html

2.4.1 The role of an enterprise service bus

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

An ESB supports the concepts of SOA implementation by:

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

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

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

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

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

► How service interactions are secured
► How the integrity of business transactions and data are maintained (for example, through reliable messaging, the use of transaction monitors, or compensation techniques)

> ► How the invocation of alternative service providers is handled in the event that the default provider is unavailable

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

- ► Map service requests from one protocol and address to another.
- ► Transform data formats.
- ► Support a variety of security and transactional models between service consumers and service providers and recognize that consumers and providers might support or require different models.
- ► Aggregate or disaggregate service requests and responses.
- ► Support communication protocols between multiple platforms with appropriate qualities of service.

Provide messaging capabilities such as message correlation and publish/subscribe to support different messaging models such as events and asynchronous request/response.

2.5 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.

2.5.1 ESB capabilities

The following sections describe various characteristics of an ESB.

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 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.

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.

2.5.2 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 2-2.

Table 2-2   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?

Attribute	Description
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 2-2 on page 37 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 2.5, "ESB capabilities and decision attributes" on page 34.

► 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, while 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.

3

Product descriptions

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

► IBM WebSphere Enterprise Service Bus V6

► IBM WebSphere Message Broker V6

► IBM WebSphere MQ V6

► IBM WebSphere DataPower

► IBM WebSphere Service Registry and Repository

► IBM WebSphere Adapters

► IBM WebSphere Partner Gateway V6.0

► IBM WebSphere Transformation Extender

► IBM WebSphere Process Server

► IBM Tivoli® Federated Identify Management (FIM) and Tivoli Access Manager

► IBM Tivoli Composite Application Manager for SOA

41

3.1 Primary products discussed in this book

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

3.1.1 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

We should mention that WebSphere Integration Developer (WID) is a common tool for building SOA-based integration solutions across WebSphere Process Server, WebSphere ESB, and WebSphere Adapters.

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

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

3.1.2 IBM WebSphere Message Broker V6

WebSphere Message Broker incorporates WebSphere Event Broker and extends its function to provide a message broker solution driven by business rules. Messages are formed, routed, and transformed according to the rules defined by an easy-to-use graphical user interface (GUI).

Diverse applications can exchange information in dissimilar forms, with brokers handling the processing required for the information to arrive in the right place in the correct format, according to the rules that you have defined. The applications do not need to know anything except their own conventions and requirements.

Applications also have much greater flexibility in selecting which messages they want to receive, because they can specify a topic filter or a content-based filter, or both, to control the messages that are made available to them.

WebSphere Message Broker provides a framework that supports supplied, basic functions along with user-defined enhancements to enable rapid construction and modification of business processing rules that are applied to messages in the system.

You can find more information at:

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

3.1.3 IBM WebSphere MQ V6.0

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

3.1.4 DataPower

IBM WebSphere DataPower SOA Appliancesrepresent an important element in the holistic IBM approach to service-oriented architecture (SOA). IBM SOA appliances are purpose-built, easy-to-deploy network devices that simplify, help secure, and accelerate your XML and Web services deployments while extending your SOA infrastructure. These new appliances offer an innovative, pragmatic approach to harness the power of SOA while simultaneously enabling you to leverage the value of your existing application, security, and networking infrastructure investments.

► WebSphere DataPower Integration Appliance XI50
► WebSphere DataPower XML Security Gateway XS40
► WebSphere DataPower XML Accelerator XA35

http://www-306.ibm.com/software/integration/datapower/index.html?S_TACT=102A9W01&S_CMP=campaign

3.1.5 WebSphere Service Registry and Repository

WebSphere Service Registry and Repository is an industrial-strength tool that helps you achieve more business value from your SOA by enabling better management and governance of your services. Through its robust registry and repository capabilities and its tight integration with IBM SOA Foundation, WebSphere Service Registry and Repository can be an essential foundational component of your SOA implementation.

The WebSphere Service Registry and Repository system enables you to store, access, and manage information about the services (commonly referred to as service metadata) in your SOA. This information is used to select, invoke, govern, and reuse services as part of a successful SOA.

You can use WebSphere Service Registry and Repository to store information about services in your systems, or in other organizations' systems, that you already use, that you plan to use, or that you want to be aware of. For example, an application can check with WebSphere Service Registry and Repository just before it invokes a service to locate the most appropriate service that satisfies its functional and performance needs. This capability helps make your SOA deployment more dynamic and more adaptable to changing business conditions.

You can find more information about WebSphere Service Registry and Repository at:

http://www-306.ibm.com/common/ssi/rep_ca/0/897/ENUS206-230/ENUS206-230.PDF

http://www.redbooks.ibm.com/redpieces/abstracts/sg247386.html?Open

3.1.6 WebSphere Adapters

IBM WebSphere Adapters allow you to quickly and easily create integrated processes that exchange information between Enterprise Resource Planning, Human Resources, Customer Relationship Management, and supply chain systems. The adapters service-enable your applications by connecting them to the Enterprise Service Bus, which powers your service-oriented architecture:

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

3.1.7 WebSphere Partner Gateway

WebSphere Partner Gateway V6.0 extends process integration beyond the enterprise, providing a consolidated business-to-business (B2B) gateway to lower the costs of B2B integration. It integrates external processes and partner communities with internal processes and infrastructures, combining extensive partner profile management capabilities with a simple, reliable, and secure exchange for B2B messages.

WebSphere Partner Gateway is the re-branded, follow-on version to WebSphere Business Integration Connect V4.2, carrying forward all of its capabilities.

WebSphere Partner Gateway enables a single unified B2B partner management environment that supports traditional EDI and XML-based message data and protocols, such as AS1, AS2, and RosettaNet, for B2B integration.

Transforming your business with WebSphere Partner Gateway lets your business become more responsive to demand. It becomes an enterprise whose business processes are integrated end-to-end across the company and with key partners, suppliers, and customers so that they can respond rapidly to changes in customer demand, market opportunity, or external threat. For more information see:

http://www-306.ibm.com/common/ssi/rep_ca/8/897/ENUS205-158/ENUS205-158.PDF

3.1.8 WebSphere Transformation Extender for Message Broker

WebSphere Transformation Extender is a leading universal transformation engine, ready to be deployed in any infrastructure to integrate data across the enterprise. This transaction-oriented solution automates the transformation and validation of high-volume and complex data without the need for hand coding.

WebSphere Transformation Extender is the *develop once, deploy anywhere* solution for data transformation.

WebSphere Transformation Extender delivers:

► Consistent data transformation across the enterprise, independent of data structure, data location, infrastructure, and operating environment

► Reduced application development and maintenance costs with increased application deployment speed by reusing transformation assets

► Increased application quality by working in a code-free environment for transformation and validation of highly complex data

► Faster standards compliance and improved data quality with automated data validation using industry and regulatory standards

► Multiple execution options to support right-time, right-style transformation — batch, real time, or embedded

► Standards-based transaction support for unique industry transformation requirements such as X-12, EDIFACT, HIPAA, HL7, SWIFT, and NCPDP

For more information see:

http://www-306.ibm.com/software/integration/wdatastagetx/index.html

3.2 Related products

In this section we mention those products related to our primary products that are used for runtime functionality.

3.2.1 WebSphere Process Server

WebSphere Process Server is at the very heart of your business process management solutions. It ensures that the processes you design in WebSphere Business Modeler or WebSphere Integration Developer are executed consistently, reliably, securely, and with transactional integrity. Built on open standards, it deploys and executes processes that orchestrate services (people, information, systems, and trading partners) within your SOA or non-SOA infrastructure. When combined with the power of WebSphere Business Monitor, processes can be optimized to meet changing business requirements, giving the business a competitive advantage. WebSphere Process Server is built upon, and contains, the WebSphere ESB functionality. For more information see:

http://www-306.ibm.com/common/ssi/rep_ca/4/897/ENUS206-244/ENUS206-244.PDF

http://www-306.ibm.com/common/ssi/rep_ca/3/897/ENUS206-243/ENUS206-243.PDF

3.2.2 TFIM/TAM

IBM Tivoli Federated Identity Management (FIM) provides a simple, loosely coupled model for managing identity and access to resources that span companies or security domains. Rather than replicate identity and security administration at both companies, Tivoli Federated Identity Manager provides a simple model for managing identities and providing them with access to information and services in a trusted fashion. For companies deploying service-oriented architecture and Web services, FIM provides policy-based integrated security management for federated Web services. The foundation of FIM is trust, integrity, and privacy of data.

On this foundation, organizations can share identity and policy data about users and services. The sharing of trusted identities and policies is the key to delivering a richer experience for users navigating between federation sites. Trust enables companies to loosely couple their disparate identity management systems.

A federated model simplifies administration and enables companies to extend identity and access management to third-party users and third-party services. For more information see:

http://www-306.ibm.com/software/tivoli/products/federated-identity-mgr/

IBM Tivoli Access Manager for Business Integration is a multi-platform security management solution for IBM WebSphere MQ that upgrades the native security services of IBM WebSphere MQ to those provided by IBM WebSphere MQ Extended Security Edition. It provides application-level data protection for

WebSphere MQ-based applications, without the need to modify or even recompile them.

Application-level data protection differs from link level or channel level data protection in that the integrity and confidentiality of messages can be demonstrated, not just while messages are in transit from system to system, but also while they were under the control of WebSphere MQ itself (for example, resident in a queue). This is critical for customers using WebSphere MQ to process personally identifiable information or other types of sensitive data, such as high value financial transactions.

IBM Tivoli Access Manager for Business Integration Host Edition provides extended security services for mainframe applications using WebSphere MQ for z/OS. Unless noted otherwise, equivalent features are available on both the mainframe and non-mainframe environments.

http://www-306.ibm.com/software/tivoli/products/access-mgr-bus-integration/

3.2.3 IT CAM for SOA

IBM Tivoli Composite Application Manager for SOA (ITCAM for SOA) will monitor, manage, and control SOAs deployed using a wide range of IBM and third-party systems, helping you to:

► Proactively recognize and quickly isolate Web service performance problems.

► Verify that Web services are available and performing to specification.

► Alert you when Web service performance is degraded.

► Perform automated service mediation (for example, to reject or re-route certain requests during periods of heavy load).

► Report results against committed service levels.

► Visualize service flows, end-to-end, as they cross the enterprise.

► Pinpoint source of service bottlenecks.

► Understand the impact of service problems on business processes.

ITCAM for SOA includes the Web services Navigator, a plug-in to IBM Rational and other Eclipse-based tools, which provides a deep understanding of service flows, patterns, and relationships to developers and architects using operational data from Tivoli Data Warehouse. Eclipse is an award-winning, open source platform for the construction of powerful software development tools and rich desktop applications.

ITCAM for SOA is a core component of the IBM SOA Foundation, an integrated and open set of software, best practices, patterns, and skills resources to get you started with service-oriented architectures. Visit the IBM SOA Web site for more details.

ITCAM for SOA is an integral part of the IBM IT Service Management (ITSM) solutions that are designed to help deliver services based upon a framework of best practices. For more details see:

http://www-306.ibm.com/software/tivoli/products/composite-application-mgr -soa/

Part 2

Product capabilities in relation to SOA and ESB

In this part we discuss each of the core ESB technologies. We discuss the specific capabilities that they have in a service-oriented architecture, how they relate to one another, and how they relate to other technologies within the suite.

This part contains:

- ► Chapter 4, "ESB runtime patterns and product mappings" on page 53, discusses the principle topologies of the ESB pattern and maps out the placement of the key products that are discussed throughout the remainder of the document.

- ► Chapter 5, "WebSphere Enterprise Service Bus" on page 81, introduces WebSphere Enterprise Service Bus, the J2EE-based enterprise service bus hub.

- ► Chapter 6, "WebSphere Message Broker in SOA" on page 105, introduces WebSphere Message Broker, the advanced enterprise service bus hub with extensive integration capabilities.

- ► Chapter 7, "WebSphere DataPower appliances in SOA" on page 165, introduces the WebSphere DataPower appliances, discussing where in the enterprise service bus pattern they can be used to greatest effect.

- ► Chapter 8, "ESB design options" on page 181, assesses various configurations of the above products.

4

ESB runtime patterns and product mappings

This chapter looks at the topology of simple and advanced ESB configurations, mapping the integration requirements to actual products. Specifically, it addresses the following:

► Typical topologies and product mappings for a single ESB with a range of integration requirements

► Complex topologies where there are multiple ESBs present within an organization

4.1 ESB runtime topologies

Initially, we discuss the makeup of an individual ESB. It is important to remember here that an ESB is first and foremost an architectural pattern. An ESB may be spread across several nodes within the physical topology for clustering reasons, and there may be more than one product required to fulfil all integration needs.

We start with the simplest possible ESB configuration, and then gradually build up to a more complex ESB configuration based on typical requirements.

4.1.1 ESB runtime pattern

Earlier, we discussed the purpose of an ESB in 2.4, "The enterprise service bus" on page 30. Now lets take a more detailed at the ESB pattern itself.

Figure 4-1 shows the core logical components within an ESB, and Figure 4-2 on page 60 shows an appropriate product mapping.
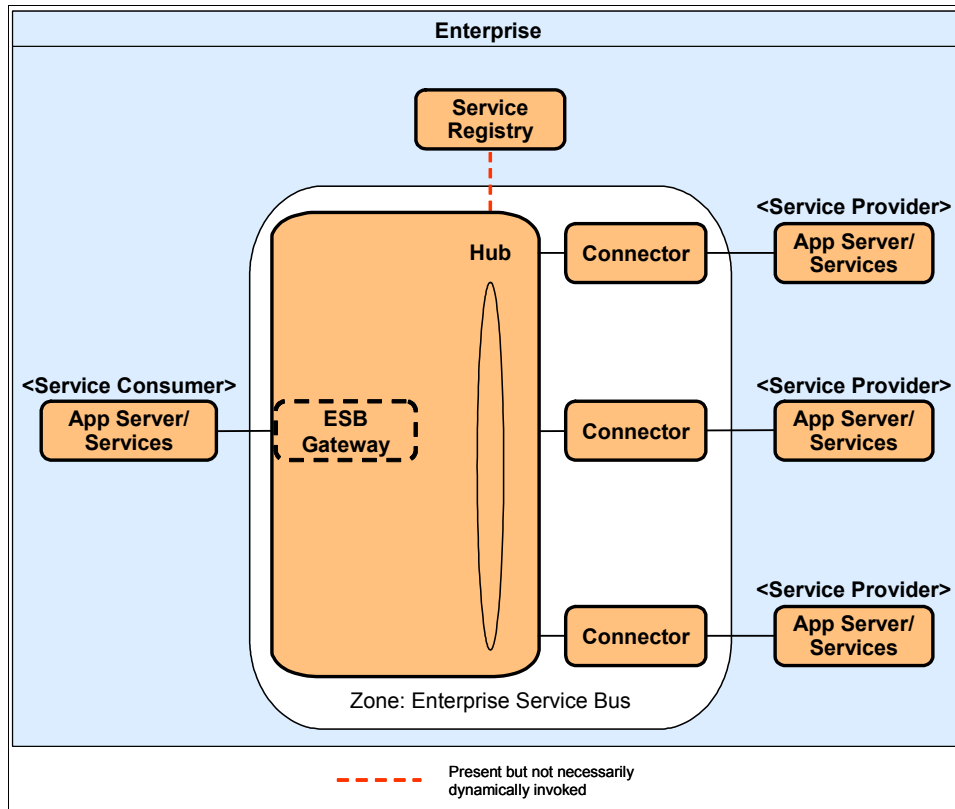
Figure 4-1   ESB runtime pattern level 2

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.

Hub node

This node supports the key ESB functions and, therefore, fulfills a large part of the ESB capabilities, and is described in some detail here. The node has a fundamental service integration role and should be able to support various styles

of interaction. Every ESB Zone requires an implicit or explicit ESB Gateway where aspect-oriented services are applied (for example, logging, security, and so on). The precise deliniation between the ESB Gateway and the Hub is described in more detail in 4.1.3, "Exposed ESB Gateway composite pattern" on page 62. For this simple topology the Hub node is assumed to include this ESB Gateway role.

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

► Routing and brokerage

This function removes the need for applications to know anything about the bus topology or its traversal. If the request that a consumer initiates is sent to one provider, then it is classed as routing. If there is more than one recipient, then it is performing the broker pattern.

► Namespace translation

A namespace in this context is defined as a collection of data representations specific to a particular domain.

Services will be exposed using a namespace or data model that is appropriate at the enterprise level, aiming for a consistent representation of data across services. This data model should be arrived at based on top-down analysis of the enterprise's core business functions, and will almost certainly be different from the namespaces of the back-end systems.

A mapping or translation between these two different data representations is performed by the ESB. This de-coupling is critical to allow implementations of services to change over time.

It is particularly important to differentiate namespace translation, which is the mapping between two different logical data models, from data parsing, where data is extracted from data encodings or data formats such as fixed width, delimited, EBCDIC, and XML. This data parsing has nothing to do with namespace translation, and although it might be provided by the Hub node for some cases, it may often be pushed down into the connector adapter nodes.

► Service virtualization

Service virtualization complements routing to provide location transparency and supports service substitution. Service endpoint addresses (the addresses of the actual provider of the service) are transparent to the service consumer and can be transformed by the node. The node obtains the service endpoint address from the Service Registry.

► Messaging styles

The node should support an appropriate variety of messaging styles. The most common are request/response, fire and forget, events, publish/subscribe, and so on.

► Transport protocols

The node should support at least one transport that is or can be made widely available, such as HTTP/S. The node can provide protocol transformation. If a protocol transformation is required that is not supported by the node, then a specific connector can be used to perform the transformation.

► Service interface definition

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

► Service messaging model

The node should support at least one model, preferably using common standards such as SOAP/XML.

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

► Integration

Additional integration services that may be provided include service mapping 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 node 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 node 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)

▶ Administration

An ESB should be controlled by a single administration infrastructure. This is not shown separately on the diagram, but is typically separated from the runtime ESB node itself.

This node provides administration services that, at a minimum, should support service addressing and naming. Other core capabilities of this node are ESB configuration, service provisioning and registration, .ogging, 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 node should support security capabilities such as authentication, authorization, non-repudiation, confidentiality, and security standards, such as Kerberos and WS-Security.

Service Registry

The role of the Service Registry is to provide details of services that are available to perform business functions identified within a taxonomy. In its simplest form, the ESB Hub contains its own repository. This makes sense to a limited extent since the ESB is the central point through which all services are invoked.

However, it quickly becomes apparent that there are much wider uses for a registry that extend far beyond the reach of the ESB. For example, it is valuable to trace, and indeed govern, the life cycle of services from their inception rather than only have knowledge of them once they have reached production.

It is therefore relevant to separate this capability from the ESB.

From the ESBs point of view, a Service Registry could provide the following capabilities:

▶ Uploading of service metadata (for example, publishing of services so that other clients may make use of them)

- ► Access to the registry at development time for data such as service interfaces and endpoints
- ► Searchable access to the registry at run time for information such as dynamic endpoints and policy
- ► Enables the components within the ESB pattern to dynamically respond to changes in the desired use and administration of services

The link to the Hub node from the Service Registry is dotted to show that dynamic lookup in the Service Registry at run time is still the exception rather than the rule.

The importance of the registry becomes even more apparent when the service infrastructure progresses to contain more than one integration product and more than one ESB (4.2, "Multiple ESBs within an organization" on page 65).

Connectors

Connectors can be separated into two different types: *path connectors* and *Adapter Connectors*.

Path connectors specify a protocol, data representation, and transport mechanism used to connect two systems together. The assumption here is that both sides are using the same protocol (for example, HTTP) and have agreed on the same wire format (for example, XML), and both have access to the same transport mechanism. A transport mechanism could be as wide as the Internet, or a simple as an area of shared storage.

In some cases, nothing more than a *path connector* is needed to connect between two systems since they share the same protocol, data format, and transport mechanism. An example would be two systems connecting via XML/HTTP over the Internet or an intranet.

Often in pattern diagrams, *path connectors* are not shown explicitly as a node, and are simply represented by a link optionally showing the protocol and format.

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).

The *Adapter Connector* is responsible for hiding the complexities of connectivity, protocol, and back-end data representation. Any Application Programming Interface (API) required to talk to the system, and complexities such as connection pooling, flow control, batching of transactions, *keep-alive* heartbeat signals, and status health checks should be encapsulated in the Adapter Connector.

Any parsing of incoming data or formatting of outgoing data should be taken care of by the *Adapter Connector* so that business events are presented to the hub in a standard form, immediately usable by the tooling of the hub. Equally, the enterprise service bus can pass business data to the adapter without understanding the subtleties of the back end system.

The Adapter Connector is not, however, aware of how the data will be moved — in other words what the transport mechanism will be — and this is where the *path connector* comes in, and often an *Adapter Connector* implicitly includes a *path connector*.

4.1.2 ESB runtime pattern product mapping

Figure 4-2 shows an example mapping of products to the ESB runtime pattern.
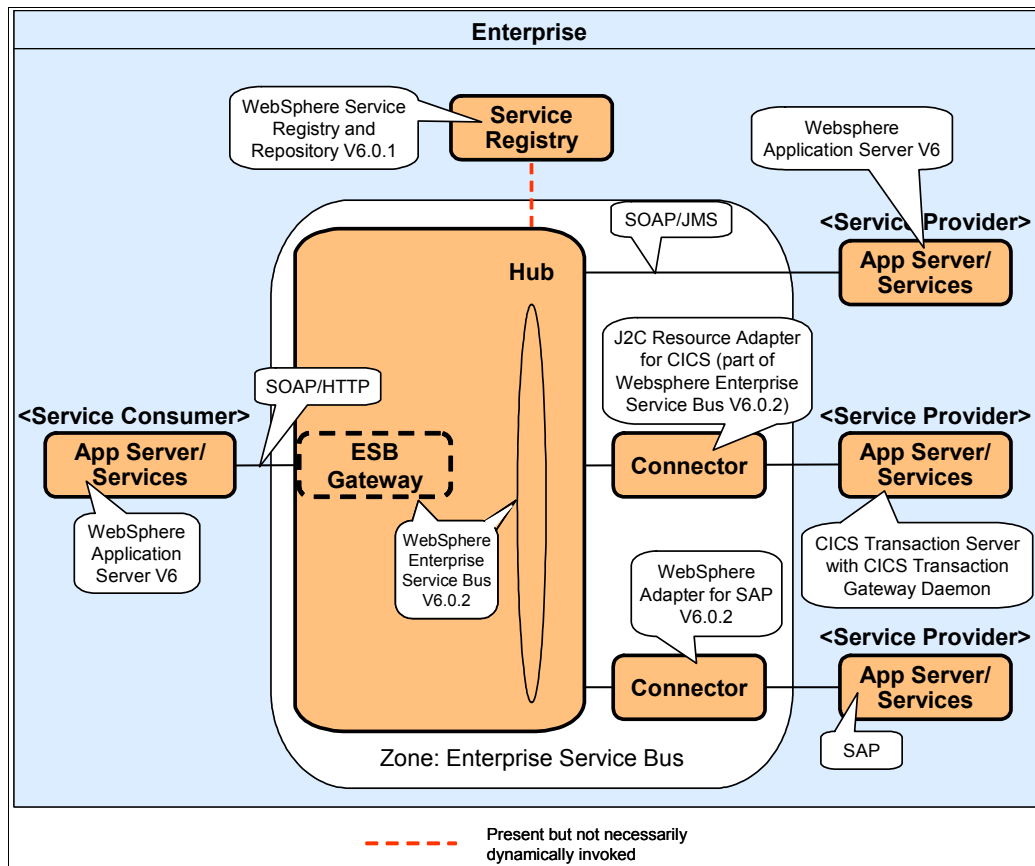
Figure 4-2   ESB runtime pattern level 2 product mappings

App server/services node

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.

This node also applies to J2EE application servers such as WebSphere Application Server, which provide an enterprise grade container service implementation.

Hub node

WebSphere Enterprise Service Bus is targeted specifically at the capabilities of this node, and exhibits several of the capabilities noted above. In particular, it provides an implicit ESB Gateway capability within the product. The product has a strong focus on providing a clear mechanism for exposing SOA services. It provides visual tooling for the key aspects of tasks such as routing, mapping, interface definition, and discovery. Transport protocols, quality of service, and many other administration functions can all be configured rather than resorting to code. More information about WebSphere Enterprise Service Bus can be found in Chapter 5, "WebSphere Enterprise Service Bus" on page 81.

The product mapping shows WebSphere Enterprise Service Bus, but WebSphere Message Broker is an equally valid choice here, and would be chosen over WebSphere Enterprise Service Bus when advanced connectivity and performance characteristics are required. The key features of WebSphere Message Broker are discussed in Chapter 6, "WebSphere Message Broker in SOA" on page 105.

Service Registry

WebSphere Service Registry and Repository is designed specifically to address the needs of a registry for a service-oriented architecture. It addresses a variety of mechanisms for uploading, managing, and retrieving metadata regarding enterprise services. A more detailed discussion on the capabilities of WebSphere Service Registry and Repository is given in *WebSphere Service Registry and Repository* Handbook, SG24-7386.

Adapter Connectors

A suite of WebSphere Adapters is available covering standard technologies and packages. This book focuses on connectivity to back-end systems. More details on various adapters can be found in Chapter 8, "ESB design options" on page 181.

4.1.3 Exposed ESB Gateway composite pattern

In 4.1.1, "ESB runtime pattern" on page 54, we described the minimum topology required to provide an ESB. Now we look at how this basic topology could be extended to provide services across a larger, more complex enterprise, or indeed beyond the walls of the enterprise itself.

Figure 4-3 shows the topology of an Exposed ESB Gateway composite pattern, which adds access to and from external parties over the basic pattern.
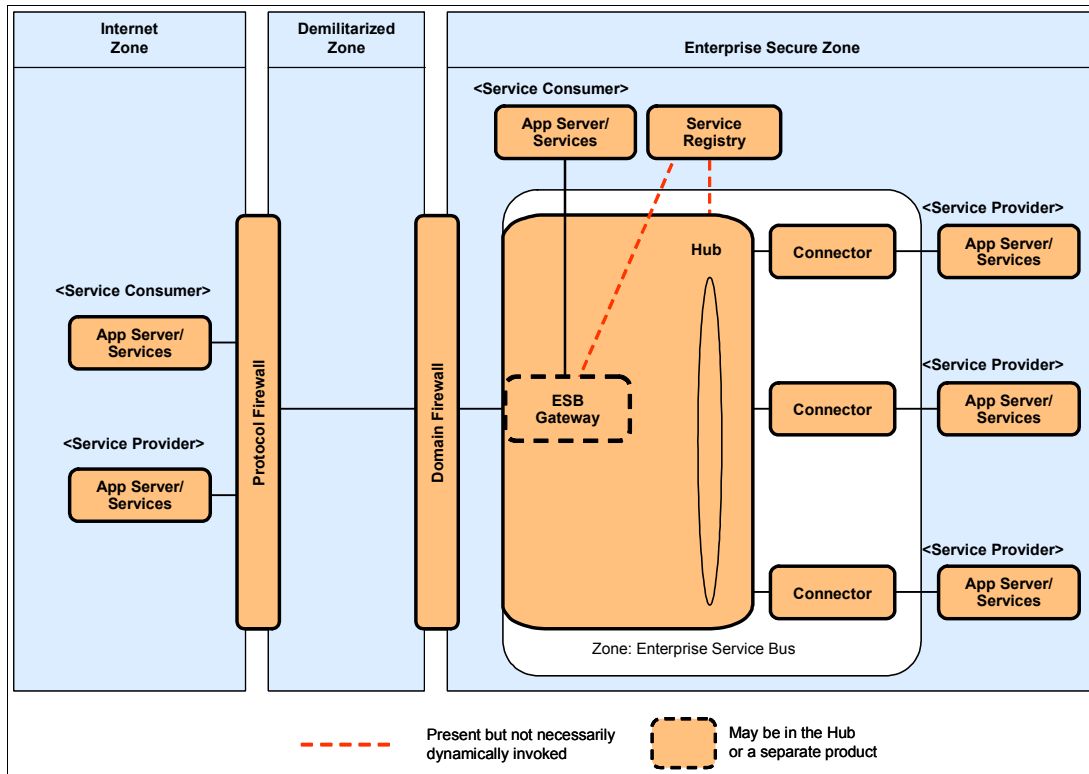
Figure 4-3   Exposed ESB Gateway composite pattern

Most of the nodes in Figure 4-3 have already been described in 4.1.1, "ESB runtime pattern" on page 54. Only new nodes, or nodes that are performing an extended function, are described below.

Hub

The Hub node in this Exposed ESB Gateway composite pattern may have a more extensive integration role, having a richer set of integration capabilities,

including built-in capabilities to understand industry standard data meta models, and sophisticated data parsing capabilities.

ESB Gateway

The ESB Gateway functions in the ESB runtime pattern were handled by the ESB Hub node. In the Exposed ESB Gateway composite pattern, it may be valuable to separate out this functionality into a separate node for reasons of security, and new options for scalability.

The ESB Gateway acts as a proxy to provide controlled access to the ESB. A principal use of the ESB Gateway is to expose services in a consistent fashion. This node allows generic actions to be defined and performed on all calls to services such as logging, auditing, monitoring, security, and threat protection.

It is likely that in large enterprises the benefits of this protection and consistency may well be just as valid even within the enterprise. As services start to be used by other domains of the organization that have different governance, it will be appropriate to treat these domains as external parties. In this situation service consumers within the organization will also call services via the ESB Gateway.

The ESB Gateway talks to the Service Registry at run time to gather information about the location of appropriate implementations of services, and also for policy to be applied to the service requests in terms of security and auditing, for example.

4.1.4 Exposed ESB Gateway product mapping

Figure 4-4 shows an example product mapping for the Exposed ESB Gateway composite pattern.
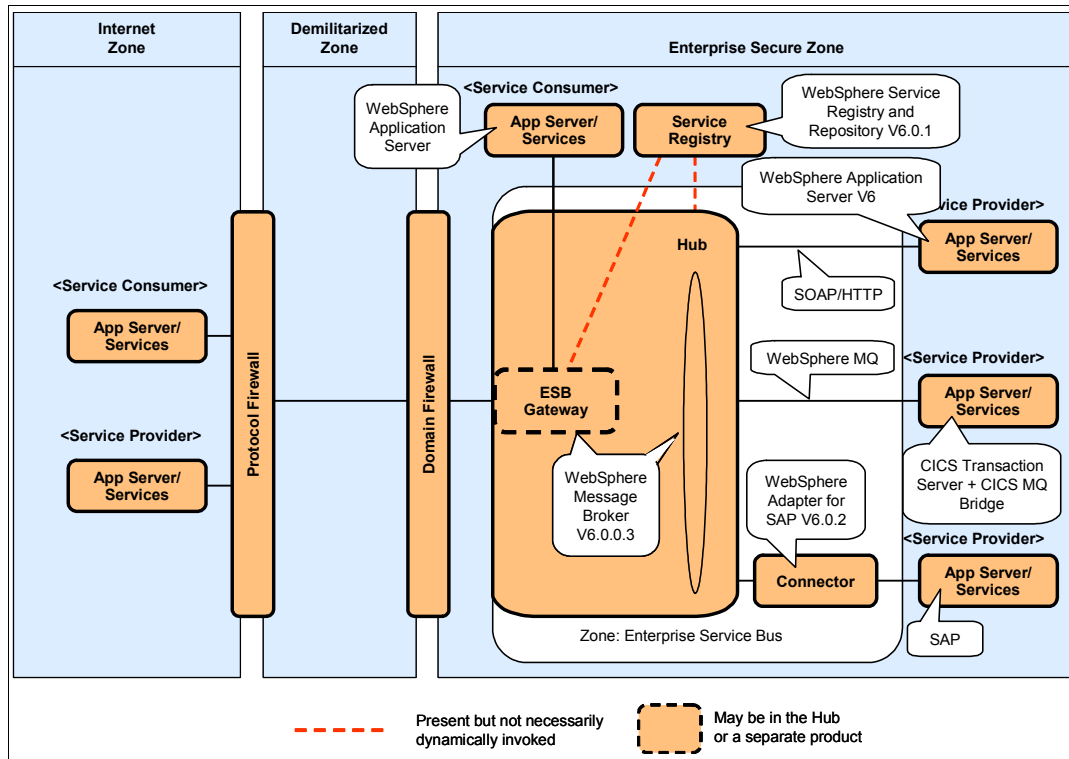
Figure 4-4   Exposed ESB Gateway composite pattern product mapping

Hub

As with the ESB runtime pattern, the choice between WebSphere Enterprise Service Bus and WebSphere Message Broker is based on the integration capabilities required. Which product to use, and whether there is a need for both products is discussed in Chapter 8, "ESB design options" on page 181.

ESB Gateway

It is important to separate the ESB Gateway capabilities at least from a design point of view, even if they are still being provided by the same product as the ESB Hub. As such, both WebSphere Enterprise Service Bus and WebSphere Message Broker can perform the function of the ESB Gateway. However, WebSphere DataPower appliances, and specifically the XI50 and XS40, which specialize in security and Web Service proxy capabilities, should also be

investigated for this role. These appliances parse, transform, and route XML at wire speed; add significant levels of threat protection; significantly reduce the CPU and memory overhead of the XML processing on the ESB Hub; and are described in Chapter 7, "WebSphere DataPower appliances in SOA" on page 165.

4.2 Multiple ESBs within an organization

In this section we discuss the issues surrounding environments that have more than one ESB present. This topic can, however, easily be misinterpreted. Let us first make a clear distinction between two very different design problems, the *second* of which is the focus of the rest of this section:

► Multiple hub technologies within an ESB

This is where two hub technologies are coupled together within a *single* ESB. The combination of two technologies gives this single ESB a wider breadth of capabilities. For example, WebSphere Enterprise Service Bus might be used as the main entry point for all services, handling basic routing functionality and access to back-end systems via JCA adapters. However, when the back-end systems use more complex industry standard formats or are more reliant on sophisticated MQ features for integration, WebSphere Enterprise Service Bus might route the request through WebSphere Message Broker. So more than one hub technology is used, but we are only dealing with one ESB. Another way of looking at this is using one ESB technology as the main entry point and using a second ESB technology to provide additional protocol and data adapter options. While a perfectly valid design, this is *not* the focus of this current section.

► Multiple ESBs

This is where several ESB initiatives have been implemented separately and unrelated, perhaps by different departments. This is very likely to occur when an enterprise has an ESB already, and acquires another company that also has its own implementation of an ESB. It can even happen within a single company when two different departments or domains independently go ahead and create ESBs for their own domain. Each domain provides a self-contained Enterprise Service Bus, owning its own namespaces, providing its own capabilities for monitoring, auditing, clustering, and so on. In this situation we are not dealing with simply a combination of technologies, but more a combination of separate ESB initiatives with different governance, funding, and strategy. This is covered in the remainder of this section and in *Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture*, SG24-67730.

4.2.1 Multiple ESBs

The intent of the ESB is to facilitate integration across the entire enterprise, hence the name *enterprise service bus*. A single controlling entrypoint for all services (for example, a single ESB) must surely be the optimum solution? Well, not necessarily.

There are several reasons, technical and organizational, which we discuss in this section.

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

The following sections describe these reasons in more detail.

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?" 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 find multiple CIO and CTO positions within an enterprise that has grown this way.

Some enterprises use a franchise or co-operative 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.

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 or 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.

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 on-going costs of certifying the ESB components needed for the healthcare organizational unit would not have a business benefit.

As said earlier, 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.

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.

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.

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 re-implement 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.

Conclusion

A single ESB, if it satisfies the needs of the enterprise, will of course be simpler to sustain, but there are many reasons why this may not be practical. Each enterprise must look at the above strategic and tactical issues to determine whether one or several ESBs will be required. This should be examined on a periodic basis to determine whether the original decision remains the best one, and whether ESBs can reasonably be merged, or maybe even should be split. If you require more than one ESB, the remainder of this chapter will be of help, as it discusses some of the issues you may face.

4.2.2 ESB topology patterns

As noted in the previous section, for an organization of significant size or complexity, the simplistic ESB patterns shown above are just the first steps on the road to a full SOA. New challenges soon arise where departments have grown their SOA independently from one another, or when trying to integrate new departments following an acquisition such that more than one ESB exists within the enterprise.

Figure 4-5 shows the typical progression of topologies as the complexity increases, or arguably as the SOA matures.
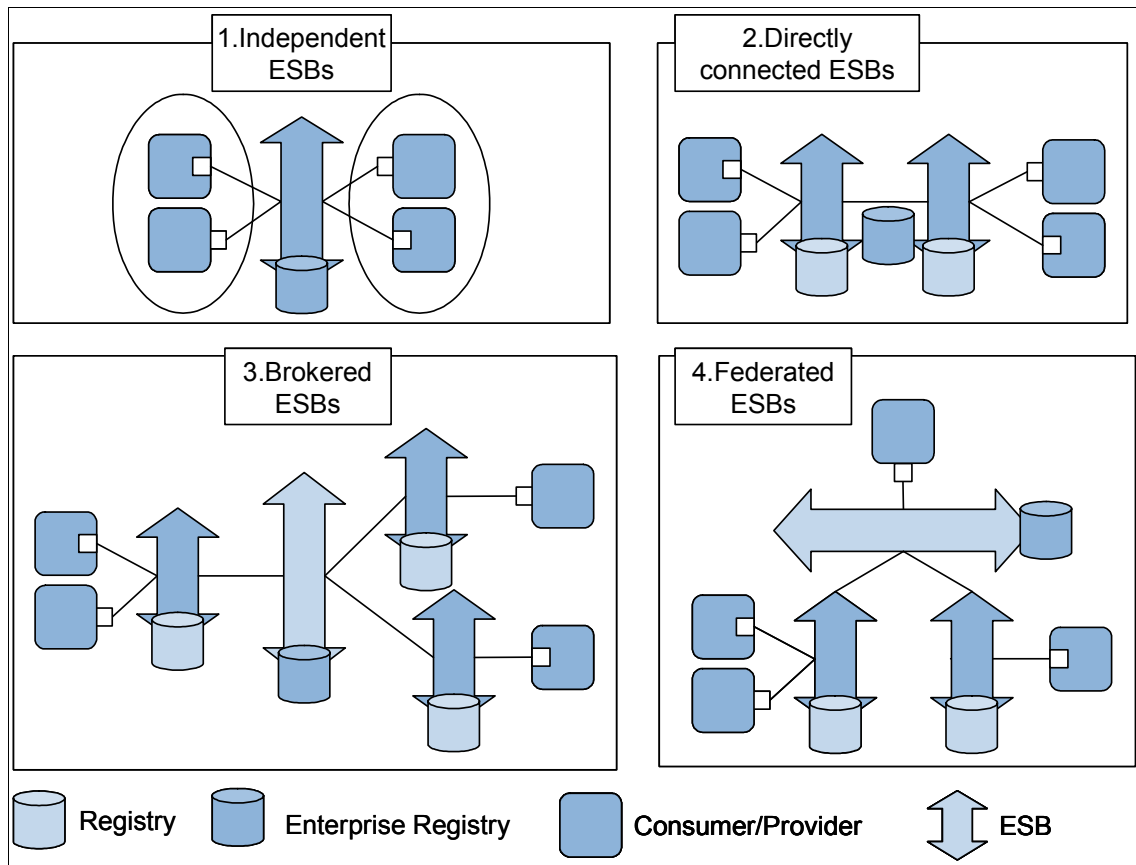
Figure 4-5   ESB topology patterns

These are:

- Independent ESBs

  This represents the situation where one or more independent ESBs are present in the organization, as discussed throughout 4.1, "ESB runtime topologies" on page 54.

- Directly connected ESBs

  This is used where services are provided and managed by a line of business but made available enterprise-wide. This topology soon suffers from the same issues as would be present with point-to-point integration, especially when the number of ESBs goes beyond two. The ESBs have to know where the other ESBs are, and we are back to the basic problem of maintaining multiple

interfaces that the ESBs were introduced to avoid. The introduction of a common Service Registry improves the situation, but does not centralize any of the core capabilities such as monitoring and auditing.

► Brokered ESBs

These introduce a broker or gateway that selectively exposes services to partners in other domains. The gateway regulates sharing among multiple ESB installations that each manage their own namespace. Departments develop and manage their own services, but share a few of them, or selectively access services provided across the enterprise. The broker or gateway ESB does not provide connectivity to back-end systems and focuses more on aspects such as routing, namespace transformation, security, and monitoring. If we look back to the topology of the Exposed ESB Gateway, as described in 4.1.3, "Exposed ESB Gateway composite pattern" on page 62, you will see the similarity between the ESB Gateway and the brokering ESB in this multiple ESB topology. Effectively, the Exposed ESB Gateway product pattern had already separated out the gateway functionality, and this capability can then be shared across multiple ESBs as the entry point to the SOA.

► Federated ESBs

This is an extension to the Brokered ESB with the key difference that consumers can talk directly to the central hub. Effectively, one master ESB to which several dependent ESBs are federated. Service consumers and providers connect to the master or to a dependent ESB to access services throughout the network. This is used by organizations that want to federate a set of moderately autonomous departments under the umbrella of a supervising department. The Federated ESB recognizes that there will be some composite services or business processes that choreograph services from multiple ESBs. These processes cannot therefore live within any one ESB's domain and must make their requests at the level of the federated bus.

These patterns are described in more detail in *Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture*, SG24-67730, but for now it is sufficient to understand the different ways that ESBs can be combined and connected such that we can go on to discuss what effect this has on the administration of policies across the ESBs.

4.2.3 Handling policy with ESB Gateways and Service Registries

Multiple ESBs arise ultimately as a result of different domains of governance. Whether that be differing governance over the technology or political divisions within an organization, the one common factor is that the $policy$ imposed by each domain is different. Each domain will have its own requirements and standards to which they expect all services to comply.

Examples of policies that are relevant to be applied to services are:

► System monitoring
► Logging and traceability
► Auditing
► Security
► Data protection
► Performance
► Service level agreements (SLAs)
► Service virtualization
► Usage statistics

Service-oriented architecture give us an opportunity to implement at least some of these policies in a single place, rather than embed them into the heterogeneous back-end technologies.

In an environment with a single relatively simplistic ESB we will most probably only be dealing with a single, or at least small, set of policy rules. In this circumstance it may be reasonable to implement the policy directly in the ESB since this is the single point through which all services will be passed. However, it is worth taking a moment to discuss a more appropriate separation of responsibilities that will better centralize the administration of services and will provide better scalability as the service architecture grows. Specifically, we discuss the use of the *ESB Gateway* and the *Service Registry* to assist with the implementation and administration of policy.

The Exposed ESB Gateway runtime pattern shown in Figure 4-3 on page 62 allows for the *ESB Gateway node* to be either implicitly part of the Hub node or defined explicitly as a separate node from the Hub. The ESB Gateway is the single standard external entrypoint to the Hub and is therefore the ideal place to implement generic external policies.

When we have multiple ESBs, and therefore several different governance domains and policies, the benefits of this separation of responsibility become clear. The ESB Gateway allows us to introduce a new ESB Hub to the environment and centralize the administration of policy.

The Exposed ESB Gateway patterns also separate the *Service Registry* from the ESB Hub. This is because the service definition stretches beyond the ESB. It will be used at development time, by monitoring tools, and at various other times during the life cycle of the service. The policies discussed above may be relevant to more then just the ESB, so it is reasonable to assume that these should also be stored outside the ESB in the Service Registry alongside the service definition, further centralizing the administration of the service-oriented architecture.

The configuration in the registry can be used to dynamically apply different policies to any of the services, controlling everything from who has access to them through to how they are logged, audited, and monitored. Policies can be re-used in order to apply policy by domain or to provide categorizations of services.

Now we have established that policy can be administered more effectively and made available to a broader range of uses if its metadata is moved out into a separate service repository. We have also seen that the function of the ESB Gateway is broadly to provide access to services while implementing those policies. Now we are in a position to consider how we can handle the variations in policy required to support a complex multiple-ESB scenario with many governance domains.

4.2.4 Patterns for multiple governance zones

In Figure 4-5 on page 70 we saw that there were a number of ways to organize multiple ESBs within an organization. These separate ESBs typically belong to different zones of governance with different policy requirements. We now revisit some of those topologies in more detail and consider how the multiple runtime policies that we have discussed above should be managed.

> **Note:** Governance is a huge topic reaching far beyond simply the administration of runtime policies discussed here. A wider discussion of the reach of governance in relation to service-oriented architecture can be found in *WebSphere Service Registry and RepositoryWebSphere Service Registry and Repository Handbook*, SG24-7386.

The three main multiple ESB topologies from Figure 4-5 on page 70 are:
- ► Directly connected ESBs
- ► Brokered ESBs
- ► Federated ESBs

We look at each of the topologies in turn, offering two possible implementations of the Brokered ESBs, and establish how this affects the positioning of registries and gateways within the topology.

Directly connected ESBs

Figure 4-6 shows the *directly connected ESBs* topology where the ESB in domain 1 simply treats the service in domain 2 as another back-end system. Domain 2 continues to own the service and implement its policy and registry definition.
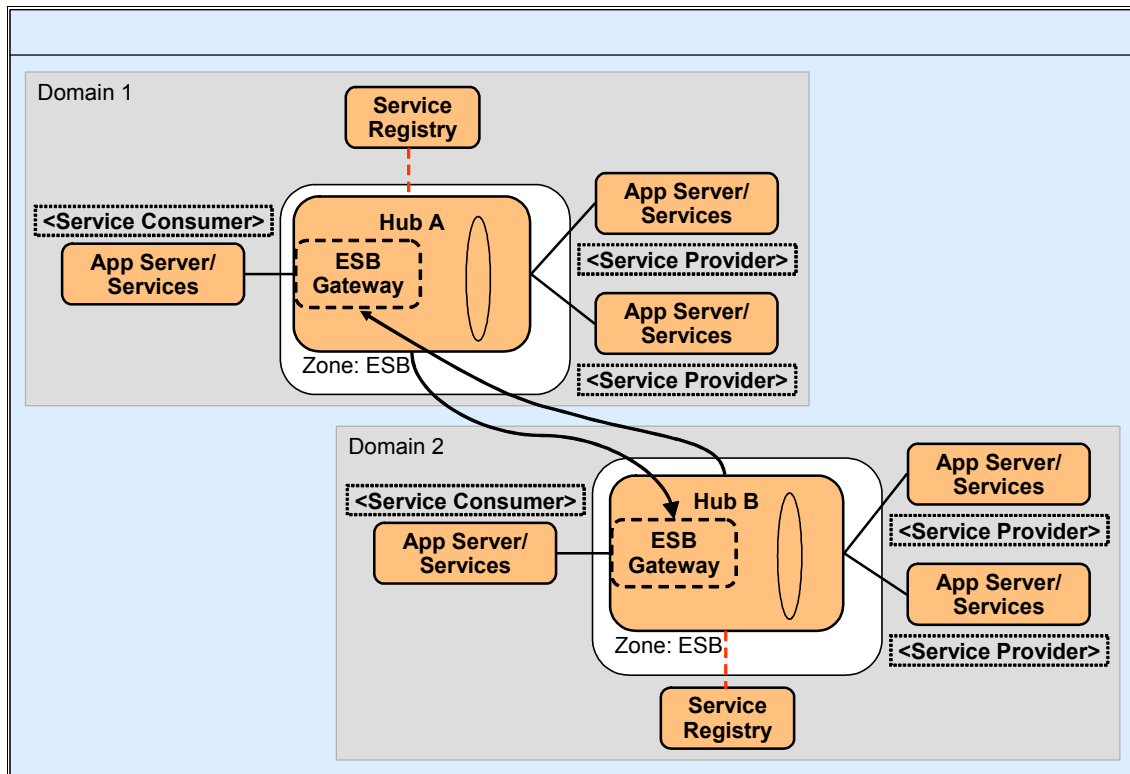
Figure 4-6   Directly connected ESBs

Pros:

► Relatively simple to configure. The domain 2 service is likely to be exposed in a suitable standard format for easy consumption by the domain 1 ESB.

► Complete decoupling. Neither domain is aware that there is an inter-ESB interaction taking place.

► Local services consumers in domain 2 continue as before since policy is still implemented by the domain 2 ESB.

Cons:

► A copy of at least part of domain 2's service definition must copied into the domain 1 repository. This will have to be done explicitly for each individual

service exposed in this way. It becomes difficult to have a centralized view of the service catalogue.

► The service is likely to suffer from the costs of implementing policy in both domain 1 and domain 2. In addition to this policy overhead, the request is traversing twice the number of hubs and gateways necessary.

► We are effectively performing point-to-point integration between ESBs. This has the same issues as point-to-point between applications — as the number of ESBs increases, the number of interactions grows exponentially.

Brokered ESBs

For Brokered ESBs, one ESB is chosen to act as a master domain or *broker* through which all interactions traverse when they need to make use of a service in another domain. The two key options for this configuration are explored in Figure 4-7 and Figure 4-8 on page 77.

Figure 4-7 shows the most obvious pattern where domain 1 has been chosen as the master domain. Consumers from other domains such as domain 3 always use domain 1 to get to services that are outside their domain, rather than creating a further connection from their own ESB.
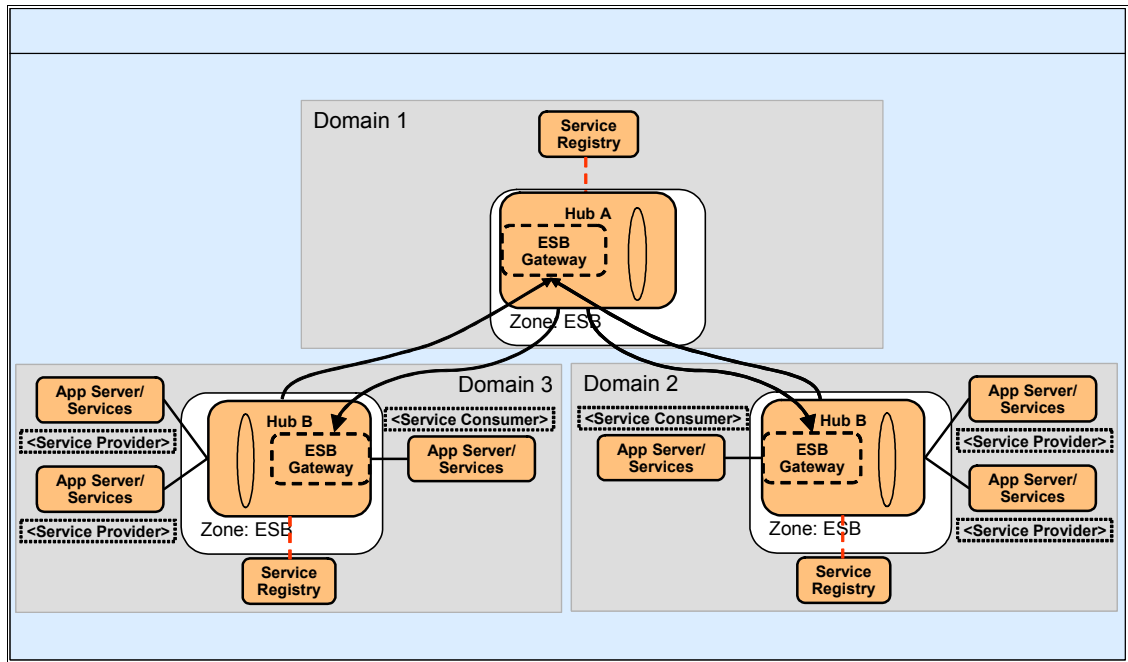
Figure 4-7 Brokered ESBs

Pros:

- ► Domain 1's registry now represents a catalogue of all the services available in the enterprise.

- ► If service endpoint details change, they only need to be altered in domain 1.

- ► Local services consumers in domain 2 continue as before since policy is still implemented by the domain 2 ESB.

Cons:

- ► Domain 1's registry has satellite copies of parts of it's catalogue spread out among the other ESBs, leading to extra maintenance effort, and opportunities for erroneous configuration.

- ► We have the overhead of going through three sets of hubs and gateways, and no doubt implementing repeated policies. Now this is probably slightly unfair, since in reality communication would most probably only need to go via the three gateways, with minimal transformation, and then through just the one hub at the end of the chain to reach the provider.

Figure 4-8 shows the Brokered ESB's variation 1. Here we show a more realistic Brokered ESB topology. Note that some domains such as Domain 3 may not have an ESB and so they would call the domain 1 gateway directly. We also show that domain 1 will most probably also provide services itself, and will have consumers of those and other services.
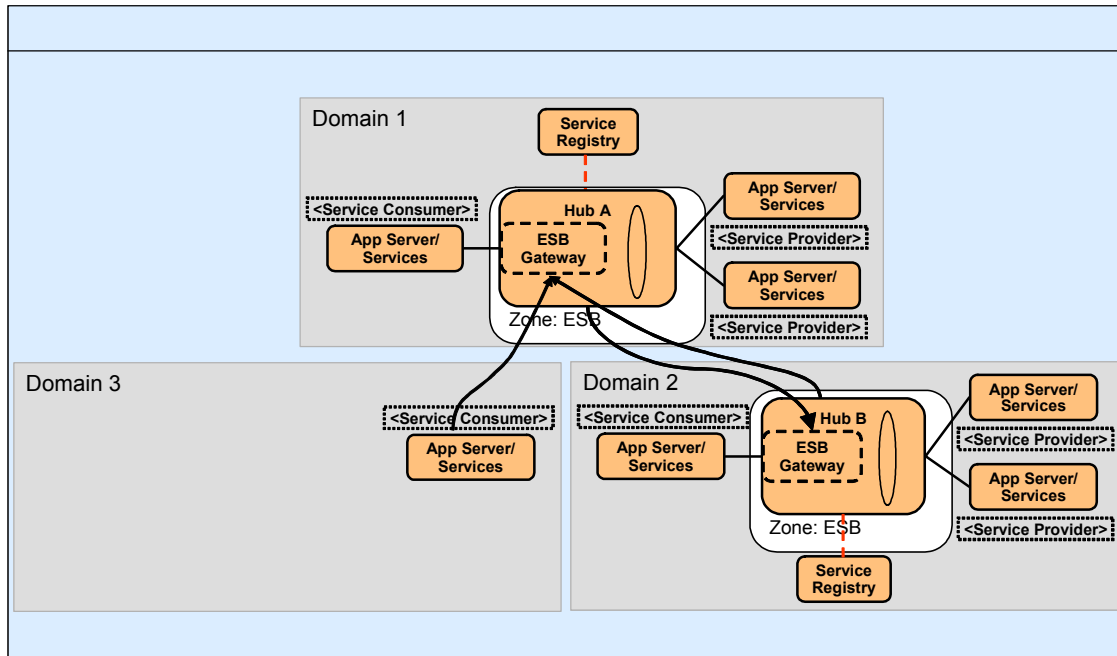
Figure 4-8   Brokered ESBs variation 1

Federated ESBs

At the heart of this final pattern is the concept that each of the domain's ESBs are autonomous, and yet they all have a knowledge of the wider enterprise-level services. Figure 4-9 shows a possible topology for federated ESBs in which any consumer can call services in any domain without necessarily having set up the communication paths in advance.
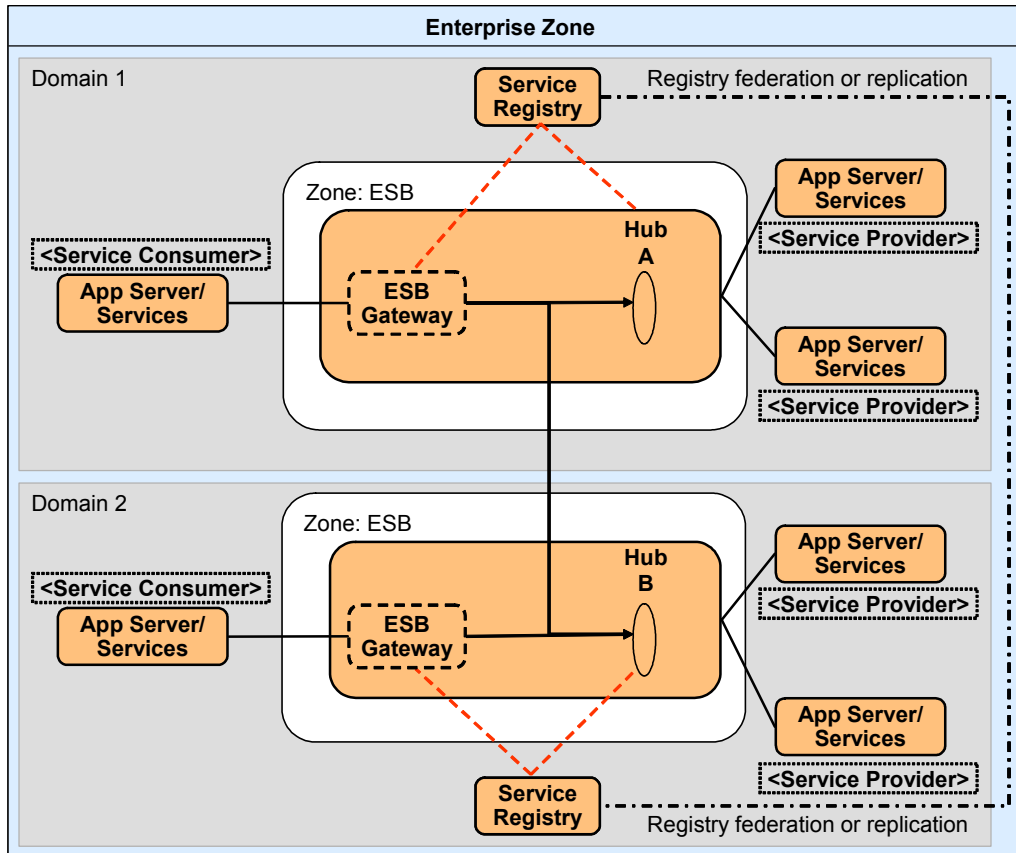
Figure 4-9   Federated ESBs

There are some key capabilities that would need to be in place for this pattern to be possible:

► Peer-to-peer lookup or replication facilities between registries

► Standardization of protocols used between ESB Gateways and Hubs

► Trusted channels for intra-ESB traffic between domains

► Standardization of policy attributes

- Capability of all ESB Gateways to be able to retrieve and use service information at run time
- Capability of all ESB Gateways to perform service agnostic policy based on dynamically retrieved policy metadata
- Policy executed to create the same ultimate effect regardless of the gateway implementing the policy

The peer-to-peer relationship is made possible by the fact that the domains' registries have knowledge of one another and are able to look up or replicate endpoint information for services that are outside of their domain.

> **Tip:** This peer-to-peer registry lookup capability is similar to the model used by the Domain Name System (DNS) used on the Internet to look up IP addresses based on host names. Such queries ripple through a chain of DNS servers in a hierarchical fashion, enabling matches to any host name to be found regardless of which DNS server you initially connected to. In this case we are looking for a particular service and its endpoint details based on a set of service criteria, but the principle is largely the same.

For this pattern to work, there must be some level of agreement at the enterprise level about what protocols and interaction types are allowable between ESB Gateways and Hubs, and there need to be suitable channels opened up across domain boundaries to allow service calls to pass. The two ESB zones represent a set of nodes that communicate with an agreed upon set of protocols over trusted channels.

An interesting good part of the fact that we have so rigidly defined the ESB Gateway to Hub protocols is that if an application provides a suitable interface already, it could in theory directly publish itself in the registry as an endpoint. In other words, the application may not need to be connected to a Hub in order to be exposed to the enterprise. It can then be called directly by the ESB Gateways using the same standard protocols used to converse with Hubs. Consumers of the service are already decoupled from future change to the endpoint by virtue of the fact that they call the ESB Gateway and will draw any changes to the service endpoint from the registry at run time.

Pros:

- Provides a highly scalable architecture for distributed enterprise service buses.
- Service endpoints can be made available to the enterprise by being added to any domain and by publishing them to the local registry with appropriate enterprise-level policy.

- ► Service consumers do not have to directly make cross-domain requests.
- ► New domains can be added with minimal configuration to existing domains.
- ► The standardization on ESB Gateway to Hub protocols and interaction types allows new services (conforming to those protocols) to be published without hub configuration.
- ► No double hops are required to go through unnecessary gateways and hubs.

Cons:

- ► Significant requirements on maturation of service architecture
- ► Standardization in some areas required across the enterprise
- ► Complex requirements on gateways and hub nodes
- ► Domains must meet a minimum criteria to take part

5

WebSphere Enterprise Service Bus

This chapter provides an introduction to WebSphere Enterprise Service Bus, its runtime architecture, development environment, and related concepts and products. We discuss these with a specific focus on their place in a service-oriented architecture.

For a detailed description of WebSphere Enterprise Service Bus refer to *Getting Started with WebSphere Enterprise Service BusGetting Started with WebSphere Enterprise Service Bus V6*, SG24-7212.

Section 5.5, "WebSphere ESB V6.0.2 release notes" on page 103, highlights the enhancements in v6.0.2 of WebSphere ESB.

5.1 Product overview

WebSphere Enterprise Service Bus delivers an Enterprise Service Bus (ESB) infrastructure to enable connecting applications that have standards-based interfaces (typically a Web service interface described in a WSDL file). It provides mechanisms to process request and response messages from service consumers and service providers that connect to the ESB.

WebSphere Enterprise Service Bus is the mediation layer that runs on top of the transport layer within WebSphere Application Server. As such, WebSphere Enterprise Service Bus provides prebuilt mediation functions and easy-to-use tools to enable rapid construction and implementation of an ESB as a value-add on top of WebSphere Application Server.

Figure 5-1 gives an overview of WebSphere Enterprise Service Bus, the components in the product, and the features and functions that are associated with the product.
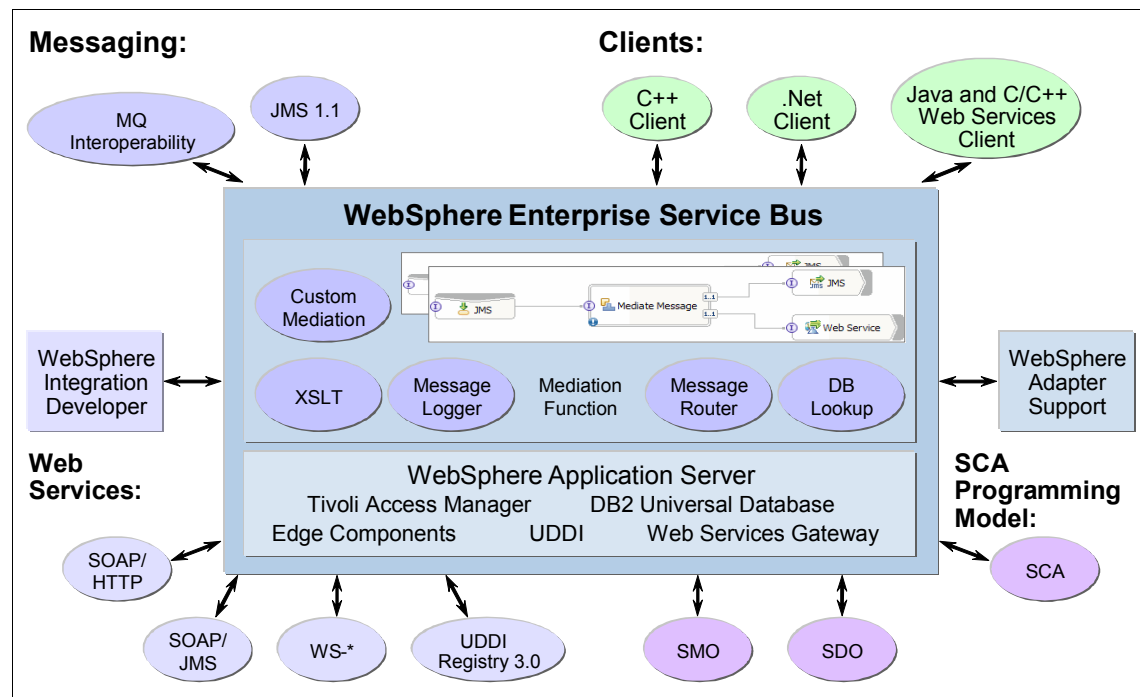
Figure 5-1   WebSphere Enterprise Service Bus at a glance

WebSphere Enterprise Service Bus leverages WebSphere Application Server Network Deployment qualities of service, with its clustering, failover, scalability, security, and a built-in messaging provider. Along with these qualities,

WebSphere Enterprise Service Bus includes a number of key WebSphere Application Server related features, including UDDI as a Service Registry, the Web services gateway, Tivoli Access Manager, DB2® Universal Database™, and Edge components.

WebSphere Enterprise Service Bus adds the following value to the application server:

► Provides built-in mediation functions, which can be used to create integration logic for connectivity.

► The SCA programming model supports rapid development of mediation flow components.

► WebSphere Integration Developer is an easy-to-use tool that supports WebSphere Enterprise Service Bus.

► Leveraging WebSphere Application Server, WebSphere Enterprise Service Bus offers JMS messaging and WebSphere MQ interoperability for messaging, as well as a comprehensive clients package for connectivity.

► Offers support for J2EE Connector Architecture based WebSphere Adapters.

To implement an SOA properly, it is necessary to have a single invocation model and a single data model. Service Component Architecture (SCA) is this invocation model — every integration component is described through an interface. These services can then be assembled in a component assembly editor, thus enabling a very flexible and encapsulated solution.

WebSphere Enterprise Service Bus introduces a new component type to the SCA model — the *mediation flow component*. From the perspective of the SCA, a mediation flow component is not different from any other service component.

Business objects are the universal data description. They are used as data objects are passed between services and are based on the Service Data Object (SDO) standard. In WebSphere Enterprise Service Bus, a special type of SDO is introduced, the Service Message Object (SMO).

Also part of the infrastructure is the Common Event Infrastructure (CEI), which is the foundation for monitoring applications. IBM uses this infrastructure throughout its product portfolio, and monitoring products from Tivoli as well as WebSphere Business Monitor exploit it. The event definition (Common Business Event) is standardized through the OASIS standards body so that other companies as well as customers can use the same infrastructure to monitor their environment.

5.2 Key terms in WebSphere Enterprise Service Bus

Table 5-1 summarizes the key terms in the context of WebSphere Enterprise Service Bus that are introduced in this chapter.

Table 5-1   Key terms relating to WebSphere Enterprise Service Bus

Term	Explanation
Mediation	A service request interception by an ESB that typically centralizes logic such as routing, transformation, and data handling.
Mediation module	The basic building block in WebSphere Enterprise Service Bus for creating mediations.
Export	Exposes the interfaces of a mediation module and contains the bindings.
Stand-alone reference	The external publishing of an interface for SCA clients only (without a WSDL description).
Import	Represents the service providers that are invoked by a mediation module.
Binding	The protocols and transports that are assigned to exports and imports.
Mediation flow component	The container for mediation logic inside a mediation module that provides interfaces and that uses references.
Interface	Define access points and are defined using WSDL.
Operation	Represent interactions that can be 1-way (only input parameters) and 2-way (input and output parameters).
Partner reference	The declaration of the referenced interfaces of an mediation flow component.
Wire	An association between components inside a mediation module and exports/imports/stand-alone references.
Mediation flow	The processing steps that are defined for each interface in the form of a request flow and usually a response flow.
Mediation primitive	Units of message processing inside a mediation flow that provide different terminals.
Service message object (SMO)	A data object that represents the context, the content, and the header information of an application message that is created during a mediation flow.
Business object	Data type definitions (specified in XML schema) that can be used for input/output parameters.

5.3 Structure of WebSphere Enterprise Service Bus

This section explores the structure of WebSphere Enterprise Service Bus by working through the different layers of the product architecture in a top-down manner.

5.3.1 Mediations, service consumers, and service providers

A service interaction in SOA defines both service consumers and service providers. The role of WebSphere Enterprise Service Bus is to intercept the requests of service consumers and fulfill additional tasks in mediations in order to support loose coupling. When the mediation completes, the relevant service providers should be invoked. The mediation tasks include:

► Centralizing the routing logic so that service providers can be exchanged transparently

► Performing tasks like protocol translation and transport mapping

► Acting as a facade in order to provide different interfaces between service consumers and providers

► Adding logic to provide tasks such as logging

As shown in Figure 5-2, mediations customize the protocol and the details of a request and also modify the results of the reply.
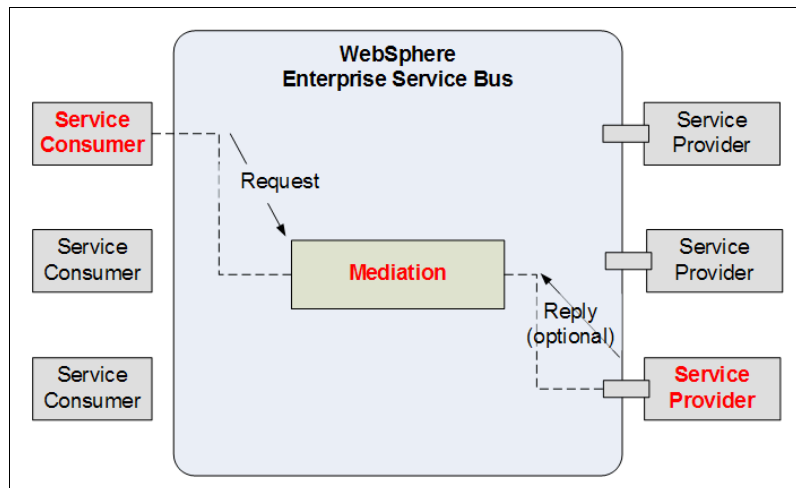
Figure 5-2   Enterprise Service Bus and mediations

WebSphere Enterprise Service Bus can interconnect a variety of different service consumers and providers using standard protocols including:

► JMS
► SOAP over HTTP (for Web services)
► SOAP over JMS (for Web services)
► MQ

For back-end applications (such as SAP®) several IBM WebSphere Adapters (based on JCA) are available.

WebSphere Enterprise Service Bus supports diverse messaging interaction models to meet your requirements, including the following models:

► One-way interactions
► Request-reply
► Publish/subscribe

5.3.2 Mediation modules

The *mediation module* is a new type of SCA component that can process or mediate service interactions. As illustrated in Figure 5-3, the mediation module is externalized or made available through an *export*, which specifies the interfaces that are exposed. These interfaces are defined in a WSDL document. *Stand-alone references* provide the externalized interface only for SCA clients. They do not define a WSDL document. Instead, they specify the interface declaration in Java (called a *reference*).
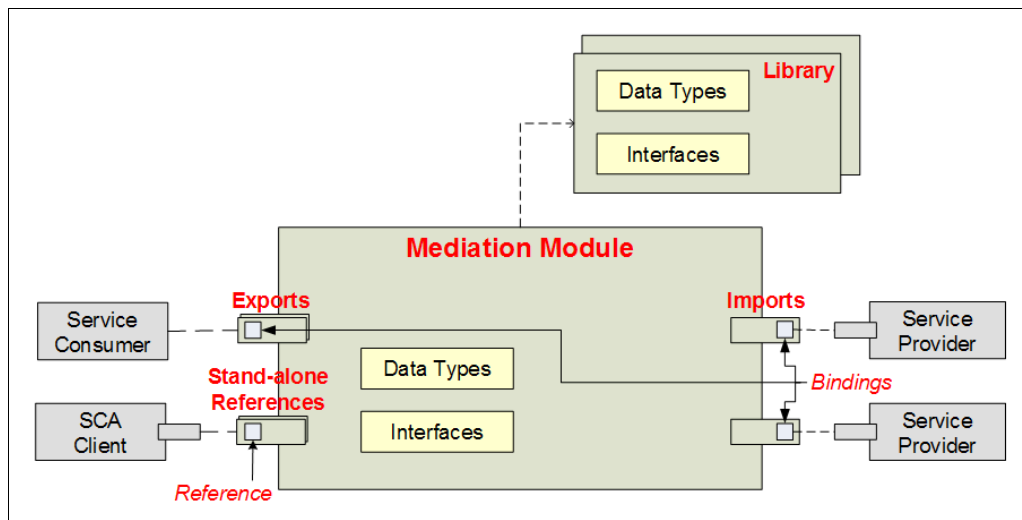
Figure 5-3   Mediation modules

The mediation module typically invokes other service providers. These providers are declared with the creation of an *import*, which represents an external service to be invoked.

For each export and import, an interface needs to be specified. Each interface has multiple operations, which in turn can have multiple input and output parameters that are associated with either simple data types or business objects. A one-way operation has only input parameters.

Every export and import has to be associated with a *binding*. A binding identifies a specific type of invocation for a service consumer or provider. WebSphere Enterprise Service Bus supports several bindings:

► JMS binding leveraging the JMS V1.1 that is delivered in WebSphere Application Server V6 using the service integration bus

► Web services using SOAP/HTTP and SOAP/JMS

► JCA-compliant WebSphere Adapters

► SCA bindings, which are the default bindings that are used for communication between SCA modules

> **Note:** Wiring of SCA components can been done either at development time within WebSphere Integration Developer or administrators can modify those bindings dynamically using the WebSphere Enterprise Service Bus administrative console to *rewire* component interactions (see "Changing bindings" in *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212.).

► Enterprise Java Beans (EJBs), which are only valid for import bindings

Finally, data types (business objects) and interfaces can be defined on the module level, but they can also be defined and referenced in *libraries* in order to centralize them.

5.3.3 Mediation flow components

Inside a mediation module there can be one *mediation flow component*. Mediation flow components offer one or more *interfaces* and use one or more *partner references*. Both get resolved, assigning them to exports or imports via wires, as shown in Figure 5-4.

Important: You should not try to compare the notions and semantics of components and interfaces of the Java programming language with the ones in the WebSphere Enterprise Service Bus model, because this is not applicable in several cases.
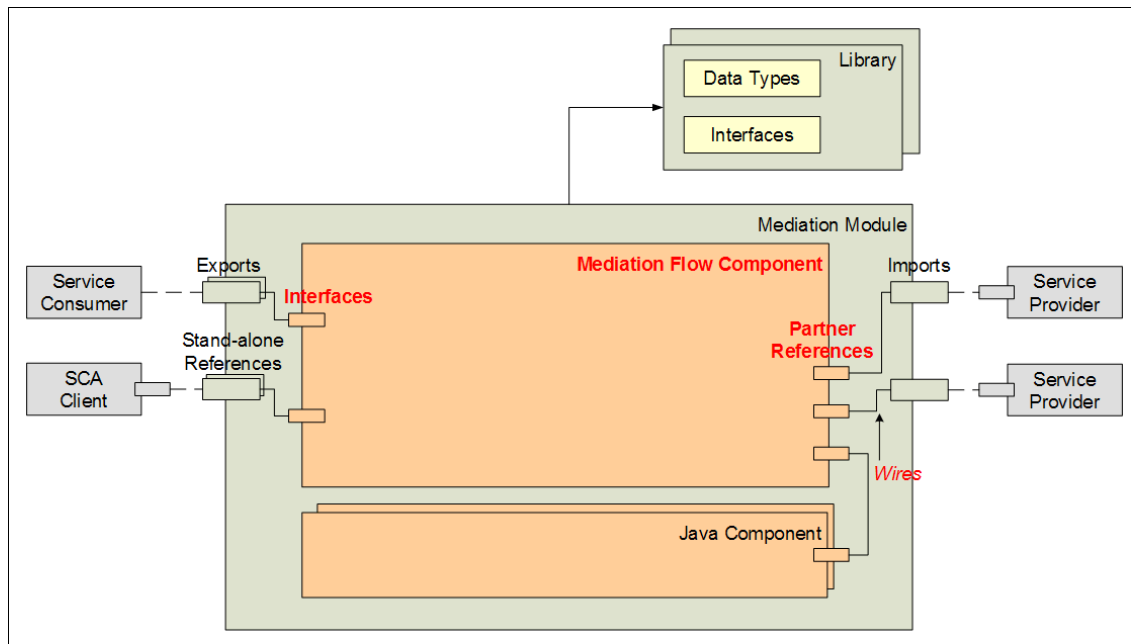
Figure 5-4   Mediation flow component

In addition to the mediation flow component inside a mediation module, one or more Java components can be created using custom mediation implementations.

Restriction: WebSphere Integration Developer does not stop you from creating more than one mediation flow component per mediation module, but only one is allowed (as described in the product documentation). Therefore, there is a one-to-one relationship between a mediation module and a mediation flow component.

5.3.4 Mediation flows

Mediation flows (Figure 5-5) contain the high-level mediation logic. Thus, the different processing steps of a request are declared in a graphical way. In WebSphere Enterprise Service Bus, the processing of requests is separated from the processing of responses. Therefore, we distinguish between a *request flow* and a *response flow*. In both directions, logic can be added or modifications can be applied.

> **Note:** Mediation flows need to be defined *for every operation* that gets exposed via an export of a mediation module. For those operations that do not need any additional functionality to the wrapped interface, you wire them from input-to-input response.
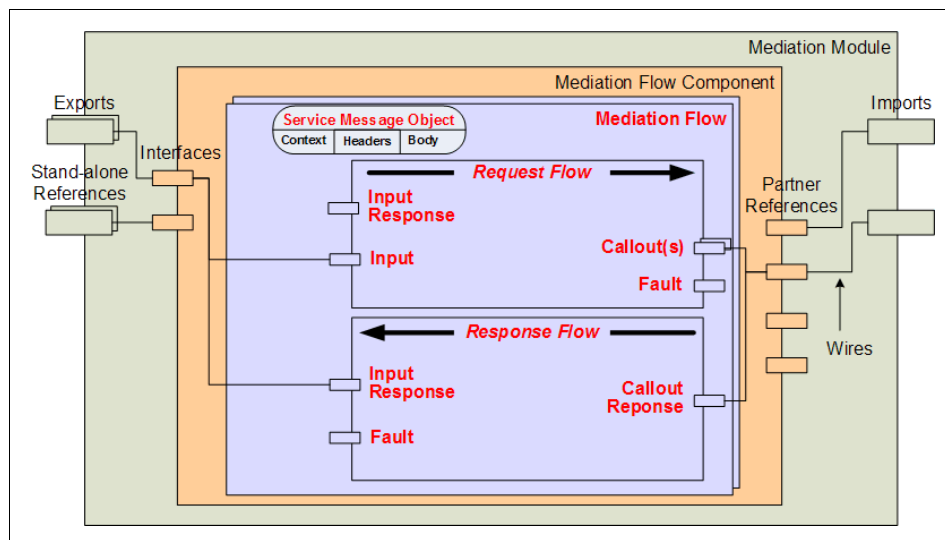
Figure 5-5   Mediation flows

Mediation flows consist of a sequence of processing steps that are executed when an input message is received. A *request flow* begins with a single *input* for the source operation and can have multiple *callouts*. If a message is to be returned to the source directly after processing, it can be wired to an *input response* in the request flow. If fault messages are defined in the source operation, an *input fault* is also created.

A *response flow* begins with one or more *callout responses* and ends with a single input response (and optionally a callout fault). Both a request flow and a

response flow are associated with a mediation flow. The request flow can map data to a correlation context and the transient context.

In terms of the actual data, WebSphere Enterprise Service Bus introduces the Service Message Object (SMO). SMO is a special kind of a service data object that represents the content of an application message as it passes through a mediation flow component. As well as the payload in the body, it contains context and header information, which can be accessed and acted upon inside the mediation flows.

5.3.5 Mediation primitives

Mediation primitives (Figure 5-6) are the smallest building blocks in WebSphere Enterprise Service Bus. They are wired and configured inside mediation flows. They let you change the format, content, or target of service requests; log messages; do database lookups; and so forth.
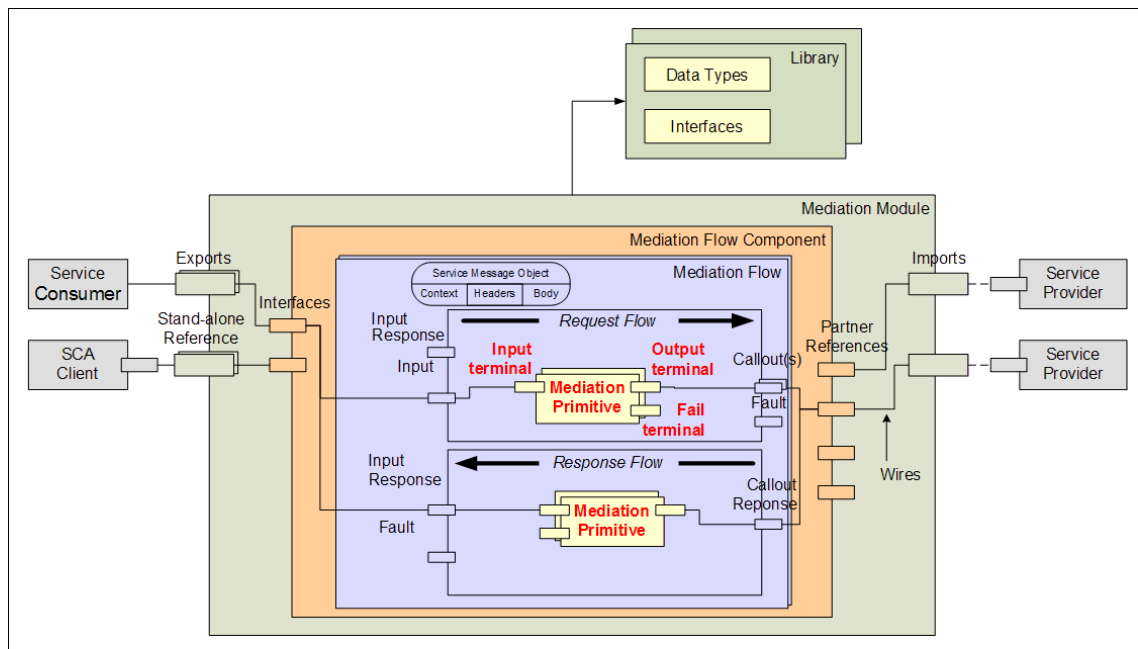
Figure 5-6   Mediation primitives (in the complete overview)

Mediation primitives are the smallest unit in WebSphere ESB that are used for developing mediation flows. Each primitive contains terminals that can be wired to build the mediation logic. The following table lists the primitive nodes that are available with WebSphere ESB V6.0.2

Table 5-2   Primitive nodes available with WebSphere ESB V6.0.2

Mediation primitives	Symbol	Description
Message Logger		Logs a copy of a message to a database for future retrieval or audit. The integration developer can customize the primitive by, for example, naming the database.
Message Filter		To filter messages selectively forwarding them on to output terminals based on a simple condition expression.
Database Lookup		To access information in a database and store it in the message.
XSLT		To manipulate or transform messages using XSL transformation.
Stop		To stop a path in the flow without generating an exception.
Fail		To stop a path in the flow and generate an exception.
Custom		For custom processing of a message. Uses a custom SCA Java component for custom message processing.
Event Emitter		Emit CBE Events from within a mediation flow.
Message Element Setter		Facilitate data changes in mediation flow. SMOs can be updated without custom coding and without having to define XML $\rightarrow$ XML maps.
Endpoint Lookup		Can be configured to search for service endpoints using various selection criteria. Uses WebSphere Service Registry and Repository as the registry.

5.4 Related technologies

This section explores some of the accompanying features of WebSphere Enterprise Service Bus in more detail.

5.4.1 Service message objects

Messages can come from a variety of sources, so the payload has to be able to carry a number of different types of messages. Mediation primitives need to be able to operate on these messages, and service message objects (SMOs) represent the common representation that is needed.

The types of messages that are handled by WebSphere Enterprise Service Bus include:

► SDO data object
► SDO data graph
► SCA component invocation message (request, reply, or exception)
► SOAP message
► JMS message

The SMO model is extensible so it can support other message types in the future, such as COBOL structures. SMO extends SDO with additional information to support the needs of a messaging subsystem.

SMO structure

All SMOs have the same basic structure, defined by an XML schema. An SMO has three major sections:

► The *body* contains the application data (payload) of the message, particularly the input or output values of an operation.

► The *headers* contain the information relevant to the protocol used to send the message.

► The *context* covers the data specific to the logic of a flow or failure information.

Figure 5-7 shows a sample SMO when calling the stock quote sample that is provided with the WebSphere Enterprise Service Bus.
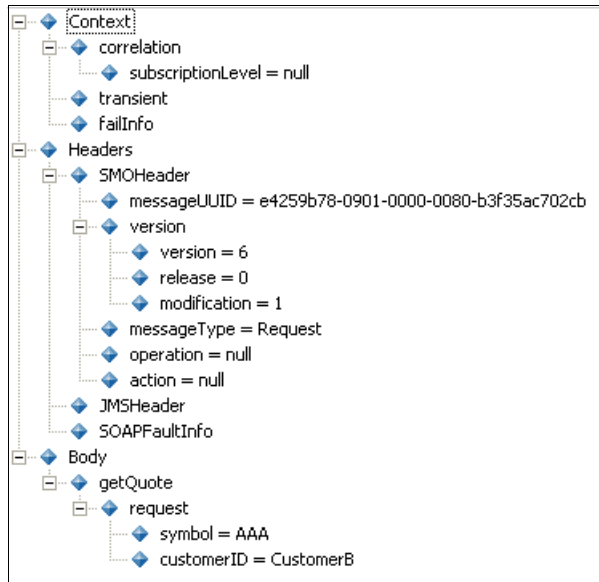
Figure 5-7   Sample SMO

Data section

The data that is carried in the SMO body is the operation that is defined by the interface specification and the inputs/outputs/faults that are specified in the message parts set in the business object definition (Figure 5-8).
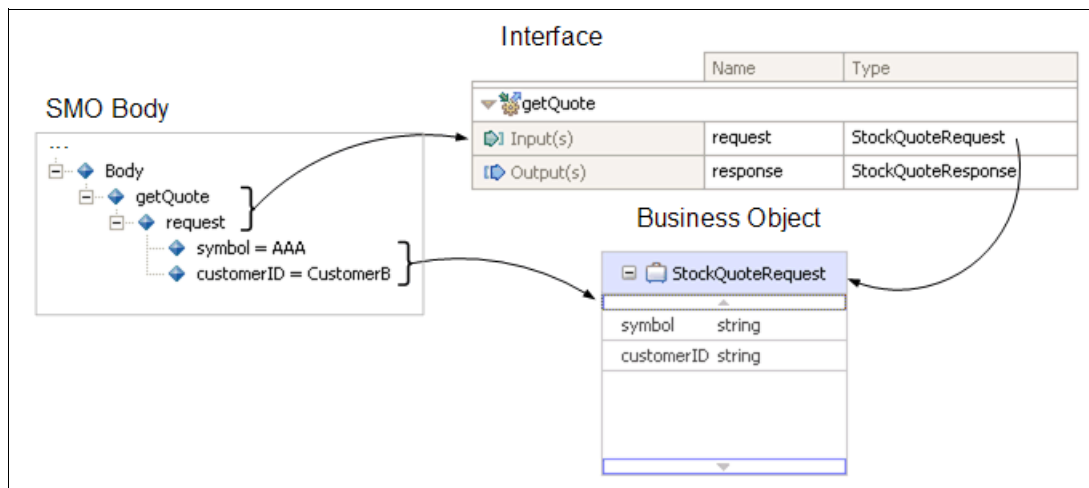
Figure 5-8   Content of the SMO body

Context section

The context includes the correlation and transient context information. Correlation is used to maintain data across a request/response flow, while transient maintains data only in one direction. Both of these contexts are used to pass application data between mediation primitives. They are described as business objects, which contain XML schema that are described data objects and that are specified on the mediation flows input node properties.

The context also includes the failInfo, which is added to the SMO when a fault terminal flow is used. The information that is provided includes the failureString (nature of the failure), origin (mediation primitive in which the failure occurred), invocationPath (the flow taken through the mediation), and predecessor (previous failure).

Header section

The header section of a SMO contains the following supplemental information:

► SMOHeader: information about the message (message identifier, SMO version)

► JMSHeader: used when there is a JMS import or export binding

► SOAPHeader: used when there is a Web services import or export binding

► SOAPFaultInfo: contains information about SOAP faults

► Properties[]: arbitrary list of name value pairs (for example, JMS user properties)

SMO manipulation

During the execution of mediation flows, the active mediation primitives can access and manipulate the SMO. There are three different ways to access SMOs:

► XPath V1.0 expressions

  The primary mechanism that is used by all mediation primitives.

► XSL stylesheets

  Used by the XSLT mediation primitive, and are the common way to modify the SMO type within a flow. These can also be used to modify the SMO without changing the type (using XSLT function and logical processing with XSL choose statements).

► Java code

  Using the Custom Mediation primitive, you can access the SMO either using the generic DataObject APIs (commonj.sdo.DataObject, which is loosely typed) or the SMO APIs (com.ibm.websphere.sibx.smobo, strongly typed).

5.4.2 WebSphere Enterprise Service Bus bindings

Bindings identify a specific type of invocation for a service consumer or provider. Bindings can be applied to mediation module imports or exports. Exports let a mediation module offer a service to consumers. They define interactions between SCA modules and service consumers. Export bindings define the specific way that an SCA module is accessed by others.

Imports let a mediation module access external services (services that are outside the SCA module) in a transparent manner. Imports define interactions between SCA modules and service providers. Import bindings define the specific way that an external service is accessed.

WebSphere Enterprise Service Bus supports the following bindings:

▶ Web service binding

Using a Web service binding on an export, it exposes the module as a Web service. To invoke an external Web service, an import with a Web service binding is used. This binding always uses SOAP messages, and two transports are available:

– SOAP/HTTP
– SOAP/JMS

▶ SCA binding

SCA bindings connect SCA modules with each other. This is the default binding.

▶ WebSphere Adapter binding

– WebSphere Adapters enable interaction with Enterprise Information Systems (EIS).

– The Enterprise Service Discovery tool can be used to create import and exports representing applications on EIS systems. To use EIS bindings a resource adapter is needed.

▶ Java Message Service (JMS) V1.1 binding

– JMS can exploit various transport types, including TCP/IP and HTTP(S).

– There are predefined JMS bindings that support JMS text messages containing Business Object (BO) XML. The predefined JMS bindings also support JMS object messages containing serialized Java Business Objects.

– You can use JMS custom bindings to support other types of JMS messages. However, custom bindings require some coding to translate the message.

– If you want a module to receive a JMS message from a queue or topic, you need to use an export with a JMS binding. If you want a module to send a JMS message, you use an import with a JMS binding.

> **Note:** The publish/subscribe interaction model can be applied in WebSphere Enterprise Service Bus using the JMS binding.

► EJB bindings (only for imports)

An import component can have a stateless session EJB binding.

5.4.3 Quality of service

Qualifiers in SCA allow developers to place quality of service requirements on the SCA run time. There are several different categories of qualifiers available in SCA:

► Security
► Transactions (with ActivitySessions as a special type)
► Reliable messaging

Each qualifier has a particular scope within the Service Component Definition Language (SCDL) specification for a SCA component where the qualifier can be added (interface, implementation, partner reference).

For example, some qualifiers can be specified at the partner reference level, while others might only be valid at the interfaces or implementation level. Figure 5-9 shows the conceptual model for SCA service qualifiers.
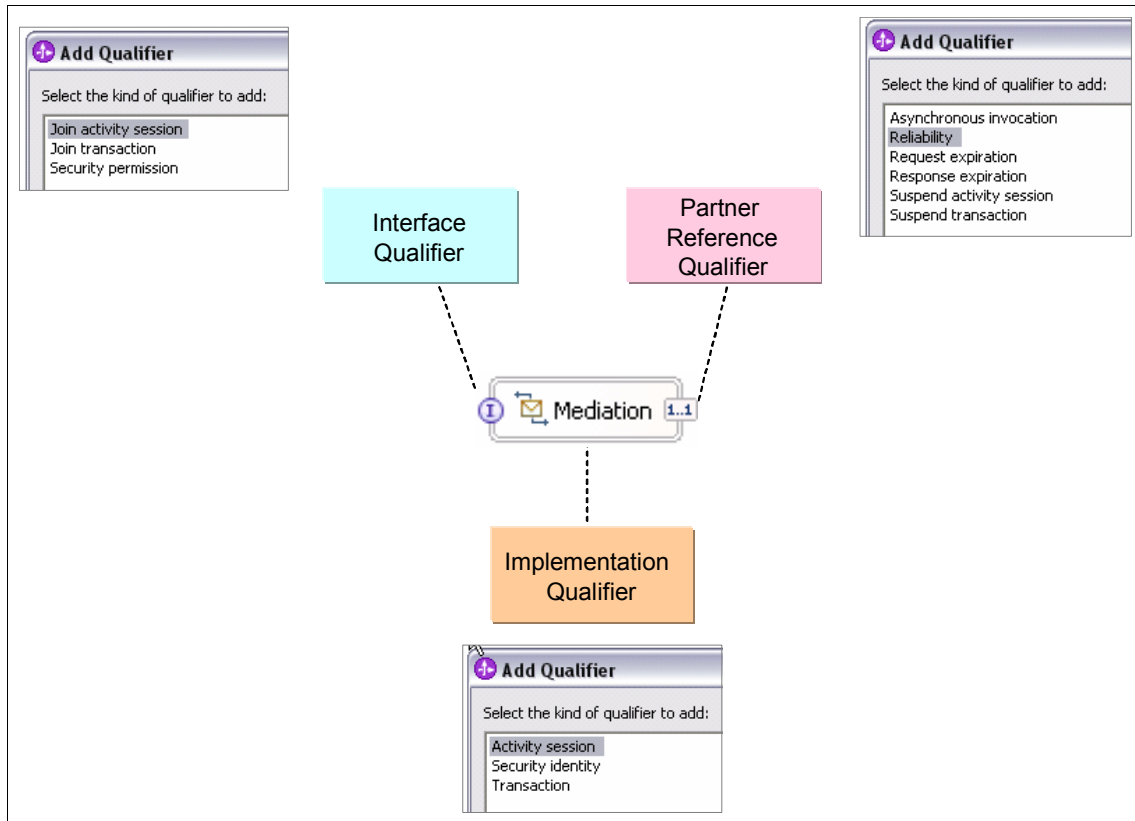
Figure 5-9   SCA quality of service qualifier model

In the following subsections we briefly describe the various qualifiers that are available, and the valid scope for each will be examined.

Security

In WebSphere Integration Developer you specify security attributes for mediation flow components in the properties view at the boundaries and the implementation of an component.

At the interface level you can define the permission for every operation (Figure 5-10). At the mediation flow component implementation level you can define under which identity the component gets executed (initiating a role change), as shown in Figure 5-11 on page 99.
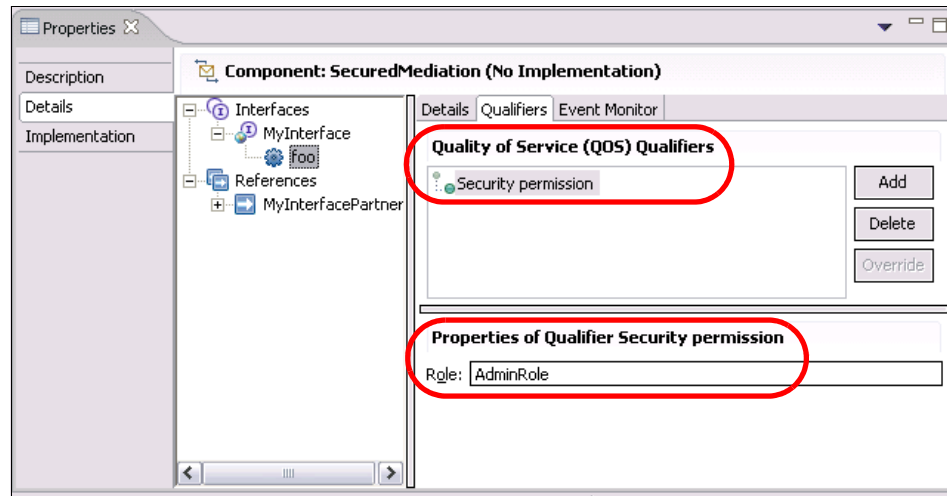
Figure 5-10   Security permission qualifier on interfaces

Use the *security permission* qualifier to specify a role, which is a semantic grouping of permissions that a given type of user must have to use an operation in an interface. The identity of the caller must have this role in order to be permitted to call the interface or operation. If no security permission is specified, then no permissions are checked and all callers are permitted to call the interface or operation.

The *security identity* qualifier is a privilege specification that you can use to provide a logical name for the identity under which the implementation executes at run time (Figure 5-11). An implementation has to be created for this qualifier to be specified. If this qualifier is not specified, then the implementation executes under the identity of its caller. Alternatively, it is executed under the hosting container's identity if no caller identity is present. Roles are associated with the identity and the roles dictate whether the implementation is authorized to invoke other components.
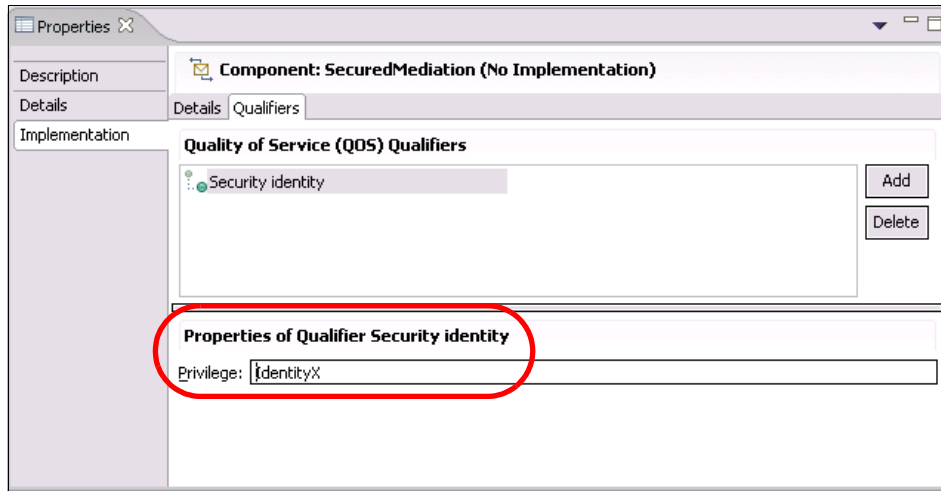
Figure 5-11   Security identity qualifier for mediation components

Depending on the bindings you have created, WebSphere Enterprise Service Bus will generate the relevant J2EE artifacts. In order to integrate remote clients (for example, using the Web service security specifications) with the J2EE application infrastructure, a proper distributed security infrastructure needs to be built. For additional information about securing Web services, see *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461.

Activity sessions

This qualifier determines whether the components processing will be executed under an *activity session*, which provides an alternate unit-of-work scope to the one provided by global transaction contexts. An activity session context can have a longer lifetime global transaction context and can encapsulate global transactions.

> **Note:** Activity sessions are an extension of J2EE that were introduced with WebSphere Application Server V5. For more information see:
>
> http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.
> websphere.base.doc/info/aes/ae/welc6tech_as.html

You can specify the activity session qualifier at all three levels:

▶ Interface level

Can optionally join a propagated (client) activity session.

▶ Implementation level

For the implementation, the qualifier specifies whether the component can run under an established activity session. The default is that if an activity session has been propagated from the client, the runtime environment will dispatch methods from the component in the activity session. Otherwise, the component will not run under any activity session.

▶ Partner reference level

By default, activity session context is always propagated to a target component when it is invoked using the synchronous programming model. If the client does not want a target component to federate with the client's activity session, further qualification of the partner reference is required using the *suspend activity session* qualifier.

Transactions

This qualifier determines the logical unit of work that the component processing executes. For a logical unit of work, all of the data modifications made during a transaction are either committed together as a unit or rolled back as a unit.

▶ On an interface level, the *join transaction* qualifier determines whether the hosting container will join any propagated transaction.

► On a implementation level, the *transaction* qualifier can be set to *global* (where multiple resource managers are required), *local (default)* (running in a local transaction), or *any* (dispatching the global transaction context if existent).

> **Note:** The different combinations of the interface and implementation qualifiers define the behavior for the target component. Not all combinations are allowed.

► For a partner reference, you can specify the *Suspend transaction* qualifier, which can be set to *false* (so the synchronous invocations run completely within any global transaction) and *true* (where synchronous invocations occur outside any client global transaction).

In addition, the *asynchronous invocation* determines whether asynchronous invocations should occur as part of any client transaction. When set to *call* (default), the asynchronous invocations using the partner reference occur immediately, while with *commit* the partner reference is transacted as part of any client global transaction or extended local transaction, which postpones the availability of the request.

Asynchronous reliability

To support asynchronous invocation of components, asynchronous reliability qualifiers can be specified for the partner reference only. They take effect when asynchronous programming calls are used by the client to invoke a service. The reliability qualifier specifications are:

► Reliability

The reliability qualifier determines the quality of an asynchronous message delivery. In general, better performance usually means less reliable message delivery. With an *assured* specification, the client application cannot tolerate the loss of a request or response message. With a *best effort* specification, the client application can tolerate the possible loss of the request or response message.

► Request expiration (milliseconds)

Request expiration is the length of time after which an asynchronous request will be discarded if it has not been delivered, beginning from the time when the request is issued. Zero denotes an indefinite expiration.

► Response expiration (milliseconds)

Response expiration is the length of time that the runtime environment must retain an asynchronous response or provide a callback, beginning from the time when the request is issued. Zero denotes an indefinite expiration.

5.4.4 Common event infrastructure

The common event infrastructure CEI is a core component of WebSphere Enterprise Service Bus leveraged from WebSphere Application Server and provides facilities for the runtime environment to persistently store and retrieve events from different programming environments. This section briefly introduces the basic event-related concepts:

► Common Event Infrastructure (CEI)
► Common Base Events (CBE)

Common Event Infrastructure

In WebSphere Enterprise Service Bus, the CEI is used to provide basic event management services, such as event generation, transmission, persistence, and consumption. CEI was developed to address industry-wide problems in exchanging events between incompatible systems, many of which employed different event infrastructures, event formats, and data stores.

Common Base Event

Although CEI provides an infrastructure for event management, it does not define the format of events. This is defined by the Common Base Event specification, which provides a standard XML-based format for business events, system events, and performance information. Application developers and administrators can use the Common Base Event specification for structuring and developing event types.

The key concept in the Common Base Event model is the *situation*, which is any occurrence that happens anywhere in the computing system, such as a user login or a scheduled server shutdown. The Common Base Event model defines a set of standard situation types, such as *StartSituation* and *CreateSituation*, that accommodate most of the situations that might arise.

In the Common Base Event model, an event is a structured notification that reports information related to a situation. An event reports three kinds of information:

► The situation that has occurred

► The identity of the affected component

► The identity of the component that is reporting the situation, which might be the same as the affected component

In the WebSphere Integration Developer editors the specification of event monitoring is based on the operation level.

5.5 WebSphere ESB V6.0.2 release notes

At the time of writing, Version 6.0.2 of WebSphere ESB introduced a number of key features. This section discusses what was introduced in that release and notes where these features have been demonstrated in the practical scenarios later in the book.

► Administrative configuration

  – Administrative configuration of endpoints: Administrators can modify the definition of a service endpoint, removing the need for a code/test cycle for purely configuration changes.

  – Administrative configuration of meditations: Administrators can dynamically change the behavior of a running mediation. Selected properties of mediations can be exposed to the administrative interface to allow them to be changed at run time.

  – Dynamic endpoint selection: Endpoints can be changed programmatically, based on a value contained in a message, the time of the day, by looking up an endpoint definition from a registry, or any other predetermined parameters.

► Dynamic configuration

  – Integration with WebSphere Service Registry and Repository allows WebSphere ESB to use metadata from the registry and repository to govern its behavior and help ensure that runtime changes are governed by approved policies.

  – New dynamic endpoint lookup primitive: The administrator can query service endpoint information from WebSphere Service Registry and Repository with the dynamic endpoint lookup primitive without redevelopment and deployment.

  – ITCAM for SOA feeds service performance information to WebSphere Service Registry and Repository, giving WebSphere ESB better control for its dynamic selection of service endpoints.

► Simplified and reduced time to market

  – Bundled technology adapters via WebSphere Integration Developer packaging.

  – Increased business activity management (BAM) flexibility: With the common event infrastructure (CEI) primitive so that you can define and feed events into WebSphere Business Monitor.

► Usability improvements

  – Message element setter: Allows you to set values in a message element, without creating message transformation code.

- Remote access to DB2 artifacts on System z™ from a distributed environment.
- Support for IBM DB2 on z/OS as a remote database management system (DBMS).
- Provides new samples to facilitate the development of custom data bindings (datahandler support).
- Delivers simplified server configuration using the WebSphere Application Server Network Deployment console for clusters and multiple cells.

► Additional platform support

Provides additional platform support for Solaris™ 10 (SPARC and x86-64), HP-UX 11i2 (PA-RISC), SUSE Linux Enterprise Server 10, Red Hat Enterprise Linux 4, and Linux on zSeries® (64-bit).

► New bindings

- WebSphere MQ JMS binding provides faster and simpler access to WebSphere MQ JMS based assets.
- WebSphere MQ native binding support provides easier and faster integration with WebSphere Message Broker and WebSphere MQ.

► Performance

Significantly improved performance.

The scenarios later in this book show examples of the WebSphere Registry and Repository lookup, dynamic endpoints, and the MQ and JMS MQ bindings.

6

WebSphere Message Broker in SOA

This chapter discusses how to design a WebSphere Message Broker implementation as an Enterprise Service Bus in a service-oriented architecture and how WebSphere Message Broker provides support for Web service.

6.1 WebSphere Message Broker overview

This section provides an overview for WebSphere Message Broker V6. This book is based on the latest version of the WebSphere Message Broker run time at the time of writing, which is 6.0.0.3, and the latest WebSphere Message Broker toolkit version, which is 6.0.0.2.

6.1.1 Product positioning

Generally speaking, WebSphere Message Broker is a broker engine that can perform message transforming and routing from different participants to different destinations based on user-defined rules, so that diverse applications can exchange information in dissimilar forms, with brokers handling the processing required for the information to arrive at the right place in the correct format.

Unlike a process server, WebSphere Message Broker is not a good place to store long-running business process states and choreograph different business processes. In fact, WebSphere Message Broker provides a connectivity layer for process engines that choreograph the flow of activities between services. It is WebSphere Message Broker's responsibility to deliver service requests, rerouting or transforming them if appropriate.

Similarly, WebSphere Message Broker is also not a good place to implement business logic. It is a bad idea to encapsulate business logic inside the message flow. While WebSphere Message Broker has the ability to route and transform a message based on business rules and values, the real, complex business logic should be put into an application server like a CICS transaction server or WebSphere Application Server.

One of the WebSphere Message Broker's strengths is its powerful capability for parsing different formats of messages coming from different channels or protocols, manipulating the message, and serializing the message in different wire formats and protocols. WebSphere Message Broker is flexible enough to support different messaging paradigms like synchronous and asynchronous behavior, different message interaction patterns like fire-and-forget, request-reply, publish-subscribe, and its aggregation capability.

6.1.2 WebSphere Message Broker runtime architecture

WebSphere Message Broker consists of a development environment in which message flows and message sets are designed and developed, and a runtime environment in which the message flows execute. Figure 6-1 shows an overview of the runtime architecture.
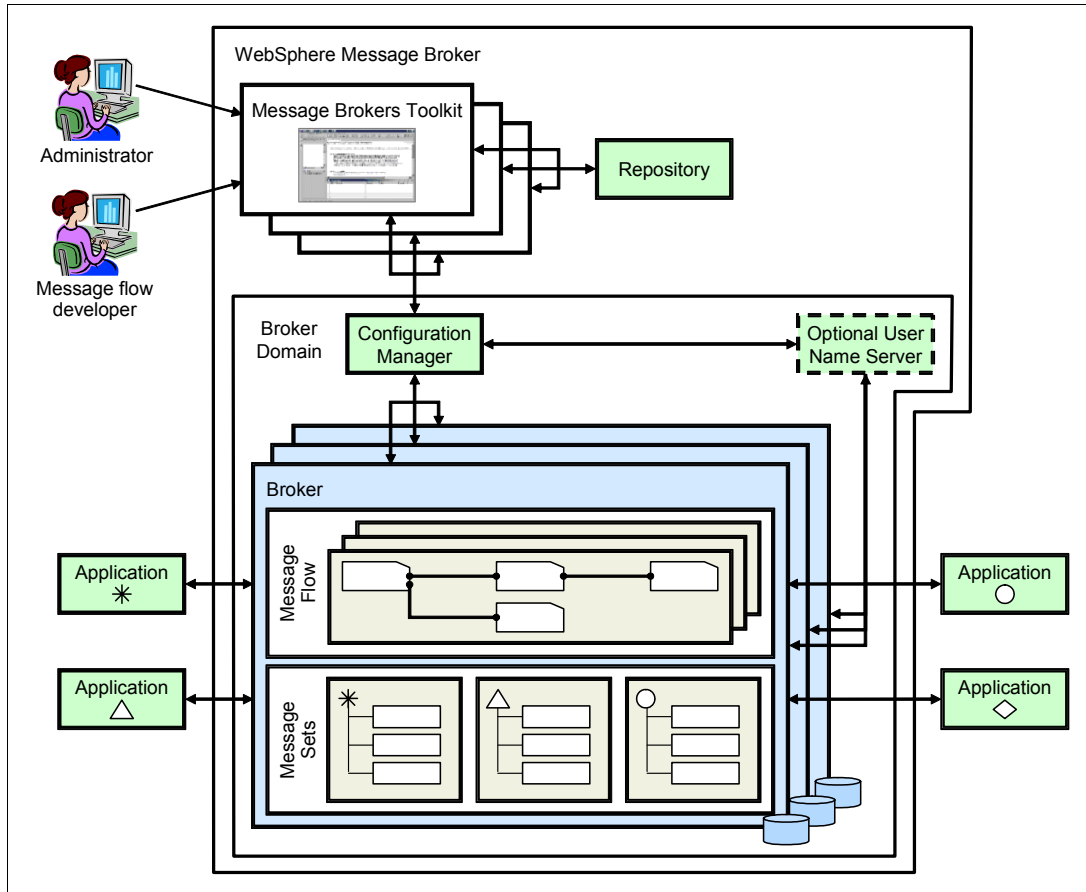
Figure 6-1   Overview of WebSphere Message Broker architecture

Let us take a brief look at each runtime component.

Broker

The broker is a set of application processes that host and run message flows. When a message arrives at the broker from a business application, the broker processes the message before passing it on to one or more other business

applications. The broker routes, transforms, and manipulates messages according to the logic defined in their message flow applications.

Each broker uses a database to keep the broker's configuration information and the message sets together with message flows deployed to it, which will be loaded at the start time.

Execution group

Execution groups enable message flows within the broker to be grouped together. Each broker contains a default execution group. Additional execution groups can be created as long as they are given unique names within the broker.

Each execution group is a separate operating system process and, therefore, the contents of an execution group remain separate from the contents of other execution groups within the same broker. This can be useful for isolating pieces of information for security because the message flows execute in separate address spaces or as unique processes.

Message flow applications are deployed to a specific execution group. To enhance performance, you can deploy additional message flow instances to an execution group. That means that there will be more threads available in the execution group to process the incoming message at run time.

Configuration manager

The Configuration Manager is the interface between the Message Brokers Toolkit and the brokers in the broker domain. The Configuration Manager stores configuration details for the broker domain in an internal repository, providing a central store for resources in the broker domain.

The Configuration Manager is responsible for deploying message flow applications to the brokers. The Configuration Manager also reports back on the progress of the deployment and on the status of the broker. When the Message Brokers Toolkit connects to the Configuration Manager, the status of the brokers in the domain is derived from the configuration information stored in the Configuration Manager's internal repository.

User name server

A user name server is an optional component that is required only when publish/subscribe message flow applications are running, and where extra security is required for applications to be able to publish or subscribe to topics.

The user name server provides authentication for topic-level security for users and groups that are performing publish/subscribe operations.

Broker domain

Brokers are grouped together in broker domains. A broker domain contains one or more brokers and a single Configuration Manager. It can also contain a user name server. The components in a broker domain can exist on multiple machines and operating systems, and are connected together with WebSphere MQ channels. A broker belongs to only one broker domain.

Note: For more information about WebSphere Message Broker, refer to the WebSphere Message Broker Web page and *WebSphere Message Broker Basics*, SG24-7137.

6.2 WebSphere Message Broker as enterprise service bus

SOA defines an approach to reuse and extend current application assets in a very flexible way. This is done by separating the service interface and the service implementation. Each application that provides business logic is exposed as a service. For example, the *create customer account* service might be a CICS transaction, while a customer balance query is served by a J2EE application running on WebSphere Application Server. By creating a standard interface for these applications, their business logic becomes available for reuse in many business process applications.

To enable these applications to talk using the same language is not easy to do. Older applications may use technology that is completely different from current technology.

An ESB provides virtualization and management of service interactions between communicating participants. It not only provides the connection layer between the service provider and service consumer, but it also helps with the translation between two participants trying to interact with different protocols. An ESB can locate the service providers for a service and route the request to the most suitable provider, eliminating the need for the service consumer to know where the provider is.

WebSphere Message Broker provides Exposed ESB Gateway capabilities, including:

► Service virtualization
► Transport protocol support and conversion
► Message models and transformation
► Dynamic message routing
► Custom mediation support

- ► Interaction pattern support
- ► Integration with other enterprise information systems
- ► Quality of service (QoS) support
- ► Service Registry access
- ► Ease of administration

6.2.1 Service virtualization

Service virtualization is a core functionality of an ESB. The service requestor, both in its application logic and deployment, does not need to be aware of the realization of the service provider. The service requestor does not need to be concerned about the programming language the service provider is written in, its runtime platform, its network address, its communication protocol, or even whether there is a service provider implementation available. The requestor only has the responsibility of connecting to the bus and placing the request.

WebSphere Message Broker uses message flows to accept incoming requests using different formats, protocols, and communication channels. It can determine the destination for the request, convert the request if necessary, and then send the request to the service provider.

The service provider takes the request from WebSphere Message Broker (the bus) in its native format, processes the request, and sends the reply back to the bus, which sends the response to the service requester. The provider does not need to know the origin or format of the request.

6.2.2 Transport protocol support and conversion

The use of an ESB eliminates the need for the service requester and service provider to use the same transportation protocol.

WebSphere Message Broker provides support for a variety of transport protocols for both inbound and outbound connectivity that extends the reach, scope, and scale of the ESB to mobile and handheld devices, along with embedded devices such as sensors or actuators.

The transports supported by WebSphere Message Broker include:

- ► WebSphere MQ Enterprise Transport

  Used by WebSphere MQ. This transport supports WebSphere MQ applications that connect to WebSphere Message Broker to benefit from message routing and transformation options.

  The message flow nodes for this transport include MQInput, MQOutput, MQReply, and Publication.

▶ WebSphere MQ Mobile Transport

This is used exclusively by WebSphere MQ Everyplace® clients.

▶ WebSphere MQ Multicast Transport

This is used predominantly for the publish/subscribe model. The nodes provided to support this protocol are Real-timeInput, Real-timeOptimizedFlow, and Publication.

▶ WebSphere MQ Real-time Transport

This is a lightweight protocol optimized for use with non-persistent messaging. It is used exclusively by JMS clients and provides high levels of scalability and message throughput.

The nodes provided to support this protocol are Real-timeOptimizedFlow, Real-timeInput, and Publication.

▶ WebSphere MQ Telemetry Transport

This is used by specialized applications on small footprint devices that require a low-bandwidth communication, typically for remote data acquisition and process control.

The nodes provided to support this protocol are SCADAInput, Publication, and SCADAOutput.

▶ WebSphere MQ Web services Transport

This is used for messages in XML. This uses the standardized HTTP protocol running over TCP/IP. HTTP and HTTPS are the most commonly used protocols over the Internet and intranets. Many firewalls on the Internet are configured to only allow HTTP/HTTPS packets to flow through. For the same reason SOAP over HTTP(S) is the most widely used transportation for Web services.

The nodes provided to support this protocol are HTTPInput, HTTPReply, and HTTPRequest.

▶ WebSphere Broker JMS Transport

This is used to send and receive JMS messages that conform to the Java Message Service Specification Version 1.1. By providing native JMS interoperability, WebSphere Message Broker can act as a bridge between any combination of JMS providers, enabling seamless interaction with other vendors' message platforms.

The nodes provided to support this protocol are JMSInput and JMSOutput. The MQJMSTranform and JMSMQTranform nodes can be used to transform the protocol between MQ and JMS, bringing even greater flexibility.

WebSphere Message Broker can even support flat files, VSAM files, QSAM files, CICS EXCI, TCP/IP Socket, FTP and SCADA based telemetry protocols. These

can also be considered different transportation protocols for delivering messages.

WebSphere Message Broker also supports the WBI Adapters. These adapters provide support for other transport protocols and, for the purpose of this document, are classed as protocol bridges.

Furthermore, WebSphere Message Broker interacts with WebSphere Transformation Extender for Message Broker, which provides an additional set of connectors, and with WebSphere Message Broker File Extender, which brings support for file access to WebSphere Message Broker.

6.2.3 Message models and transformation

Much of the business world relies on the exchange of information between applications. The information is contained in messages that have a defined structure that is known and agreed upon by the sender and the receiver. Applications typically use a combination of messages, including those that are defined by the following structures or standards:

► C and COBOL data structures
► Industry standards such as SWIFT or EDIFACT
► XML DTD or schema

You can model a wide variety of the message formats that can be understood by WebSphere Message Broker message flows. When the message format is known, the broker can parse an incoming message bit stream and convert it into a logical message tree for manipulation by a message flow. After the message has been processed by the message flow, the broker converts the message tree back into a message bit stream.

Here is an overview how a message bit-stream is parsed when it reaches the input node of a message flow. The message is constructed as a logic tree structure and in preparation for processing by the message flow. Finally, the message is flattened again and sent out to the outside world.
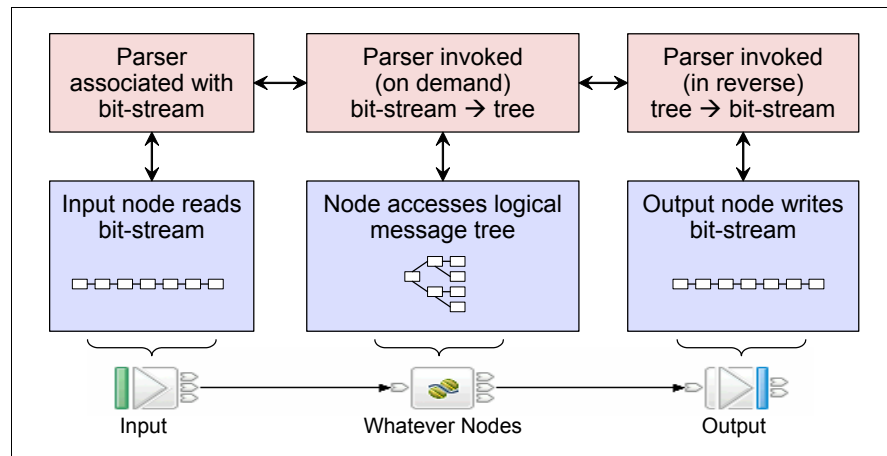
Figure 6-2   Message model and transformation overview

WebSphere Message Broker typically supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. An example of a self-defining message format is XML. In XML the message itself contains metadata as well as data values, enabling an XML parser to understand an XML message even if no model is available. Most message formats, however, are not self-defining. As examples, a binary message originating from a COBOL program and a SWIFT formatted text message do not contain sufficient metadata to enable a parser to understand the message. The parser must have access to a model that describes the message to parse it correctly.

To speed up creation of message models, importers are provided that take metadata such as C header files, COBOL copybooks, XML Schema and DTDs, and WSDL files and create message models from that metadata. Alternatively, IBM has pre-built models for common industry standard message formats such as SWIFT, EDIFACT, X12, FIX, HL7, TLOG, and so on.
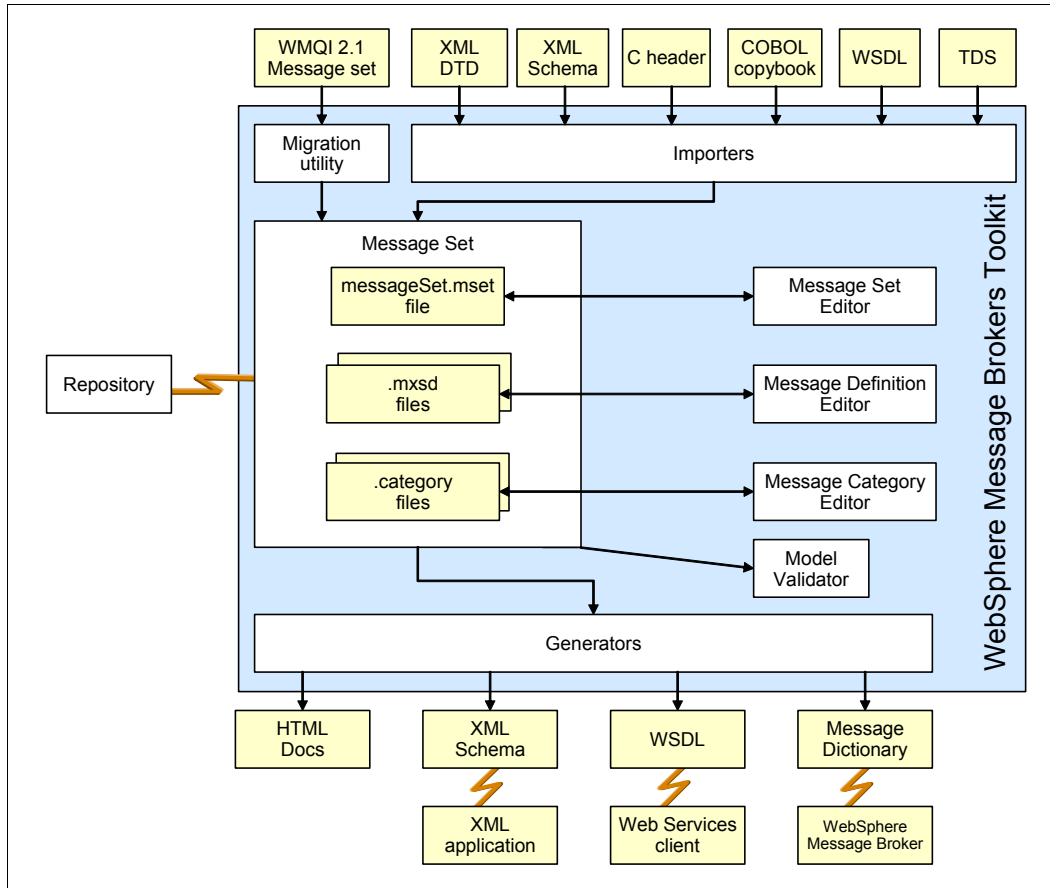
Figure 6-3   Message modeling support in WebSphere Message Broker

Table 6-1 lists the message domains that WebSphere Message Broker supports.

Table 6-1   WebSphere Message Broker supported message domain

Message model	General usage
MRM	For modeling a wide range of messages including XML, fixed-format binary, and formatted text, MRM can also be used to handle JMS byteMessage and textMessage accepted by JMSInput node.

Message model	General usage
XMLNSC, XMLNS, or XML	For messages conforming to the W3C XML standard, including XML over JMS textMessage.
JMSMap or JMSStream	For messages produced by the WebSphere MQ implementation of the Java Messaging Service standard, JMSInput node will reformat the JMS MapMessage and StreamMessage to XML.
IDOC	For messages in SAP IDoc format.
MIME	For handling multipart MIME messages such as SOAP with attachments or RosettaNet.
BLOB	The BLOB message domain includes all messages with content that cannot be interpreted and subdivided into smaller sections of information.

With WebSphere Transformation Extender for Message Broker, WebSphere Message Broker extends its capability with enhanced data format processing and support for industry standard data formats.

Once the inbound message has been constructed as an internal logic tree structure, WebSphere Message Broker provides different ways to process it:

► ✓ Validate node

Use the validate node to check that the message that arrives on its input terminal is as expected using the message template properties such as message domain, message set, and message type. You can also check that the content of the message is correct by selecting message validation.

The checks that can be performed depend on the domain of the message. See Table 6-2.

Table 6-2   Validation types

Check	Domain
Check message domain	All domains
Check message set	MRM and IDOC only
Check message type	MRM only
Validate message body	MRM and IDOC only

You can use the validate node to confirm that a message has the correct message template properties and valid content before allowing the message

into the rest of the flow. This means that subsequent nodes can rely upon the message being correct without doing their own error checking.

► Compute node

Use the Compute node to construct one or more new output messages. These output messages might be created by modifying the information that is provided in the input message, or the output messages might be created using only new information, which might (or might not) be taken from a database. Elements of the input message (for example, headers, header fields, and body data), its associated environment, and its exception list can be used to create the new output message.

You specify how the new messages are created by coding ESQL in the message flow ESQL resource file. Extended Structured Query Language (ESQL) is a programming language defined by WebSphere Message Broker to define and manipulate data in any message model within a message flow. You can both create and modify the components of the message using ESQL expressions, and can refer to elements of both the input message and data from an external database. An expression can use arithmetic operators, text operators (for example, concatenation), logical operators, and other built-in functions.

Use the Compute node to:

- Build a new message using a set of assignment statements.
- Copy messages between parsers.
- Convert messages from one code set to another.
- Transform messages from one format to another.

You define the ESQL statements in a module associated with this node in the ESQL .esql file associated with this message flow. You must create this file to complete the definition of the message flow. Example 6-1 is a sample skeleton of the ESQL procedure.

Example 6-1   An ESQL skeleton for the Compute node

```
CREATE COMPUTE MODULE SampleFlow_Compute
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      -- CALL CopyMessageHeaders();
      -- CALL CopyEntireMessage();
      RETURN TRUE;
   END;

   CREATE PROCEDURE CopyMessageHeaders() BEGIN
      DECLARE I INTEGER;
      DECLARE J INTEGER;
      SET I = 1;
```

```
        SET J = CARDINALITY(InputRoot.*[]);
        WHILE I < J DO
            SET OutputRoot.*[I] = InputRoot.*[I];
            SET I = I + 1;
        END WHILE;
    END;

    CREATE PROCEDURE CopyEntireMessage() BEGIN
        SET OutputRoot = InputRoot;
    END;
END MODULE;
```

► ![icon] Java Compute node

The Java Compute node is a new general-purpose programmable node. It is based on J2SE™ 1.5 and supports XPath 1.0 to query on all four message trees in the message assembly. You can also use the Configuration Manager Proxy API to perform administrative tasks on the broker.

You can examine an incoming message in any domain and, navigating the message tree, optionally create a new copy of the message to make a change on it. Depending on its content, you can propagate the message to one of the node's two output terminals. The node behaves in a similar way to a Filter node, but uses Java instead of ESQL to decide which output terminal to use. Or you can create and build a new output message that is totally independent of the input message.

Database support is also provided via two routes. You can use the standard JDBC type 4 non-XA driver to access the database, or you can use the MbSQLStatement API to access a database that actually uses the ODBC driver under the cover, with the full transaction support.

The Java code that is used by the node is stored in an Eclipse Java project, and full IDE support of Eclipse Java Development Tools (JDT) is supplied for the development. Full debugging support is provided by utilizing Eclipse Java debugger integrated with the Message Flow Visual Debugger.

Example 6-2 provides the code skeleton for a Java Compute node.

Example 6-2   Code skeleton for Java Compute node

```
import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.*;

public class SamplelNode extends MbJavaComputeNode {

    public void evaluate(MbMessageAssembly assembly) throws MbException
{
```

```
        MbOutputTerminal out = getOutputTerminal("out");
        MbOutputTerminal alt = getOutputTerminal("alternate");

        MbMessage message = assembly.getMessage();
        // -----------------------------------------------------------
        // Add user code below

        // End of user code
        // -----------------------------------------------------------

        // The following should only be changed
        // if not propagating message to the 'out' terminal
        out.propagate(assembly);
    }
}
```

► Mapping node

Use the Mapping node to construct one or more new messages and populate them with new information, with modified information from the input message, or with information taken from a database. You can modify elements of the message body data, its associated environment, and its exception list.

Use the Mapping node to:

– Build a new message.
– Copy messages between parsers.
– Transform a message from one format to another.

The components of the output message can be defined using mappings that are based on elements of both the input message and data from an external database. You create the mappings associated with this node in the mapping file associated with this node by mapping inputs (message or database) to outputs. You can optionally modify the assignments made by these mappings using supplied or user-defined functions and procedures. For example, you can convert a string value to uppercase when you assign it to the message output field.
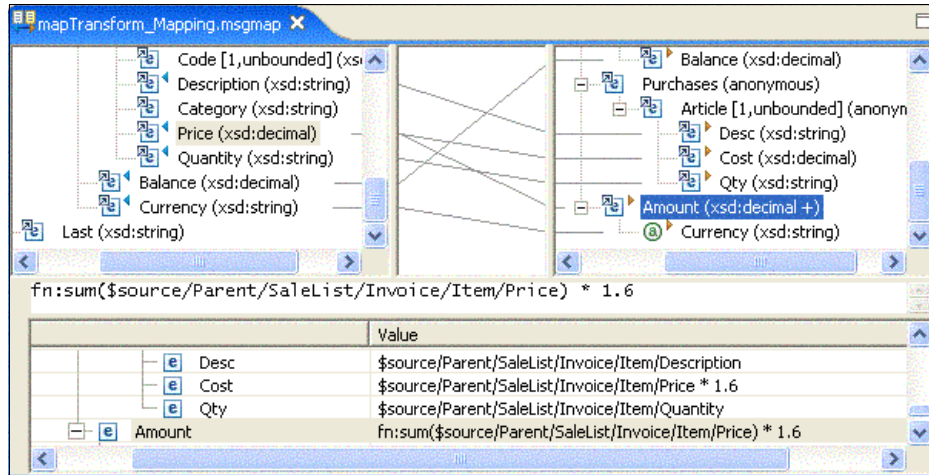
Figure 6-4   Image of the mapping editor

The Mapping node provides a graphical interface for users with limited programming skills to manipulate messages, but provides less flexibility than the Compute node or Java Compute node. There is a very nice graphical debugger to debug the maps in run time. The Mapping node will generate ESQL at deployment time.

The source and target must be defined by a database schema or MRM message set definition. The mapping node cannot operate on other message domains. This support is expected to expand in future releases.

► XMLTransformation node

XMLTransformation uses eXtensible Stylesheet Language for Transformations (XSLT) to transform an XML message to another format (which may or may not be XML) according to the rules provided by an eXtensible Stylesheet Language (XSL) style sheet.
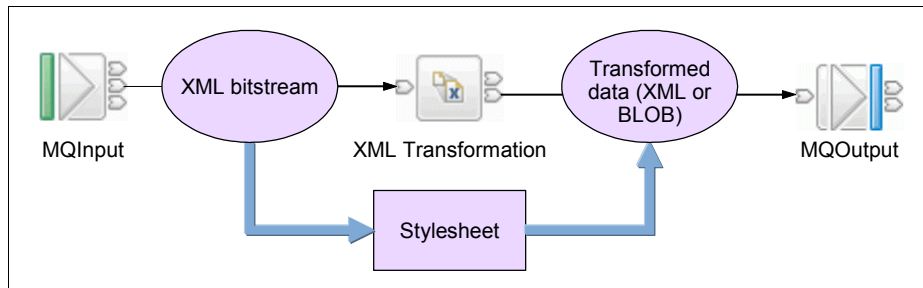
Figure 6-5   XMLTransformation node overview

The XMLTransformation node will be generated as Java at deployment time. WebSphere Message Broker V6 supports compiled stylesheets, which improves the performance of the transformation.

The XMLTransformation nodes depend on the inbound being XML. It does not have to be in one of the XML domains, but it must be a well-formed XML bitstream. The output will be in the BLOB domain.

► WebSphere Transformation Extender for Message Broker

Refer to 3.1.8, "WebSphere Transformation Extender for Message Broker" on page 46.

6.2.4 Dynamic message routing

In a service-oriented architecture, an ESB should have the capability to decide the message destination dynamically, thus providing the location transparency of services. With WebSphere Message Broker, the routing can be based on message header or content, or based on customer-defined policy.

WebSphere Message Broker can access the message header attributes at run time, making it possible to carry the routing information in the message header like MQRFH2 or MQMD, and route the message based on the header information.

It is worth mentioning here that in WebSphere Message Broker V6, one of the significant ESQL enhancements is the new data types: ROW and SHARED. These are very useful if the solution is designed to route the message to different

destinations by looking up a rule table. Usually our approach is to save the rule table in a database so that it is accessible to all the message flows. However, a better alternative in Version 6 is to use the SHARED ROW variable (Example 6-3). You can set up the lookup table in memory, where it can be shared by all threads of the message flow instances. Memory access also avoids the overhead of accessing a database for each incoming message.

Example 6-3   Code snippet: usage of shared long-live memory cache in ESQL

```
-- A shared variable table that can be used by instances of a flow
declare CacheQueueTable SHARED ROW;
-- User code to setup the CacheQueueTable and extract routing criteria
......
-- Get destination dynamically from the in-memory cache tables
SET OutputLocalEnvironment.Destination.MQDestinationList.DestinationData[]
=
( SELECT   S.QUEUE_MANAGER as queueManagerName,
           S.QUEUE_NAME as queueName
   FROM CacheQueueTable.DestinationData[] as S
   WHERE    S.VARIABLE2 = Variable2 and
            S.VARIABLE3 = Variable3
);
```

Note: For more information about the SHARED ROW data type, refer to the product information center and also the message routing sample in the sample gallery.

Several decision-making nodes are supplied to enhance the routing mechanism:

► Filter node

   Use the Filter node to route a message according to message content. You define the route by coding a filter expression in ESQL. You can include elements of the input message or message properties in the filter expression. You can also use data held in an external database to complete the expression. The output terminal to which the message is routed depends on whether the expression evaluates to true, false, or unknown.

► FlowOrder node

   Use the FlowOrder node to control the order in which a message is processed by a message flow. The input message is propagated to the first output terminal, and the sequence of nodes connected to this terminal processes the message. When that message processing is complete, control returns to the FlowOrder node. If the message processing completes successfully, the input message is propagated to the second output terminal and the sequence of nodes connected to this terminal processes the message.

► Label node and RouteToLabel node

Use the Label node in combination with a RouteToLabel node to dynamically determine the route that a message takes through the message flow. The RouteToLabel node interrogates the LocalEnvironment of the message to determine the identifier of the Label node to which the message must next be routed.

Another common scenario is to use these nodes together with the WebSphere Service Registry and Repository plug-in node to provide dynamic routing capability base on the service location from the service repository.

► TimeoutControl node and TimeoutNotification node

The TimeoutControl node receives an input message that contains a time-out request, validates the request, stores the message, and propagates the message without any changes to the next node in the message flow. Example 6-4 shows a sample time-out request message in XML.

Example 6-4   A sample timeout request message in XML

```
<TimeoutRequest>
   <Action>SET | CANCEL</Action>
   <Identifier>String (arbitrary, only alphanumerics)</Identifier>
   <StartDate>String (TODAY | yyyy-mm-dd)</StartDate>
   <StartTime>String (NOW | hh:mm:ss)</StartTime>
   <Interval>Integer (seconds)</Interval>
   <Count>Integer (greater than zero or -1)</Count>
   <IgnoreMissed>TRUE | FALSE</IgnoreMissed>
   <AllowOverwrite>TRUE | FALSE</AllowOverwrite>
 </TimeoutRequest>
```

The TimeoutNotification node is an input node that can be used in one of two ways:

– Paired with one or more TimeoutControl nodes

The TimeoutNotification node processes timeout request messages that are set by the TimeoutControl nodes with which it is paired, and propagates copies of the messages (or selected fragments of the messages) to the next node in the message flow.

– Standalone

Generated messages are propagated to the next node in the message flow at time intervals that are specified in the configuration of this node. You can accomplish automatic time outs with a single Timeout Notification node running in automatic mode. You must provide an interval in seconds as a configuration parameter to determine the period between adjacent flow invocations.

Use of automatic time outs is great for implementing time-based event-driven activities (for example, a *heartbeat* flow used to monitor the status of a system or a maintenance flow used to archive a database). You can also use both nodes to implement event-driven Service Level Agreements (SLAs), that is, to check that a certain event has occurred after a required time. Otherwise start performing the exception processing.

► Validate node

You can use the validate node to ensure that the message is routed appropriately through the message flow. For example, you can configure it to direct a message that requests stock purchases through a different route from that required for a message that requests stock sales.

► XML Validator node

The XML Validator node provided by SupportPac™ IA9A validates an XML message against an XML schema. If the message conforms to the schema, the message is passed to the Out terminal unchanged. However, if errors are encountered, these are added to the environment tree, and passed to the invalid terminal, along with the unchanged message. The XML Validator node checks the bitstream of a message, as opposed to the logical structure, and therefore supports any message format (for example, XML, XMLNS, BLOB, and so on). Configurable parameters are supplied to allow the schema defined in the XML document to be overridden, and can be set either via the node properties or at run time, by the use of Environment/LocalEnvironment variables.

This SupportPac could be used as part of a Web services message flow if there is a requirement to validate incoming or outgoing XML messages against schemas. The ability to validate against a different schema for each message enables the node to be used in situations where there are a large variety of messages passing through the flow. For instance, if the broker is acting as a Web services intermediary, several types of messages could be validated by one message flow using this node.

It is also useful when the content of a schema changes frequently, such as during application development stages, as no re-deploy of the message flow is required to use the changed schema.

Note: For more information about message routing nodes and capability, refer to the WebSphere Message Broke information center.

In summary, routing and transformation logic can be performed in WebSphere Message Broker versus in the applications, enabling you to achieve separation of concerns. The endpoint applications contain the business logic that applies the business processing rules that are important to your enterprise, while operations

such as protocol and message transformation or routing are contained solely within the WebSphere Message Broker environment, making your IT infrastructure much more flexible.

6.2.5 Custom mediation support

WebSphere Message Broker supports customized mediation capability through the use of the following built-in or SupportPac nodes:

► Database nodes

   The database nodes can be used to enrich messages at run time from an ODBC data source or to perform operation on database data. These nodes provide a flexible interface with a wide range of functions. You can specify transaction behavior in the message flow specifying whether changes are committed immediately or after the entire message flow is successful.
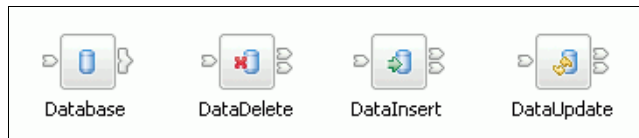
Figure 6-6   Built-in database-related nodes

► File and dataset related nodes

   WebSphere Message Broker provides a set of SupportPac nodes that can be used to enrich messages or used as logging facilities for monitoring purpose.

   – QSAM dataset adapter

      The QSAM dataset adapter nodes provided in SupportPac IA11 can be used to operate on the QSAM (for example, sequential dataset on the mainframe).
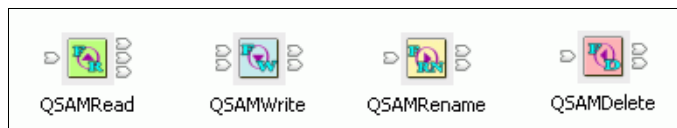
Figure 6-7   QSAM dataset related nodes

– VSAM dataset adapter nodes

The VSAM dataset adapter is provided in SupportPac IA13 can be used to operate VSAM dataset on the mainframe.

Figure 6-8   VSAM dataset related nodes

– WebSphere Message Broker File Extender

This provides a set of nodes for the WebSphere Message Broker V6 to provide capabilities into the domain of files, allowing users to natively exploit huge amounts of information from the broker that many enterprises store in flat files on distributed platforms.

> **Note:** For more information about WebSphere Message Broker File Extender, refer to the product page:
>
> http://www-306.ibm.com/software/integration/wbimessagebroker/file extender/v5/index.html

► Other supplied nodes

An LDAP plug-in node provided by SupportPac IA08 can be used to perform an LDAP lookup using certain parameters in the message, and then to enrich the message using the result from the LDAP server. You can also use the Sendmail node provided by SupportPac IA07 to construct and send e-mail using the content in an XML message.

► Custom extensions

WebSphere Message Broker provides extensive APIs to allow users to write custom extensions to perform customer mediation and message model handling. These include:

– Writing user-defined nodes in Java
– Writing user-defined nodes and parsers in C

> **Note:** For more information about how to write a custom extension for WebSphere Message Broker, refer to the "Developing user-defined extensions" chapter in the WebSphere Message Broker information center.

6.2.6 Interaction pattern support

When two or more participants interact with others, they always fall into one or more of these interaction patterns:

- ► One-way interaction
- ► Request-response
- ► Aggregation
- ► Publish/subscribe (pub/sub)

One-way interaction

Fire-and-forget is one common scenario in asynchronous mode. The sender sends the notification to the receiver, and that is all. The sender does not need to wait for the reply to continue his job. For example, the broker in the airline control system periodically sends the latest flight information to a portlet application to display on the Web site.
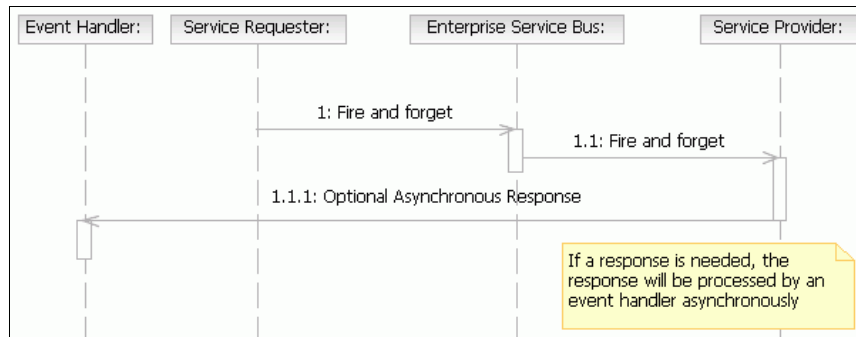
Figure 6-9   Sequence diagram for one-way interaction pattern

Sometimes when necessary, the receiver needs to send feedback or a response to the sender. In this situation, the requester must implement an event handler to capture the response when it comes back asynchronously and to correlate each response with the corresponding request.

WebSphere Message Broker supports asynchronous service interaction using MQ or JMS transportation. It also supports one-way Web services using SOAP over HTTP 1.1 as a service requester and service provider. The requester does not need to wait for a valid SOAP reply, but a simple 200 or 202 HTTP response status code is the indicator that the request has been accepted by the HTTP layer. Figure 6-10 is a simple example.
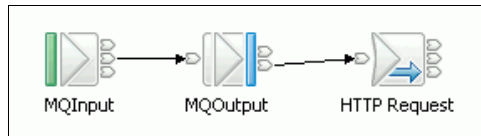
Figure 6-10   Sample message flow for asynchronous interaction pattern

In the example, the entire flow acts as a Web service provider that only accepts requests that come in to the MQ queue. Within the flow, it sends two Web service requests using an MQOutput node and an HTTPRequest node without waiting for the reply. However, as HTTP is a synchronous protocol, the HTTPRequest waits for a quick 202-Accept or 200-OK empty HTTP reply from the one-way service provider. The HTTPRequest node treats the 200 series status codes as a success and the response is routed to the out terminal of the node.

Request-response

This is possibly the most widely used scenario in an asynchronous or synchronous mode. When it is synchronous, the requester sends the request to the receiver or server application, waits for the reply, and then goes on with its work. The requester must wait for the reply or feedback. In no reply is received after a certain period of time, a time-out exception will be raised to handle the error. For example, the customer submits a balance query command on the ATM and waits for the reply from the core-banking CICS system.
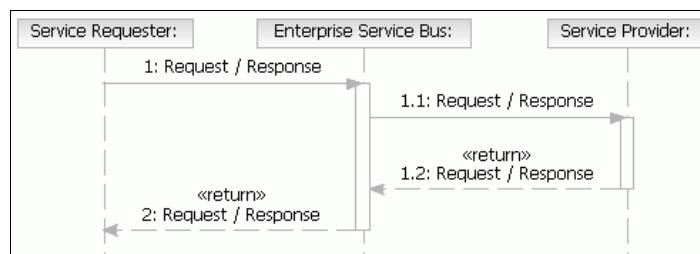
Figure 6-11   Sequence diagram for request-response interaction pattern

WebSphere Message Broker provides options for synchronous interaction both as a requester and a provider over a variety of transport protocols. Figure 6-12 shows a sample of a request-response implementation with WebSphere Message Broker.

Figure 6-12   Sample message flow for synchronous interaction pattern

In this message flow, WebSphere Message Broker is itself a Web service provider over the MQ transportation protocol. It also acts as a Web service requester within the flow twice, once over HTTP and a second time over MQ. All interactions are synchronous.

Aggregation

Aggregation is the generation and fan-out of related requests derived from a single input message and the fan-in of the corresponding replies to produce a single aggregated reply message. For example, a customer places an order from the Internet. The broker constructs several independent requests to all of the suppliers involved in fulfilling the order, and then consolidates the results into one response to the customer about the estimated delivery time.
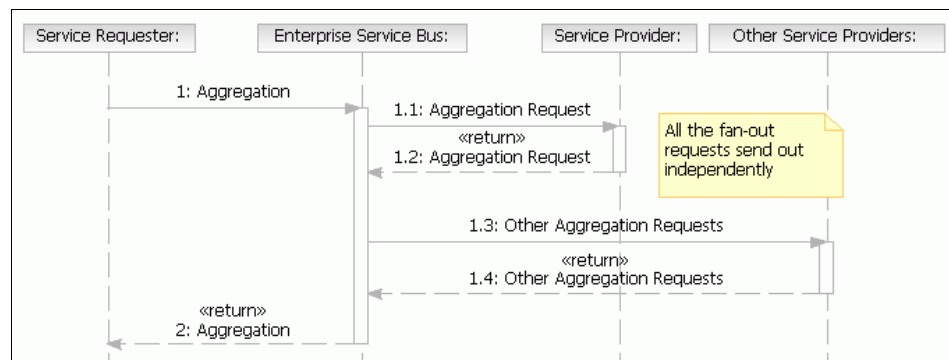
Figure 6-13   Sequence diagram for aggregation interaction pattern

Several aggregation nodes are supplied to provide this capability. They are:

► AggregateControl node
► AggregateRequest node
► AggregateReply node

A general scenario, shown in Figure 6-14, contains a request fan-out message flow and a response fan-in message flow.
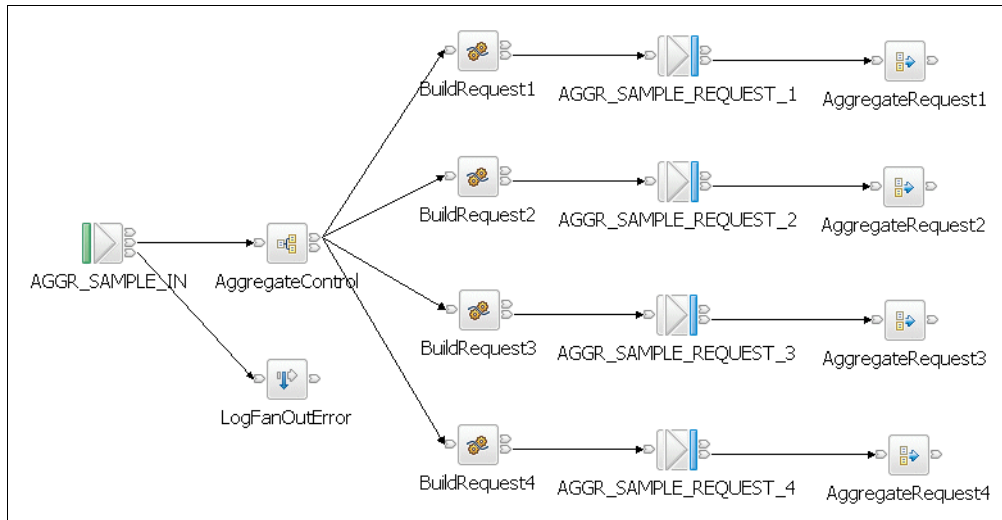
Figure 6-14   Aggregation request fan-out message flow

When a composite request comes in and arrives at the AGGR_SAMPLE_IN input node, it is manipulated and decomposed into four separate requests that are sent to different services to get fulfilled. Some control information about the aggregation is saved in the local environment by the AggregationControl node and AggregateRequests node. This information is necessary for aggregating the replies by the AggregateReply node in the fan-in message flow below.
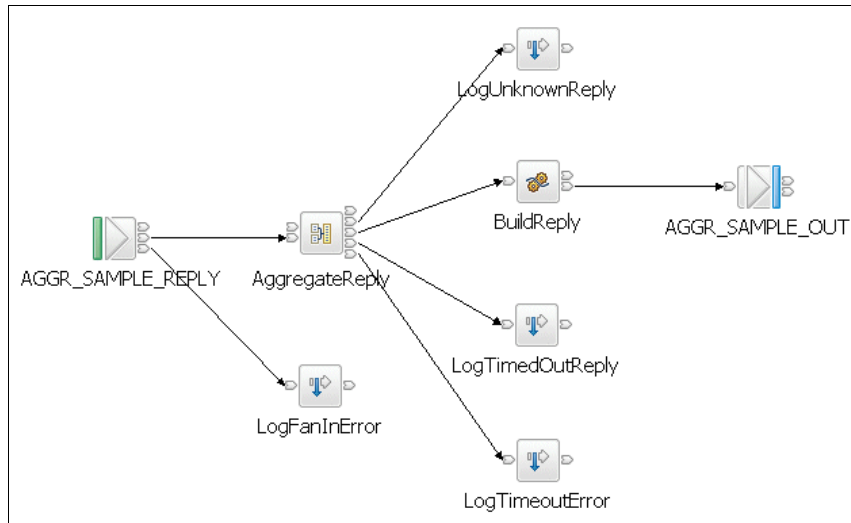
Figure 6-15   Aggregation reply fan-in message flow

Once the requests are sent to their corresponding service provider, the AggregateReply node keeps track of the responses. After all replies come back to the AGGR_SAMPLE_REPLY input node (within the specified time-out period), control is passed to the BuildReply node to construct the final composite reply message, which is sent back to the service requester.

In WebSphere Message Broker V6, the aggregate nodes are based on MQ store for aggregation status. Database tables are no longer required to keep track of the aggregation state, thus improving the aggregation performance.

Note: For more information about aggregation nodes, refer to Aggregation Sample in the Sample Gallery of WebSphere Message Broker Toolkit.

Publish/subscribe (pub/sub)

Publish/subscribe involves the sending of data by one application to any number of receivers. This introduces a much more complex relationship between the sender and the receiver, where there is an abstraction between them so that they are unaware of each other. This abstraction layer is usually provided by a broker

service, which manages incoming data (publications) from sending applications (publishers) and matches them to interested receivers (subscribers). The broker then forwards the data to the registered subscribers.
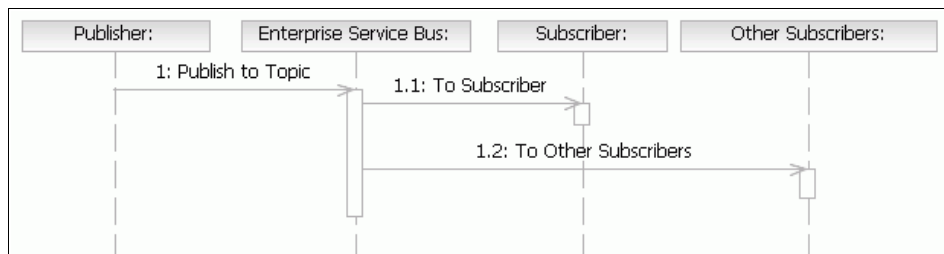
Figure 6-16   Sequence diagram for pub/sub interaction pattern

The correlation between publishers and subscribers is defined by using topics, which are contained in a hierarchical arbitrary namespace (that is, application defined). Subscribers register interest in a topic (or topics) with the broker, and publishers publish messages using those topics. The names of topics usually relate to the information carried within them. The broker routes publications on particular topics to all those subscribers that have registered an interest in those topics.

For example, a broker publishes the stock price for different companies using different topics, one per company. Users only subscribe to the topic that corresponds to the company they are interested in. The users receive a notice when information arrives on the topic.

In a WebSphere Message Broker scenario, typically a publish/subscribe system has more than one publisher and more than one subscriber, and often more than one broker. An application can be both a publisher and a subscriber. The publisher application generates a message that it wants to publish and defines the topic of the message. A message flow running in the broker retrieves the message from its input node and passes the message to a Publication node for distribution to all subscribers that have registered an interest in the topic. A user name server is provided to achieve the security control on topics.

The input node might be one of the following built-in nodes:

► An MQInput node, which represents a WebSphere MQ queue

► A Real-timeInput node, which receives messages from a JMS application using WebSphere MQ Real-time Transport

► SCADAInput, which represents a SCADA input port

A subscriber registers a request for a publication by specifying one of the following items:

► The topics of the published messages that it is interested in.

► The subscription point from which it wants to receive publications.

► The content filter that should be applied to the published message.

► The name of the queue (known as the subscriber queue) on which publications that match the criteria selected should be placed. This can be the name of a cluster queue so that publications can be distributed to clustered subscribers.

6.2.7 Integration with other enterprise information systems

An ESB should have the capability to interact with the existing enterprise information systems within the enterprise. There are many adapters available for interaction with other components within the environment, in particular *off the shelf* applications like Siebel®, PeopleSoft®, SAP R/3®, and also IBM enterprise environments, for example, CICS Transaction Server and IMS.

WebSphere Message Broker supports integration with WebSphere Business Integration Adapters (JMS based) to interact with existing enterprise information system.

WebSphere Message Broker also provides the CICSRequest node in SupportPac IA12 to synchronously execute CICS programs from a message flow. The node takes an input message tree and executes a CICS External Call Interface (EXCI) request.

By configuring the properties of the CICSRequest node, you specify the communication area (known as COMMAREA) and the CICS program name used in the CICS EXCI request. The COMMAREA returned is placed in the message tree. The location of the sent COMMAREA in the input message tree and the returned COMMAREA in the output message tree may be specified by properties of the node.

Other CICS control parameters are also specified by properties of the node, for example, the CICS user ID, program name and transaction identifier to be used.

Errors in the CICS programs, or in the EXCI interface, are reported back to the message flow and may optionally be directed to an error terminal or result in a failure.

The CICSRequest node is supported only by brokers running on a z/OS system.

6.2.8 Quality of service (QoS) support

From an architecture point of view, an ESB should only provide capability to support service virtualization, but there are always a lot of quality of service issues affecting an ESB. Some important QoS aspects relevant to an ESB are:

► Authentication and authorization
► Non-repudiation and confidentiality
► Transactions
► Various assured delivery paradigms
► Performance and throughput
► High availability
► Logging, metering, and monitoring
► Integration to systems management and administration tooling

More information related to security and transaction features provided by WebSphere Message Broker are discussed in 6.5, "Security considerations" on page 159, and 6.6, "Transaction considerations" on page 162.

6.2.9 Service Registry access

As the number of services in SOA infrastructures increases, it becomes increasingly important to have a system of services that can do the following:

► Find and publish services.
► Manage service life cycle.
► Support policy-driven service interactions.
► Change notification.
► Central service metadata repository.

A Service Registry manages the service metadata, enabling selection, invocation, management, governance, and reuse of services leading to a successful SOA. As the service backbone in a SOA architecture, the ESB should have the full ability to access the Service Registry in the enterprise.

SupportPac IA9L is provided to enable WebSphere Message Broker to access WebSphere Service Registry and Repository from within the message flow, thus providing service location transparence capability and flexible service deployment and management.

> **Note:** For more information about WebSphere Message Broker support on integrating with WebSphere Service Registry and Repository, refer to the *WebSphere Service Registry and RepositoryWebSphere Service Registry and Repository Handbook*, SG24-7386.

6.2.10 Ease of administration

An ESB should be easy to manage. WebSphere Message Broker includes the Message Brokers Toolkit, an administration tool that provides both development and runtime administration capabilities. See Figure 6-17.
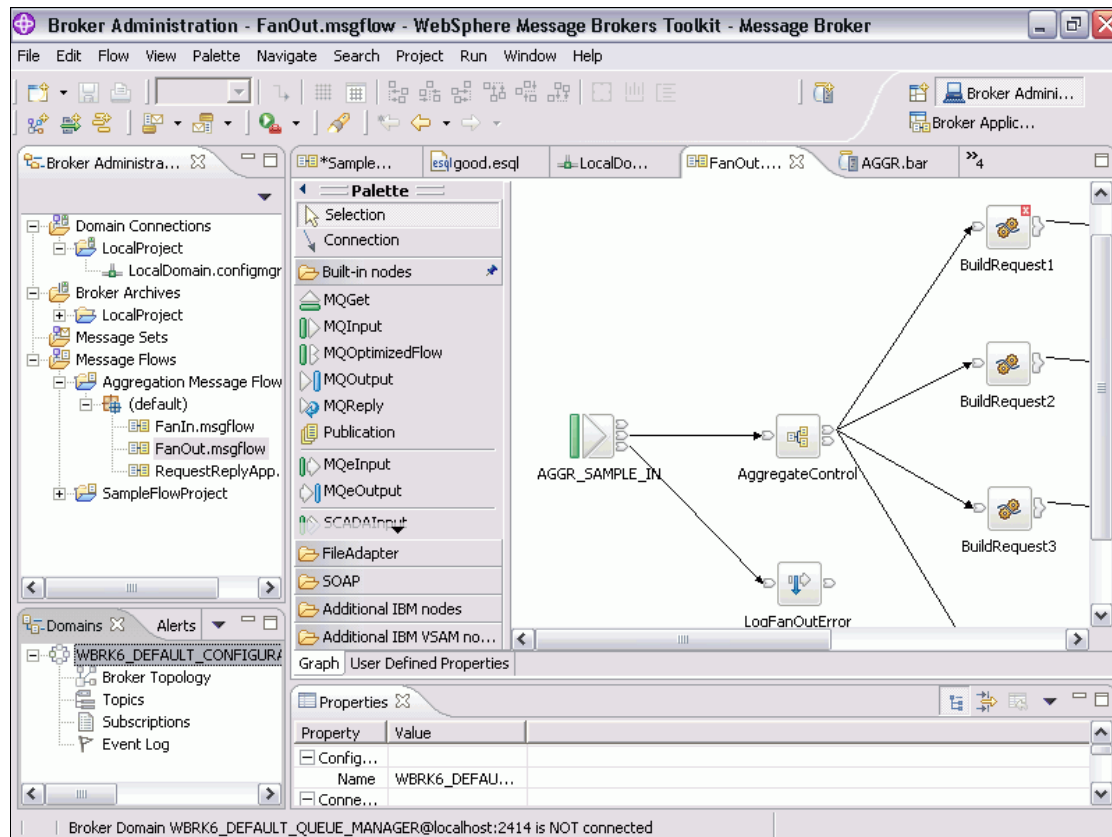
Figure 6-17   Message Brokers Toolkit

In addition to using the graphical interface of the Message Brokers Toolkit, you also have the following options:

► SupportPac IS02: WebSphere Message Broker Administrator Explorer Plug-in for MQ Explorer

This SupportPac enables the WebSphere MQ Explorer to perform some common WebSphere Message Broker administrative tasks from within the MQ Explorer without installing the Message Brokers Toolkit, but provides a more friendly user interface than the command line.

For more information see:

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24012457&loc=en_US&cs=utf-8?=en

► Configuration Manager Proxy API

The Configuration Manager Proxy (CMP) is an application programming interface that your applications can use to control broker domains through a remote interface to the Configuration Manager. More information about the CMP can be found in the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/topic/com.ibm.etools.mft.doc/ae20620_.htm

6.3 Web service support in WebSphere Message Broker

WebSphere Message Broker, as a universal message hub, supports many Web service standards. With WebSphere Message Broker, you can develop message flows to work with SOAP messages over any of the supported transports, including HTTP or HTTPS, JMS, or WebSphere MQ transportation.

6.3.1 Choose the message domain for SOAP

Because SOAP is an XML message format, your message flow would use one of the message domains XMLNS, XMLNSC, or MRM. If your Web service uses SOAP with attachments, you would have to use the MIME domain. Both SOAP 1.1 and 1.2. are supported by WebSphere Message Broker V6.

MRM domain

In previous versions of WebSphere Message Broker, the term MRM, short for Message Repository Manager, represented the component of the Configuration Manager that managed message set definitions in the message repository. In Version 6.0, there is no Message Repository Manager. In V6, MRM is just a name for a message domain. MRM can be used for modeling a wide range of messages. Its features includes:

► Support for modeling messages from applications written in C, COBOL, PL/I, and other languages, using the Custom Wire Format (CWF) physical format. This support includes the ability to create a message model directly from a C header file or COBOL copybook.

► Support for modeling XML messages, including those that exploit XML namespaces, using the XML Wire Format (XML) physical format. This support includes the ability to create a message model directly from an XML DTD or XML schema file.

► Support for modeling formatted text messages, perhaps with field content identified by tags or separated by specific delimiters or both, using the Tagged Delimited String Format (TDS) physical format. This includes industry standards such as HL7, SWIFT, EDIFACT, and X12.

► Easy transformation from one physical format to another. For example, a model can be created by importing a C header file, and the equivalent XML schema can be generated for use by a different application.

► The MRM domain comes with a specialized parser that is used by runtime products such as WebSphere Message Broker. Once you have completed your message set, you generate a message dictionary, which is deployed to the runtime product. The MRM parser then uses the dictionary to parse and write messages that are defined within that message set.

► The MRM parser can perform runtime validation of messages against the deployed dictionary.

XML domains

The XML domains can be used for modeling messages conforming to the W3C XML standard.

► XMLNSC domain

This domain may be used for any XML message, including those that use XML namespaces. The accompanying parser uses XML4C under the cover and reduces the amount of memory taken up by the logical message tree created from the parsed message. Its default behavior is to discard non-significant whitespace and mixed content, comments, processing instructions, and embedded DTDs, though controls are provided to retain mixed content, comments, and processing instructions if desired.

► XMLNS domain

If the XMLNSC domain does not meet your requirements, use this namespace-aware domain.

► XML domain

This domain is not namespace aware and is provided for compatibility with earlier releases of WebSphere Message Broker.

MIME domain - SOAP with attachments

SOAP with attachments is the W3C standard for Web services that need to incorporate attachments such as image data in their messages. It is based on multipart MIME, in which a message consists of a number of parts, each of a defined type and delimited by a boundary string. The first part is typically a regular SOAP (XML) message and refers to the other parts (the attachments), which may contain any type of data (encoded if necessary).

The broker provides a new MIME domain and parser. The MIME document format and the corresponding logical tree are described in the online help. The payload for each part is held as BLOB data. Canned message definitions support ESQL content assist for MIME when creating message flows. You select the appropriate message definition and import it into a message set using the New Message Definition File wizard and the IBM Supplied Messages option.

Typically you would create a message flow using the MIME domain to parse the outer level MIME envelope and then further parse the resulting parts in ESQL or using a ResetContentDescriptor node. For example, you might use the MRM XML parser to parse the first part of the message as SOAP.

The MIME domain that is implemented in WebSphere Message Broker does not support the full MIME standard, but it does support the MIME formats in use in message-based applications, including SOAP with Attachments and RosettaNet.

Which message domain to choose

Choose the domain that best fits your messaging needs:

► If your messages are in XML, use either the XML Wire Format in the MRM domain or use the XMLNS or XMLNSC domains. Usually, you will find that the MRM domain offers more flexibility, for the following reasons:

– You can generate a message dictionary for use by the MRM parser in WebSphere Message Broker. This enables the MRM parser to interpret the data in an XML message in an advanced manner. For example:

• The dictionary indicates the real data type of a field in the message, instead of always treating it as a character string.

• The MRM parser can validate XML messages against the dictionary.

• Base64 binary data can be automatically decoded.

• Date and time information can be extracted from a data value using a specified format string.

– You have extra control over the rendering of your data. For example, you might have a data field that is rendered as an XML element in one message, and as an XML attribute in another message. Or you could have a data field that is known by a particular name in one message, and a different name in another message. This advanced rendering can be specified using MRM XML Wire Format properties.

– You can share a common logical message structure between physical formats. For example, if you have a message that is created by a COBOL application, you can use an MRM XML Wire Format to easily and quickly define the equivalent XML message.

- If none of the above reasons is your concern and you really need a better performance or less memory consumption, use XMLNSC.

- Never use the XML domain for new message flows. The XML domain is only supplied to provide compatibility.

- If your messages come from an application written in a language such as C, COBOL, or PL/I, or consist of fixed-format binary data (possibly including null-terminated strings), use the Custom Wire Format in the MRM domain.

- If your messages consist of formatted text, perhaps with field content identified by tags or separated by specific delimiters or both, use the Tagged Delimited String Format in the MRM domain.

- If you use JMS messages, use either the XML domain or one of the JMS domains.

- If your message has multiple parts or uses SOAP attachment, use MIME message domain.

Alternatively, you can specify your own domain. You would typically do this if you were using WebSphere Message Broker and had written a user-defined parser to parse your messages instead of using the MRM or XML parsers.

6.3.2 Processing SOAP messages

SOAP consists of a standard XML message wrapper, called an envelope, that embeds payload data within the header and body elements. Example 6-5 shows a simple SOAP message. Typically the business data appears in the mandatory body element and collateral information (transaction IDs, security tokens, and so on) appear in the optional header element. A simple SOAP message is shown in Example 6-5.

Example 6-5   SOAP example

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
   <env:Header>
      <!-Set of headers processed before Body -->
   </env:Header>
   <env:Body xmlns:m="http://www.18m.com/salary">
      <m:GetSalary>
         <m:EmpSerial>902688</m:EmpSerial>
         <m:DeptCode>ZSW</m:DeptCode>
      </m:GetSalary>
   </env:Body>
</soap:Envelope>
```

The SOAP envelope has its own namespace. The WS-I recommends that the application-specific data that occurs in the SOAP body and header are also namespace-qualified.

In this section, we provide several approaches to modelling the data in the SOAP envelope, including the header and payload message in the envelope body, so the message can be accessed and manipulated in a message flow.

When importing WSDL into a message set project, a wizard will help you generate the message model for the SOAP message, but you will find that the entire SOAP body becomes a wildcard element and the internal structure is invisible in the message model (see Figure 6-18). Using the wildcard element is to ensure that at least the SOAP structure can be extracted correctly even if the data in the SOAP body does not match the message definition perfectly. However, this brings us a lot of difficulties when manipulating the SOAP message.
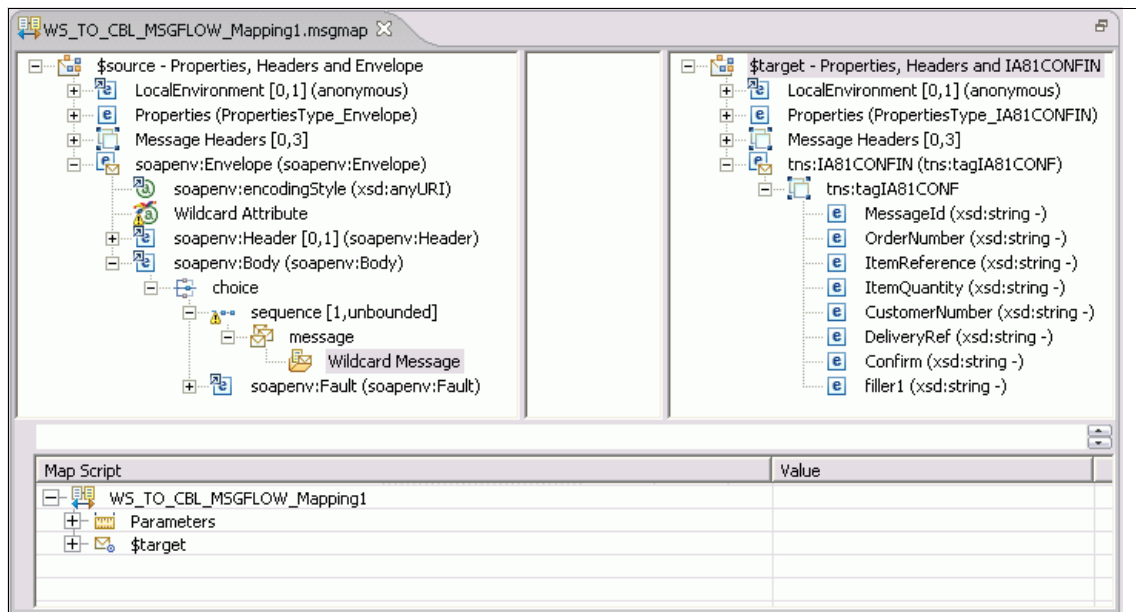
Figure 6-18   SOAP envelope body is a wildcard element in the message tree

There are two ways to make the internal structure available, as discussed next.

Accessing the SOAP body using a submap
This approach creates a submap in the Mapping node editor to map the wildcard message to a message definition somewhere else in the message set.

A new submap can be created by selecting **Create New Submap** from the
context menu of the wildcard mapping element (Figure 6-19).
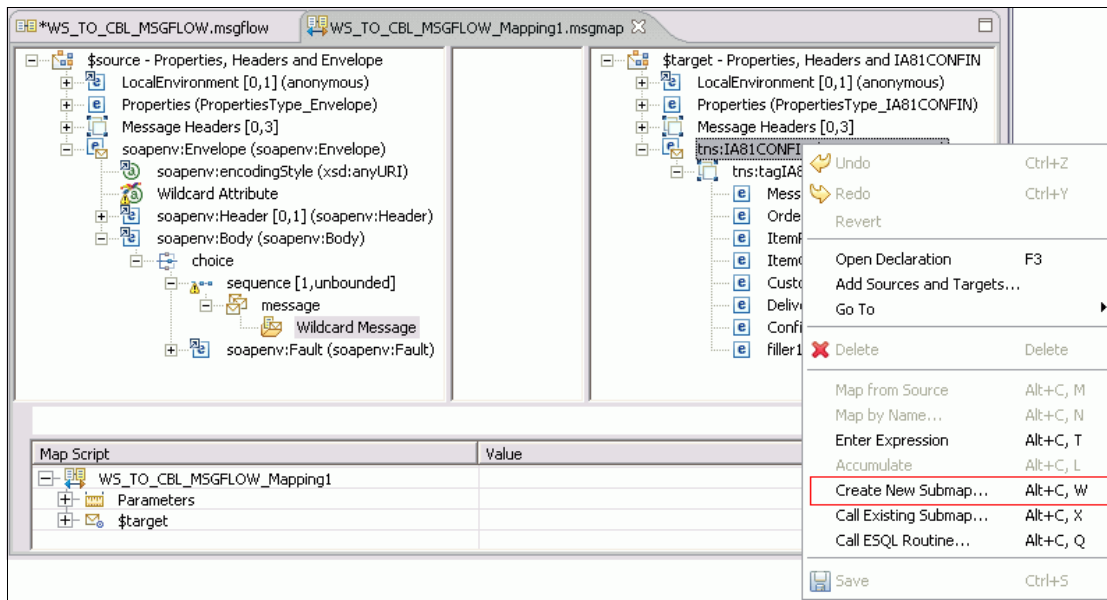
Figure 6-19   Create submap on wildcard element in SOAP message

Then select the concrete item to replace the source wildcard (Figure 6-20).
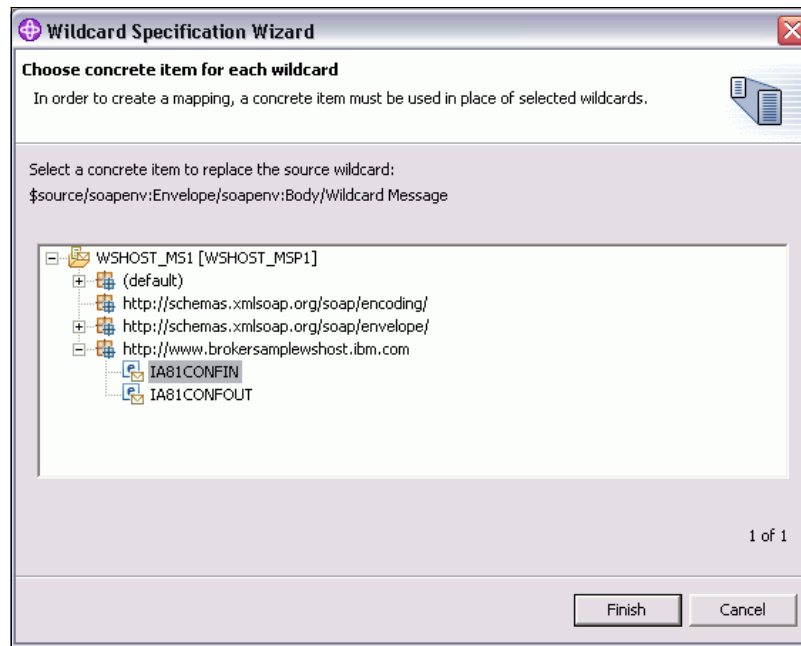
Figure 6-20   Wildcard Specification Wizard

Then you can do the specific mapping in the submap editor. Using this approach you can perform the mapping on the inside structure of the SOAP body.

Modelling SOAP envelope using message definition linking

A second approach is to link the body of the message definition to the whole SOAP message definition.

To do this, open the SOAP message definition file by double-clicking it. The SOAP body is a wildcard element in the message definition (Figure 6-21).
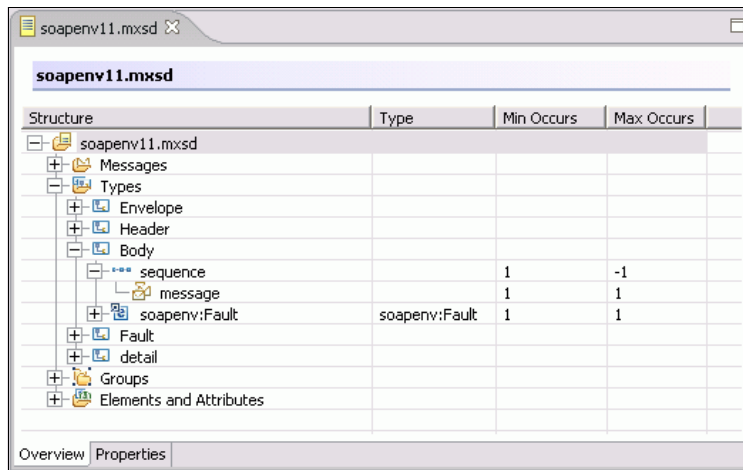
Figure 6-21   The SOAP body is a wildcard message in the message definition

Select the **Properties** tab at the bottom of the message definition editor, and import or include the body message definition file into the SOAP message definition (Figure 6-22). Use import if the two messages are not in the same namespace. Use include if they are in the same namespace.
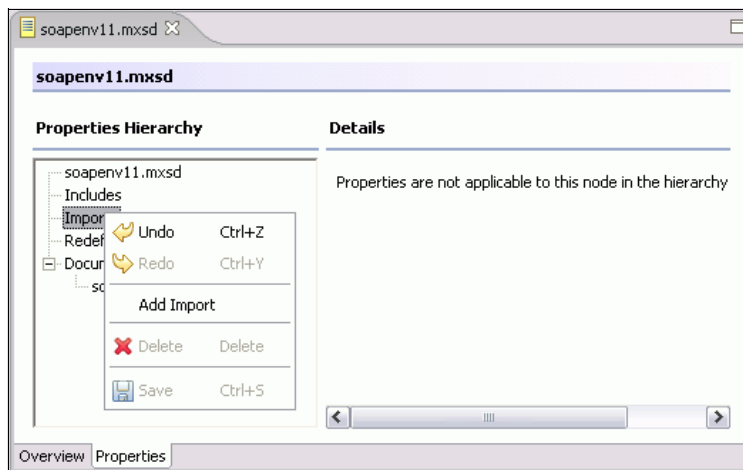
Figure 6-22   Import other message definition to the SOAP message definition

Select the message definition file for the SOAP body (Figure 6-23).
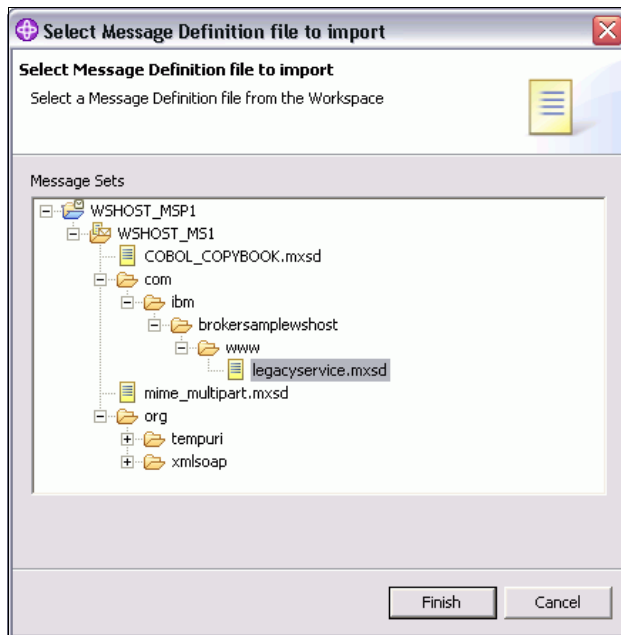
Figure 6-23   This is the SOAP body message definition file

Once the import is complete, return to the Overview tab of the message definition editor and add the body message under the SOAP body wildcard element.

Right-click the message in the body and select **Add Message** from the pop-up menu (Figure 6-24).
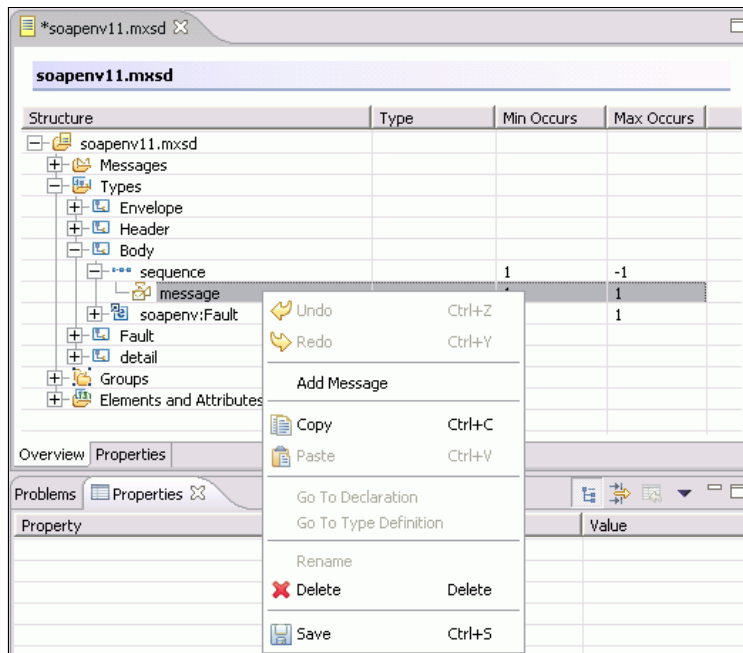
Figure 6-24   Add imported message to the SOAP body

Select the concrete message from the drop-down box.
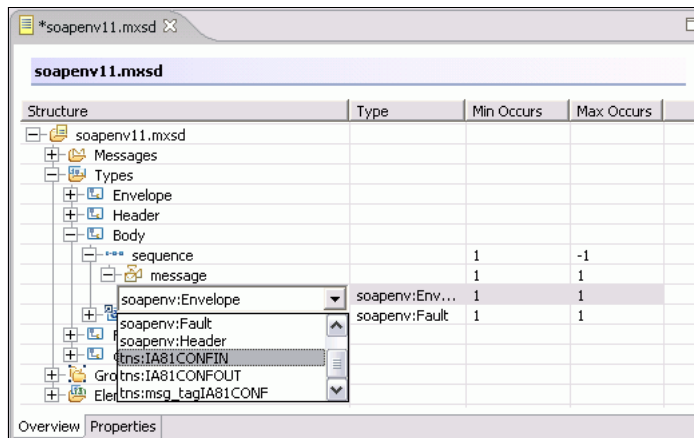
Figure 6-25   Select the concrete message, which is the SOAP body

Now the message definition is complete and both the SOAP header and the SOAP body can be manipulated, allowing you to use any mediation capability to operate on any part of the SOAP message.
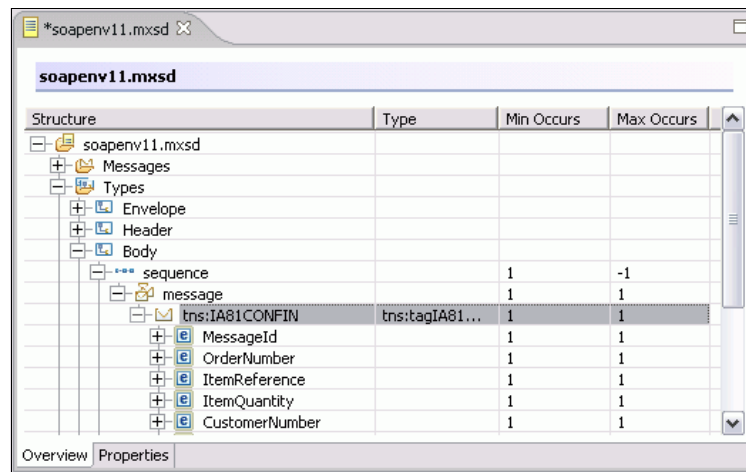
Figure 6-26   The internal structure of the SOAP body is revealed

You must make sure that the incoming SOAP body message matches the definition. Otherwise a validation exception will occur.

Using SOAPExtract and SOAPEnvelop nodes

If you are not interested in the SOAP header information, SupportPac IA9O is exactly what you want. WebSphere Message Broker V6.0 Fixpack 3 is needed to provide runtime support for these two nodes.

Figure 6-27   SOAPEnvelope node and SOAPExtract node

The new SOAPExtract and SOAPEnvelope nodes are message-processing nodes for use within a message flow. Their aim is to simplify the processing of SOAP messages in two respects:

► First, they allow the flow developer to remove or add SOAP envelopes from a SOAP message at either default or user-specified locations. This makes it possible to just deal with the SOAP message payload (child of body).

► Second, the SOAPExtract node allows routing of different SOAP operations to different parts of the message flow, making it easier to handle specific types of SOAP requests without explicit programming logic.

The most common scenarios using these SOAP envelope processing nodes are as follows:

► As a Web service requester

This flow may receives a simple XML input message on an MQInput node, and passes it through a SOAPEnvelope node to create a new SOAP envelop and add it to the message. Then it is passed to a HTTPRequest node, which calls a Web service URI. When the message returns back to this flow it is passed to a SOAPExtract node, which removes the SOAP envelope from the message for simplified processing. It then passes to a Compute node, which removes the HTTP headers, creates an MQMD, and copies across the payload. The output message is then routed to the MQOutput node.
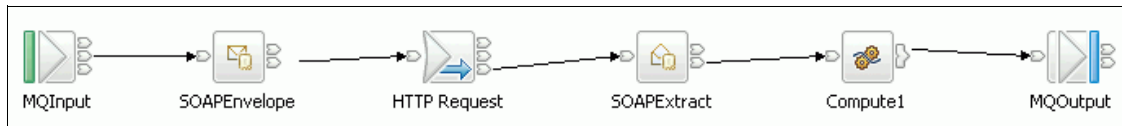
Figure 6-28   Use SOAP nodes to construct a SOAP envelope and call a Web service

► As a Web service provider

This flow can be called by a Web service client. The message is received on the URL of the HTTPInput node and passed to the SOAPExtract node, which removes the SOAP envelop for simplified processing and stores it in the LocalEnvironment. It is then passed to a Java Compute node, which generates an output message by some business logic. This is then passed to a SOAPEnvelope node, which adds the SOAP envelope stored earlier in the Local Environment to the message. Finally, the message is passed to the HTTPReply node, and the message returns back to the requester client.
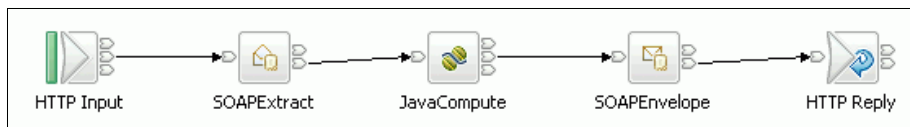
Figure 6-29   Use SOAP nodes to make the message flow a Web service provider

6.3.3 WSDL support

The format of the SOAP messages exchanged between a Web service client and a Web service provider is described using WSDL.

WebSphere Message Broker V6 supports WSDL 1.1. Using the Message Brokers Toolkit, you can import WSDL using the WSDL Importer to create a broker message model. A message model can help you develop your message flows and offers runtime validation if your flow uses the MRM domain. You can also generate a WSDL definition from an existing message model using the WSDL Generator. A WSDL editor is provided for editing WSDL files.

The mapping between the WSDL message definitions and the broker message model depends on the WSDL style, document, or Remote Procedure Call (RPC). In the case of RPC-style WSDL, additional messages are generated based on the WSDL operation names.

Message Broker Toolkit V6.0.2 will support the drag-and-drop operation of a WSDL file to automatically generate the nodes for the processing flow. If the message flow is the Web service provider, it will generate HTTPInput and HTTPOutput nodes linked with the SOAPEnvelope and SOAPExtract nodes. If the message flow is the Web service requestor it will generate the corresponding HTTPRequest node linked with SOAP processing nodes.

Generating and importing WSDL

In WebSphere Message Broker V6, the WSDL generator offers a range of generation options: single and multiple file formats, and document-literal and RPC-literal WSDL styles.

You invoke the WSDL Generator as the New WSDL Definition wizard. The WSDL Generator wizard lets you generate a WSDL definition from a message model. WSDL generation is driven using message categories, and you need to create a message category for each WSDL operation in your Web service:

► For document-style WSDL, the messages you include in your category definitions will be the payload messages expected in your SOAP documents at run time.

► For RPC-style WSDL, the payload of your SOAP body is based on the category name itself (the WSDL operation), and the messages you include in your category definitions define the names of its parameters.

Where possible, use document-style WSDL to improve interoperability.

The enhanced importer of WSDL in the Message Brokers Toolkit can accept a variety of WSDL document styles, as mentioned above. Before importing the WSDL to your message set project, you must make sure that a XML physical format layer has been added to your message set and the namespace support has been enabled.

In order to improve Web services interoperability, you should avoid unnecessary customization of the XML physical format layer for messages participating in Web services processes. Any required SOAP envelope and SOAP encoding message definitions are automatically added to your message set during the import. If required, you can also import these manually via the New Message Definition File wizard by selecting the new IBM-supplied message option.

WS-I compliance validation for WSDL

The W3C standards for Web services allow for more than one interpretation, and the Web services Interoperability Organization (WS-I) introduced a separate standard called the Basic Profile to tighten up their use to improve Web service interoperability. For example, the WS-I Basic Profile does not allow the use of SOAP encoding, even though this is often seen in practice.

The WS-I Validator can be used to check your WSDL definitions against the Basic Profile. The validator can be run either manually against a specific WSDL resource in the workbench, or automatically when generating or importing a WSDL definition. You are able to set the validation failure action to either ignore, warn, or error.

6.3.4 Web service transport capabilities

The transport protocol for the Web service SOAP data in WebSphere Message Broker can vary. The most widely used protocols for Web service transport are HTTP, MQ, and JMS.

HTTP or HTTPS

The HTTP nodes are shown in Figure 6-30.
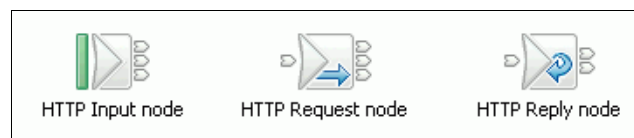
Figure 6-30   Built-in HTTP nodes

The HTTPInput and HTTPOutput nodes along with the SOAP handling technique can be used to make a message flow that provides an HTTP Web service endpoint to the outside world.

The HTTPRequest node can send a request out and wait for the reply. This can be used in a message flow to call a Web service and enrich the message with the response from the Web service provider.

In WebSphere Message Broker V6, the HTTP nodes have been enhanced to support HTTP 1.0, HTTP 1.1 (default port is 7080), and HTTPS (default port is 7083).

HTTP 1.1 provides capabilities such as chunked encoding and persistent connections, which allows multiple requests to be sent using the same TCP connection. This is especially important in the context of HTTPS. HTTPS supports SSLv3, SSLv2, and TLS protocol. You can specify the SSL ciphers to be used for the secure connection.

In the HTTPRequest node, you can specify the method for the request, GET or POST. You can use the message domain MRM with XML, XMLNS, or XMLNSC on HTTP nodes to handle the SOAP data. Note that these are general capabilities since V5.
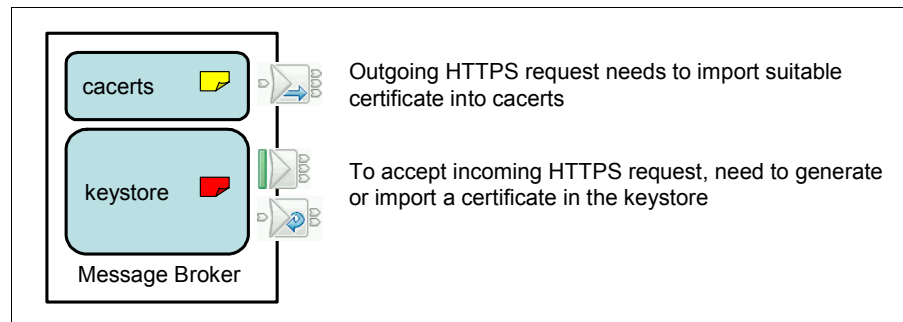
Figure 6-31   HTTPS support and certificates

WebSphere Message Broker includes a Java Runtime Environment (JRE™) that supplies a keystore manipulation program, which is called keytool. To allow HTTPInput and HTTPReply nodes to interact with the requester using the HTTPS protocol, you must generate a testing certificate using keytool. For production, you need to purchase an official certificate from a certificate organization.

Example 6-6 shows a sample of using the keytool to generate a key store file and test certificate.

Example 6-6   Commands to generate a testing purpose certificate

```
C:\IBM\MQSI\6.0\jre\bin>keytool -genkey -keypass <password> -keystore
<keystore_filename> -alias certalias
Enter keystore password: <password>
What is your first and last name?
  [Unknown]:  YUNPENG GE
What is the name of your organizational unit?
```

```
   [Unknown]:  IBM SWG
What is the name of your organization?
   [Unknown]:  IBM CHINA
What is the name of your City or Locality?
   [Unknown]:  CHINA
What is the name of your State or Province?
   [Unknown]:  SHANGHAI
What is the two-letter country code for this unit?
   [Unknown]:  CN
Is CN=YUNPENG GE, OU=IBM SWG, O=IBM CHINA, L=CHINA, ST=SHANGHAI, C=CN
correct? (type "yes" or "no") [no]:  yes
```

The next step is to enable and configure SSL in WebSphere Message Broker.
The commands are shown in Example 6-7. The sample assumes that
WebSphere Message Broker is installed on a Windows® platform, but the steps
are almost identical for other platforms.

Example 6-7   Command to enable and configure SSL in WebSphere Message Broker

```
C:\IBM\MQSI\6.0\jre\bin>mqsichangeproperties WBRK6_DEFAULT_BROKER -b
httplistener -o HTTPListener -n enableSSLConnector -v true
BIP8071I: Successful command completion.

C:\IBM\MQSI\6.0\jre\bin>mqsichangeproperties WBRK6_DEFAULT_BROKER -b
httplistener -o HTTPSConnector -n keystoreFile -v <keystore_filename>
BIP8071I: Successful command completion.

C:\IBM\MQSI\6.0\jre\bin>mqsichangeproperties WBRK6_DEFAULT_BROKER -b
httplistener -o HTTPSConnector -n keystorePass -v <password>
BIP8071I: Successful command completion.

C:\IBM\MQSI\6.0\jre\bin>mqsichangeproperties WBRK6_DEFAULT_BROKER -b
httplistener -o HTTPSConnector -n port -v <SSL_port>
BIP8071I: Successful command completion.
```

Now you can create a message flow using an SSL-enabled HTTPInput node to
accept an incoming HTTPS request on the port you specified.

Figure 6-32 shows how SSL is enabled on an HTTPInput node.
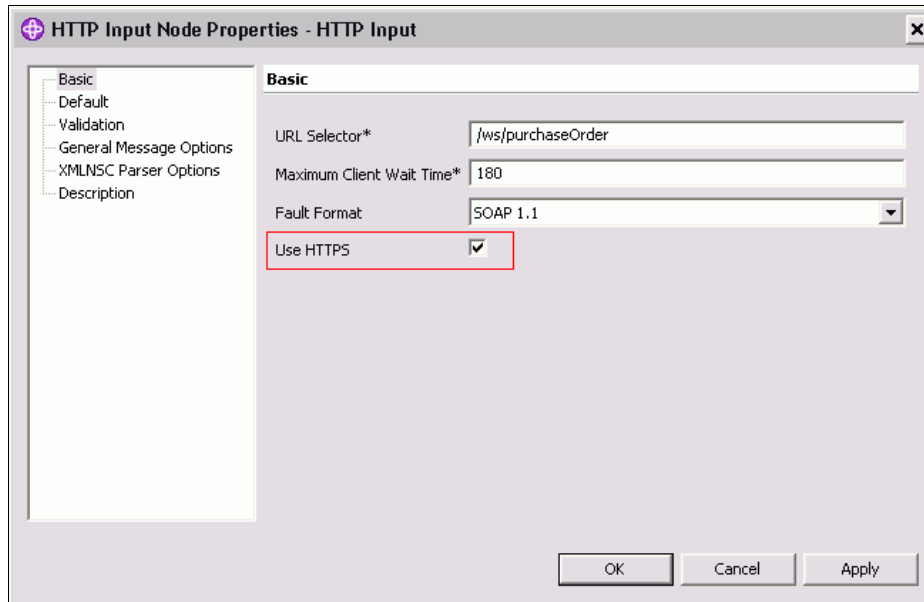
Figure 6-32   Enable HTTPS on an HTTPInput node

If your message flow acts as a client to send an HTTPS request through the HTTPRequest node, you must locate a suitable certificate for the remote Web service and import it into the cacerts file of your WebSphere Message Broker, usually located at %MQSI_FILEPATH%\jre\lib\security. The commands used to import the certificate are shown in Example 6-8.

Example 6-8   Commands to import certificate into the cacerts file

```
C:\IBM\MQSI\6.0\jre\bin>keytool -import -alias <key_alias> -file
<cert_filename> -keystore ../lib/security/cacerts -keypass changeit
Enter keystore password:  changeit
Owner: CN=YUNPENG GE, OU=IBM SWG, O=IBM CHINA, L=CHINA, ST=SHANGHAI,
C=CN
Issuer: CN=YUNPENG GE, OU=IBM SWG, O=IBM CHINA, L=CHINA, ST=SHANGHAI,
C=CN
Serial number: 4576e4dd
Valid from: 12/6/06 10:42 AM until: 3/6/07 10:42 AM
Certificate fingerprints:
   MD5:  BD:02:29:7C:59:35:1C:BE:CD:54:D4:2D:EF:AA:A3:F4
   SHA1: 7C:73:95:4A:BD:4E:C2:87:81:84:39:3A:46:9F:20:0F:DE:69:47:7C
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

> **Note:** The default password for cacerts file is *changeit*. You should change this password as soon as possible by using the keytool utility.

MQ transport

Using WebSphere MQ as a transport provides a simple way to implement an asynchronous Web service pattern. In some cases, using an MQ transport rather than HTTP can improve client response time because the client does not wait for a HTTP 200 or 202 response from the service provider. The MQ transport is also more reliable thanks to the assured delivery provided by WebSphere MQ.

Figure 6-33   Part of built-in MQ nodes

Use the MQInput node to receive messages from clients that connect to the broker using the MQ transport, and from clients that use the MQI and AMI application programming interfaces. The MQInput node receives message input to a message flow from a WebSphere MQ message queue defined on the

broker's queue manager. The node uses MQGET to read a message from a specified queue, and establishes the processing environment for the message. If appropriate, you can define the input queue as a WebSphere MQ clustered queue or shared queue.

Message flows that handle messages that are received across WebSphere MQ connections must always start with an MQInput node. You can set the properties of the MQInput node to control the way that messages are received, by causing appropriate MQGET options to be set. For example, you can indicate that a message is to be processed under transaction control. You can also request that data conversion is performed on receipt of every input message.

Use the MQOutput node to send messages to clients that connect to the broker using the MQ transport and that use the MQI and AMI application programming interfaces. The MQOutput node delivers an output message from a message flow to a WebSphere MQ queue. The node uses MQPUT to put the message to the destination queue or queues that you specify. You can configure the MQOutput node to put a message to a specific WebSphere MQ queue defined on any queue manager accessible by the broker's queue manager, or to the destinations identified in the LocalEnvironment (also known as the DestinationList) associated with the message.

For message flows working directly with WebSphere MQ, a new MQGet node enables messages to be retrieved midway through a message flow, providing a convenient way for a message flow to invoke a Web service over the WebSphere MQ transport and then receive the response within the same flow. A sample usage for enriching the message using MQGet node in the middle of the message flow is shown in Figure 6-34.
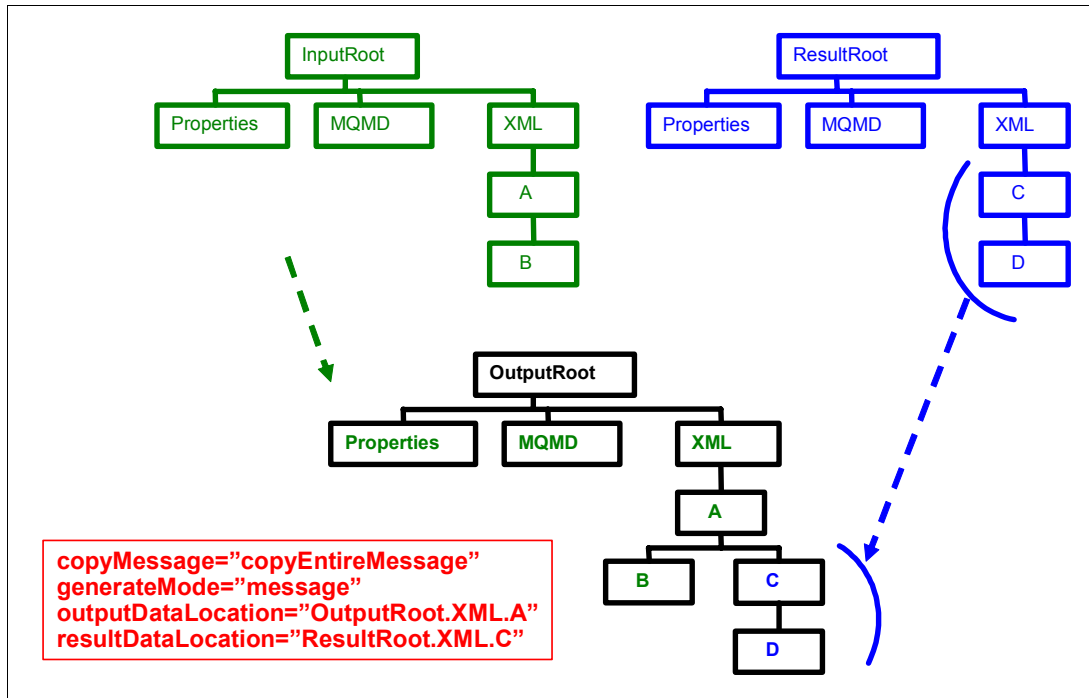
Figure 6-34   A sample usage of enriching the message using MQGet node

6.3.5 Java Message Service (JMS) transport

WebSphere Message Broker V6 introduces JMSInput and JMSOutput nodes, enabling a message flow to send messages to and receive messages from JMS destinations. These nodes behave like JMS clients and work with the WebSphere MQ JMS provider, WebSphere Application Server V6 service integration bus, and any other JMS provider that conforms to JMS Specification V1.1.

Figure 6-35   Built-in JMS nodes

The exchange of JMS messages is achieved by using the JMSInput and the JMSOutput nodes. These two nodes allow a message flow to receive messages from JMS destinations or to send messages to JMS destinations. These destinations are accessible through connection to a JMS provider. After the message is received, the JMSInput node will populate a message tree with the format shown in Figure 6-36, and the message payload can be parsed and manipulated using a different message domain and parser.
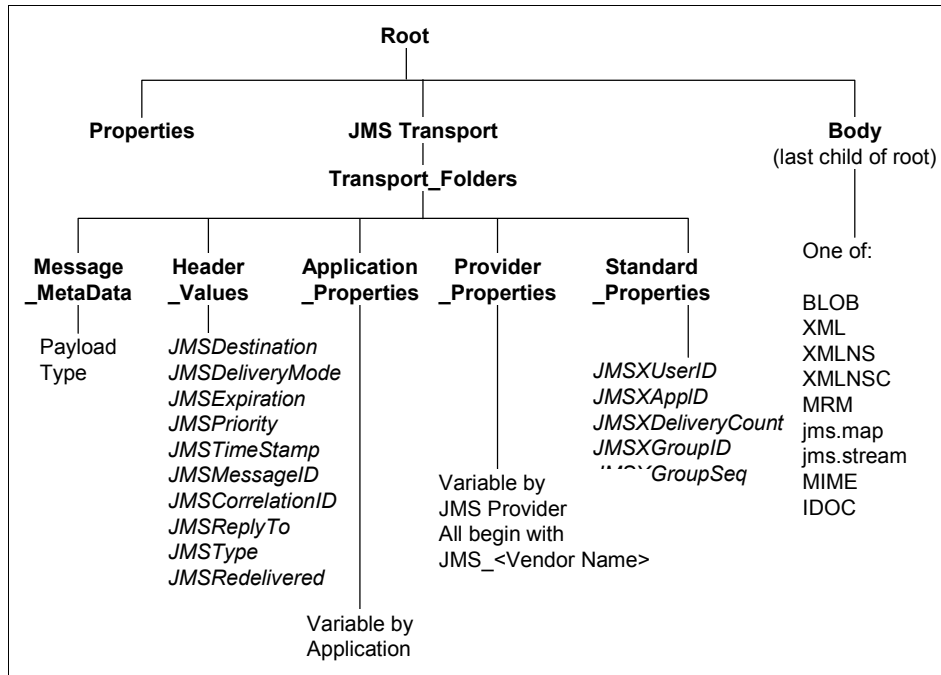
Figure 6-36   Representation of messages across the JMS Transport

Two transformation nodes allow the JMSInput and JMSOutput nodes to inter-operate with nodes that expect a propagated message to contain an MQMD (and RFH2) header. These nodes are the JMSMQTransform node and the MQJMSTransform node.

The JMSMQTransform node takes the output of the JMSInput node, which is in native JMS format, and produces a message that can be handled by an MQOutput node, which is in MQ JMS format.

The MQJMSTransform node transforms a message with an MQMD (and RFH2) header into a message that is expected by the JMSOutput node. See the transform diagram in Figure 6-37.
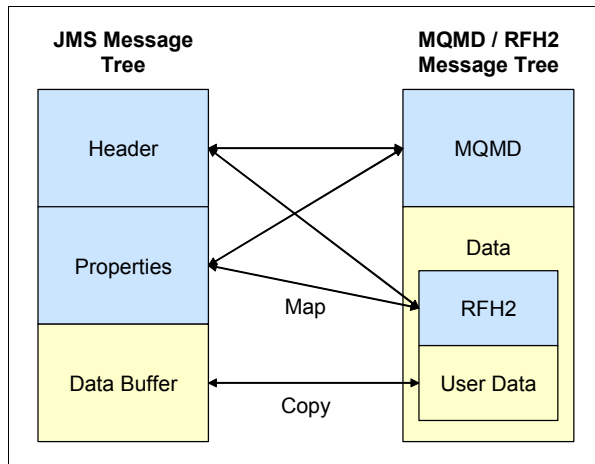
Figure 6-37   Mapping between native JMS message and a MQ JMS message

6.4 Using message flows for mediation

A message flow in WebSphere Message Broker is not the appropriate place to implement business logic in the form of a Web service. This is better done in an application server such as WebSphere Application Server.

When using WebSphere Message Broker as an ESB, message flows can front the Web service to perform additional processing or routing. Ways to use the Web services capabilities of WebSphere Message Broker to mediate messages as they pass through the ESB include:

► Using a message flow as a Web service requester, invoking existing Web services from a message flow (Figure 6-38)
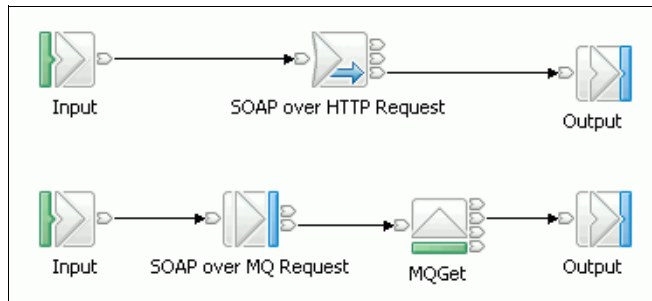
Figure 6-38   Outline of WebSphere Message Broker as service requester

► Acting as an intermediary to providing a new interface or adding some value to an existing Web service (Figure 6-39)
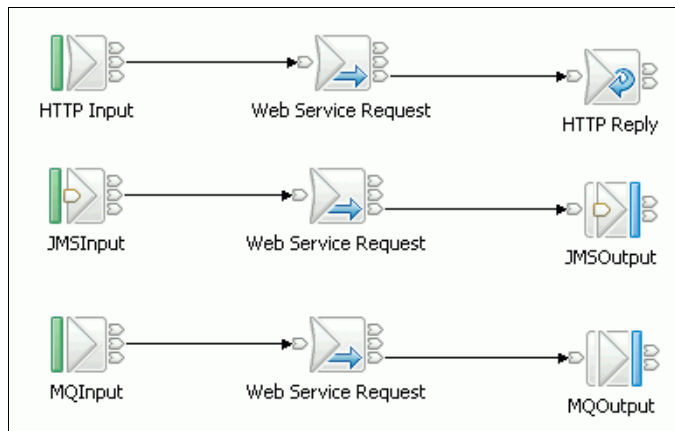
Figure 6-39   Outline of WebSphere Message Broker as intermediary

▶ Implementing a Web service interface to an existing application (such as an MQ-enabled application) (Figure 6-40)
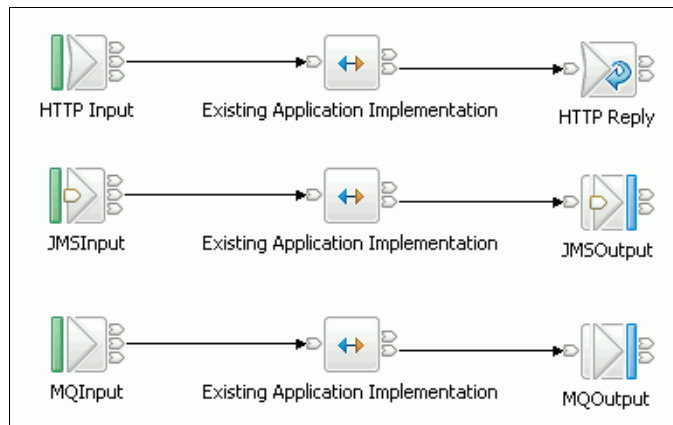
Figure 6-40   Outline of WebSphere Message Broker as wrapper

6.4.1 Service Registry lookup

To gain an understanding of Service Registry lookup and review an associated scenario to this topic, refer to 16.8.5 of the *WebSphere Service Registry and Repository Handbook*, SG24-7386:

http://www.redbooks.ibm.com/redpieces/abstracts/sg247386.html

6.5 Security considerations

There are many aspects to be considered before implementing WebSphere Message Broker. For example:

▶ Is security in the communications infrastructure acceptable, for example, in the use of Secure Socket Layer mutual authentication between EAI middleware servers, or in the use of the HTTPS protocol?

▶ Is individual, point-to-point security acceptable between participating systems, or is an end-to-end model required? For example, is there a need to propagate client identity through intermediate systems such as brokers to the end providers of service implementations?

▶ Is security in the application layer acceptable? For example, can the client code perform basic HTTP authentication with a user ID and password, or can it pass such information to the service as application data?

► Is compliance to a security standard security, such as Kerberos or WS-Security, required?

> **Note:** These questions represent the aspects that should be considered in the ESB solution, but some of these questions usually do not have an answer. It highly depends on the environment and enterprise architecture.

In this section, we discuss the security options that are provided by WebSphere Message Broker.

6.5.1 WebSphere Message Broker security

This section discusses areas of security that are important in the development and execution of message flows.

Authorization for configuration tasks

Authorization is the process of granting or denying access to a system resource. For WebSphere Message Broker, authorization consists of controlling who has permission to access WebSphere Message Broker resources and ensuring that users who attempt to work with those resources have the necessary authorization to do so.

For more information about this subject refer to:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp?topic=/com.ibm.etools.mft.doc/ab00025_.htm

Authorization to access runtime resources

Runtime resources are objects that exist at run time in the broker domain. Each runtime object has an Access Control List (ACL), which determines which users and groups can access the object. The ACL entries for an object can permit a user or group to view the object or view and modify the object from the workbench, the command line, or using the Configuration Manager Proxy (CMP).

Use topic-based security to control which applications in your publish/subscribe system can access information about which topics. For each topic to which you want to restrict access, you can specify which principals (user IDs and groups of user IDs) can publish to the topic, and which principals can subscribe to the topic. You can also specify which principals can request persistent delivery of messages.

For more information about enabling topic-based security, refer to the InfoCenter chapter "Enabling topic-based security."

Communication security

You can use the WebSphere MQ channel security exit to ensure that the participant on the other end of the channel is genuine, or that the communication between the Message Brokers Toolkit and the Configuration Manager is secure. Alternatively, you can use SSL when communicating between the Configuration Manager Proxy and the Configuration Manager.

For an overview of security exits, refer to the section "Channel security exit programs" in the WebSphere MQ Intercommunication manual. Visit the following Web site:

http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US&FNC=SRX&PBL=SC34658700

6.5.2 Web services security

New standards and applications for Web services security are being continually developed and deployed, making it a dynamically changing area. Web services security as of today can be achieved at two levels:

► Security at the transport level

  Transport security addresses the non-functional requirement that the communication between a customer and his business partner should not be able to be viewed by a third party as it travels on the Internet.

  Security at the transport level uses the built-in security features of transport technologies like HTTP, IBM WebSphere MQ, and so on. WebSphere Message Broker now supports SSL authentication for the HTTP nodes.

  This section discuss the security of Web service transportation, and the WebSphere Message Broker provides three types of transportation for the Web service: HTTP, MQ, and JMS. But JMS standard does not contain security suggestions, so only HTTP and MQ are discussed here.

  Messages using WebSphere MQ transport or JMS Message over MQ can be secured by the MQ channel secure facility. For more information about MQ channel security, refer to WebSphere MQ Security manual at:

  http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US&FNC=SRX&PBL=SC34658801

► Security at the SOAP or messaging level

  This level is currently being extensively researched and specifications are being developed by groups like W3C or OASIS. This involves usage of digital signatures, certificates, and so on at the XML document level.

  One of the examples would be WS-Security. Generally speaking, WS-Security is a W3C standard and defines how a Web service message can

be encrypted and decrypted at the SOAP message level to address the requirements of message non-repudiation and avoid messages from being tampered with.

From the product design point of view, WebSphere Message Broker as an ESB is not the right place to implement WS-Security functionality. However, WebSphere Message Broker is able to deliver the SOAP message with security credentials to its destination.

6.6 Transaction considerations

Transactions are a fundamental concept in building reliable distributed applications. A transaction is a mechanism to insure all the participants in an application achieve a mutually agreed upon outcome. Traditionally, transactions have held the following properties, collectively referred to as ACID:

► Atomicity: If successful, then all of the operations happen, and if unsuccessful, then none of the operations happen.

► Consistency: The application performs valid state transitions at completion.

► Isolation: The effects of the operations are not shared outside the transaction until it completes successfully.

► Durability: Once a transaction successfully completes, the changes survive failure.

A message flow consists of the multiple nodes that each perform a function, for example:

► An input queue
► The message flow
► Some update to database tables
► One or more output queues

In any specific application, especially a critical application, we want the message flow to be transactional. That is, all of the elements in the message flow should work in coordination. For example, if an update database operation fails at the last step, the message flow should have the capability to roll back all of the changes done in the flow before the database update. In this context, this is a global transaction that involves multiple resource managers, typically (but not limited to) JMS resources and database resources.

WebSphere Message Broker supports global transactions. It is possible to be selective as to which resources are included within the message flow's transaction and which are a part of their own transaction. Complex transaction processing, where parts of a message flow can be isolated within separate

transactions, can also be implemented in WebSphere Message Broker. There is also the ability to have try/catch blocks of mediation processing.

6.6.1 Message flow transaction

Message flows support two transaction styles:

- ► Coordinated message flows ensure that all updates to resources are committed or rolled back together within a single transaction.
- ► Uncoordinated message flows allow updates to resources to occur independently. The updates are not affected by the overall success or failure of the flow.

Coordinated message flows

A message flow that includes interaction with an external database or other recoverable resource can be configured so that all of its processing is coordinated within a single global transaction. This coordination ensures that either all processing is successfully completed, or no processing is completed. The transaction is committed (if all processing is successful) or rolled back (if at least one part of the processing is unsuccessful). This means that all affected resources (queues, databases, and so on) are maintained in a consistent state, and that data integrity is preserved.

Updates made by a coordinated flow are committed when the flow successfully completes processing the input message. The updates are backed out if the following are true:

- ► Any node within the flow throws an exception that is not caught by a node other than the input node.
- ► The input node's catch terminal is not connected.

To configure a message flow as coordinated, set the coordinated property on the message flow.

For some input nodes (such as MQInput) or SCADA nodes, you can set the transaction mode property on the nodes in the flow to automatic. This means that messages will be part of the global transaction, and the flow marked as transactional if the input message is persistent, and uncoordinated if the input message is not persistent. Subsequent nodes in the flow that set the transaction mode property to Automatic are included in the global transaction if the flow was marked transactional by the input node.

Transaction coordination of message flows is provided on distributed platforms by WebSphere MQ and on z/OS systems by RRS. Message flows are always

globally coordinated on z/OS, regardless of whether the message flow's coordinated property is specified as coordinated.

Using a Java Compute node

Special attention must be paid when using a Java Compute node to perform a database-related operation. If you use a JDBC type 4 driver to access the database, the transaction is not supported, because the Java Compute node only supports a non-XA type JDBC driver. However, you can use a MbSQLStatement API to operate database tables, which fully supports transactions.

If you catch the exceptions thrown by the broker in a Java Compute node, which is inherit from MbException, you probably need to re-throw the exception. Otherwise, the message is lost and the transaction is not rolled back. Generally, it is not necessary to catch the exception in the Java Compute node.

Uncoordinated message flows

Uncoordinated flows are flows for which the Coordinated property is not set. Updates to resources used by a uncoordinated flow are managed by the separate resource managers. Some resource managers, such as WebSphere MQSeries, allow updates to be made non-transactionally, or as part of a resource-specific transaction. Other resource managers, such as database managers, always use a resource-specific transaction. A resource-specific transaction is a transaction whose scope is limited to the resources owned by a single resource manager, such as a database or queue manager.

Resource-specific transactions are typically used only when there is just one type of recoverable resource used in a flow. (An example of such a flow is one that contains an MQInput and an MQOutput node, but that does not access any databases.) Resource-specific transactions should not be used when there is more than one resource and data integrity must be maintained.

Updates made to a resource accessed non-transactionally are committed immediately. An MQInput node configured to be non-transactional removes messages from the queue immediately, and if the flow fails the messages are lost.

Some input nodes (such as MQInput) or SCADA nodes can be part of a transaction, depending on the persistence of the input message, by setting the transaction mode to automatic. Messages are made part of the transaction, and the flow marked as transactional, if the input message is persistent, and non-transactional if the message is not persistent.

7

WebSphere DataPower appliances in SOA

This chapter explains how DataPower appliances can be used within an SOA environment. The chapter is organized into the following topics:

► An overview of DataPower appliances, noting their SOA-related features

► Roles for DataPower appliances within the ESB patterns discussed earlier in the book

► Combining DataPower appliances with a registry

► Descriptions of each of the DataPower appliance models

See Chapter 10, "Scenario: DataPower in an SOA" on page 323.

7.1 DataPower overview

In this section we take a moment to discuss the capabilities of the DataPower appliances.

What the DataPower SOA Appliances are

IBM SOA appliances are purpose-built, easy-to-deploy network devices (shown in Figure 7-1) that simplify, help secure, and accelerate XML and Web services deployments while extending an SOA infrastructure.
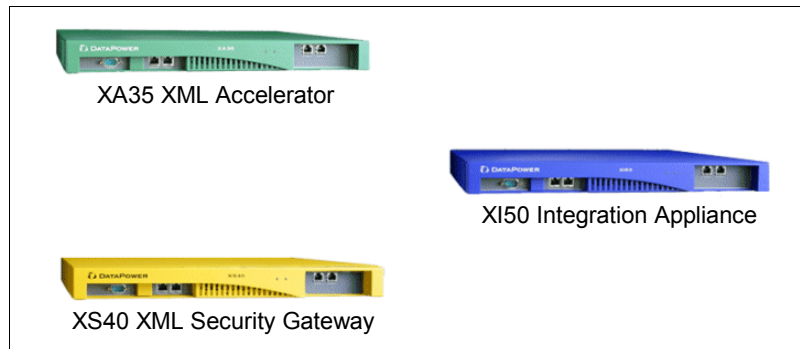
Figure 7-1   DataPower appliances

The IBM WebSphere DataPower SOA Appliancesfamily contains 1U (1.75-inch thick) rack-mountable network devices that deliver the following:

► Enhanced security

   An easy-to-install and easy-to-maintain network appliance that can satisfy both application and network operational groups, supporting current and emerging security standards, as well as XML Web services standards out-of-the-box. Key support includes, but is not limited to, the XML/SOAP firewall, field-level XML security, data validation, XML Web services access control, service virtualization, and SSL acceleration.

► Simplicity

   Common message transformation, integration, and routing functions in a network device, helping to cut operational costs, reduce complexity, and improve performance.

► Acceleration

   Drop-in solution that can streamline XML and Web service deployments, helping lower total cost of ownership and accelerate return on your assets, as you continue to move to SOA. SOA appliances are purpose-built hardware

devices capable of offloading overtaxed servers by processing XML, Web services, and other message format at wirespeed.

In summary, the IBM WebSphere DataPower SOA Appliancesoffer:

► 1U (1.75-inch thick) rack-mountable, purpose-built network appliances

► XML/SOAP firewalling, field-level XML security, data validation, XML Web services access control, and service virtualization

► Lightweight and protocol-independent message brokering, integrated message-level security and fine-grained access control, and the ability to bridge mission-critical transaction networks to SOAs and ESBs

► High performance, multi-step, wirespeed message processing, including XML, XSLT, XPath, and XSD

► Centralized Web services policy and service-level management

► WS-* standard support, such as WS-Security, SAML 1.0/1.1/2.0, portions of Liberty Alliance protocol, WS-Federation, WS-Trust, XKMS, Radius, XML Digital Signature, XML-Encryption, WSDM, WS-SecureConversation, SOAP, WSDL, UDDI, and others

► Transport layer flexibility, which supports HTTP/HTTPS, MQ, SSL, FTP, and others

► Scalable, wirespeed, any-to-any message transformation, such as arbitrary binary, flat text, and XML messages, which include COBOL Copybook, CORBA, CICS, ISO 8583, ASN.1, EDI, and others

There are three types of DataPower appliance available, each building on the features of the last:

► IBM WebSphere DataPower XML Accelerator XA35

Accelerates common types of XML processing by offloading this processing from servers and networks. It can perform XML parsing, XML Schema validation, XPath routing, Extensible Stylesheet Language Transformations (XSLT), XML compression, and other essential XML processing with wirespeed XML performance.

► IBM WebSphere DataPower XML Security Gateway XS40

Provides a security-enforcement point for XML and Web services transactions, including encryption, firewall filtering, digital signatures, schema validation, WS-Security, XML access control, XPath and detailed logging.

► IBM WebSphere DataPower Integration Appliance XI50

Transport-independent transformations between binary, flat text files, and XML message formats. Visual tools are used to describe data formats, create mappings between different formats, and define message choreography. This

appliance can transform binary, flat-text, and other non-XML messages to help offer an innovative solution for security-rich XML enablement, enterprise message buses, and mainframe connectivity.

For more details on each of the appliance types see 7.4, "DataPower appliance models" on page 177.

For full product information about IBM WebSphere DataPower SOA Appliances see:

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

7.1.1 Key SOA features

The key SOA features are listed in this section.

XML Acceleration

Extensible Markup Language (XML) has proven to be a great force in the software industry, and all the more so with the current focus on SOA and the related increased adoption of Web services. We need XML for our SOA, but we do not want to spend our precious CPU cycles looking after it, or our design time deciding how to implement standard mechanisms for validation, encryption, translation. It is time to push the XML processing down onto the firmware, and focus our applications on doing actual business logic.

XML's flexible, self-describing, language-independent format makes decoupling partner systems much easier. However, the heavy reliance on XML for data transfer between services also presents some problems. For example, XML can result in lengthy message payloads and large amounts of overhead for schema validation and parsing. The processing overhead of dealing with XML can tax application servers and middleware infrastructure, drastically decreasing performance.

The evolution of network infrastructure has seen an increasing trend toward replacing general purpose software systems with dedicated hardware for increased performance. In this same way, there is an evolution towards using dedicated hardware for performing repetitive XML tasks such as parsing, schema validation, and XML Stylesheet Language (XSL) translation.

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

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

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

► Incoming/outgoing data validation

► Schema validation

► XML security, access control, authentication, and authorization

► Integration with various security and monitoring products such as IBM Tivoli Enterprise™ Monitoring, Tivoli Access Manager, Netegrity SiteMinder, and so on

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

XML Protection

Traditional firewalls only protect traffic at the Internet Protocol (IP) level. Web services effectively tunnel through this layer via standard HTTP(S) and expose the organization's applications to completely new threats. We need to ensure that only valid requests for valid services from genuine clients get inside the enterprise boundary. Essentially, we need an XML firewall.

The seriousness of these new threats should not be understated. The following developerWorks article clearly defines the breadth and seriousness of different attacks that are possible to any service exposed using XML:

http://www-128.ibm.com/developerworks/websphere/techjournal/0603_col_hines/0603_col_hines.html

The article concludes with the following comments:

To truly harden a system using Web services, you need to perform several important security steps (recommended by Gartner and others), including:

– Inspect messages for well-formedness.

– Validate schema.

– Verify digital signatures.

- Sign messages.

- Implement service virtualization to mask internal resources via XML transformation and routing.

- Encrypt data at the field level.

Systems hosting Web services, particularly public Internet-facing ones, should seriously consider the case for hardened gateway devices acting as XML firewalls to protect your systems from XML threats.

DataPower appliances address all of the above issues and more, creating a robust XML firewall for the enterprise. Examples are given below.

DataPower appliances introduce sophisticated checks on the incoming XML including the following:

▶ XML Threat Protection
▶ Single Message XML Denial of Service Protection
▶ Multiple Message XML Denial of Service Protection
▶ Message Tampering Protection
▶ Protocol Threat Protection
▶ XML Virus Protection
▶ Dictionary Attack Protection

Security

Protection from XML threats, as described in the previous section, is clearly only one way in which enterprise systems need to be protected. DataPower appliances provide a sophisticated set of security capabilities, a selection of which are noted below:

▶ XML Web services access control: DataPower appliances support a variety of access control mechanisms, including WS-Security, WS-Trust, X.509, SAML, SSL, LDAP, RADIUS, and simple client/URL maps. They can control access rights by rejecting unsigned messages and verifying signatures within SAML assertions.

▶ Authentication and authorization: Appliances directly support IBM Tivoli Federated Identity Manager (TFIM) with capabilities such as mapping identities for downstream access and IBM Tivoli Access Manager retrieve authorization.

▶ Field level message security: DataPower appliances can selectively share information through encryption/decryption and signing/verification of entire messages or of individual XML fields. These granular and conditional security policies can be based on nearly any variable, including content, IP address, host name, or other user-defined filters.

Service virtualization

This is when services are found to be sufficiently re-usable that they deserve to be promoted to enterprise-level services. Users of the enterprise services expect to be decoupled from the implementation of the service in at least the following ways:

▶ Routing: The hosting of a service is likely to change over time as demands for its availability and resilience increase. Its URL, and possibly even its hosting server, may change. In a mature SOA, there may even be more than one implementation of the service, that must be chosen between at run time. Clients of the service should remain unaffected by these changes. DataPower appliances allow configuration of a Web services proxy, decoupling the client completely from the implementation. Adjustments and translations can be made to all relevant metadata of the service from URL re-writing through to WS-Addressing and HTTP header manipulation. Routing information can be supplied in a variety of different ways, including direct configuration, setting routing data as part of a transformation, and database or registry lookups.

▶ Data model and namespace: Wherever possible, enterprise services should aim to expose a standardized data model. This model may, and arguably should, be different from that of the actual implementation of the service such that changes to the service do not effect the clients of the service. DataPower Appliances allow wire-speed translation of data models using XSLT, completely decoupling the client from the implementation.

▶ Versioning: The above two capabilities can be brought together to assist with issues such as service versioning. Clients need to be insulated from version changes to a service interface. In an SOA we cannot always assume that we can notify all users of a service if the interface needs to change. So we need to retain the existing interface as well as provide the new one. Rather than maintain several versions of the implementation of the service, however, DataPower can translate between the old and new URLs, host names, data models, headers, and any other relevant metadata.

Protocol switching

Services can be exposed using different formats from the ones in which they are implemented. Furthermore, they can be exposed using several different protocols at once to support a wide range of clients.

▶ Protocols. Services can be exposed and called using any combination of the typical protocols used for passing SOAP messages in an SOA, such as HTTP, HTTPS, and JMS. Direct communication with WebSphere MQ is also supported.

▶ Any-to-any transformation engine: If the enterprise's standard protocols reach beyond the commonly accepted Web services data formats, appliances can parse and transform arbitrary binary, flat text, and XML messages, including

EDI, COBOL Copybook, ISO 8583, CSV, ASN.1, and ebXML. Unlike approaches based on custom programming, DataPower's patented DataGlue technology uses a fully declarative, metadata-based approach.

Monitoring and management

DataPower appliances offer a number of different mechanisms for monitoring the traffic through the devise, from the very low level of XML statistics up to the service level management.

► Statistics: real-time visibility into critical XML statistics such as throughput, transaction counts, errors, and other processing statistics. Data network-level analysis is provided, and includes server health information and traffic statistics, as well as management and configuration data.

► Remote management: supports SNMP, script-based configuration, and remote logging to integrate seamlessly with leading management software.

► Web services management/service level management: support for Web services Distributed Management (WSDM); Universal Description, Discovery, and Integration (UDDI); Web services Description Language (WSDL); Dynamic Discovery; and broad support for Service Level Management configurations.

► Integration with various monitoring products such as IBM Tivoli Enterprise Monitoring and Netegrity SiteMinder.

7.2 Roles for DataPower in an SOA environment

Now that we have a clear picture of the capabilities of the DataPower appliance, let us look back at the ESB pattern where DataPower appliances are best placed. The key roles are:

► XML firewall
► ESB Gateway
► Gateway to multiple ESBs
► Back-end resource gateway
► XML Accelerator

They are described in more detail in the following sections.

Figure 7-2 shows our product mapping first described in 4.1.3, "Exposed ESB Gateway composite pattern" on page 62. Here, the two primary roles for DataPower appliances in an SOA are shown — the XML firewall and the ESB Gateway. These and other roles for the appliance are described below.
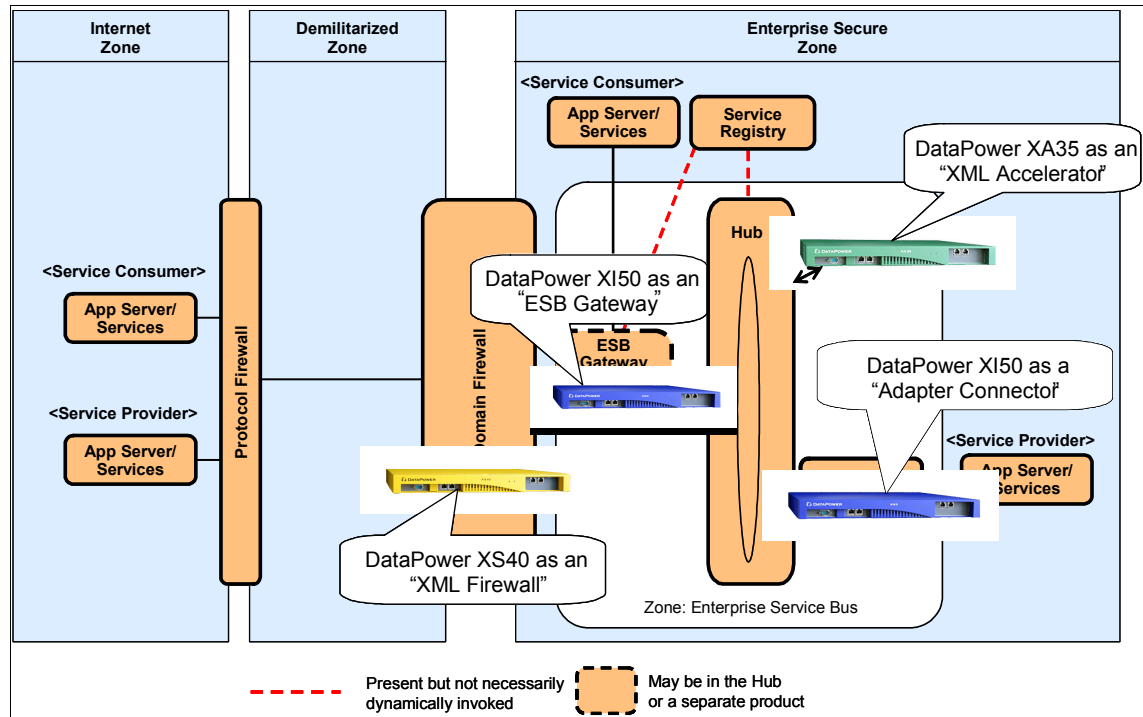
Figure 7-2   Roles for DataPower within an ESB

7.2.1 XML firewall

As discussed in 4.1.3, "Exposed ESB Gateway composite pattern" on page 62, an XML firewall should protect the applications, and indeed the Enterprise Service Bus, from threats by:

▶ Blocking XML attacks before they hit critical systems
▶ Shielding direct access to the internal systems
▶ Providing the most up-to-date security features for the SOA medium

The DataPower XS40 is most appropriately suited to this role, offering the XML threat protection, service virtualization, security, and encryption features described in 7.1.1, "Key SOA features" on page 168.

The above capabilities are handled at wire speed, adding minimal latency to the requests through the firewall.

7.2.2 ESB Gateway

As discussed in 4.1.3, "Exposed ESB Gateway composite pattern" on page 62, an ESB Gateway acts as a proxy to provide controlled access to the ESB Hub node.

Products that perform the role of the ESB Hub often exhibit some of the capabilities of an ESB Gateway themselves, but many of these capabilities are extremely CPU intensive.

Use of a DataPower appliance such as the XI50 allows offloading of this processing to a device specifically designed to perform security, encryption, XML validation, and transformation at wire speed, leaving the ESB Hub to focus on the business of mediating protocols, formats, and back-end specific considerations.

The most likely mechanism of connectivity between the DataPower appliance and the ESB Hub would be HTTP(S), but it should be noted that the XI50 can also work with JMS messages. It can talk to both WebSphere MQ and WebSphere Platform Messaging as JMS messaging providers.

Separation of this ESB Gateway to a separate dedicated node is also the first step on the way to handling more complex ESB scenarios that inevitably arise where more than one ESB Hub or when more than one ESB Hub technology comes into play, as discussed in the next section on Hierarchical Gateways.

7.2.3 Hierarchical ESB Gateways

A logical extension of the ESB Gateway usage above is to have a hierarchy of ESB Gateways. In this scenario, multiple gateways are used to administer groups of services, each owning access to a particular set of the services available in the enterprise. Each of these will have different policies associated with its services for generic aspects such as security, auditing, monitoring, and so on.

These multiple gateways do not necessarily imply multiple appliances. All of these different logical gateways could be handled by a single DataPower appliance by making use of the fact that DataPower provides a concept of domains. Each domain can provide access to its own set of services, and the domains are separately administrable using a typical role-based administration mechanism.

The DataPower XI50 is ideally suited to making use of its security features to control access to given domains and restrict access to the underlying ESB Hubs, also adding the ability to implement common policies across a given domain, and integration capabilities to provide the required communication protocols for each of the hubs.

7.2.4 Adapter Connector

If could be argued that a DataPower appliance such as the XI50 with its integration capabilities could be used as an integration hub in its own right, and in certain constrained scenarios this might be appropriate, but we need to understand the appropriate scope for this usage and, as we will see, it is more accurately described as a dedicated Adapter Connector.

The XI50 has integration capabilities allowing it to communicate over messaging protocols as well as HTTP, with support for back-end data formats such as COBOL Copybooks. This makes it a possible candidate for providing secure virtualized access to back-end resources such as mainframes, where the specific integration capabilities align well with the integration needs. The appliance can act in a dedicated role to provide access to a specific set of back-end systems, using its high-performance characteristics and security to protect the investment in the system. However, this is not the same capability as a full ESB Hub, and we should be clear about the difference.

Ultimately, an ESB Hub would need to have a rich and extensible suite of integration capabilities, including a persistent messaging environment, the ability to initiate and co-ordinate XA transactions, a general purpose programming environment, and a wide range of technology, application, and package adapters. This is where products such as WebSphere Enterprise Service Bus and WebSphere Message Broker fit in. In a typical case the ESB will have grown up around these ESB Hub products and DataPower will then be introduced to play the specific roles to which it is well suited.

7.2.5 XML Acceleration

Finally, we discuss the more fundamental use of the DataPower appliance. It is possible to make use of the appliance purely to offload XML processing from an application server. Here we would have recognized that a particular application was highly reliant on parsing, transformation, or encryption of XML. Here the XA35 can be used as an extension of the application server, providing wire speed XML processing capabilities.

The reason this role has been discussed last is because typically when high reliance on XML processing is found in an application, it usually points to a need to extend or re-align the architecture introducing the appliance in one of the previous roles discussed. Adding XML acceleration to an application may simply be perpetuating an architecture that is reaching its natural limits rather than resolving the core problem.

7.3 Combining DataPower with a registry

DataPower's clear fit for gateway style use in the ESB pattern means that it becomes the single entrypoint for service requests. This is the primary place to perform service virtualization and impose common policies on the services of the enterprise. Those common policies and information concerning where services are to be found can be configured directly into the gateway, but they are really part of the wider cataloging of services for the enterprise. It would be preferable if the gateway simply acted upon the policies set at enterprise level. This is just one of the reasons why there is a need for a Service Registry.

With the introduction of the WebSphere Service Registry and Repository product, all of the related products in the SOA Foundation suite are being upgraded to include mechanisms to contact the registry at run time or interact with the registry during configuration time. We can expect a similar set of registry connectivity features in DataPower to those that we are seeing introduced into the other key products such as WebSphere Enterprise Service Bus and WebSphere Message Broker.

Examples of themes in registry usage that may be relevant to DataPower appliances follow, noting to which roles (as defined in 7.2, "Roles for DataPower in an SOA environment" on page 172) these are most applicable:

► Validation

Access to schemas for validation of XML messages at configuration time. This is principally relevant to the XML firewall role for DataPower appliances, but potentially also to the ESB Gateway.

► Service virtualization

Access to the registry at run time in order to acquire the actual endpoint based on a logical profile for the service. This is particularly applicable when DataPower appliances are used in the role of an ESB Gateway.

► Policy

Runtime and configuration time access to policies such as WS-Policy from the registry. This could be for either the XML firewall or for the ESB Gateway roles, with different types of policy information being relevant to each. This is also particularly relevant to the Gateway to Multiple ESBs role, where layers of ESB Gateways each have a policy.

► Availability/performance

Runtime access to data-enabling choices between implementations of services based on their availability and performance characteristics. This is particularly relevant to the ESB Gateway role.

► Namespace translation

Access to standard transformations between known namespaces, most probably provided as xslt files. This is particularly relevant to the ESB Gateway pattern where routing between different versions of a service may be required to seamlessly support old clients against new implementations of a service. This could also be relevant where services are promoted up the gateway hierarchy and need to conform to different object/namespace structures.

The reach of the registry is much wider than this alone. More details on WebSphere Service Registry and Repository can be found at:

http://www-306.ibm.com/common/ssi/rep_ca/0/897/ENUS206-230/ENUS206-230.PDF

http://publib-b.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg247386.html?Open.

7.4 DataPower appliance models

Below is a more detailed description of each of the three models of WebSphere DataPower appliances.

IBM WebSphere DataPower XML Security Gateway XS40

This appliance provides a security-enforcement point for XML and Web services transactions, including encryption, firewall filtering, digital signatures, schema validation, WS-Security, XML access control, XPath, and detailed logging, and includes:

► An XML/SOAP firewall: The DataPower XML Security Gateway XS40 filters traffic at wirespeed, based on information from layers 2 through 7 of the protocol stack; from field-level message content; and SOAP envelopes to IP address, port/host name, payload size, or other metadata. Filters can be predefined with easy point-and-click XPath filtering GUI, and automatically uploaded to change security policies based on the time of day or other triggers.

► XML/SOAP data validation: With its unique ability to perform XML Schema validation as well as message validation, at wirespeed, the XS40 ensures that incoming and outgoing XML documents are legitimate and properly structured. This protects against threats such as XML Denial of Service (XDoS) attacks, buffer overflows, or vulnerabilities created by deliberately or inadvertently malformed XML documents.

► Field level message security: The XS40 selectively shares information through encryption/decryption and signing/verification of entire messages or of individual XML fields. These granular and conditional security policies can

be based on nearly any variable, including content, IP address, host name, or other user-defined filters.

► XML Web services access control: The XS40 supports a variety of access control mechanisms, including WS-Security, WS-Trust, X.509, SAML, SSL, LDAP, RADIUS, and simple client/URL maps. The XS40 can control access rights by rejecting unsigned messages and verifying signatures within SAML assertions.

► Service virtualization: XML Web services require companies to link partners to resources without leaking information about their location or configuration. With the combined power of URL rewriting, high-performance XSL transforms, and XML/SOAP routing, the XS40 can transparently map a rich set of services to protected back-end resources with high performance.

► Centralized policy management: The XS40's wirespeed performance enables enterprises to centralize security functions in a single drop-in device that can enhance security and help reduce on-going maintenance costs. Simple firewall functionality can be configured via a GUI and can be running in minutes. Using the power of XSLT, the XS40 can also create sophisticated security and routing rules. Because the XS40 works with leading Policy Managers, it is an ideal policy execution engine for securing next-generation applications. Manageable locally or remotely, the XS40 supports SNMP, script-based configuration, and remote logging to integrate seamlessly with leading management software.

► Web services management/service level management: With support for Web services Distributed Management (WSDM); Universal Description, Discovery, and Integration (UDDI); Web services Description Language (WSDL); Dynamic Discovery; and broad support for Service Level Management configurations, the XS40 natively offers a robust Web services management framework for the efficient management of distributed Web service endpoints and proxies in heterogeneous SOA environments, as well as SLM alerts and logging, and pull and enforce policies, which helps enable broad integration support for third-party management systems and unified dashboards, in addition to robust support and enforcement for governance frameworks and policies.

IBM WebSphere DataPower Integration Appliance XI50

This appliance provides transport-independent transformations between binary, flat text files and XML message formats. Visual tools are used to describe data formats, create mappings between different formats, and define message choreography. This appliance can transform binary, flat-text, and other non-XML messages to help offer an innovative solution for security-rich XML enablement, enterprise message buses, and mainframe connectivity.

- Any-to-any Transformation Engine: XI50 can parse and transform arbitrary binary, flat text, and XML messages, including EDI, COBOL Copybook, ISO 8583, CSV, ASN.1, and ebXML. Unlike approaches based on custom programming, DataPower's patented DataGlue technology uses a fully declarative, metadata-based approach.

- Integrated Message Level Security: XI50 includes mature message-level security and access control functionality. Messages can be filtered, validated, encrypted, and signed, helping to provide more secure enablement of high-value applications. Supported technologies include WS-Security, WS-Trust, SAML, and LDAP.

- Lightweight Message Brokering.

- Sophisticated multi-step message routing, filtering, and processing.

- Multiple synchronous and asynchronous transport protocols.

- Detailed logging and audit trail, including non-repudiation support.

IBM WebSphere DataPower XML Accelerator XA35

This product can help speed common types of XML processing by offloading this processing from servers and networks. It can perform XML parsing, XML Schema validation, XPath routing, Extensible Stylesheet Language Transformations (XSLT), XML compression, and other essential XML processing with wirespeed XML performance.

- Unmatched performance: DataPower's purpose-built message processing engine can deliver wirespeed performance for both XML to XML and XML to HTML transformations with increased throughput and decreased latency.

- Ease of use: The self-learning XA35 provides drop-in acceleration with virtually no changes to the network or application software. No proprietary schemas, coding, or APIs are required to install or manage the device, and it supports popular XML Integrated Development Environments (IDEs) to help reduce the number of hours spent in the development and debugging of XML applications.

- Helps reduce infrastructure costs: Unlike simple content switches that only redirect business documents, the DataPower XML Accelerator XA35 fully parses, processes, and transforms XML with wirespeed performance and scalability to help reduce the need for stacks of servers. The XA35 also supports accelerated SSL processing to help further lessen the load on server software.

- Helps cut development costs: The XA35 can enable multiple applications to leverage a single, uniformed XML processing layer for all XML processing needs. By standardizing on high-performance hardware appliances, enterprises can deploy sophisticated applications while helping to eliminate

unnecessary hours of application debugging and tuning for marginal performance gains.

► Intelligent XML processing: In addition to wirespeed processing, DataPower appliances support XML routing, XML pipeline processing, XML compression, XML/XSL caching, as well as other intelligent processing capabilities to help manage XML traffic.

► Advanced management: The DataPower XML Accelerator XA35 provides real-time visibility into critical XML statistics such as throughput, transaction counts, errors, and other processing statistics. Data network level analysis is provided, and includes server health information and traffic statistics, as well as management and configuration data.

For full product information about IBM WebSphere DataPower SOA Appliances:

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

8

ESB design options

This chapter presents ESB topologies that make use of WebSphere Enterprise Service Bus, WebSphere Message Broker, and other supporting products. Specifically, two sets of topologies are presented — one that uses WebSphere ESB as the main ESB backbone and one that uses WebSphere Message Broker as the main ESB backbone. Supporting products such as WebSphere DataPower, WebSphere Transformation Extender, WBI Adapters, and J2C Adapters are also leveraged to make the ESB environment more comprehensive.

The primary ESB functions are discussed in each topology to show how IBM WebSphere products are used together to implement the ESB's most important functions in a relatively simple way. Each topology is a one-ESB topology. The following ESB functions are discussed:

► Data format transformations
► Transport protocol transformations
► Service virtualization
► Dynamic routing
► Qualities of service

The supported transport protocols between the ESB components are shown where multiple products are used to implement an ESB.

A resilient configuration is shown and discussed for each ESB topology. The resiliency of the ESB topology where WebSphere ESB and WebSphere Message Broker are used together can be derived from the resiliency

discussions presented in the WebSphere ESB-based topology and WebSphere Message Broker-based topology.

An ideal candidate environment is discussed for each ESB topology presented.

Details are provided in this chapter on how to set up HTTPS communication between WebSphere ESB and WebSphere Message Broker.

8.1 WebSphere ESB-based architecture

The WebSphere ESB-based architecture is a J2EE-based architecture based on the simple fact that WebSphere ESB runs inside WebSphere Application Server, which implements the J2EE specification. Figure 8-1 shows a relatively simple WebSphere ESB-based architecture.
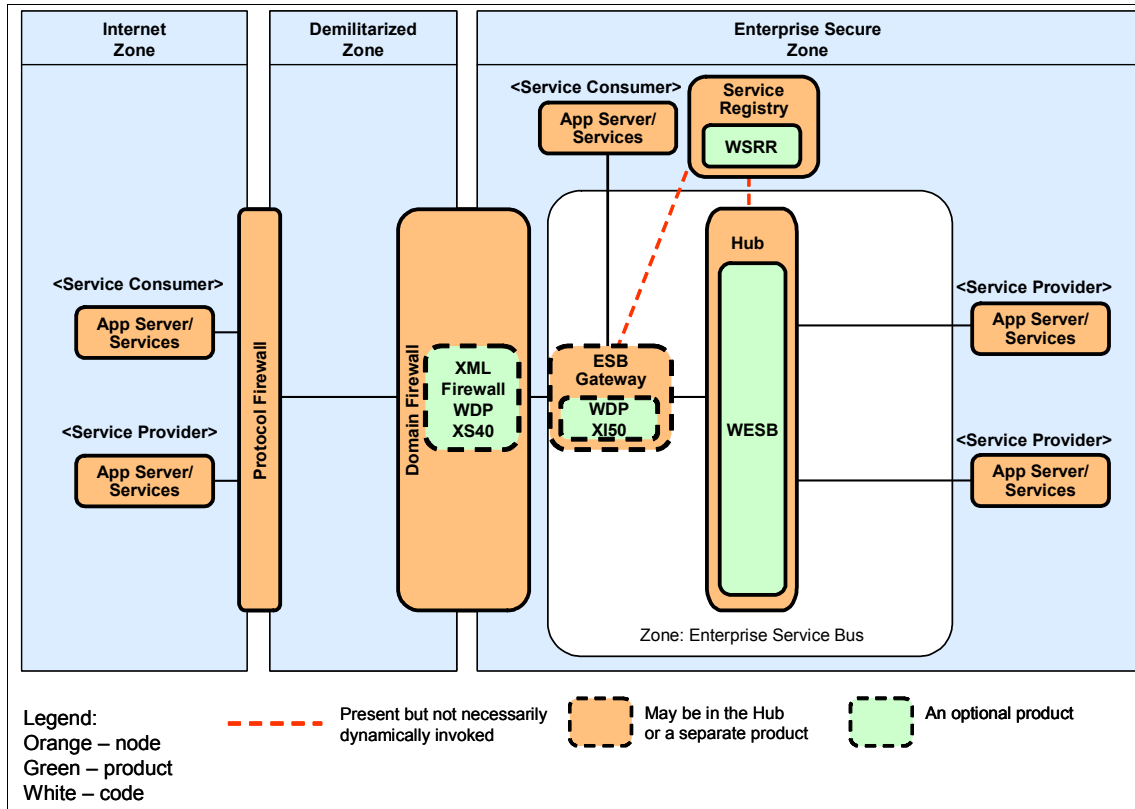
Figure 8-1   A simple WebSphere ESB-based architecture

Figure 8-1 shows WebSphere ESB to be at the heart of the ESB topology. It receives various types of service requests from the Internet and intranet. It routes these service requests intelligently to the correct service providers. The configuration shows that optionally either a WebSphere DataPower XS40 may be deployed as an XML firewall or a WebSphere DataPower XI50 may be deployed as an explicit ESB Gateway.

Through mediation flows developed in WID and deployed in the WebSphere ESB run time, WebSphere ESB routes the service request to the service provider that

may be located in the intranet, like a CICS service. WebSphere ESB may leverage a number of IBM WebSphere products such as WebSphere Transformation Extender, WebSphere DataPower, a J2C adapter, or a WBI adapter to interact with the service provider such as a CICS program or an SAP BAPI®. WebSphere ESB may also leverage WSRR to locate a service provider dynamically before invoking the service provider.

Service requests may be sent to WebSphere ESB from the intranet by Java and non-Java applications. To simplify interactions with WebSphere ESB from non-Java applications, IBM packages WebSphere ESB with two clients that can be leveraged by non-Java applications -- Message Service Clients for C/C++ and .Net and Web services Clients for C/C++.

The intranet service requests may also be sent to WebSphere DataPower, a J2C Adapter, a WBI adapter, or WebSphere Transformation Extender to process before they are sent to WebSphere ESB.

Similar to Internet-originated requests, intranet-originated requests invoke facade services exposed by WebSphere ESB. Then, through WebSphere ESB mediation flows, the actual service providers that may be located internally or externally are invoked to service these requests.

Note that the ESB zone shown in Figure 8-1 on page 183 contains only WebSphere ESB and one optional WebSphere DataPower appliance. But it may also contain WebSphere Transformation Extender, J2C Adapters, WBI Adapters, and WebSphere MQ leveraged primarily by WebSphere ESB.

8.1.1 Platforms support

Table 8-1 shows the support of various platforms by various ESB-related products shown in Figure 8-1 on page 183 or referenced earlier.

Table 8-1   Platform support of the ESB-related components

Product	Supported Platforms
WebSphere ESB	AIX®, HP-UX, Linux, Solaris, Windows, and Z/OS.
WebSphere Transformation Extender	AIX, Solaris, HP-UX, Linux, Windows, and Z/OS.
WebSphere DataPower	Not Applicable. it is a hardware device or appliance.
WSRR	AIX, HP-UX, Linux, Solaris, Windows, and Z/OS.

Product	Supported Platforms
J2C Adapters[a]	AIX, HP-UX, Linux, Solaris, Windows, and Z/OS.
WBI Adapters[b]	Windows, AIX, Solaris, HP-UX, and Linux.

a. For more details see the following link:
http://www-1.ibm.com/support/docview.wss?uid=swg27008539
b. For more details see the following link:
http://www-1.ibm.com/support/docview.wss?uid=swg27008539

8.1.2 WebSphere ESB-based candidate environment

WebSphere ESB is a good fit for an environment that has the following characteristics.

Strong J2EE environment

The J2EE developer will have no problem developing artifacts for WebSphere ESB. These artifacts are used by the integration developer who may be the same as the J2EE developer. The integration developer uses the visually-rich WID tool to build a mediation flow component by wiring existing pre-built mediation primitives or custom mediation primitives that may be implemented by the J2EE developer. The mediation flow component is part of an SCA mediation module that gets packaged as an EAR file and deployed in WebSphere Application Server as a J2EE application.

WebSphere Application Server already exists in the environment

WebSphere ESB runs inside WebSphere Application Server and leverages its services and capabilities. WebSphere ESB administration is done through WebSphere Application Server Administration Console, which has been augmented to enable the administrator to manage WebSphere ESB-specific artifacts.

► Business process management need

   If there is a need for a business process development and management or a need that may arise in the foreseeable future, it is relatively easy to add WebSphere Process Server on top of WebSphere ESB. As a matter of fact, WebSphere Process Server contains WebSphere ESB. So if an environment already makes use of WebSphere ESB, it will be relatively easy to add WebSphere Process Server and make use of the existing WebSphere ESB environment. Besides, WID is used to develop both types of artifacts — SCA

modules for WebSphere Process Server and SCA mediation modules for WebSphere ESB.

► Strong standard-driven environment

The WebSphere ESB programming model is based on SCA and the data model is based on SMO, which is an extension of SDO. SDO is primarily a joint effort between IBM and BEA. A number of companies have contributed to SCA. These companies are BEA, IBM, Interface21, IONA, Oracle®, SAP, Siebel, and Sybase. SDO has been standardized. SCA is also strongly driven toward standardization. Web services-related standards are also implemented in WebSphere ESB. WebSphere ESB supports the following WS-related standards: SOAP, SOAP with attachments, WSDL 1.1, WS-I Basic Profile 1.0 and 1.1, WS-Security, WS-Atomic Transactions, UDDI 3.0, and JAX-RPC handlers.

► Flexible programming model need

SCA defines a very flexible programming model for SOA. An integration developer wires the mediation flow from pre-built or custom mediation primitives. SCA is also the programming model used in the WebSphere Process Server. If a primitive or a component needs to be changed, it is easy to replace it with another implementation. This ease of component replacement promotes component reuse.

► Component reuse

The flexible SCA programming model implemented by WebSphere ESB promotes component reuse.

► Applications standard connectivity need

WebSphere Adapters or J2C Adapters implement the J2EE Connector Architecture specifications (currently, WebSphere ESB supports Siebel, PeopleSoft, SAP, JD Edwards®, e-mail, Flat File, and JDBC WebSphere Adapters). The number of supported applications will increase in the future.

► Dynamic routing needed

Routing does not have to be hardcoded in mediation flows. WebSphere ESB supports the dynamic routing through an enhancement introduced to the SMO header in WebSphere ESB 6.0.2. The dynamic routing capability can leverage the service endpoint lookup primitive to look up a match for a service endpoint in WSRR.

8.1.3 Data Format Transformation

A WebSphere ESB mediation module uses import components to invoke services located outside the module and uses export components to expose services to applications located outside the module.

To invoke a service exposed by a WebSphere ESB mediation module, the service requester must provide the data in the XML format. If the data is not in the XML format, WebSphere ESB supports WebSphere Transformation Extender, WebSphere DataPower, J2C Adapters, or WBI Adapters to transform the message from non-XML to XML.

The XML data is de-serialized as *Service Message Object* (SMO) when it enters the WebSphere ESB run time. The data may change as it flows through the mediation flow, but always as SMO as long as the data is in the WebSphere ESB run time.

Data may have to be transformed to a non-XML format so that the service provider can process it. To transform the data from XML to non-XML, WebSphere ESB can leverage, again, WebSphere DataPower, WebSphere Transformation Extender, J2C Adapters, or WBI Adapters.

Service Message Object

Figure 8-2 shows a portion of the SMO tree. Note the addition of the MQHeader subtree to support the MQ binding in WebSphere ESB 6.0.2. Any part of the SMO tree can be accessed via an XPath expression. For example, the XPath expression for the root object that is the smo element is /. The XPath expression for the context subtree is /context, and so on.

► The application data or payload is stored in the body subtree. Most of the data mapping is performed on the body subtree.

► The SMOHeader subtree is always there.

► The JMSHeader subtree is created when the JMS binding is used for the export or import component.

► The SOAPHeader subtree is created when the Web service binding is used for the export or import component.

► The MQHeader subtree is created when the MQ binding is used for the export or import component.

► Note that there may be more than one SOAPHeader and more than one properties subtree.
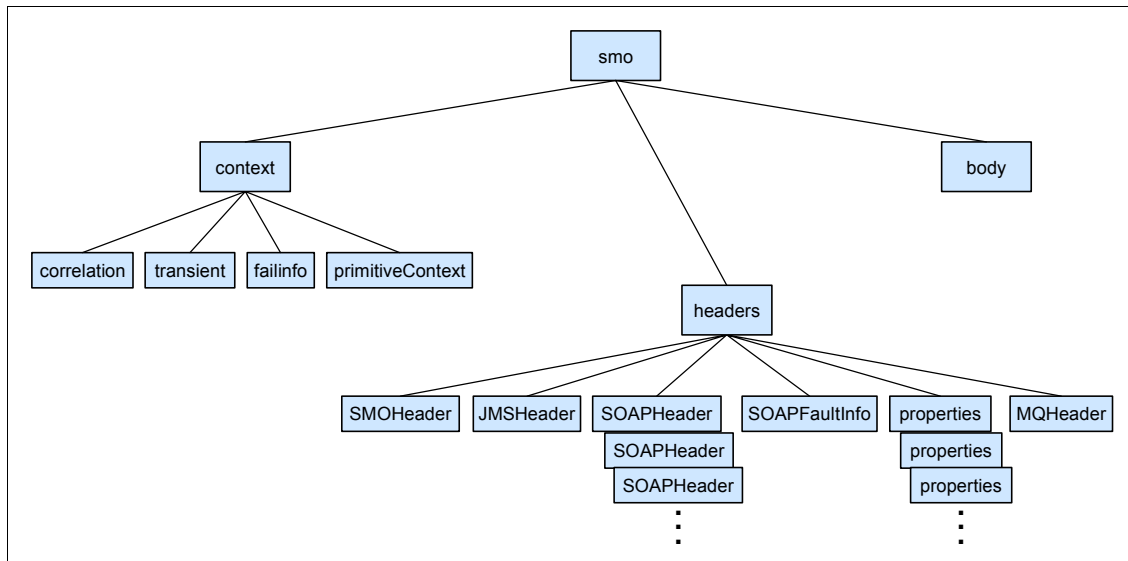
Figure 8-2   SMO context, headers, and body subtrees

Example 8-1 shows an XML representation of a sample SMO tree.

Example 8-1   XML representation of a sample SMO tree

```
<?xml version="1.0" encoding=UTF-8"?>
<smo:smo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmins:accounts="urn://www.example.com/accounts"
xmins:smo="http://www.ibm.com.com/websphere/sibx/smo/v6.0
 <context>
  <correlation xsi:type="accounts:CorrelateByAgentId">
   <agentid>SMI537654-2</agentid>
  </correlation>
  <transient xsi:type="accounts:ExchangeRate">
   <from>GBP</from>
   <to>USD</to>
   <multiplier>1.74068</multiplier>
  </transient>
 </context>
 <headers>
  <SMOHeader>
   <MessageUUID>b048778f-0701-0000-0080-80c5b8a4d8b8<MessageUUID>
   <Version>
    <Version>6</Version>
    <Release>0</Release>
    <Modification>1</Modification>
   </Version>
  </SMOHeader>
 </headers>
 <body xsi:type="accounts:processPaymentRequestMsg">
  <processPayment>
   <agentid>SMI537654-2</agentid>
   <priority>2</priorty>
   <payment>
    <value>415.26</value>
    <currency>USD</currency>
    <date>2005-01-16Z</date>
    <account>546219G</account>
   </payment>
  </processPayment>
 </body>
</smo.smo>
```

XML mapping in the mediation flow

Mapping data from one XML format to another is the only mapping type supported in WebSphere ESB.

The XML mapping capabilities are primarily provided by the *XSL Transformation mediation primitive*. XSL Transformation usage is necessary when the output terminal data type (SMO) of the source primitive or node is different from the input terminal data type (SMO) of the target primitive or node. The XSL Transformation primitive is effectively used to map one XML-serialized SMO tree to a different XML-serialized SMO tree. This means that when the XSL Transformation primitive is reached in a mediation flow, the SMO tree gets serialized to XML before the XML transformation occurs.

When there is no need to create another SMO tree and only a change for the same SMO tree is needed, the *Message Element Setter primitive* can be used to set values to leaf elements of the SMO tree. It can also be used to copy subtrees from one part of the SMO tree to another as long as the source subtree and the target subtree are of the same type. The Message Element Setter primitive can also be used to delete element instances in the SMO tree. The Message Element Setter primitive is provided with WebSphere ESB 6.0.2. Without this primitive, the simple tasks of setting element values or copying subtrees require an XSL transformation or a custom mediation primitive. Using the Message Element Setter primitive to update the SMO tree offers better performance than using the XSL Transformation primitive because the Message Element Setter primitive updates the same SMO tree, while the XSL Transformation primitive creates another SMO tree.

So far, this section discussed XML-to-XML transformations mechanisms inside a mediation flow component using the XSL Transformation and the Message Element Setter primitives.

Data transformation using external means

Figure 8-3 shows additional message transformation possibilities.
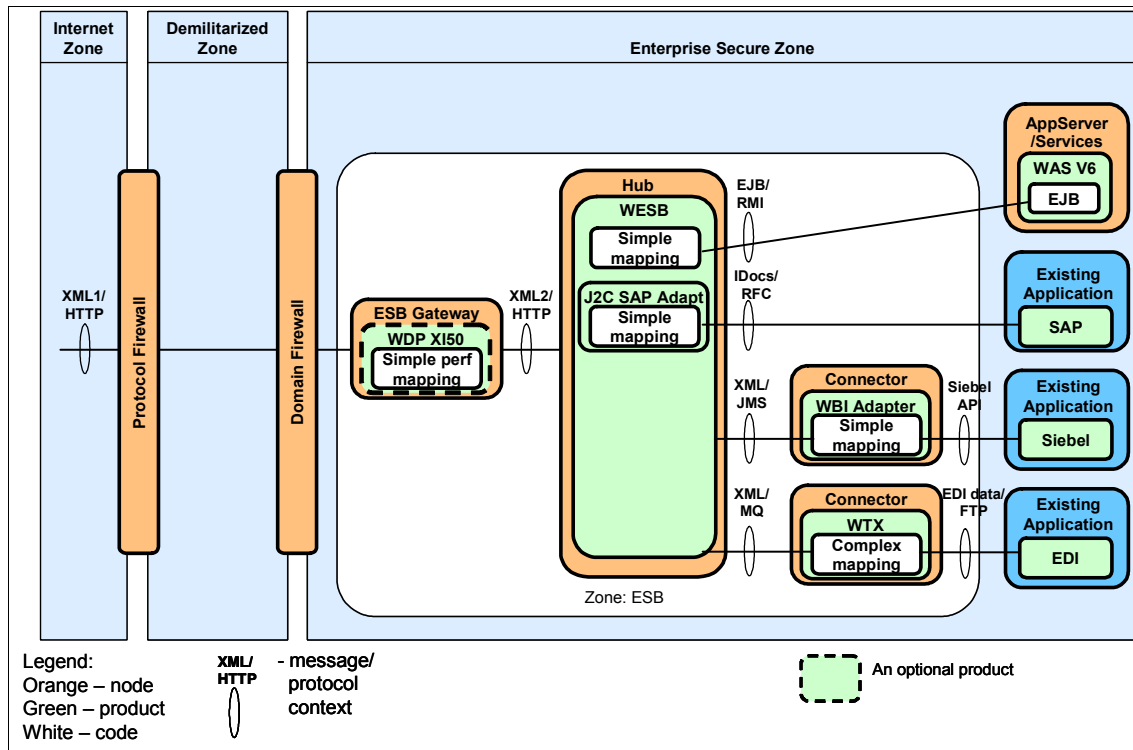
Figure 8-3   Data mapping performed by various components

The request may possibly get translated at high speed by WebSphere DataPower to another XML format before entering WebSphere ESB. Figure 8-3 also shows the data getting translated from XML to EIS format by a J2C SAP adapter, from XML to non-XML format by WebSphere Transformation Extender V8.1, and from XML to EIS by a WBI adapter. The J2C SAP adapter is shown inside the WebSphere ESB box because the J2C SAP adapter runs in the same JVM™ as WebSphere ESB. The WBI adapter is shown outside the WebSphere ESB box because the WBI adapter runs in its own JVM.

It is relatively straightforward to develop a binding for WebSphere Transformation Extender on an export and import the SCA component. (Such a binding has been developed by the IBM Software Services for WebSphere group.)

The WebSphere Application Server box shown in Figure 8-3 could be the same WebSphere Application Server environment used by WebSphere ESB or a remote WebSphere Application Server environment. The data sent to the EJB

box shown in Figure 8-3 on page 191 is supported by the stateless session EJB binding on an import SCA component. But a Java SCA component is needed to convert the SMO object to a Java object that can be consumed by EJB. WID 6.0.2 uses the java2wsdl (JAX-RPC) command to generate this Java component automatically.

Transformations using WebSphere adapters

The strategic direction for adapters is J2C Adapters, which currently do not support many technologies and applications, but the number of supported technologies and applications will soon increase. J2C Adapters include the following:

► IBM WebSphere Adapter for JDBC
► IBM WebSphere Adapter for FTP
► IBM WebSphere Adapter for E-mail
► IBM WebSphere Adapter for Flat File
► IBM WebSphere Adapter for IMS
► IBM WebSphere Adapter for CICS
► IBM WebSphere Adapter for Oracle E-Business Suite
► IBM WebSphere Adapter for SAP
► IBM WebSphere Adapter for PeopleSoft
► IBM WebSphere Adapter for Siebel
► IBM WebSphere Adapter for JD Edwards EnterpriseOne.

For more details see the following:

http://www-306.ibm.com/software/integration/wbiadapters/v60/

WBI Adapters can be used where J2C Adapters are used. WBI Adapters support many technologies and applications. Where an adapter functionality is desirable, but not supported in J2C Adapters, a WBI adapter can be used. For more details see the following:

http://www-306.ibm.com/software/integration/wbiadapters/v60/

The export and import components with EIS binding in a mediation module can be associated with either adapter type, J2C or WBI, to enable a mediation flow to interact with an EIS application. Note that the J2C adapter runs in the same JVM as WebSphere ESB, but the WBI adapter runs in its own JVM outside the WebSphere ESB JVM and requires the WBI Adapter Framework product. J2C and WBI adapters provide relatively simple interaction capabilities, primarily create, read, update, and delete (CRUD) operations against EIS applications or technology providers.

Transformations using WebSphere Transformation Extender

WebSphere Transformation Extender has a powerful and fast many-to-many transformation, the C/C++ engine.

Each input and output of a WebSphere Transformation Extender transformation map is associated with a data format and an adapter. For example, inputs to a WebSphere Transformation Extender map could be XML using the MQ adapter, SOAP using the HTTP adapter, flat file using the FTP adapter, and binary using the E-mail adapter, and the outputs from the WebSphere Transformation Extender map could be EDI X12 using the MQ adapter and SOAP using the HTTP adapter.

To support various industry data format standards and adapter functionality, IBM provides many data format packs and resource adapters for WebSphere Transformation Extender. Packs for WebSphere Transformation Extender include Pack for Web services, Pack for EDIFACT, Pack for HL7, Pack for HIPAA EDI, and so on. Resource adapters for WebSphere Transformation Extender include the database adapters such as DB2 and Oracle, the Internet adapters such as E-mail and HTTP, the message-oriented middleware adapters such as MQ and MSMQ, and so on. For a technology or an application that does not have a supported adapter, WebSphere Transformation Extender provides the adapter toolkit for building custom adapters.

WebSphere Transformation Extender maps can be invoked from within applications such as those written in Java, C, and C#. Java applications can invoke WebSphere Transformation Extender maps by using WebSphere Transformation Extender Programming Interface Java API, RMI API, CORBA API, or by using the JCA Connector. So WebSphere Transformation Extender maps can be invoked from within a custom mediation inside a WebSphere ESB mediation module or from within a custom JMS binding Java class of an import component inside a WebSphere ESB mediation module. Alternatively, WebSphere Transformation Extender and WebSphere ESB may interact indirectly through one of the supported protocols like MQ.

WebSphere Transformation Extender is a great fit in situations where more than one input data source needs to be mapped to one or more output data targets.

For more information about WebSphere Transformation Extender see:

http://www-306.ibm.com/software/integration/wdatastagetx/index.html

Transformations using WebSphere DataPower

There are three different versions of WebSphere DataPower: XA35, XS40, and XI50. XA35 is a subset of XS40, which is a subset of XI50. XA35 is effectively a fast XML transformation engine. XS40 has the same capabilities as XA35 in

addition to a more robust XML and SOAP security processing. XI50 has the same capabilities as XS40 in addition to more robust data transformation capabilities for XML and non-XML data.

WebSphere DataPower in the DMZ should be at least at the XS40 level. WebSphere DataPower in the intranet could be used primarily for a specialized data handling like a field-level encryption and a fast data transformation. Depending on application requirements, any version of WebSphere DataPower may perform the specialized data handling task in the intranet.

For more information see Chapter 7, "WebSphere DataPower appliances in SOA" on page 165.

8.1.4 Protocol Transformation

A WebSphere ESB mediation module is used to mediate between a service requester and a service provider. The mediation module contains at the most one mediation flow component, one or more export components to enable service requesters to invoke facade services, one or more import components to enable the module to invoke service providers, and possibly a Java component. A WebSphere ESB mediation module that does not have a mediation flow component, but has an export and an import component, can be used primarily for transport protocol transformations. For example, the service requester invokes the facade service with a SOAP/HTTP call, but the service provider is invoked via SOAP/JMS call.

The SCA export component is defined via a WSDL interface to expose a service operation and has a binding type to define the access mechanism for use by the service requester to call the service operations.

The SCA import component is defined via a WSDL interface that defines a service operation and has a binding type to define the access mechanism to use to invoke the service provider.

Note that the SCA import component with the stateless-session EJB binding has a Java-described interface. This is why a wsdl-to-Java conversion must happen between the WSDL-defined interface and a Java-defined interface.

Figure 8-4 shows the content of a simple mediation module.
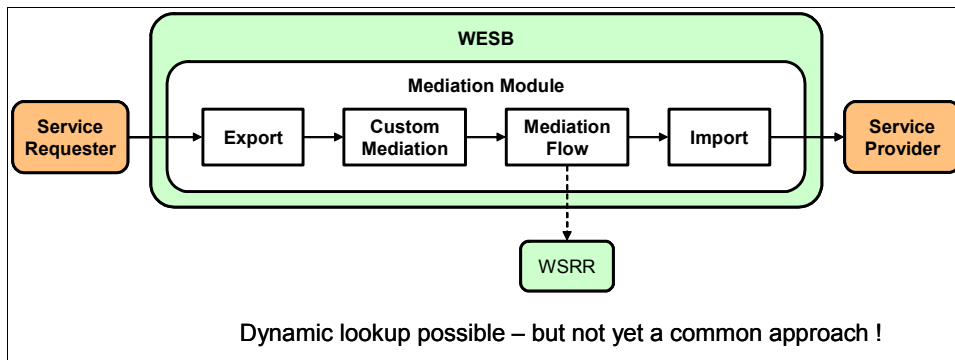
Figure 8-4   Content of a simple mediation module

Note that the custom mediation component is only needed when the mediation flow component cannot achieve the mediation task. When the mediation module only contains the export and the import components, the mediation module can only be used to perform a transport protocol transformation, as shown in Figure 8-5.
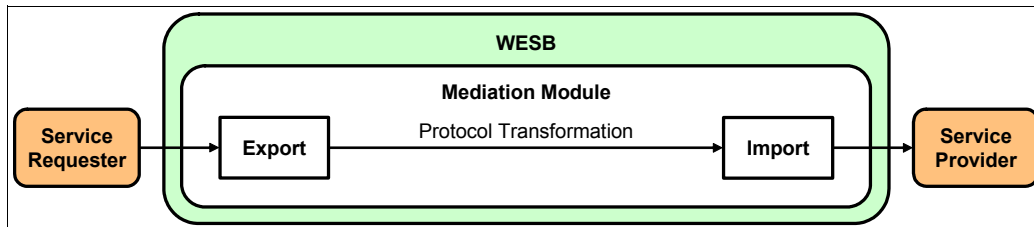
Figure 8-5   Mediation module for transport protocol transformation only

Figure 8-6 shows the various product choices available to perform the protocol transformations required. The SMTP, TCP/IP, and FTP transformations are supported by WebSphere Transformation Extender. (WebSphere Message Broker is shown in a later table in this chapter.)

Out In	HTTP	JMS	MQ	MQ-JMS	SMTP	FTP
HTTP	WESB, WDP, WTX	WESB, WDP, WTX	WESB, WTX, WDP	WESB, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX
JMS	WESB, WDP, WTX	WESB, WDP, WTX	WESB, WDP, WTX	WESB, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX
MQ	WESB, WDP, WTX	WESB, WDP, WTX	WESB, WDP, WTX	WESB, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX
MQ-JMS	WESB, WTX	WESB, WTX	WESB, WTX	WESB, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX
SMTP	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX
FTP	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX	WESB with J2C Adapter, WTX

Figure 8-6   Protocol transformation capabilities

Figure 8-7 shows calls made at the component level of ESB. WebSphere ESB
6.0.2 supports the following bindings on an export component: MQ, JMS, MQ
JMS, SCA, EIS, and Web service.
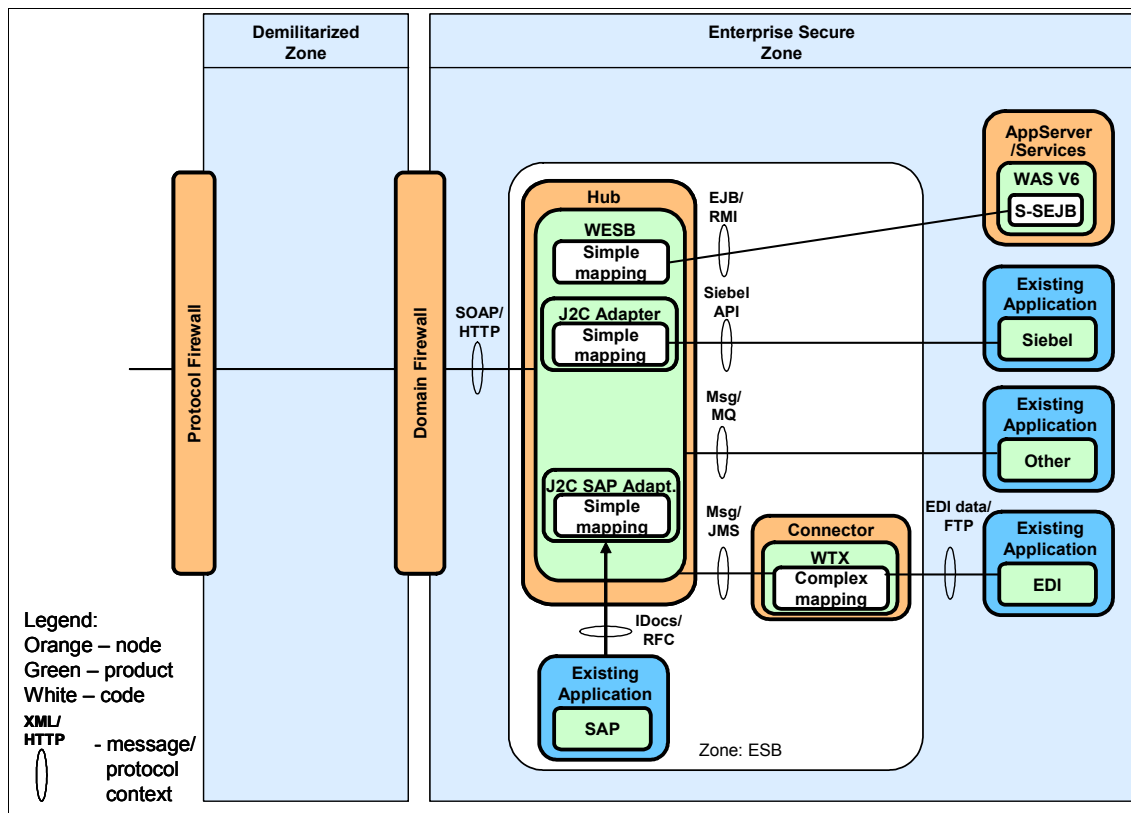
Figure 8-7   Calls made from various ESB clients and ESB components

The requests coming into WebSphere ESB in Figure 8-7 are supported via the
export components. Note the use for the J2C SAP adapter to support the export
component with EIS binding.

The import component supports all the bindings supported on the export
component in addition to the stateless session EJB binding.

The requests going out of WebSphere ESB in Figure 8-7 are supported via the
import components: an import component with S-SEJB binding to call a
stateless-session EJB running in WebSphere Application Server; an import

component with EIS binding to call Siebel. Note that WebSphere ESB leverages WebSphere Transformation Extender to send an FTP request in Figure 8-7 on page 197.

Note that the J2C Siebel adapter and the J2C SAP adapter are shown inside WebSphere ESB because they run in the same JVM as WebSphere ESB, while WebSphere Transformation Extender is outside WebSphere ESB.

The most efficient binding is the SCA binding. In the SCA binding, there are typically two SCA modules involved. One module has an SCA export component that is imported into another module with SCA binding. At run time, the call from the module that has the import component to the module that has the export component is a direct call. The other binding types have a performance overhead. Stateless session EJB binding is likely faster than the Web service binding because a Web service call involves a SOAP parsing performance overhead. JMS, MQ, and MQJMS binding are used where reliability is more important than performance.

One of the capabilities of ESB is transport protocol transformations. A synchronous transport protocol may be transformed to an asynchronous transport protocol. This has to be taken into consideration to understand its effects on client interactions with ESB. For example, let us assume that a service requester sends a request synchronously using SOAP over HTTP to WebSphere ESB. WebSphere ESB forwards the request to WebSphere DataPower using XML over MQ. Then WebSphere DataPower forwards the request synchronously to the service provider using SOAP over HTTP. The service provider creates a reply that is sent synchronously to WebSphere DataPower. At this point, WebSphere DataPower needs to send the correct reply back to WebSphere ESB because WebSphere ESB sent the request asynchronously via MQ. To send the correct reply, WebSphere DataPower needs to copy the MsgId field of the MQMD header from the request message into the CorrelId field of the MQMD header of the reply message. Then, when WebSphere DataPower sends the reply to WebSphere ESB, WebSphere ESB will be able to correlate the reply with the request. WebSphere ESB will then send the reply back to the correct service requester.

8.1.5 Virtualization of Service

One of the major benefits of an ESB is loose coupling or even decoupling. When an ESB pattern is implemented, the service provider can change location, platform, and implementation; add new functions; and change interface, data types, or qualities of service. The service requester knows how to communicate with the service facade exposed by ESB. The service requester interacts with the service facade, not knowing that the real service provider is located somewhere else.

Figure 8-8 shows schematically one service facade and multiple service providers that differ in shape to reflect the possible differences in platform, location, implementation language, interfaces, operations exposed, operation parameters required, and so on. The service facade exposed as a Web service receives a request from a service requester. The Web service could be defined through a wsdl-described interface export component with a Web service binding in a mediation module or could be exposed in WebSphere DataPower or WebSphere Transformation Extender. The WebSphere ESB run time operates on the request using a mediation flow and eventually invokes the real service provider that is implemented as a JMS service or any other supported service.
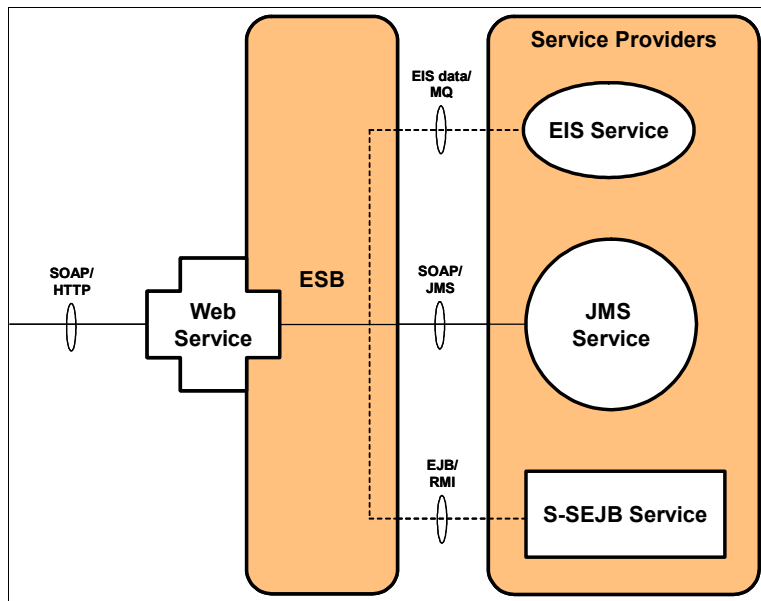
Figure 8-8   Many services can be virtualized as Web services

8.1.6 Dynamic routing

The dynamic routing of a service request is supported in WebSphere ESB 6.0.2 through the use of the /smo/headers/SMOHeader/Target/address element (shown in Figure 8-9) and by checking the box labeled **Use dynamic endpoint if set in the message header** on the Details tab of the Callout node in WID. When the Callout node is configured for dynamic routing, the Callout node associated reference does not have to be wired to any service.
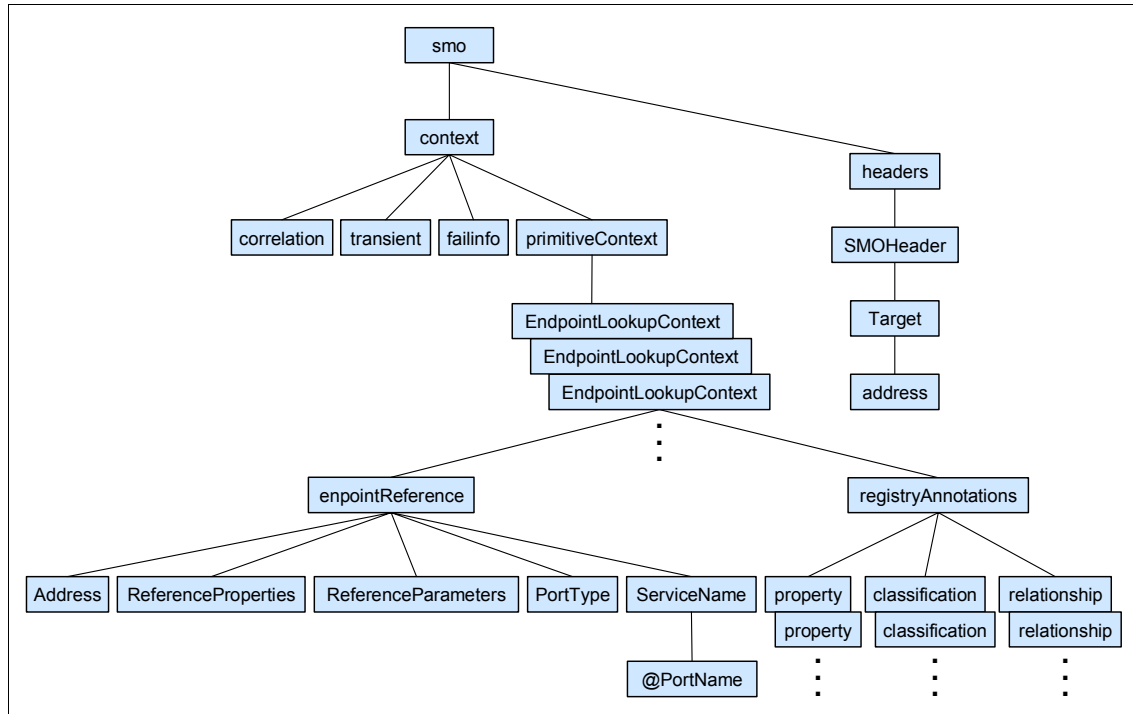
Figure 8-9   A part of the SMO tree

If the address field contains a valid value and the Callout node is configured for dynamic routing, the service identified in the address field is invoked. Or else if the address field is not set and the Callout node associated reference is wired, the default wired service is invoked. Otherwise, an exception is thrown by the WebSphere ESB run time.

The Endpoint Lookup primitive is used in a mediation flow to retrieve service endpoints and related information from WSRR based on selection criteria. The primitive submits a service search request to the registry. The search request has the following parameters:

► Port type name - usually matches the reference interface name set on the Callout node.

► Port type version - specifies a specific version of a service.

► Namespace

► Registry name - specifies the name of a registry configured in the WebSphere ESB run time. The registry name can be found under the registries entry in the navigation table of the WebSphere Administrative Console.

► Match policy - can be set to either "Return one matching endpoint" or "Return all matching endpoints".

► Classifications - a set of URIs defined in the Web Ontology Language (OWL).

► User properties - The user enters values for properties that should be matched for endpoints found in the registry.

If one endpoint is returned from the registry, the SMO address field is set to the endpoint address and the endpoint service is invoked. If more than one endpoint is returned, the SMO primitiveContext subtree shown in Figure 8-9 on page 200 is populated and the SMO tree is propagated forward to the next mediation primitives for endpoint selection logic. For example, these mediation primitives may check the registryAnnotations subtree to find the information associated with the best service endpoint to select for invocation.

To enhance the performance of the Endpoint Lookup primitive, the registry lookups are cached in the WebSphere ESB run time, so the primitive lookups do not have to be performed against the registry unless the associated cache lookup entry has been invalidated or timed out. The registry lookups cache is not available to the developer, but it is available to the administrator, who can configure the registry lookups cache.

Another primitive that can be used in conjunction with the dynamic routing mechanism available with WebSphere ESB is the DB Lookup primitive, but the use of the Endpoint Lookup primitive helps make WebSphere ESB mediation flows more flexible because the primitive can retrieve potentially many service endpoints with different properties. These service endpoints and associated information are not hard coded in the mediation flow. They are specified in the registry, which can be updated independently of the mediation flow.

If a mediation module contains a call to a WebSphere Transformation Extender map in a custom mediation component or an import component with a custom

JMS binding, the map input and output cards can be selected dynamically at run time.

8.1.7 Inter-communication

Figure 8-10 shows the communication between the components of ESB. Note that WebSphere Transformation Extender can be integrated with WebSphere ESB in a number of ways. WebSphere Transformation Extender can be connected with WebSphere ESB through JMS, MQ, or HTTP. Alternatively, WebSphere Transformation Extender maps can be invoked from within a custom mediation inside a WebSphere ESB mediation module or from within a custom JMS binding Java class of an import component inside a WebSphere ESB mediation module. WebSphere DataPower can be connected with WebSphere ESB through HTTP, JMS, and MQ.

Product / Product	WESB	WTX	WDP
WESB		MQ, JMS, MQ-JMS, SOAP/HTTP	MQ, JMS, SOAP/HTTP
WTX	MQ, JMS, MQ-JMS, SOAP/HTTP		MQ, JMS, SOAP/HTTP
WDP	MQ, JMS, SOAP/HTTP	MQ, JMS, SOAP/HTTP	

Figure 8-10   Inter-communication between the components of ESB

8.1.8 Resiliency

The WebSphere ESB product, which is the core ESB product in this section, can be made highly available by leveraging the clustering capability in WebSphere Application Server. Figure 8-11 on page 203 shows three clusters: a messaging engine cluster (ME Cluster), an application cluster (Application Cluster), and a common event infrastructure cluster (CEI Cluster), all of which are managed in a WebSphere Application Server cell by the deployment manager. Each of these clusters is shown to have its own database — MEDB for JMS messages persistence, APPDB for use by SCA mediation modules, and CEIDB for use by

the CEI server. Only one messaging engine is active in the messaging engine cluster. It is active on one of the application servers of the cluster.
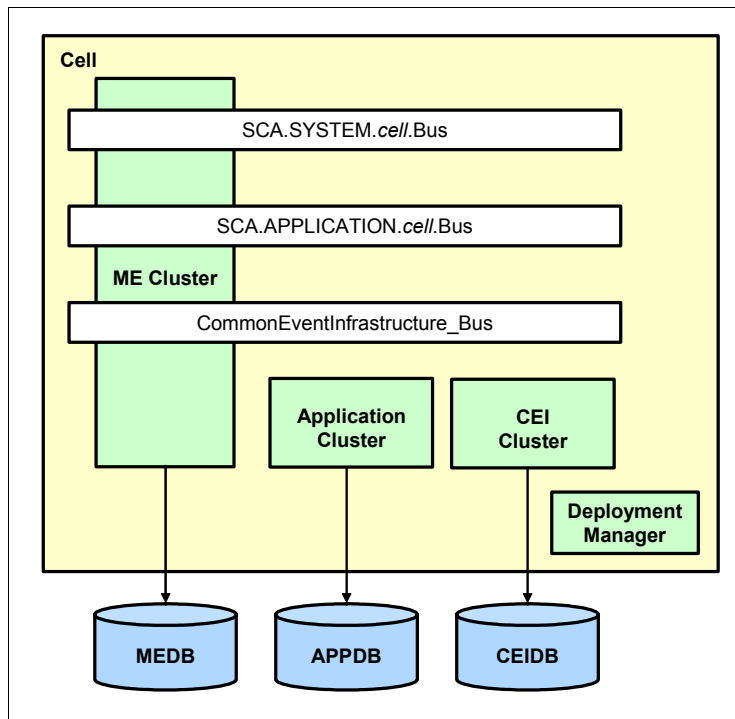
Figure 8-11   WebSphere ESB clustered configuration

If the messaging engine fails, the HA manager will start the messaging engine on another application server member of the messaging engine cluster. This configuration does not increase the message throughput of the messaging engine, but increases messaging engine availability.

The message throughput is increased through a bus destination partitioning across multiple physical queues. The workload manager will distribute all messages destined for that partitioned bus destination across all the underlying physical queues. This eliminates the potential bottle-necking behavior with a single physical queue. To accomplish the bus destination partitioning, there needs to be one messaging engine active on each application server member of the messaging engine cluster. This way, each messaging engine will host an instance of a physical queue that is mapped to the partitioned destination. Beware that destination partitioning has negative implications.

There are four different implications associated with a bus destination partitioning:

► Unbalanced connections to the physical queues
► Affinity problems between the message and the application
► Message orphans on failed messaging engines
► Message sequencing problem

Avoid the cluster topology that makes use of a bus destination partitioning unless your application is stateless. It can deal with messages that are out of sequence, and delays in processing message orphans are not a problem.

The topology shown in Figure 8-11 on page 203 enables the administrator to tune each cluster and each database independently for optimal performance, availability, and scalability. The CEI and messaging engine clusters do not run user applications. User applications like mediation flow applications run in the application cluster.

The application cluster runs the same SCA mediation module enterprise applications on every member of the cluster. This increases request throughput and availability of the mediation applications.

The CEI server is running on each application server member of the CEI cluster. This increases the request throughput and availability of the CEI server.

Figure 8-12 shows the WebSphere Application Server HA manager and the services it monitors. Note the use of a database for JMS messages, HTTP sessions, and stateful session beans persistence. Also note the use of an external shared storage for WebSphere Application Server transaction logs.
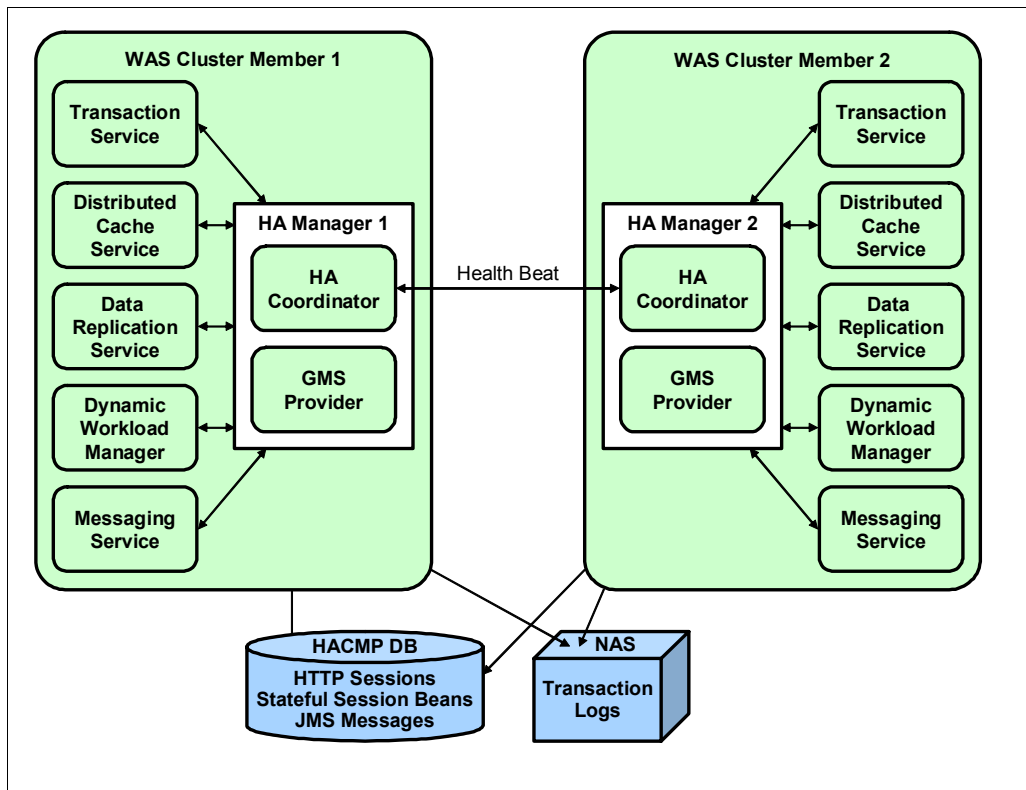
Figure 8-12   WebSphere Application Server HA manager

The application components (such as the Event Emitter primitive in a mediation flow) running in the application cluster interact with the CEI server running in the CEI cluster either synchronously or asynchronously. The synchronous interaction is done through an EJB call to the CEI server. The asynchronous interaction is done through a message queue hosted by a messaging engine running in the messaging engine cluster. Both types of interactions are highly available.

Figure 8-13 shows the two types of interactions between the application components and the CEI server. The recommendation is to use the asynchronous interaction between the application and the CEI server. Using the asynchronous interaction is preferable because the application does not block until the event is logged in the database by the CEI server. The type of interaction between the application and the CEI server is configurable via WebSphere Administrative Console.
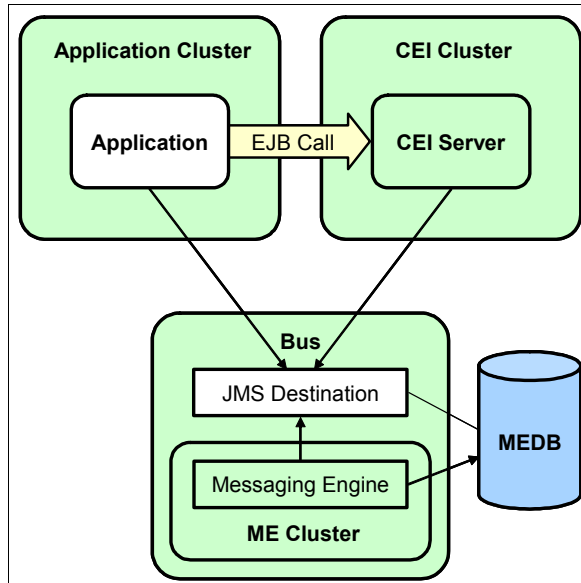
Figure 8-13   Application interactions with CEI

A simpler topology to set up is shown in Figure 8-14, but this topology is not as flexible as the topology shown in Figure 8-11 on page 203. Figure 8-14 shows that there is only one cluster for all WebSphere ESB components. Each component still has its own database.
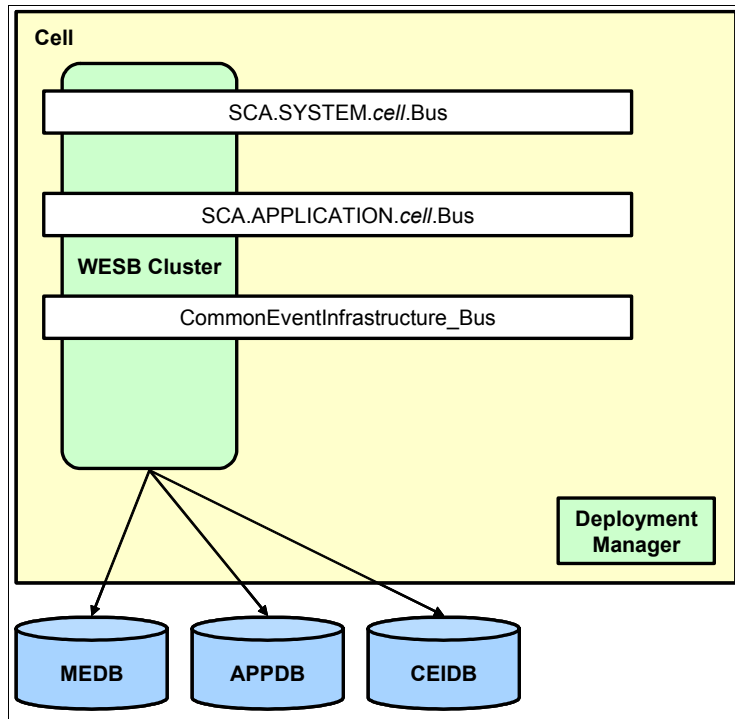
Figure 8-14   One WebSphere ESB cluster for all components with separate databases

Figure 8-15 shows the simplest, but least flexible topology, where there is only one WebSphere ESB cluster where all components share the same database.
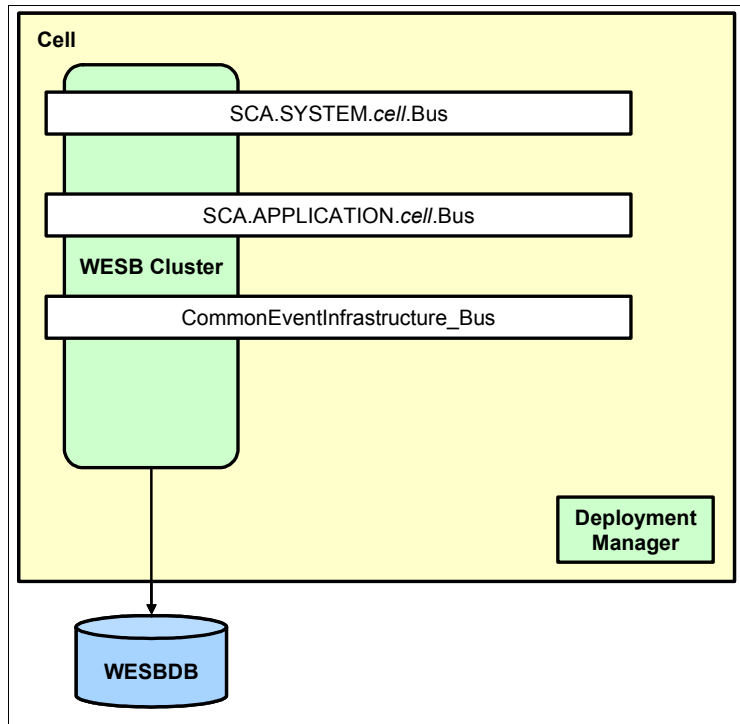
Figure 8-15   The least flexible WebSphere ESB topology

8.1.9 Qualities of service

In the WebSphere ESB-based architecture, the qualities of service are set at different levels and scope in the WebSphere ESB runtime artifacts. Table 8-2 lists the qualities of service that can be set in different scopes of the runtime artifacts of WebSphere ESB.

Table 8-2   Qualities of service available in the WebSphere ESB run time

Artifact type	Qualities of service
Interface	Join activity session Join transaction Security permission
Operation	Event sequencing Join activity session Join transaction Security permission
Reference	Asynchronous invocation Reliability Request expiration Response expiration Suspend activity session Suspend transaction

A quality of service set at the interface level can be overridden at the operation level, but cannot be removed at the operation level. The quality of service can be removed at the interface level.

The type of interactions between two SCA components in a mediation module can be set to either synchronous or asynchronous. Also, the interaction type between the SCA mediation module and a component outside WebSphere ESB is determined via the binding type of the export or the import component used to interact with the component outside the WebSphere ESB run time.

There are a number of different delivery qualities of service provided by the WebSphere Application Server messaging engine. When the WebSphere Application Server messaging engine is used (as is the case with the internal asynchronous communication used in the WebSphere ESB run time) the following delivery qualities of service are available: assured persistent, reliable persistent, reliable nonpersistent, express nonpersistent, and best effort nonpersistent.

Assured persistent messages have the most negative impact on performance but are the most reliable, while the best effort nonpersistent is best for performance but is the least reliable.

Table 8-3 lists the characteristics of the various levels of the delivery quality of service.

Table 8-3   Delivery qualities of service

Characteristic	Best effort nonpersistent	Express nonpersistent	Reliable nonpersistent	Reliable persistent	Assured persistent
JMS Delivery mode	Nonpersistent	Nonpersistent	Nonpersistent	Persistent	Persistent
Transactionally atomic?	No	Yes, but messages do not survive restart	Same as express nonpersistent	Yes	Yes
Messages hardened?	No	Only as a result of resource shortage	Same as express nonpersistent	Yes, asynchronously	Yes, synchronously
Discarded in normal operations?	Yes	No	No	No	No
Duplicated	No	Possibly	Possibly	Possibly	No
Messages survive planned shutdown?	No	No	No	Yes	Yes
Messages survive client communication failure?	No	No	Yes	Yes	Yes
Messages survive engine communication failure?	No	Yes	Yes	Yes	Yes
Messages survive engine crash?	No	No	No	Hardened messages are recovered	Yes

For a message received from WebSphere MQ into the WebSphere Application Server messaging engine, the delivery qualities of service are mapped by default. See Table 8-4.

Table 8-4   Delivery qualities of service mapping

WebSphere MQ	WebSphere Application Server Messaging Engine
Persistent	Assured persistent
Nonpersistent	Express nonpersistent

For a message sent to WebSphere MQ from the WebSphere Application Server messaging engine, the delivery qualities of service are mapped as follows (Table 8-5) when it is received by MQ.

Table 8-5   Delivery qualities of service mapping

WebSphere Application Server Messaging Engine	WebSphere MQ
Reliable persistent	Persistent
Assured persistent	Persistent
Reliable nonpersistent	Nonpersistent
Express nonpersistent	Nonpersistent
Best effort nonpersistent	Nonpersistent

Note that the MQ link needed to connect WebSphere MQ to the WebSphere Application Server messaging engine is made highly available through SupportPac MR01 found here:

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24013895&loc=en_US&cs=utf-8&lang=en

8.2 WebSphere Message Broker-based ESB architecture

The WebSphere Message Broker-based ESB architecture is based on a C/C++ high-performance broker engine. Figure 8-16 shows a relatively simple WebSphere Message Broker-based ESB architecture. Figure 8-16 is quite similar to Figure 8-1 on page 183, which shows a simple WebSphere ESB-based architecture. Architecturally, both WebSphere Message Broker and WebSphere ESB are used as brokers, but each one is suited for a specific environment.
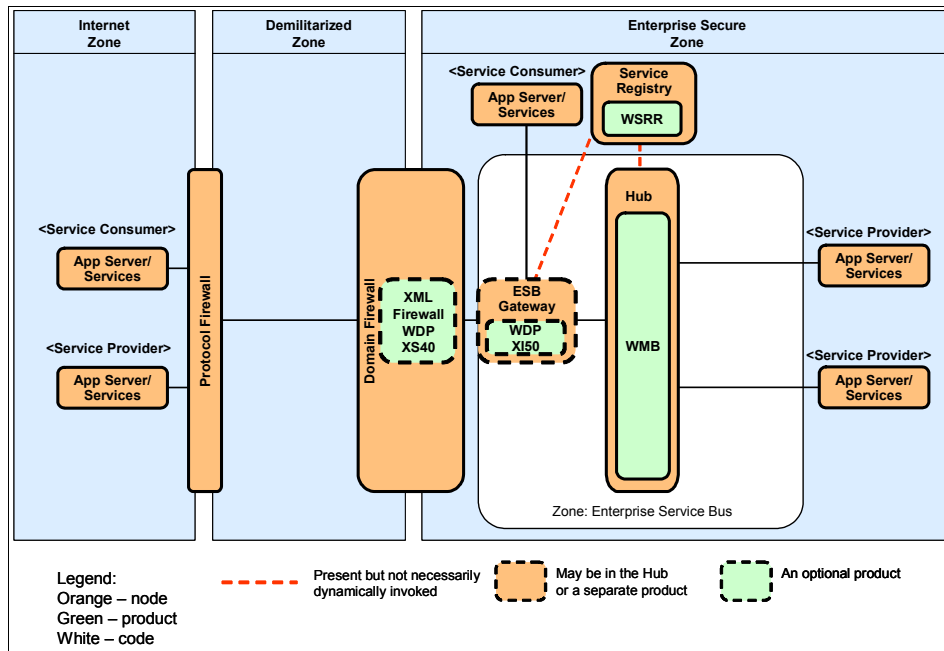
Figure 8-16   A simple WebSphere Message Broker-based ESB environment

Figure 8-16 shows WebSphere Message Broker to be at the heart of the ESB topology. It receives service requests from the Internet and intranet. It routes these service requests intelligently to the correct service providers. The configuration shows that optionally either a WebSphere DataPower XS40 may be deployed as an XML firewall or a WebSphere DataPower XI50 may be deployed as an explicit ESB Gateway. Using a message flow developed in Message Brokers Toolkit and deployed in the WebSphere Message Broker run time, WebSphere Message Broker routes the service request optionally coming from WebSphere DataPower to the service provider that may be located in the intranet, as shown in Figure 8-16.

WebSphere Message Broker may leverage a number of IBM WebSphere products such as WebSphere Transformation Extender, WebSphere DataPower, and WBI Adapters to interact with the service providers such as a CICS program or an SAP BAPI. WebSphere Message Broker may also leverage a number of SupportPacs to interact with EIS systems. For the dynamic routing of service requests, WebSphere Message Broker may leverage WSRR to locate a service provider before invocation.

The service requests may be sent to WebSphere Message Broker from the intranet. To simplify interactions with WebSphere Message Broker, IBM provides the Message Service Client for C/C++ SupportPac that enables C/C++ applications to exchange messages with WebSphere Message Broker. A Java application may leverage a number of Java APIs provided by WebSphere MQ to communicate with WebSphere Message Broker. WebSphere MQ provides JMS, Java MQ, and AMI APIs. The Java application may run standalone or inside a J2EE application server such as WebSphere Application Server.

Similar to Internet-originated requests, intranet-originated requests invoke facade services exposed by WebSphere Message Broker. Then, through WebSphere Message Broker message flows, the actual service providers that may be located internally or externally are invoked to service these service requests.

8.2.1 Platforms support

Table 8-6 shows the support of various platforms by WebSphere Message Broker shown in Figure 8-16 on page 212.

Table 8-6   WebSphere Message Broker support for various platforms

Product	Supported platforms[a]
WebSphere Message Broker	AIX, HP-UX, Linux, Solaris, Windows, and Z/OS

a. Check the platform details from the following link:
http://www-306.ibm.com/software/integration/wbimessagebroker/requirements/

8.2.2 WebSphere Message Broker-based ESB candidate environment

WebSphere Message Broker is a good fit for an environment that has the following characteristics:

► WebSphere MQ already exists in the environment.

  WebSphere Message Broker leverages WebSphere MQ qualities of service like message delivery, transaction management, authorization, and so on.

- ► High-performance is needed.

  The WebSphere Message Broker is a high-performance message flow engine. A processing speed of more than 1700 messages per second has been reported for messages of 1 KB in size.

- ► Support for various transport protocols is needed.

  If an environment needs messages to be exchanged over many different transport protocols, WebSphere Message Broker will be a great fit. WebSphere Message Broker supports all MQ-related transports in addition to other types of transport like HTTP, FTP, TCP/IP, SMTP, LDAP, JMS, SOAP over HTTP/S, and over MQ.

- ► Application integration is needed.

  Through SupportPacs, WebSphere Message Broker supports native connectivity and processing for CICS, Z/OS VSAM and Z/OS QSAM, E-mail, FTP, flat file, LDAP, TLog, Zip/Unzip, and Remote Server Retail Data Extensions. WBI Adapters enable WebSphere Message Broker to integrate with many other applications.

- ► XML and non-XML data transformations are needed.

  WebSphere Message Broker has many built-in parsers that are capable of processing many data formats. If complex data transformation is needed, WebSphere Message Broker provides a number of nodes that simplify the mapping tasks for the developer. WebSphere Message Broker provides the following built-in nodes: Compute, Database, DataDelete, DataInsert, DataUpdate, Extract, JavaCompute, JMSMQTransform, MQJMSTransform, Mapping, Warehouse, and XMLTransformation, in addition to other nodes provided in SupportPacs that help you develop complex data transformations. For even more challenging data transformation tasks, WebSphere Message Broker leverages WebSphere Transformation Extender for Message Brokers product that integrates WebSphere Transformation Extender's powerful data transformation capabilities inside message flows.

- ► Support for XML and non-XML data formats is needed.

  WebSphere Message Broker supports many industry data formats like HL7, EDI X12, EDIFACT, SWIFT, and so on. If there is a need for a data format that is not supported by WebSphere Message Broker, the more comprehensive data format support provided by WebSphere Transformation Extender can be leveraged easily in WebSphere Message Broker.

- ► Complex event processing is needed.

  The SupportPac IA0S provides two powerful event processing nodes that can be included in message flows for complex event processing. Using an embedded complex-event processing engine, these nodes monitor the message flows for complex situations and intelligently react to these

situations to drive message flows. The two nodes are SituationManager and IntelligentFilter.

► A mixture of implementation languages is needed.

WebSphere Message Broker supports custom nodes written in C and in Java.

► Support for standards is needed.

WebSphere Message Broker supports JMS, WSDL 1.1, WS-I Basic Profile 1.0, SOAP, HTTP/S, SOAP over MQ, SOAP over JMS, SOAP over HTTP/S, and SOAP with Attachment. The number of supported standards will increase with WebSphere Message Broker in the future.

► Dynamic routing is needed.

WebSphere Message Broker dynamic routing capabilities are quite powerful. WebSphere Message Broker supports WSRR through a number of nodes introduced with WebSphere Message Broker Fix Pack 6.0.3. These nodes are included in message flows to query WSRR for certain information that would dynamically affect message routes through the message flows. WebSphere Message Broker supports dynamic routing through other means also.

8.2.3 Message modeling

WebSphere Message Broker provides a number of pre-built message types as SupportPacs so that developers do not have to develop the message sets and message types themselves. These message types cover a number of industries, like health care and financial industries. If a message type is not supported in a SupportPac, a custom message type can be built in the WebSphere Message Brokers Toolkit.

The WebSphere Message Broker supports different data format types like Tagged or Delimited String (TDS) format type, Custom Wire Format (CWF) type, and XML format type. Messages compliant with ACORD, AL3, EDIFACT, HL7, and X12 fall in the TDS format category where data elements in the message are separated by a well-known delimiter like the asterisk. Messages like a C structure or a COBOL copybook fall in the CWF format category where data elements in the message are not separated by a delimiter.

8.2.4 Data Format Transformation

WebSphere Message Broker does not need the service request to be in a specific data format. WebSphere Message Broker has a number of built-in parsers that are capable of parsing many data formats. The data may be transformed as it flows through the message flow using one or more of the data transformation nodes. Data may have to be transformed to another format so that the service provider accepts it. When it comes to data transformations,

WebSphere Message Broker is much more sophisticated than WebSphere ESB because WebSphere Message Broker can natively do XML-to-Non-XML mapping and vice versa. WebSphere Message Broker can perform the data transformation between any two format categories like XML-to-TDS and between two formats of the same category like XML-to-XML.

WebSphere Message Broker natively comes with a number of robust message transformation nodes:

► To transform the request message from one XML format to another XML format, the XMLTransformation node can be used to perform the transformation according to a specified XSL style sheet.

► The mapping node accepts the request message and a database as inputs to create a new output message.

► For more complex data transformation, the Compute node or the JavaCompute node can be used. The Compute node accepts the request message and a database as inputs and can create multiple output messages. To perform data transformation, the Compute node uses ESQL code.

► The JavaCompute node enables the developer to leverage the full power of the Java language in a message flow to transform request messages, to interact with databases, to call a service like an EJB or a Web service, and so on.

All of the transformation nodes can perform the data transformation on the message body as well as the message headers.

Message parsing

The message parser specified on the input node (such as the MQInput node) receives the message as a bit stream and creates a logical tree for processing such as data transformation through the message flow. There are four different subtrees that make up the logical tree:

► The message subtree
► The Environment subtree
► The LocalEnvironment subtree
► The ExceptionList subtree

Figure 8-17 shows the message tree. This tree is specific to a message received via an MQ transport. For a message received via another transport protocol, the headers will be different. If a user-defined parser was used to create the message tree, the tree may look different from this tree. The Properties element shown in the tree is used by the application.
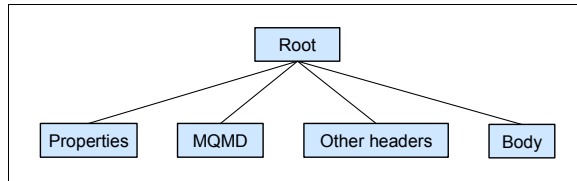
Figure 8-17   The message tree

Figure 8-18 shows the environment tree. This tree is used by the application. The application can create any variable in the tree to be used during the message flow processing. Figure 8-19 on page 218 shows the LocalEnvironment tree. There are more details provided about this tree in 8.2.7, "Dynamic routing" on page 222. The ExceptionList tree contains exception details about a message flow processing failure. The ExceptionList tree is shown in Figure 8-20 on page 218. Other exception types that may be shown in the ExceptionList tree are ParserException, ConversionException, UserException, and DatabaseException. All of these trees are available in the message flow for processing, such as transformation.
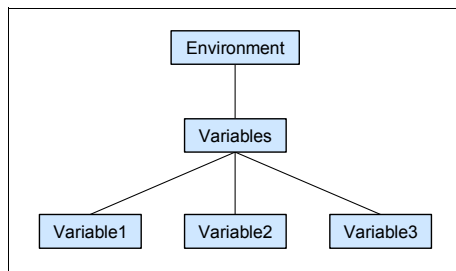
Figure 8-18   The environment tree
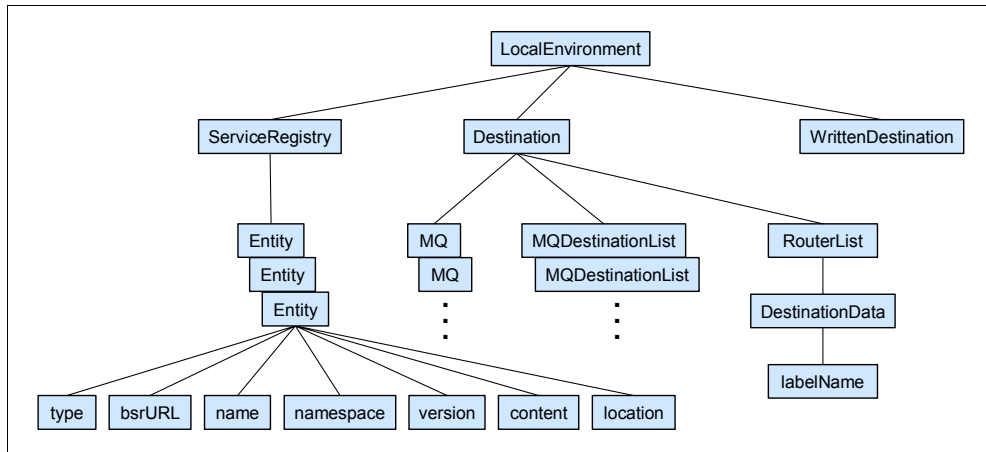
Figure 8-19   Portion of the LocalEnvironment tree
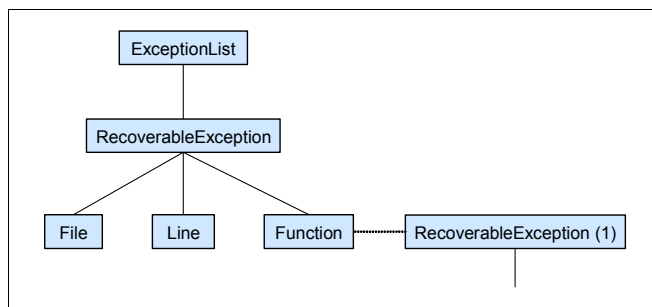
Figure 8-20   A portion of the ExceptionList tree

External data transformation

Like WebSphere ESB, WebSphere Message Broker can also leverage
WebSphere Transformation Extender for more sophisticated mapping, especially
when many-to-many mapping is required. Also like WebSphere ESB,
WebSphere Message Broker can leverage WebSphere DataPower for a
specialized, but simple mapping like encrypting a specific security-sensitive field
in a message.

Figure 8-21 shows one WebSphere Message Broker-based ESB topology used for both simple or complex data transformation. In Figure 8-21, WebSphere Message Broker is being used to transform an XML message received to possibly another XML format before passing the message to a Web service running in WebSphere Application Server. WebSphere Message Broker is also leveraging WebSphere Transformation Extender in Figure 8-21 to transform the input XML data and a database input to EDI X12.
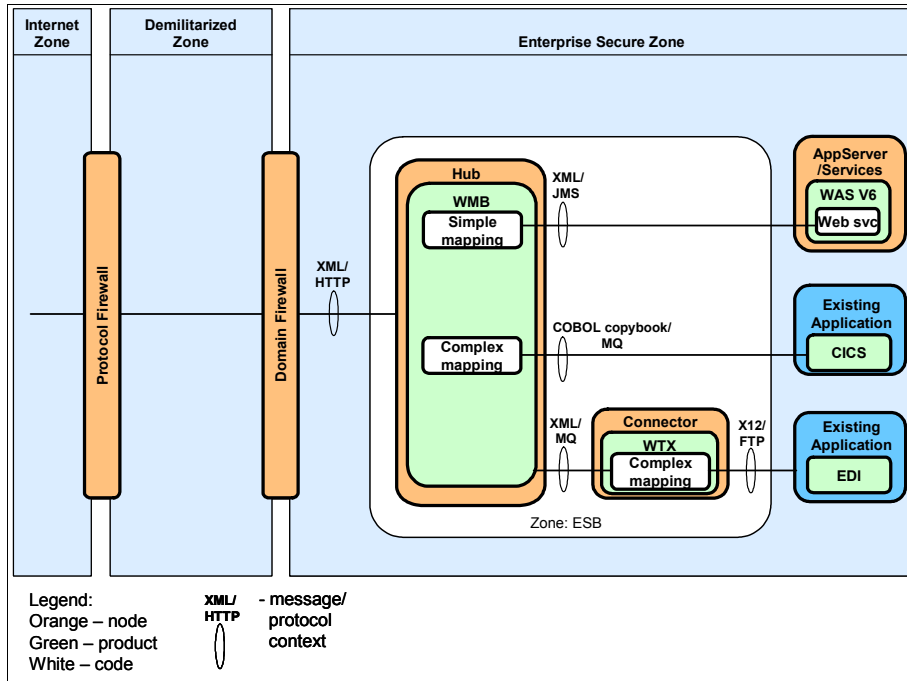
Figure 8-21   Data transformation in WebSphere Message Broker-based ESB

8.2.5 Protocol transformation

Figure 8-22 shows the various product choices available to perform the protocol transformations required. The ESB component is a logical component that may include WebSphere Message Broker, WebSphere DataPower, WebSphere Transformation Extender, and WBI Adapters.

Outbound format / Inbound format	MQ	JMS	MQ-JMS	HTTP	EIS	FTP	RMI over IIOP
MQ	WMB, WDP, WTX	WMB, WDP, WTX	WMB, WTX	WMB, WDP, WTX	WMB, WTX	WMB, WTX	WMB
JMS	WMB, WDP, WTX	WMB, WDP, WTX	WMB, WTX	WMB, WDP, WTX	WMB, WTX	WMB, WTX	WMB
MQ-JMS	WMB, WTX	WMB, WTX	WMB, WTX	WMB, WTX	WMB, WTX	WMB, WTX	WMB
HTTP	WMB, WDP, WTX	WMB, WDP, WTX	WMB, WTX	WMB, WDP, WTX	WMB, WTX	WMB, WTX	WMB
EIS	WMB, WTX	WMB, WTX	WMB, WTX	WMB, WTX	WMB, WTX	WMB, WTX	WMB
FTP	WMB, WTX	WMB, WTX	WMB, WTX	WMB, WTX	WMB, WTX	WMB, WTX	WMB

Figure 8-22   Protocol transformations by ESB

WebSphere Message Broker natively comes with many built-in nodes that support different transport protocols. These nodes can be wired together in many different ways to support protocol transformations. For example, the MQInput node supports the receipt of a message using the MQ transport protocol, and the HTTPRequest node sends a SOAP message using the HTTP protocol to a Web service. The two nodes can be wired together in a message flow in such a way that a message received via MQ is sent via SOAP over HTTP.

If a specific transport protocol is not supported in the WebSphere Message Broker built-in nodes, there are SupportPacs that are available to support additional transport protocols like TCP/IP, LDAP, SMTP, and FTP nodes. If additional transport protocols support is needed beyond what is provided in the built-in and SupportPacs nodes, WebSphere Transformation Extender can be leveraged for its transport protocols conversion capabilities.

8.2.6 Virtualization of service

One of the major benefits of an ESB is loose coupling or even decoupling. When an ESB pattern is implemented, the service provider can change location, platform, implementation; add new functions; and change interface, data types, or quality of services. The service requester knows how to communicate with the service facade exposed by the ESB. The service requester interacts with the service facade, not knowing that the real service provider is located somewhere else. Figure 8-23 shows schematically one service facade and multiple service providers that differ in shape to reflect the possible differences in platform, location, implementation language, interfaces, operations exposed, operation parameters required, and so on. The service facade exposed as a Web service receives a request from a service requester. The WebSphere Message Broker run time operates on the request using a message flow and eventually invokes the real service provider that is implemented as a JMS service or any other supported service.
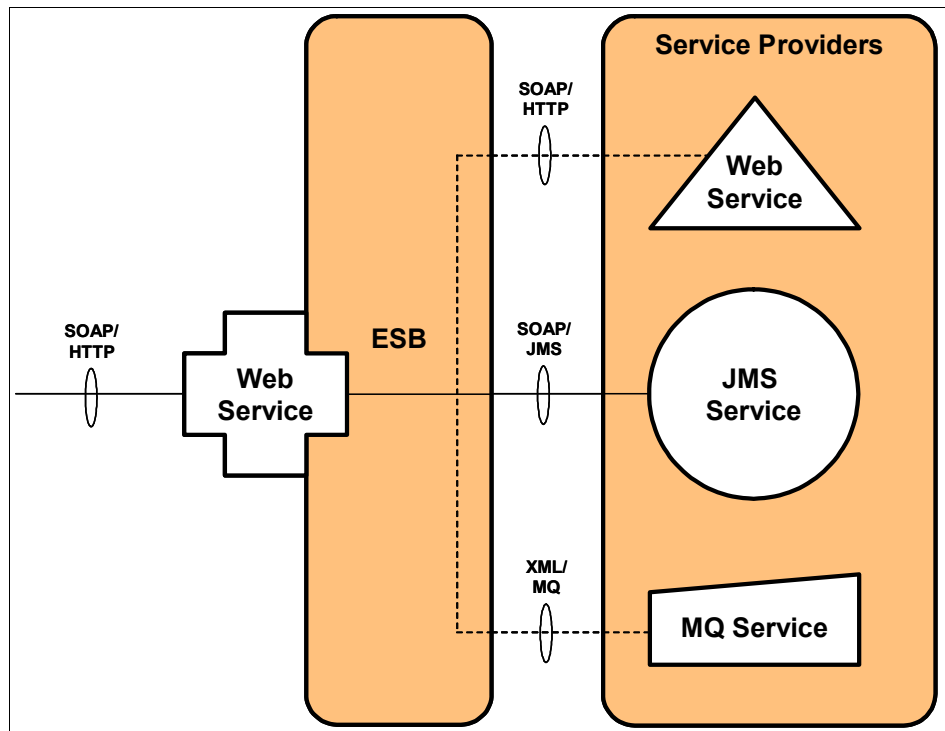
Figure 8-23   Many services can be virtualized as Web services

Figure 8-24 shows a Web service virtualized by WebSphere Message Broker as an MQ service. The MQInput node receives the MQ request, which goes through some processing in a message flow. When the HTTPRequest node is reached, a Web service call is issued to a Web service running in WebSphere Application Server. The Web service sends a reply that may get processed through the message flow that uses the MQOutput node to send a reply to the original service requester.
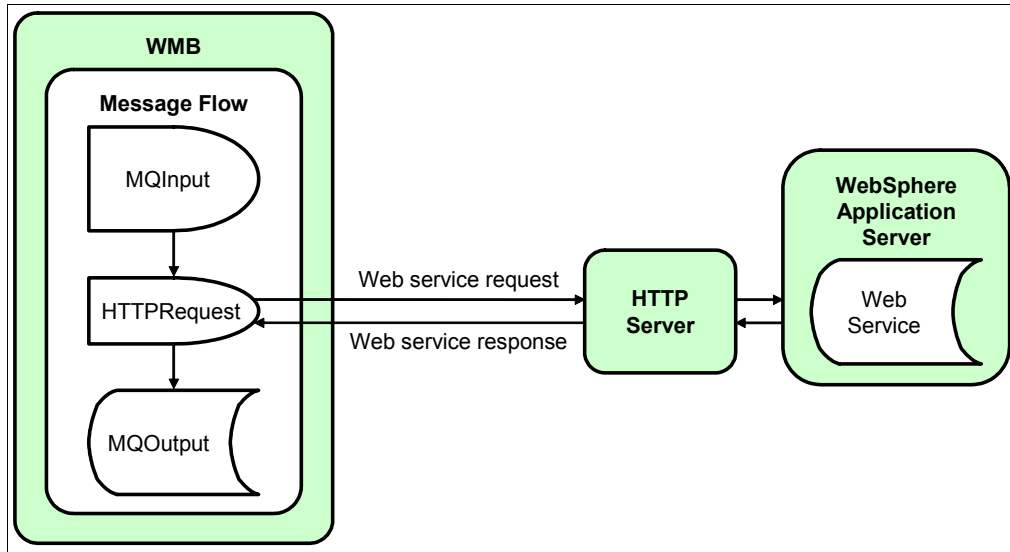
Figure 8-24   A Web service call made from an HTTPRequest node

Figure 8-24 shows a service virtualization using the various components of a multi-component ESB. Some service providers get invoked directly by WebSphere Message Broker, some get invoked by WebSphere Transformation Extender, some get invoked by WebSphere DataPower, and some may be invoked by a WBI adapter.

8.2.7 Dynamic routing

The dynamic routing in WebSphere Message Broker can be accomplished by using the RouteToLabel and Label nodes. In a message flow designed for dynamic routing, the RouteToLabel node reads the RouterList structure set in the LocalEnvironment tree of the message and routes the message to the Label node identified in the RouterList structure. The RouteToLabel node is not wired to the Label nodes. The route taken by the message is determined at run time.

Figure 8-25 shows a portion of the MQ message LocalEnvironment tree relevant to the dynamic routing capability. Note that the Entity, MQ, and MQDestinationList elements are arrays. Also note that this LocalEnvironment tree is specific to MQ messages.
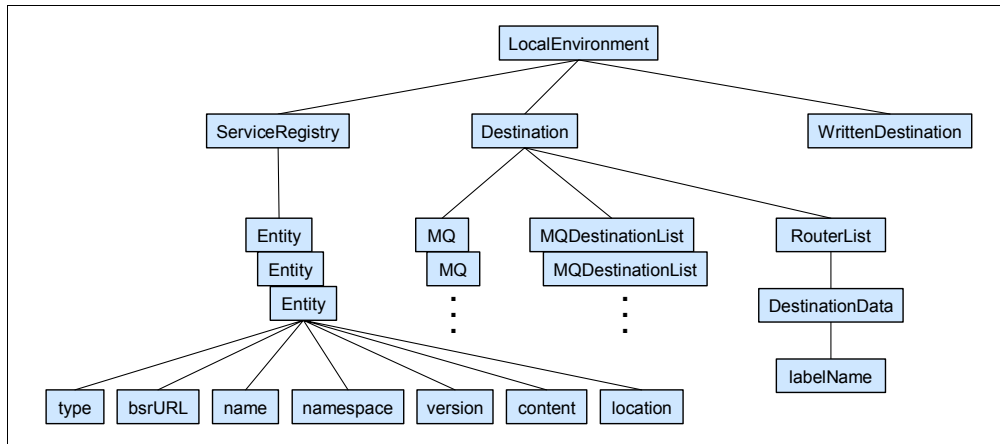
Figure 8-25   A portion of the MQ message LocalEnvironment tree

The dynamic routing in WebSphere Message Broker is enhanced with the use of WSRR and the WSRR-related nodes available with the WSRR client for WebSphere Message Broker. There are five WSRR-related nodes that can be used in WebSphere Message Broker message flows just like any built-in node. The WSRR-related nodes are SRGetVirtualService, SRProcessVirtualService, SRDispatchVirtualService, SRRetrieveITService, and SRRetrieveEntity.

At run time, the SRGetVirtualService node sets the labelName field shown in Figure 8-25 for the RouterToLabel node based on the relationship type, relationship value, and endpoint alias retrieved from WSRR. When the message is propagated from the SRGetVirtualService node to the RouterToLabel node, the RouterToLabel node reads the labelName field and routes the message to the Label node identified in the labelName field.

Figure 8-26 shows the steps involved in the dynamic routing behavior described here. The SRProcessVirtualService node works similarly to the SRGetVirtualService node, but it sets the labelName field to a value that contains a label name and a supported protocol such as MQ, URL, or JMS. At run time, the dynamic routing behavior is similar to the behavior described in the case of the SRGetVirtualService node. The SRDispatchVirtualService is used to establish the endpoint of the target service.
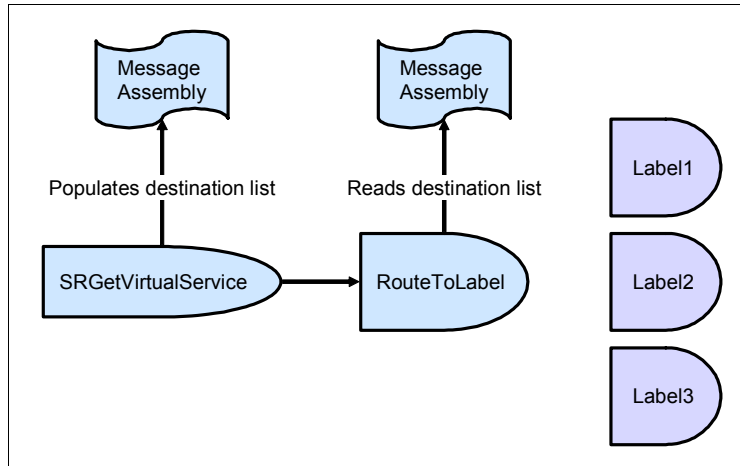
Figure 8-26   Dynamic routing support in WebSphere Message Broker

The SRRetrieveEntity node retrieves any type of documents stored in WSRR. Once these documents are retrieved, the SRRetrieveEntity node populates the ServiceRegistry structure shown in Figure 8-25 on page 223. Note that the Entity field shown in the tree is an array. It is conceivable that the information populated

in the ServiceRegistry structure can be easily used to determine a message route at run time. Figure 8-27 illustrates this point. The SRRetrieveITService node can be configured to return either one service endpoint or multiple service endpoints.
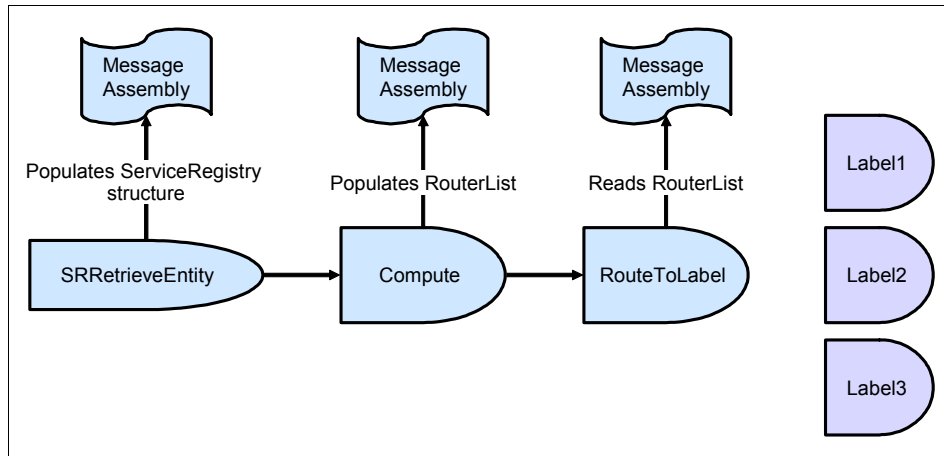
Figure 8-27   Use of SRRetrieveEntity in dynamic routing in WebSphere Message Broker

To improve the performance of WSRR lookups that can be made by all of these five nodes, WebSphere Message Broker retains a local cache of lookup data so that the lookup requests are sent to WSRR only when the cache cannot service the lookup requests.

In cases where WSRR is not used to provide information used for dynamic routing, the RouteToLabel and Label nodes can still be used, but logic has to be built before the RouteToLabel node in the message flow to populate the RouterList structure. This can be done by using a node such as a Compute or a JavaCompute node. The node populates the RouterList structure based on the message content, a database lookup, and some ESQL or Java logic.

Figure 8-28 shows the steps involved in the dynamic routing capability of WebSphere Message Broker without using WSRR. Each Label node may be connected to a message subflow that provides further processing to the message. The message subflow may end with a call to a Web service or further dynamic routing.
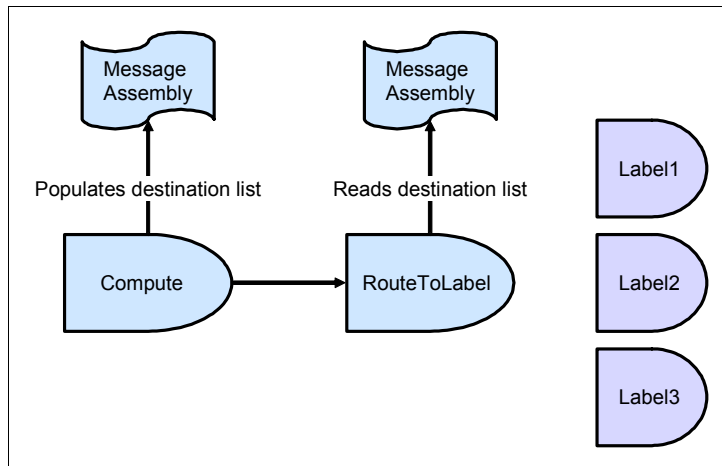
Figure 8-28   Dynamic routing in WebSphere Message Broker without WSRR

Other mechanisms can also be used in WebSphere Message Broker for dynamic routing. For example, the Filter node can be used for dynamic routing, but the nodes RouteToLabel and Label are more robust than the Filter node because they can do a more complex routing logic.

8.2.8 Inter-communication

Figure 8-29 shows the communication between the components of a WebSphere Message Broker-based ESB.

Product \ Product	WMB	WTX	WDP
WMB		MQ, JMS, SOAP/HTTP	MQ, JMS, SOAP/HTTP
WTX	MQ, JMS, SOAP/HTTP		MQ, JMS, SOAP/HTTP
WDP	MQ, JMS, SOAP/HTTP	MQ, JMS, SOAP/HTTP	

Figure 8-29   Inter-communication between the WebSphere Message Broker-based ESB components

Note that WebSphere Transformation Extender can be integrated with WebSphere Message Broker in a number of ways. WebSphere Transformation Extender can be connected with WebSphere Message Broker through JMS, MQ, and HTTP. Also, a WebSphere Transformation Extender node is available for message flow developers in the Message Brokers Toolkit. To get the WebSphere Transformation Extender node, the WebSphere Transformation Extender for Message Broker product needs to be installed.

8.2.9 Resiliency

In this section we discuss the ability to return to the original state.

HACMP for AIX platforms

High-Availability Cluster Multi-Processing (HACMP™) is IBM flagship software that enables critical software services, like databases, to be highly available on AIX-based clustered machines. The simplest HACMP environment consists of two machines in an active-passive or active-standby configuration, whereby the active machine is doing critical data processing and the passive machine is

waiting to take over data processing from the active machine when necessary. If a failover-inducing event takes place on the active machine, the data processing must be routed from the failing active machine to the healthy passive or standby machine. A more advanced HACMP environment consists of two machines in an active-active configuration, whereby both machines are doing critical data processing. If a failover-inducing event takes place on one machine, the data processing must be routed from the failing machine to the healthy machine.

Upon failover, the healthy machine, in an HACMP cluster, must be able to process its own data as well as the newly routed stream of data. This means that the software, processing the data stream on the failing machine, must be able to process the same data stream when it is routed to the healthy machine. This requirement dictates that such software must be installed and configured specifically for such an environment. To accomplish this, the software must be available, but not necessarily concurrently, on both machines. This availability requirement is met when the software is installed on disk space shared between the two machines in the HACMP clustered configuration. Due to the fact that some software cannot be completely installed on shared space, special installation or configuration is required to enable such software to process the same data stream on either machine. DB2, WebSphere MQ, and WebSphere Message Broker, for example, cannot be completely installed on shared space. They use file systems defined in the system (rootvg) volume group internal to each machine.

Often an HACMP cluster acts as a server that services clients located outside of the cluster. Any message originating from a client, outside of the HACMP cluster, is sent to a specific machine in the cluster, but the message must be processed even if it was destined to an HACMP machine that immediately fails when the message arrives. In this scenario, HACMP software will fail over all software components necessary from the failed machine to the healthy machine so that the message can be processed. All provisions must be made for a successful failover and high-availability. This includes multiple copies of data on separate physical disks, multiple network adapters, and multiple software components. If a disk fails, another disk with the same needed data must be available. If a network adapter fails, another one must be available to reach the machine in the HACMP cluster. If a software component fails, processing will still continue on the other machine if failover is necessary.

HACMP groups all critical components of a service that must be available for business data processing in what is termed a resource group. If any component in the resource group fails, HACMP will fail over the entire resource group from the failing machine to the healthy machine, if necessary, to keep the service available.

WebSphere Message Broker high availability

Figure 8-30 shows a possible configuration for a simple HACMP cluster of two machines.
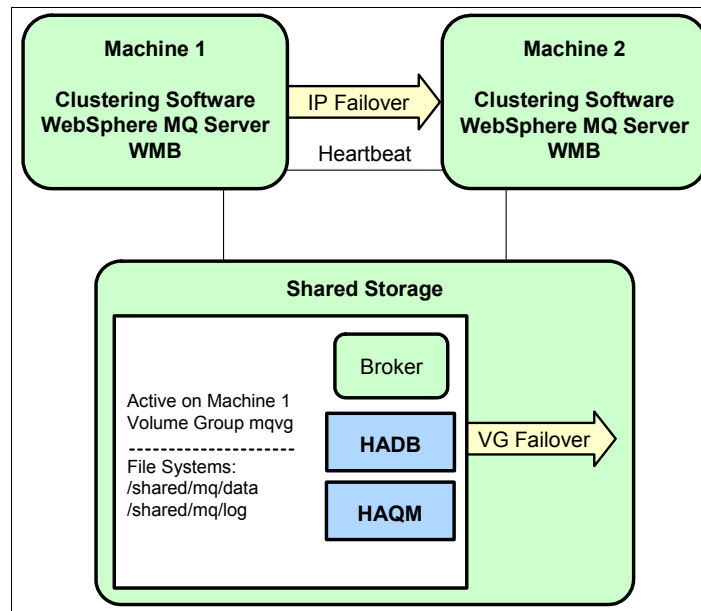
Figure 8-30   A simple active/standby HA configuration of WebSphere Message Broker

Machine 1 and machine 2 have the same software installed — HACMP software WebSphere MQ, DB2, and WebSphere Message Broker. In this HACMP configuration, if a client uses an IP address to reach a service such as an HTTPInput node in a message flow provided by machine 1 and a failover-inducing event occurs on machine 1, HACMP software will migrate the IP address to machine 2 so that the client will still be able to use the service that will now be provided by machine 2. The client must take into consideration this failure scenario so that the client will re-establish connection because the failure is not completely transparent.

Note the heartbeat line connecting machine 1 and machine 2. This heartbeat line is used by HACMP software to send HACMP packets between machine 1 and machine 2 to check on the health of the machines.

Figure 8-30 also shows a volume group created on shared storage. Note that machine 1 and the volume group mqvg both have the same color. This indicates that the volume group mqvg is being accessed exclusively by machine 1. All critical data that must be available, whether the service is provided by machine 1 or machine 2, is put in volume group mqvg. When a failover-inducing event

occurs on machine 1, HACMP software deactivates volume group mqvg on machine 1 and activates the volume group mqvg on machine 2. Note that the shared volume group mqvg has two file systems — one for the logs and one for the broker data. The logs file system contains queue manager logs, broker logs, and database logs. The broker data file system contains the queue manager data such as queues, the database table spaces, and the broker files. HADB is the broker database and HAQM is the broker queue manager. The queue manager, the database, and the broker can run on either machine, but only on one machine at a time in this configuration.

Figure 8-31 shows a topology where both machines are active. Each machine is running two brokers whose database is located remotely, preferably in another highly available cluster.
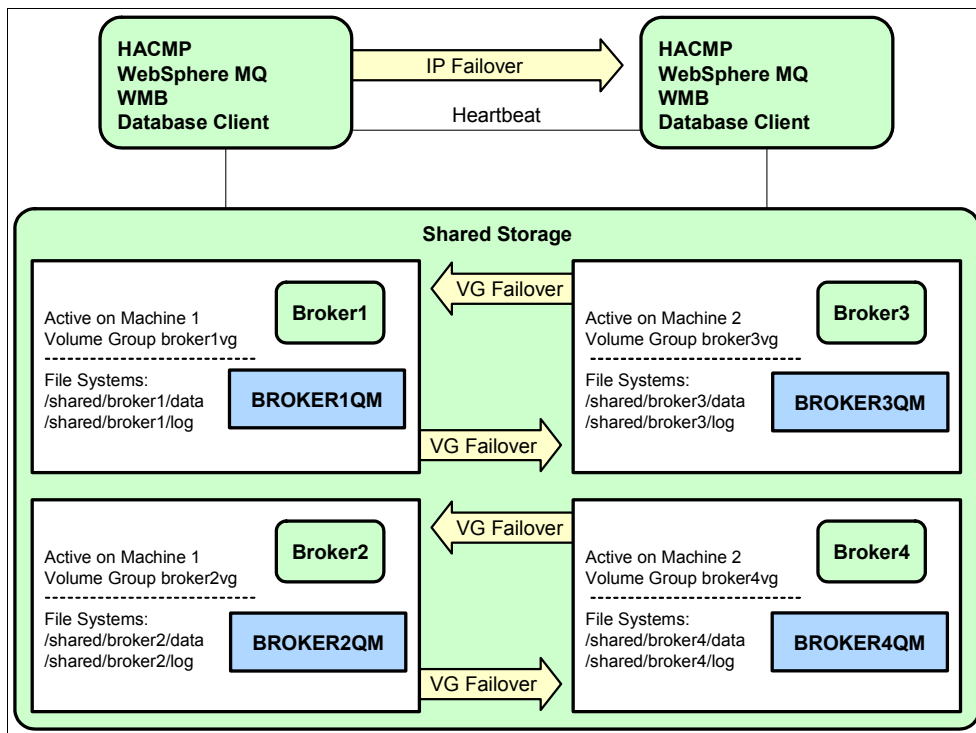
Figure 8-31   A more robust active/active HA configuration of WebSphere Message Broker

Redundancy

WebSphere Message Broker application availability is increased using the following means:

► Message flows redundancy
► Execution groups redundancy
► Local broker redundancy
► Clustered broker redundancy
► Broker queue manager cluster

Message flows and execution group redundancy

Figure 8-31 on page 230 shows two brokers on each machine in the HA cluster. So, for a two-machine HA cluster, there are four identical brokers. Each of these brokers has the following characteristics:

► It has two execution groups for a given message flow. More execution groups can be created if necessary.

► Each execution group has at least two instances of a given message flow.

So there are at least four instances of a given message flow — two instances in each execution group. This creates an execution group and a message flow redundancy. So, if one execution group goes down, the other execution group is up managing different instances of the same message flow that stopped on the failed execution group.

Local broker redundancy

Each machine in the cluster has two identical brokers running. So, if one broker goes down, the other broker on the same machine is still available to process messages.

Clustered broker redundancy

The two-machine HA cluster has four identical brokers. So, if one machine goes down, the two brokers on the other machine are still available to process messages.

Broker queue manager cluster

A well designed queue manager cluster increases availability and throughput by using clustered queues. If twenty MQ messages are sent by clients to a clustered queue defined on all broker queue managers in the cluster, five messages are sent to each queue manager in the cluster and hence five messages are processed by each broker. If one broker goes down, the twenty messages will be divided equally among the three remaining brokers.

Message sequence considerations

If a specific sequence of MQ messages must be processed by a given message flow, the high availability is decreased because if the execution group containing the desired message flow goes down, one of the following scenarios has to take place before the messages are processed:

► The execution group is restarted automatically by the broker.

► If the execution group went down because its broker went down, then the broker must be restarted automatically by the HACMP software on the same machine before the execution group is restarted.

► If the execution group went down because the whole machine went down, then the broker must be restarted automatically by the HACMP software on the other machine before the execution group is restarted.

The first scenario has the least downtime, while the last scenario has the most downtime.

Broker Web service communication

The broker uses the biphttplistener process to receive HTTP requests. The biphttplistener forwards HTTP requests to an HTTPInput node in a message flow, as shown in Figure 8-32.
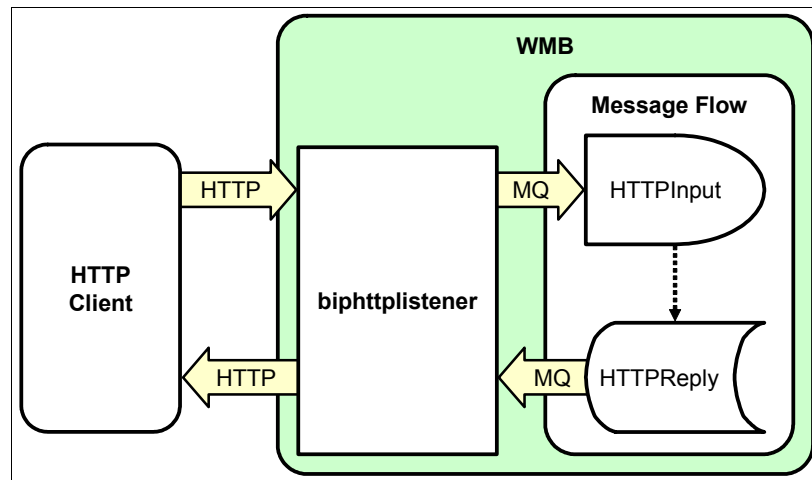
Figure 8-32   A simple way of receiving HTTP requests in a broker

The HTTP listener uses the MQ transport to communicate with the broker component. The HTTP listener puts the HTTP request in the SYSTEM.BROKER.WS.INPUT queue and receives the HTTP reply from the SYSTEM.BROKER.WS.REPLY queue. The listener uses the

SYSTEM.BROKER.WS.ACK queue to send acknowledgement messages to the HTTP flows when replies are sent to the clients. The HTTP request is processed through the message flow and a reply is sent back by the HTTPReply node to the HTTP client through the biphttplistener process.

The biphttplistener is a multi-threaded resilient process configured to accept many concurrent HTTP calls. If the process stops, it will automatically be restarted. Every broker has its own listener. In Figure 8-31 on page 230, there are four brokers running in the HACMP cluster, each with its own biphttplistener process. If any one broker fails, it can be failed over to the other machine, and the biphttplistener process is available again.

The number of HTTP requests that can be sent through the same HTTP connection and the number of concurrent HTTP connections are configurable.

► The biphttplistener process configured for fewer concurrent HTTP connections with a large number of HTTP requests enabled for each HTTP connection is ideal for a small number of client applications that require large message throughput. A large message throughput is possible because the applications do not have to establish an HTTP connection often. For example, if 100 HTTP requests can be sent through the same HTTP connection, the application can send 100 HTTP requests before it has to open another HTTP connection.

► The biphttplistener process configured for large concurrent HTTP connections and a small number of HTTP requests enabled for each HTTP connection is ideal for a large number of client applications.

The biphttplistener process is not very scalable. It can handle about 1200 concurrent HTTP requests. To handle a larger number of concurrent HTTP requests, WebSphere Message Broker can use a number of different mechanisms.

Running the same message flow in different brokers

If every message flow that makes use of an HTTPInput node runs in more than one broker, the number of concurrent HTTP requests to that message flow is increased. This configuration can be done by using a load balancer between the client applications and the biphttplistener processes of the brokers, as shown in Figure 8-33.
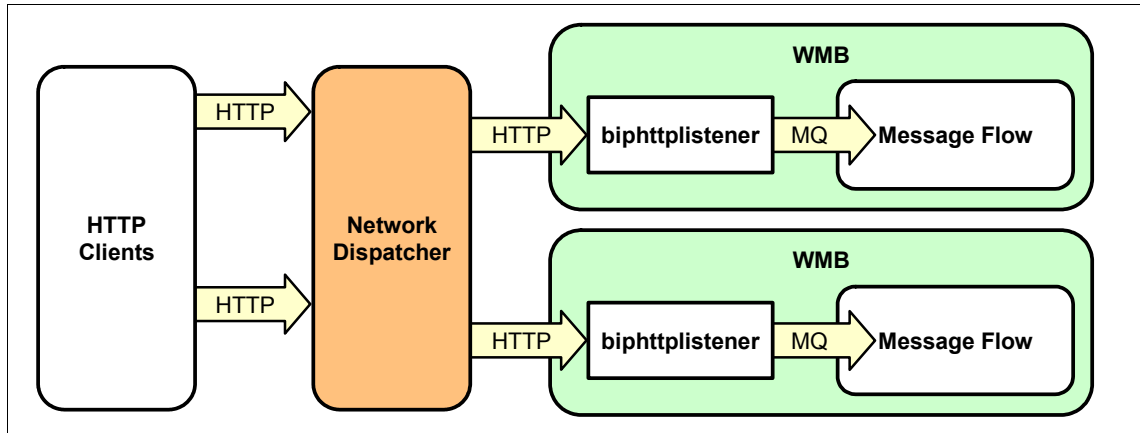
Figure 8-33   Increasing HTTP throughput through a load balancer

Using proxy servlet from SupportPac IE01

The proxy servlet runs inside a Web container like the one running in WebSphere Application Server. The proxy servlet can be deployed remotely. It communicates with the broker using broker internal MQ queues. The proxy servlet can forward HTTP requests only to one broker, but more than one proxy servlet can be configured to forward HTTP requests to one broker. Using this mechanism off loads the HTTP requests receipt to the proxy servlet instead of the biphttplistener process. This mechanism is illustrated in Figure 8-34.
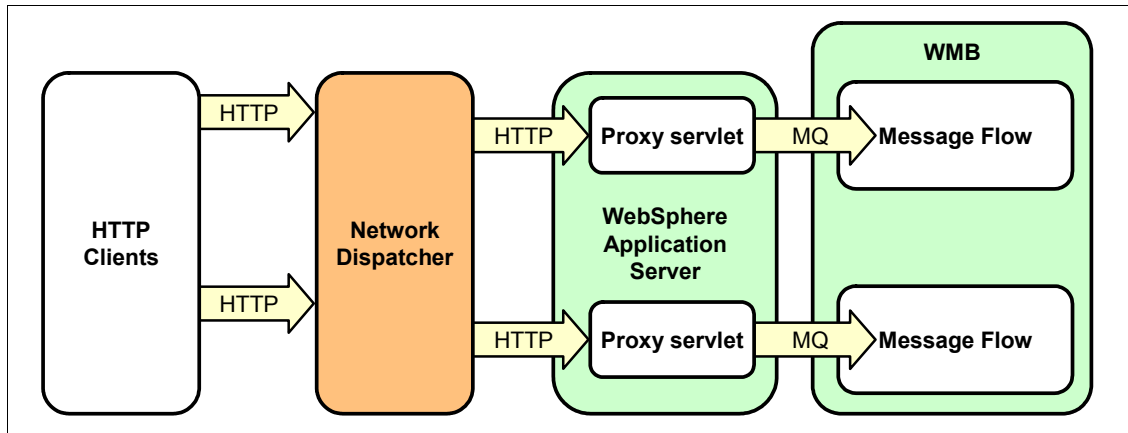
Figure 8-34   Increasing HTTP throughput through the proxy servlet

8.2.10 Qualities of service

WebSphere Message Broker leverages the underlying qualities of service provided by WebSphere MQ. This includes delivery, persistence level, priority level, and transaction management qualities of service. WebSphere MQ guarantees that the message is delivered to its destination once without duplication.

WebSphere MQ provides two levels of persistence. The message is either persistent or non-persistent. Where the message does not need to be persistent, non-persistent messages should be used because they are best for performance. The persistence level can be specified on the message queue as well as the message MQMD header in the Persistence field.

Priority level can be set in the message MQMD header in the Priority field and should be used in certain situations where messages from specific applications or clients, for example, should be processed first. The priority setting in the MQMD header overrides the default WebSphere MQ behavior of First-In-First-Out (FIFO).

If all processing steps within a message flow need to be treated together in one transaction, WebSphere Message Broker supports that when the developer activates the coordinated property on the message flow. This type of message flow is called a coordinated message flow. This type of a message flow is needed when the data integrity is critical and updates to more than one recoverable resource such as DB2 and MQ are necessary to complete the flow. Some input nodes have a property called Transaction mode. When this property is set to automatic, the message is part of the global transaction of the message flow and the message flow is marked transactional if the input message is persistent. But if the message is not persistent, the message flow is marked uncoordinated. All other nodes whose Transaction mode property is set to automatic are included in the global transaction if the input node set the message flow to transactional.

When integrating WebSphere Message Broker with WebSphere Transformation Extender via the WebSphere Transformation Extender node, the WebSphere Transformation Extender input card used in the message flow is always the card #1. This is irrelevant if the WebSphere Transformation Extender map has only one input card. The output card number is configurable on the WebSphere Transformation Extender node. The WebSphere Transformation Extender map input and output cards used in the message flow are included in the global transaction of the message flow. When the WebSphere Transformation Extender map has more than one input and more than one output card, the input and output cards that are not used in the message flow are not included in the global transaction of the message flow. So if the WebSphere Transformation Extender map updates a database using a card that is not used in the message flow and the message flow fails, the update to the database made by the WebSphere Transformation Extender map is not rolled back.

To include all WebSphere Transformation Extender map input and output cards in the global transaction, the WebSphere Transformation Extender API should be used. Using WebSphere Transformation Extender API, WebSphere Transformation Extender maps can be executed and WebSphere Transformation Extender input, output cards, adapters, and properties can be overridden. The WebSphere Transformation Extender Java API can be invoked from a JavaCompute node.

The developer can write a Java application to communicate with the configuration manager component of WebSphere Message Broker to control certain qualities of service in the broker. The Java application uses the configuration manager proxy (CMP) API to interact with the configuration manager. Using the CMP API, the user application can deploy bar files and publish/subscribe topologies, topic trees, and broker configurations. The user application can modify the publish/subscribe topology; and add and delete brokers, brokers connections, and collectives. The user application can create, modify, and delete execution groups.

8.2.11 WebSphere ESB-WebSphere Message Broker inter-communication

Figure 8-35 shows the communication protocols supported between WebSphere Message Broker and WebSphere ESB 6.0.2.
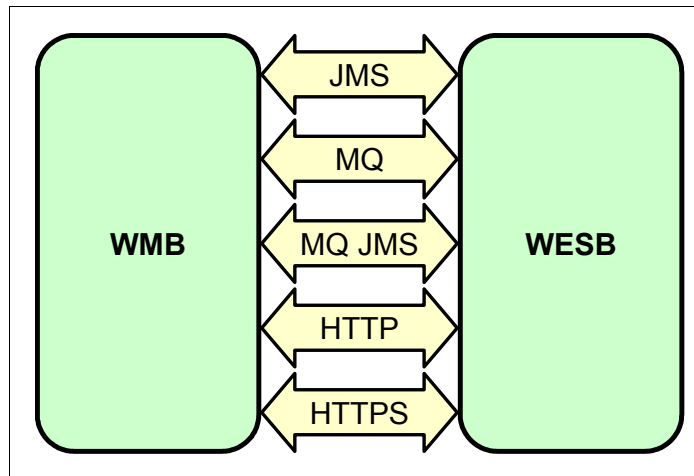
Figure 8-35   Transport protocols supported between WebSphere Message Broker and WebSphere ESB

As can be seen, both synchronous and asynchronous communication protocols are supported.

► HTTP Communication

   This transport protocol is used to carry SOAP messages. An import SCA component with Web service binding in a WebSphere ESB mediation module can invoke a WebSphere Message Broker-exposed Web service. WebSphere Message Broker can accept SOAP calls destined to one of the HTTPInput nodes in the various message flows running in the broker. The HTTPRequest node running in any message flow in the broker can send a SOAP call over HTTP to a Web service exposed in WebSphere ESB as an SCA export component with Web service binding.

► MQ Communication

   WebSphere ESB 6.0.2 supports the MQ binding on the export and import SCA components. Using an import SCA component with MQ binding, a WebSphere ESB mediation module can send an MQ message to WebSphere Message Broker. An MQInput node of a message flow running in WebSphere Message Broker can receive this MQ message coming from WebSphere ESB. WebSphere Message Broker uses an MQOutput node in a message

flow to send an MQ message to WebSphere ESB. An export SCA component with MQ binding receives the MQ message sent by WebSphere Message Broker. The MQ message payload format supported between WebSphere ESB and WebSphere Message Broker is XML.

► JMS Communication

WebSphere Message Broker may use the JMSInput node to receive a JMS message from, and the JMSOutput node to send a JMS message to, the WebSphere Application Server JMS provider, which is used by WebSphere ESB. In this scenario, WebSphere Message Broker acts as a JMS client to the WebSphere Application Server JMS provider. WebSphere ESB uses the import SCA component with JMS binding to send JMS messages to WebSphere Message Broker, which receives them with the JMSInput node. WebSphere ESB uses the export SCA component with JMS binding to receive JMS messages put by the JMSOutput node of a WebSphere Message Broker message flow. WebSphere Message Broker and WebSphere ESB 6.0.2 support all 6 JMS message types: Message, TextMessage, BytesMessage, ObjectMessage, StreamMessage, and MapMessage.

► MQ JMS Communication

WebSphere ESB 6.0.2 may use an SCA import with MQ JMS binding to send a JMS message to WebSphere Message Broker via WebSphere MQ using the JMS API provided with WebSphere MQ. WebSphere Message Broker receives the JMS message from WebSphere MQ using the JMSInput node. WebSphere Message Broker uses the JMSOutput node to send a JMS message to WebSphere ESB via WebSphere MQ. WebSphere ESB receives the JMS message from WebSphere MQ using an export SCA component with MQ JMS binding.

8.2.12 WebSphere Message Broker-WebSphere ESB HTTP secure communication

The subject of security is out of the scope of this book. But, the following procedure describes one of the simplest ways to enable HTTPS communication between WebSphere Message Broker and WebSphere ESB.

Configuring WebSphere Message Broker for HTTPS

To configure WebSphere Message Broker to receive HTTPS requests, perform the following steps:

1. Use the `keytool` command that comes with Java Runtime Environment installed with WebSphere Message Broker to create a self-signed certificate or to import a CA-signed certificate. The following command creates a new

keystore file and a self-signed certificate. The command will prompt you for certain details that are used to create the self-signed certificate. Replace *password* with the password that you want to access the keystore file, *path* with the path name to the keystore file, and *alias* with a string to identify the self-signed certificate. Export the corresponding public self-signed certificate to all the WebSphere Message Broker clients to enable them to send HTTPS requests to WebSphere Message Broker. If WebSphere Message Broker is using a CA-signed certificate, you may not have to export the corresponding public CA-signed certificate because most of public certificate stores should already have the most popular public CA-signed certificates.

```
keytool -genkey -keypass password -keystore path -alias alias
```

2. Use the `mqsichangeproperties` command to configure the biphttplistener process for SSL. The following commands will do that. In the commands below, replace *broker* with the actual name of the broker, *pathname* with the full path name to the keystore file created above, *password* with the password used to access the keystore file, and *port* with the port number used for the SSL traffic.

```
mqsichangeproperties broker -b httplistener -o HTTPListener -n
enableSSLConnector -v true

mqsichangeproperties broker -b httplistener -o HTTPSConnector -n
keystoreFile -v pathname

mqsichangeproperties broker -b httplistener -o HTTPSConnector -n
keystorePass -v password

mqsichangeproperties broker -b httplistener -o HTTPSConnector -n
port -v port
```

3. Select the **Use HTTPS** check box on the HTTPInput node of the message flow used to receive and process HTTPS requests.

To configure WebSphere Message Broker to send HTTPS requests, use the **keytool** command that comes with the Java Runtime Environment installed with WebSphere Message Broker to import the HTTP server public self-signed certificate into the cacerts file. The HTTP server refers to the server to which WebSphere Message Broker sends an HTTPS request. This could be IBM HTTP Server used in the WebSphere Application Server environment. The file cacerts is located in the security directory of the JRE installation directory. The following command imports the HTTP server certificate. Replace *alias* with a string to identify the certificate to import and *cert* with the path name of the certificate file.

The string *changeit* is the default password of the cacerts file. If the HTTP server is using a CA-signed certificate. This step may not be needed because cacerts should already have the most popular public CA-signed certificates.

```
keytool -import -alias alias -file cert -keystore cacerts -keypass
changeit
```

Configuring WebSphere ESB for HTTPS

To configure WebSphere ESB to receive HTTPS requests, perform the following steps. Note that these steps do not secure the communication between the IBM HTTP Server plug-in and the WebSphere Application Server Web container. This configuration only secures the communication between the client, like WebSphere Message Broker and IBM HTTP Server.

1. Use the `ikeyman` tool to create the key database file for IBM HTTP Server. The `ikeyman` tool is used to manage keys and certificates. The tool can be used to create and export self-signed certificates, create CA certificate requests, import CA-signed certificates, and so on.

2. Add the following lines to the httpd.conf file found in the conf directory of IBM HTTP Server. The SSLEnable directive enables SSL globally. The KeyFile directive specifies where the key database file is located for IBM HTTP Server. Replace *pathname* with the path name of the key database file. The directives SSLEnable and KeyFile can be specified for a specific virtual host only. In such a case, both directives will be nested inside a pair of virtual host entries:

```
SSLEnable
```

```
KeyFile pathname
```

```
LoadModule ibm_ssl_module modules/mod_ibm_ssl.so
```

3. If a self-signed certificate is used for IBM HTTP Server, export the corresponding public self-signed certificate to all HTTPS clients like WebSphere Message Broker to enable them to send HTTPS requests to IBM HTTP Server. If IBM HTTP Server is using a CA-signed certificate, this step may not be needed because most certificate stores already contain the most popular public CA-signed certificates.

To configure WebSphere ESB to send HTTPS requests, use the `ikeyman` tool that comes with WebSphere Application Server to import the WebSphere Message Broker public self-signed certificate into the DummyClientTrustFile.jks file. This file is located in the etc directory of the WebSphere Application Server installation directory. If WebSphere Message Broker is using a CA-signed certificate, this step may not be needed because the DummyClientTrustFile.jks file should already have the most popular public CA-signed certificates.

Part 3

Physical scenarios

In this part we look at a couple of examples of how to use the products within an enterprise service bus. This part contains the following chapters:

► Chapter 9, "Scenario: using WebSphere ESB and WebSphere Message Broker in combination" on page 243, works through the how to set up connectivity using standard SOA protocols between WebSphere Enterprise Service Bus and WebSphere Message Broker.

► Chapter 10, "Scenario: DataPower in an SOA" on page 323, shows how DataPower appliances are configured when used as an ESB Gateway.

Although not shown in this part of the book, we highly recommend that you review the scenarios on how WebSphere Service Registry and Repository can be used to augment an enterprise service bus hub. See *WebSphere Service Registry and RepositoryWebSphere Service Registry and Repository Handbook*, SG24-7386.

9

Scenario: using WebSphere ESB and WebSphere Message Broker in combination

The sample business scenario used in this chapter illustrates how to connect two separate ESBs, a WebSphere Enterprise Service Bus, and a WebSphere Message Broker into a single ESB. It focuses on the interaction between a retail system residing on a WebSphere Enterprise Service Bus and a warehouse system residing on WebSphere Message Broker.

This scenario demonstrates several new features introduced in WebSphere Message Broker V6, including the use of HTTPS for an additional level of privacy and security, SOAP over HTTP communication, and the use of the new MQGET node within the message flow to interact with the existing back end manufacturer.

In this chapter the following points are discussed:

► Design guidelines and business needs addressed by the sample scenario and the selection of the relevant ESB integration patterns

► Runtime guidelines to create and integrate the two systems including the building of the bar files and deployment steps

243

9.1 Design guidelines

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

To simplify the implementation, we import several project interchange files, consisting of the message flows, message sets, and the XML Schema Definitions (XSDs). We also provide instructions for importing on the project interchange, as well as the building of the execution groups, bar files, and the deployment steps.

To simulate the front end retail system, we utilize the IH03 - WBI Message Broker V6 - Message display, test, and performance utilities and the Test Component feature in WebSphere Integration Developer tool. The IH03 supportpac is available from the WebSphere MQ Web site. We also used the MA01 - WebSphere MQ - Q Program in simulating the back-end manufacturing application.

9.1.1 Business scenario

The business scenario implemented is a supply chain management process that is split across two organizations, as seen in Figure 9-1. This scenario focuses on the interaction between the retail system in division A and the warehouse system in division B business scenario.
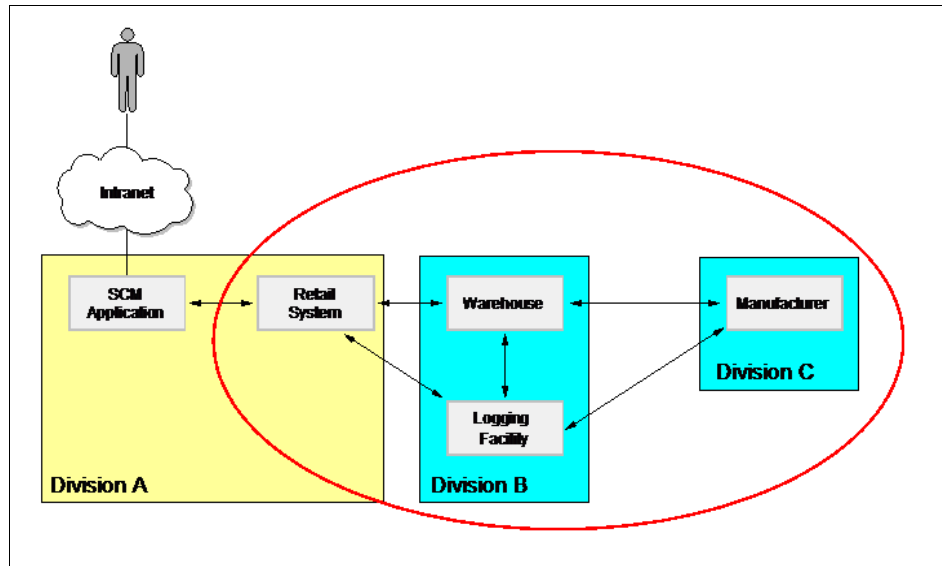
Figure 9-1   High-level business context showing the existing infrastructure

High-level business overview context

This is a supply chain management scenario in which customers access an electronics 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. Each system resides in a different division of the organization from the other components in the supply chain management scenario.

Organizational overview

As a result of growth and acquisitions, the new company now spans three separate divisions within the organization, each containing its own functional components:

► Division A

– Internet based e-commerce systems (SCM application)
– Retail system

► Division B

Warehouse system

► Division C

Manufacturer system

The challenge was to integrate these entities without any major redesign on the existing applications.

Integration of divisions

In Figure 9-2 division A uses WebSphere ESB for its ESB implementation. The primary transport used is SOAP over HTTP, though it also uses JMS over MQ.
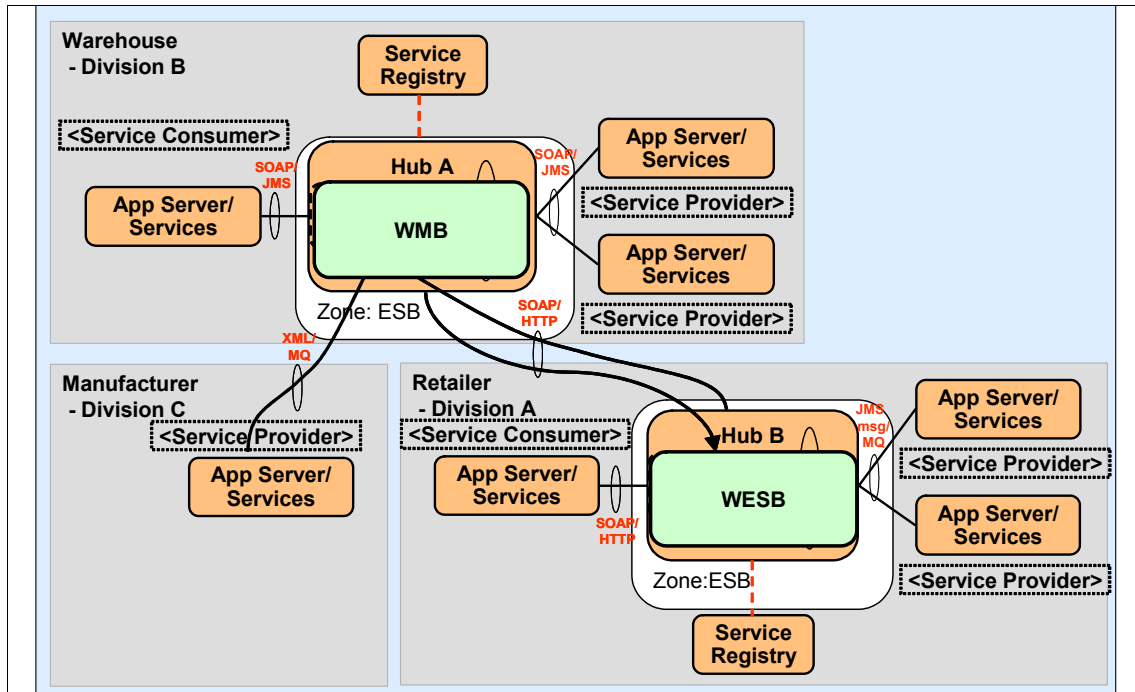
Figure 9-2   Supply chain zone

Division B uses WebSphere Message Broker. The primary transport used is SOAP over JMS with WebSphere MQ as the JMS provider.

To integrate division A with division B, the two ESB implementations will be connected using SOAP over HTTP.

The manufacturer system in division C responds to fulfillment orders using XML messages. It is linked to external buyers through WebSphere Message Broker using WebSphere MQ as the transport.

This configuration matches the Brokered ESB Variation 1 described in Figure 4-8 on page 77.

General transaction flow

The retail system requests products from the warehouse through WebSphere ESB using a PurchaseOrder request message. If the request causes the remaining stock in the warehouse to fall below a minimum threshold, the scenario requires the warehouse to re-order stock from the manufacturer.

Our focus is on the interaction between two ESB technologies used by the retail system utilizing WebSphere ESB and the warehouse system utilizing WebSphere Message Broker. The purchase order transaction can be sent from the retail system to the warehouse system over the two application systems using two possible communication methods. Prior to WebSphere Message Broker V6, the customer was utilizing SOAP over HTTP. With the implementation of WebSphere Message Broker V6, the customer can implement SOAP over HTTPS to further secure the communications.

Using JMS over MQ

The first shown in Figure 9-3 is the interaction between the retail system and the warehouse system through JMS over MQ.
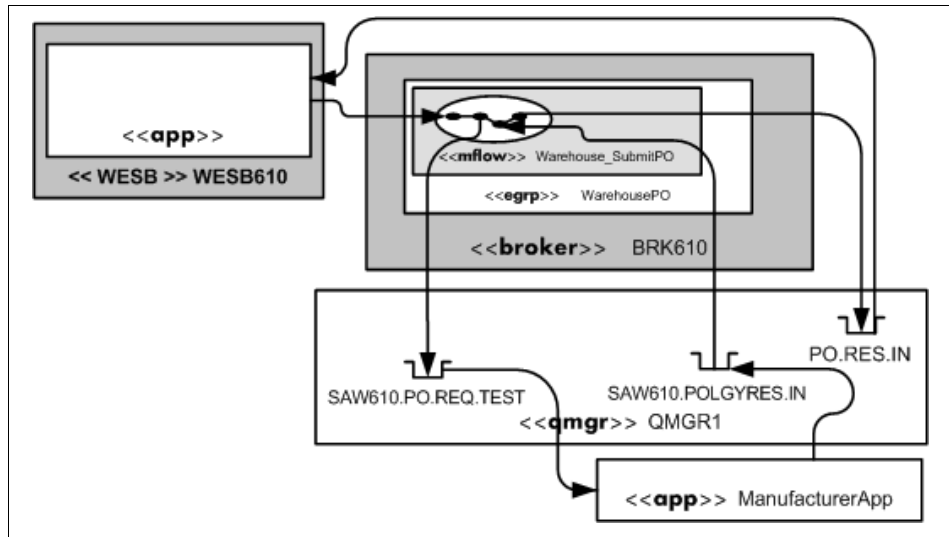
Figure 9-3   JMS/MQ warehouse PO submit

The message is sent to the warehouse system through a MQ Put to the PO.REQ.IN queue. The warehouse system retrieves the messages, and parses and stores the information. The warehouse system puts a notification message on the SAW610.POREQ.IN queue where the existing back-end application retrieves the message and sends a response back to the warehouse system on SAW610.POREQ.OUT. The warehouse system resumes processing and notifies the retailer that the purchase order has been processed.

The message flow diagram in Figure 9-3 shows the processing path from the requestor into the warehouse. The application sends the requests message to the queue manager in step 1 and issues a mqget with wait on the PO.RES.IN queue. The broker then consumes the message in step 2. During the message process, the broker sends a MQ Request message to the manufacturer application in step 3. The causes the manufacturer process to be triggered (step 4) and a response sent back in step 5. The broker then resumes processing of the request in step 6, picking up the response from the manufacturer and formatting the response for the request and inserting the response on the PO.RES.IN queue step 7, releasing the wait in step 8.

Using SOAP over HTTP

The second method of communication with the warehouse system is using SOAP over HTTP. The retail system sends the warehouse system a purchase order transaction. In this method the retail system and the warehouse system are directly communicating.

The requesting application sends a SOAP message to the broker over HTTP or HTTPS in step 1. During the message process, the broker sends a MQ Request message to the manufacturer application in step 2. The causes the manufacturer process to be triggered (step 3) and a response sent back in step 4. The broker then resumes processing of the request in step 5, picking up the response from the manufacturer and formatting the response for the request and inserting the response on the PO.RES.IN queue in step 6, releasing the wait in step 7.
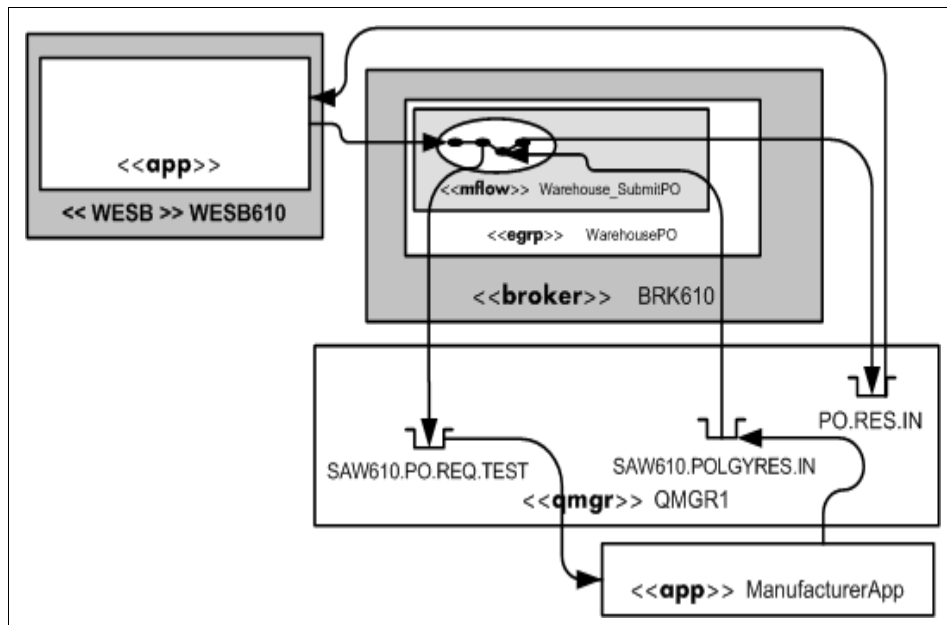
Figure 9-4   HTTP warehouse PO submit

This scenario is represented by Figure 9-1 on page 245, which uses an instantiation of the Brokered ESBs Variation 1 pattern described in Figure 4-8 on page 77.

9.2 WebSphere Message Broker

> **Note:** The message flows discussed in this section are included with the samples shipped with this book. For information about how to download the samples, see Appendix C, "Additional material" on page 377. For information about how to import these message flows into the Message Brokers Toolkit, see Appendix B, "Sample instructions" on page 371.

This section describes the WebSphere Message Broker ESB and its associated service consumers and providers.

► Messages exchanged between the two ESBs use a SOAP over HTTP or XML over JMS/MQ.

► Messages sent between WebSphere Message Broker and the back-end existing manufacturer's applications use native WebSphere MQ messages (MQ message body data is not SOAP, and the application does not use the WebSphere MQ JMS API).

Message flows in WebSphere Message Broker mediate messages from the retail system (via WebSphere ESB) for use by the manufacturer service providers. The message flows are named:

► Warehouse_SubmitPO
► Warehouse_SubmitPO_SubFlow_1
► WarehouseLogEvent
► Warehouse_SubmitSN

In order to mediate messages, WebSphere 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 Message Broker may select this parser using a message domain. Available message domains include Binary Large Object (BLOB), Message Repository Manager (MRM), XML, and XML NameSpace (XMLNS). Of these alternatives, the MRM domain is the only method that implements a parser to check on the wire messages against a predefined 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.)

To simplify the implementation, we import the project interchange, consisting of the message flows, message sets, and XML Schema Definitions (XSDs). Instructions for importing on the project interchange as well as the building of the execution groups, bar files, and the deployment steps are described in 9.1, "Design guidelines" on page 244.

To simulate the front-end retail system, we utilize the IH03 - WBI Message Broker V6 - Message display, test, and performance utilities and the Test Component feature in WebSphere Integration Developer tool. The IH03 supportpac is available from the WebSphere MQ Web site. We also used the MA01 - WebSphere MQ - Q Program in simulating the back end manufacturing application.

9.2.1 Message flow descriptions

The following section describes the message flows and their functions within the ESB. In this section of the business scenario we interface the purchase orders from the retail system into the warehouse system and the manufacturer applications (see Figure 9-5).
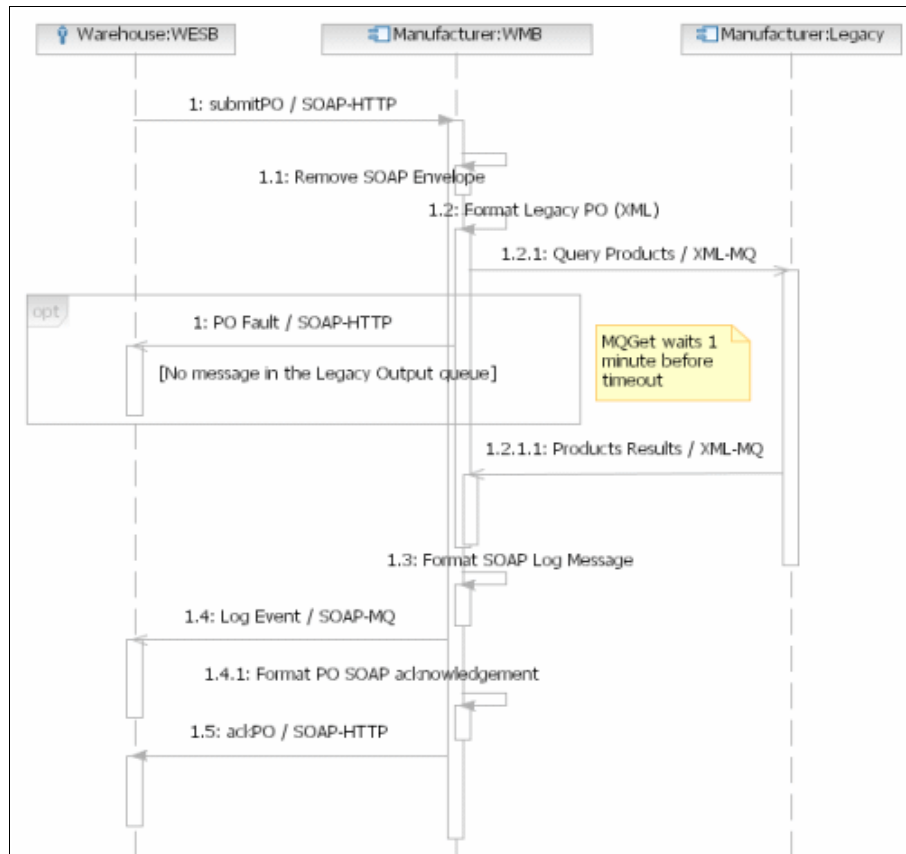
Figure 9-5   Purchase order transaction flow

The warehouse makes requests to the manufacturer using SOAP over HTTP. The manufacturer receives that message and removes the SOAP envelope from the XML content, preparing the message to be delivered to the manufacturer application.

The application receives the message and returns the Purchase Order status requested. If the purchase order (PO) request is not returned in one minute after the manufacturer delivered the request, the flow formats and sends a SOAP fault message to the warehouse saying that a error occurred.

The manufacturer receives the message from the system, logs the request in the warehouse log application (SOAP over MQ), and then returns the PO acknowledgement message to the warehouse (SOAP over HTTP), describing the status of the request.

Warehouse_SubmitPO message flow

The Warehouse_SubmitPO message flow routes messages from WebSphere ESB to the manufacturer system. Its primary purpose is to translate SOAP over HTTP into the XML format required by the back-end application.

The logic in the Warehouse_SubmitPO_Http message flow is broken down into the following functional nodes:

- ▶ JavaCompute node: TransformMessage
- ▶ JavaCompute node: FormatMessage
- ▶ JavaCompute node: FaultHandler
- ▶ JavaCompute node: FormatReply

Note: The complete JavaCompute node flow implementation can be found in Appendix A, "Java node source code" on page 353.

This message flow calls two subflows:

- ► Legacy_SubFlow
- ► logEvent_SubFlow
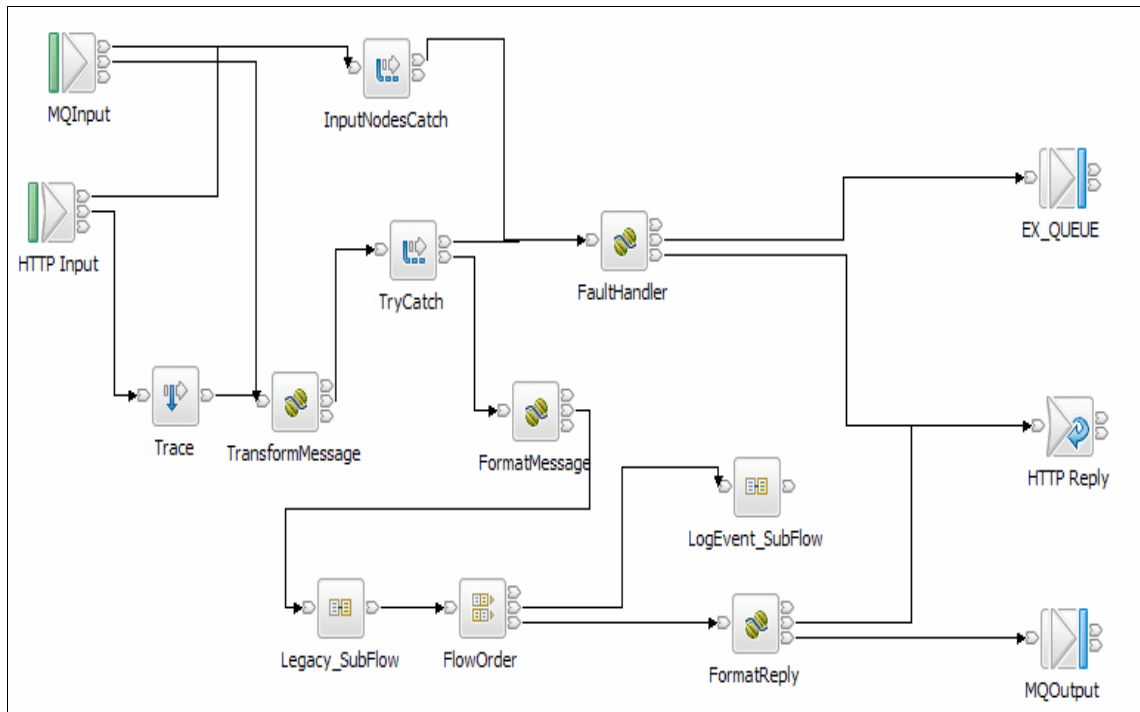
Figure 9-6   Warehouse PurchaseOrderRequest message flow

TransformMessage JavaCompute node

The TransformMessage JavaCompute node removes the SOAP message parts and saves the necessary SOAP arguments in the LocalEnvironment domain inside the broker to be used later on the flow (Example 9-1).

Example 9-1   Saving the necessary SOAP message parts in LocalEnvironment structure

```
...
SOAPMessageHelper.resetSoapHeader(envRoot); // Reset SOAP
SOAPMessageHelper.initService(envRoot,
                             "Intermediary",
                             "www.itso.ral.ibm.com",
                             "PurchaseOrderRequest");
SOAPMessageHelper.setValidHeaders(envRoot, "PurchaseOrder");
SOAPMessageHelper.setValidHeaders(envRoot, "Configuration");
SOAPMessageHelper.setValidHeaders(envRoot, "StartHeader");
...
```

This node also has the dynamic routing pattern implemented. This means that every message that comes to the manufacturer purchase order process is routed depending on its structure and implementation, MQ messages, HTTP messages, or JMS messages.

> **Note:** All Compute nodes use a utilitary class called SOAPMessageHelper, responsible for executing all operational functions defined in the flows. These functions are not just operational, but structural functions designed to realize all service virtualization implemented inside the broker.

FormatMessage JavaCompute node

This node prepares the message in the format (plain XML) so it can then be transported over MQ to the application system.

FaultHandler JavaCompute node

Exceptions within the flows are caught by a TryCatch node defined at the beginning of the manufacturer flow. If an exception is caught during the execution of the flow, the TryCatch node sends the message to a JavaCompute node to be transformed into the correct SOAP fault message format.

For example, after the XML message is delivered to the application, the flow waits for one minute (with the MQGet node) for a response. If no messages are returned during this time, the node throws an exception that is caught by the TryCatch node and sent to the exception JavaCompute node to be formatted in right SOAP message fault standards (Example 9-2).

Example 9-2   Creates the SOAP fault message to be delivered to the message requester

```
...
String err = new String(msgEx.getBuffer());
String msgErr =
    "An genereric error ocorred while processing the message. Node"+
    (err!=null?err:"");
// Creates the Soap FAULT Codes //
SOAPMessageHelper.formatSoalFaultMessage(mrm, env, msgErr);
...
```

Legacy_SubFlow Message Flow

The Legacy_SubFlow node sends a request message to the Manufacturing system to validate available stock. The message is sent to the queue SAW610PO.REQ.TEST and retrieves a response message from queue SAW610.POLGYRES.IN. The MQGet node in the flow has a wait interval of one minute. In no message is received within the wait interval, a message is returned `"Couldn't get any message from the Queue" rc=3001`. If a response is received the flow continues.
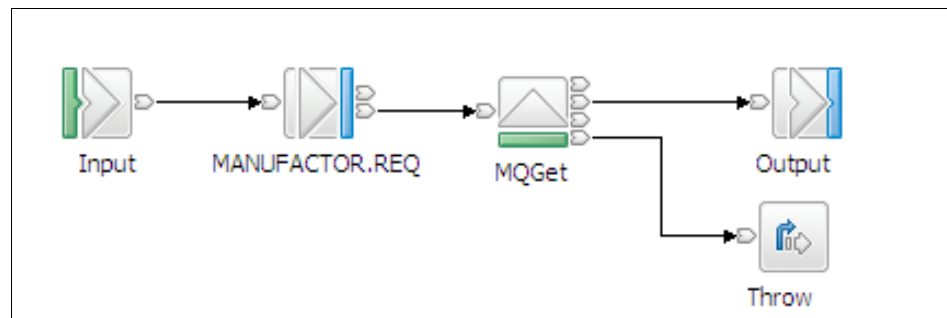
Figure 9-7   Legacy_SubFlow message flow

FormatReply JavaCompute node

The FormatReply JavaCompute node sends an acknowledge status notification message back to the requestor of the warehouse products. The message status is received by the purchase order process, and after sending the log status to the warehouse, this node then transforms the message into a SOAP envelope

response (over HTTP) using the data saved in the broker LocalEnvironment structure (Example 9-3).

Example 9-3   Creating a response to the warehouse request

```
...
SOAPMessageHelper.setOutOperation(
                inAssembly.getGlobalEnvironment().getRootElement(),
                "Response", "www.itso.ibm.com/ackPO");
// Creates the response SOAP Message //
SOAPMessageHelper.encodeAckPOSOAPMessage(inRoot, outRoot, env);
...
```

This node also implements the dynamic routing pattern so that the output message is delivered to the destination depending on its request format. If a message was requested using HTTP, a HTTPResponse is applied, and if a message was request using MQ, then a MQOutput is used.

LogEvent_SubFlow Message Flow

After the PO responds the product status, an inner flow is executed to log the request status to the warehouse. The CreateSOAP JavaCompute node is responsible to format the log message to the requester (Example 9-4 on page 258) and dynamically route the message to the right output, depending on the request format stated on the first part of the PO flow (Figure 9-8).
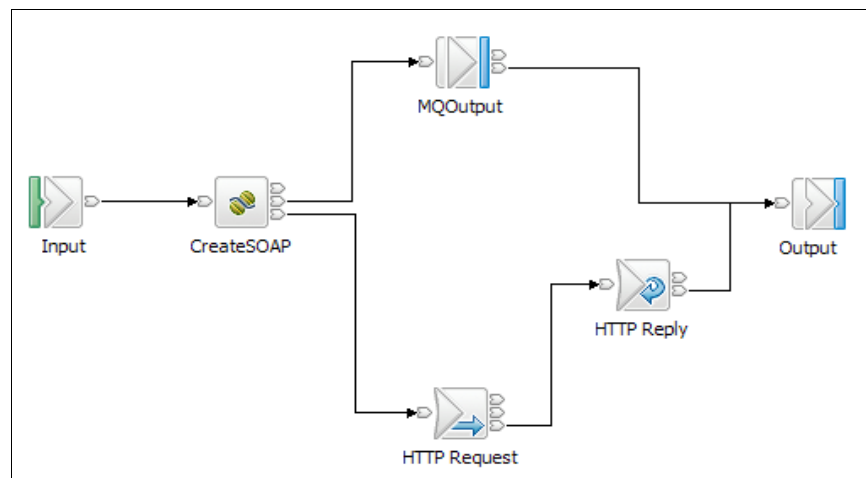
Figure 9-8   LogEvent_SubFlow message flow

> **Note:** In the case of an HTTP request, the log event is formatted as a SOAP envelope and transported to the warehouse over an HTTP request (HTTPRequest node). Otherwise the SOAP request is done using MQ (MQOutput node).

Example 9-4   Creating the log request to the warehouse system (WebSphere ESB)

```
...
SOAPMessageHelper.setOutOperation(
                inAssembly.getGlobalEnvironment().getRootElement(),
                "Response", "www.itso.ibm.com/Log");
// Takes references to the InputRoot and constructs
// an OutputRoot with a SOAP envelope defined
SOAPMessageHelper.encodeLogSOAPMessage(inRoot, outRoot,
                    inAssembly.getGlobalEnvironment().getRootElement());
...
```

WarehouseCallbackResponse message flow

The WarehouseCallbackResponse flow routes the messages to an existing back-end manufacturer application and then waits to receive a response. As stated before ("FaultHandler JavaCompute node" on page 255), if no message is returned by the purchase order process in one minute, the MQGet node throws an exception (Figure 9-9) and the flow is interrupted with an error message returned to the warehouse describing the origin of this error.
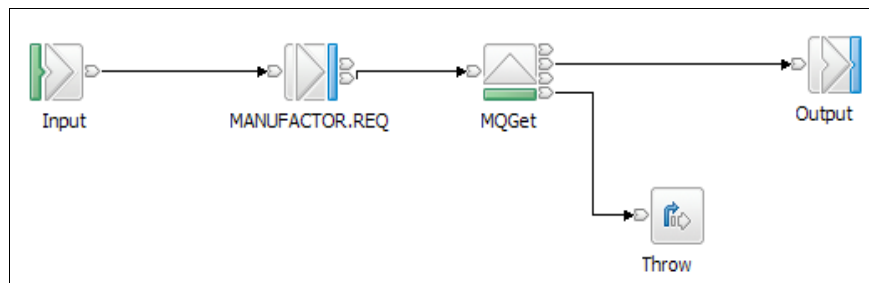
Figure 9-9   WarehouseCallbackResponse message flow

Warehouse_SubmitSN Message Flow

The warehouse submit shipping notice is a HTTP asynchronous request notifying the warehouse system of the ship date of the order. The SOAPFormat Java node wraps the XML message with the appropriate SOAP header/body information. The HTTP request is sent to the retail system. Once the retail system receives the ship notice, it is logged and a confirmation HTTP Reply is sent back to the manufacturer (Figure 9-10).
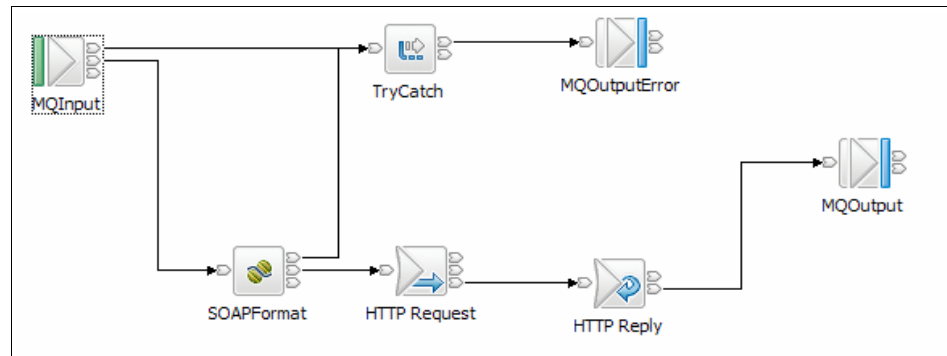
Figure 9-10   Warehouse_SubmitSN message flow

If an error occurs during the execution of the flow, the request message is automatically redirected to a MQ error queue so that the message can later be on processed by another back-end system, for example, a monitoring system.

9.2.2 Existing back-end manufacturer application

The manufacturer application is a stand-alone application. The application communicates with external systems as a WebSphere MQ messaging client. The main function of the application is to simulate an existing back-end application system and show the integration into a Web service scenario. The back-end application 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 warehouse client triggers the manufacturer's execution by sending a message into the input queue [SAW610.POREQ.IN]. The message is represented by ManufacturerSchema.xsd and is received by the MQ receiver component of the existing back-end application. The MQ receiver component receives incoming messages and transforms them into an internal object format. The request is processed and a response message is inserted on to a queue [SAW610.POREQ.OUT] for the warehouse system. The warehouse message flow consumes the message and resumes processing.

Detailed communication sequence

From a messaging point of view, the communication sequence in the warehouse application is as follows:

1. The warehouse application receives a message from the consumer or retailer 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 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 the 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.

9.3 Runtime guidelines for ESB based on WebSphere Message Broker

This section describes how to deploy the WebSphere Message Broker artifacts necessary to run this scenario. The five message flows that contribute to form the WebSphere Message Broker ESB could all be deployed to a single (for example, 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 carry out the following operations:

► Configure WebSphere MQ environment.
► Connect the toolkit to the configuration manager.
► Create execution groups.
► Create and deploy broker archive files.

9.3.1 Configure WebSphere MQ environment

You need to define several resources in WebSphere MQ for the default queue manager QMGR1, which we use as a single queue manager for supporting the configuration manager and runtime broker. In the download samples provided with this book we provide the mqsc control cards to define the queues, channels, process, and trigger service. Save the SAW610.QMGR1.tst to the workspace. Open the WebSphere MQ Explorer and define a new qmgr.

1. Create a default Queue Manager, QMGR1.

2. Create a new WebSphere MQ Listener, listening on port 1414 using the TCP protocol, then start this listener.

3. Create a WebSphere MQ ServerConn channel with the following attributes:

   – Channel name: SW610.CLNT
   – Transmission protocol: `TCP/IP`

4. Create the following WebSphere MQ objects by performing a runmqsc piping in the mqsc cards (Example 9-5).

- ☐ QLOCAL(ExceptionQueue)

- ☐ QLOCAL(LOG.IN)

- ☐ QLOCAL(SAW610.INITQ)

- ☐ QLOCAL(SAW610.POJMS.REQ.IN)

- ☐ QLOCAL(SAW610.POLGYRES.IN)

- ☐ QLOCAL(SAW610.POREQ.EXCEPT)

- ☐ QLOCAL(SAW610.POREQ.IN)

- ☐ QLOCALE(SAW610.POREQ.OUT)

- ☐ QLOCAL(SAW610.REQSN.IN)

- ☐ QLOCAL(SAW610.REQSN.OUT)

- ☐ QLOCAL(SAW610PO.REQ.TEST)
  INITQ(SAW610.INITQ) PROCESS(SAW610.RESPONSE) TRIGDATA( )
  TRIGTYPE(EVERY) TRIGGER

- ☐ PROCESS(SAW610.RESPONSE)
  APPLTYPE(WINDOWSNT) APPLICID(start "C:\Documents and
  Settings\Administrator\IBM\wmbt6.0\workspace\BackEnd.CMD")

Example 9-5   mqsc execution to define QMGR1 objects

```
C:\Documents and Settings\Administrator\IBM\wmbt6.0\workspace>runmqsc <
SAW610.QMGR1.tst
5724-H72 (C) Copyright IBM Corp. 1994, 2004.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QMGR1.

      : *
      : *  MQSC Definitions for building Qmgr QMGR1
      : *
      :
    1 : DEFINE QLOCAL ('ExceptionQueue') +
      :         MAXDEPTH(5000) MAXMSGL(4194304) +
      :         SHARE DEFSOPT(SHARED) USAGE(NORMAL) +
      :         NOTRIGGER +
      :         REPLACE
AMQ8006: WebSphere MQ queue created.
      :
      ....
   15 : DEFINE SERVICE ('SAW610.Trigger1') +
```

```
            :            STARTCMD('C:\program files\IBM\WebSphere
MQ\bin\runmqtrm') +
            :            STARTARG('-m QMGR1 -q SAW610.INITQ') +
            :            STOPCMD(' ') +
            :            STOPARG(' ') +
            :            STDOUT(' ') +
            :            STDERR(' ') +
            :            CONTROL(STARTONLY) +
            :            SERVTYPE(COMMAND) +
            :            REPLACE
AMQ8625: WebSphere MQ service created.
            :
15 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

9.3.2 Connect the toolkit to the configuration manager

In order to deploy build time artifacts (message sets and message flows) to the runtime broker, the WebSphere Message Broker Toolkit must communicate with the runtime Configuration Manager. The toolkit uses a WebSphere MQ client connection to form this communication link. The following instructions assume that you have already created a runtime configuration manager and a runtime broker named BRK610.

If you have not already done so, open the WebSphere 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 QMGR1 in the Queue Manager Name field, localhost in the Host field, and 1414 in the Port field, 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 for server project and Domain1 for 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 `BRK610` as the broker name and QMGR1 in queue manager name, and click **Finish**.

When the action successfully completes, you will see a broker named BRK610 appear in the hierarchy of the Domains view, beneath the Broker Topology level. The BRK610 broker will have a single execution group as a child, named default.

9.3.3 Create execution groups

You will need to create several execution groups, a WarehousePO and a SubmitSN to execute the message flows. 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 **BRK610** in the Domains view and select **New →
   Execution Group**.

2. Enter `WarehousePO` in the Execution Group name and click **Finish**.

3. Right-click the broker **BRK610** in the Domains view and select **New →
   Execution Group**.

4. Enter `SubmitSN` in the Execution Group name and click **Finish**.

5. Click **Finish**.

6. The new execution groups should appear beneath the broker BRK610.

9.3.4 Create and deploy broker archive files

Two 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 `SAW610.P0` in the File Name field, then click **Finish**.
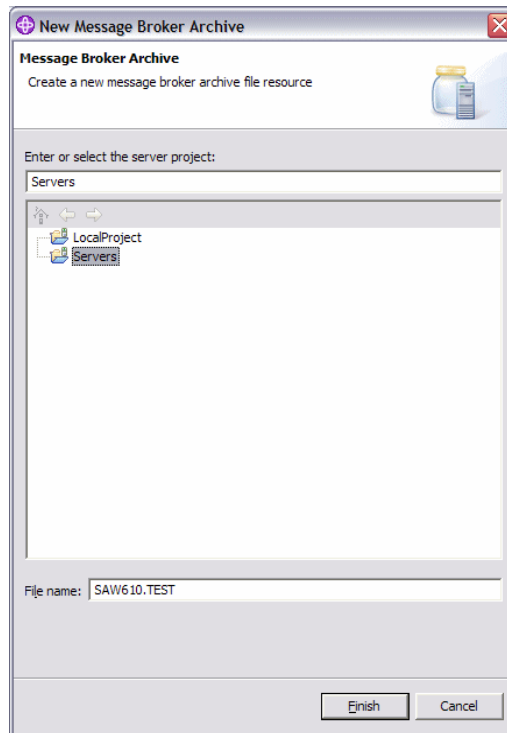
Figure 9-11   New Message Broker Archive window

3. Click the **Add** icon in the main editing window.
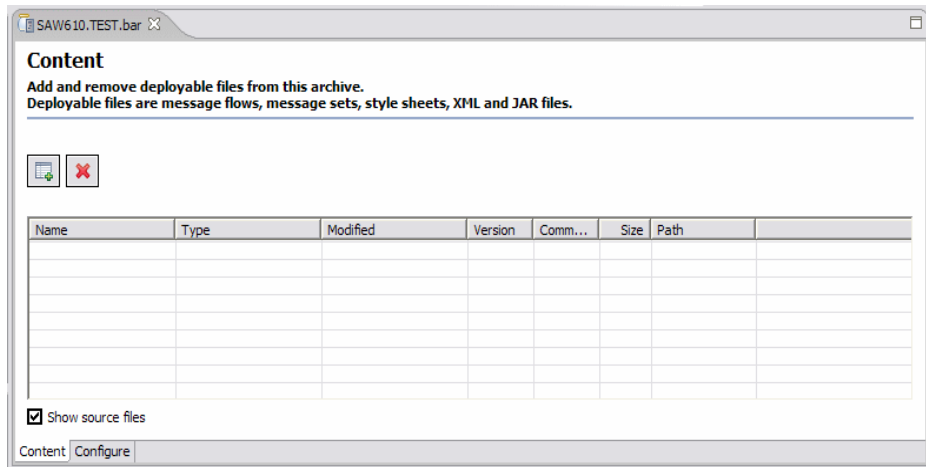
Figure 9-12   Content window

4. In the selection list, check **mfp_SW-610_Manufacturer**,
   mfp_SW-610_ManufacturerJava_SubmitPO, ms_Manufacturer, and
   ms_Other, and click **OK** to add it to the SAW610.Test.bar file. Click **OK** in the
   response dialog window.
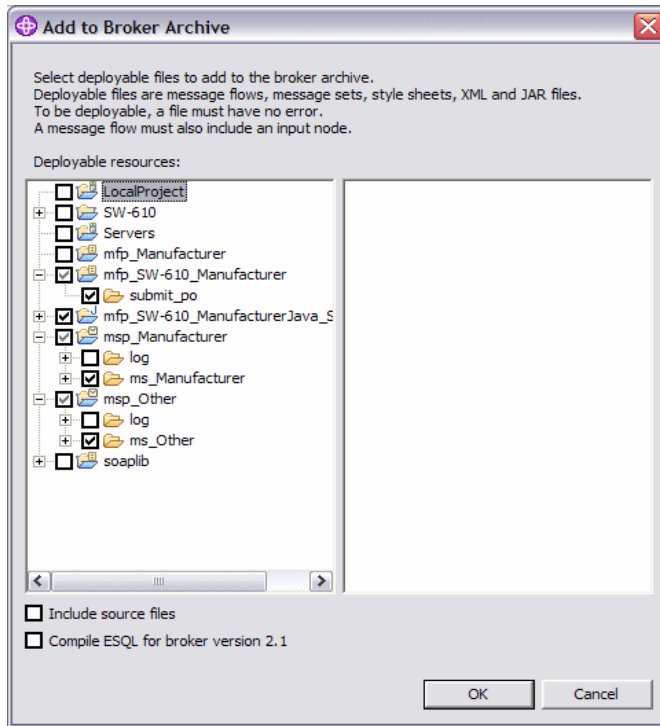
Figure 9-13   Add to Broker Archive window

5. Click **OK** in the response dialog window (Figure 9-14).
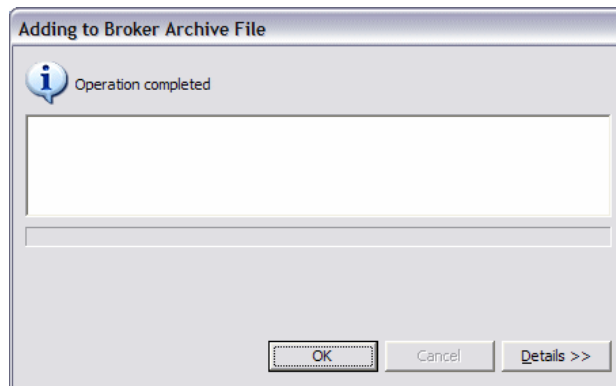
Figure 9-14   Adding to Broker Archive File window

6. Save the SAW610.PO.bar file by pressing Ctrl+S.

7. If the domain is not already connected, connect it now. In the Domains view, connect to the broker by right-clicking and selecting **Connect**.

8. Deploy the SAW610.PO.bar file by dragging it from the Broker Administration Navigator view to the execution group named WarehousePO under the broker BRK610 in the Domains view.

9. Click **OK** to acknowledge the response message from the configuration manager.

10. Double-click the Event log in the Domains view, and watch for the two successful messages with a current time stamp.

The message sets and message flow for the WarehousePO are now deployed and ready for use. Repeat this process for the SubmitSN message flow. Create an execution group SubmitSN, selecting the mfs_SW-619_Warehouse, mfp_SW-610_Warehouse, and mfp_SW-610_WarehouseJava in step 4 on page 267 for the shipping notice.

9.4 ESB based on WebSphere ESB

To implement the WebSphere ESB runtime ESB, we import a project interchange consisting of several mediation modules. The rest of this section provides instructions for connecting the WebSphere Integration Developer console to the WebSphere Enterprise Service Bus server and importing the required build time resources in order to run the scenario.

9.4.1 WebSphere Integration Developer to WebSphere Enterprise Service Bus connection

Complete the following instructions to connect WebSphere Integration Developer (WID) to the WebSphere ESB:

1. Start WID.

2. In the Servers tab window, right-click and select **NEW** → **Server**.
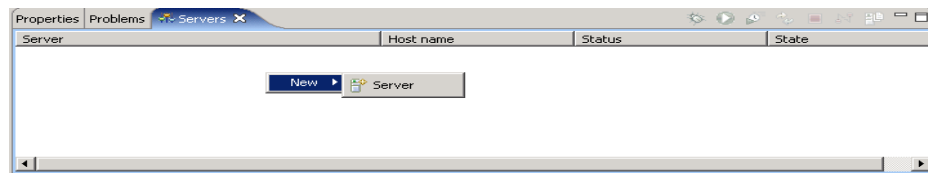
Figure 9-15   New Server connection in WebSphere Integration Developer Console

3. In the Define a New Server window, enter the host name or IP address for the WebSphere Enterprise Service Bus Server. In our example, we are local, so we use localhost. Select **WebSphere ESB Server v6.0** and click **Finish**.
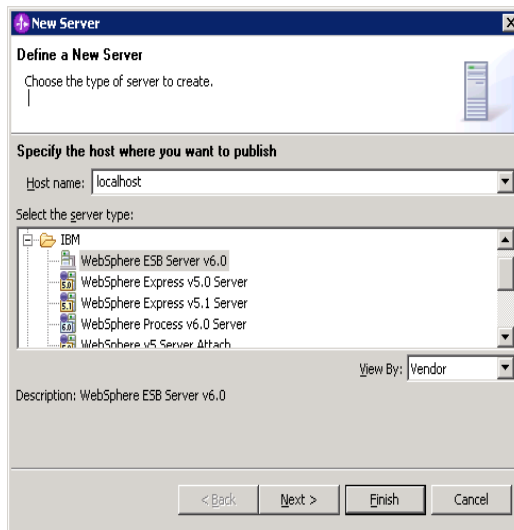
Figure 9-16   New Server window

4. This creates the connection to the WebSphere ESB. On the Server tab, you will see that the connection is in a starting state.
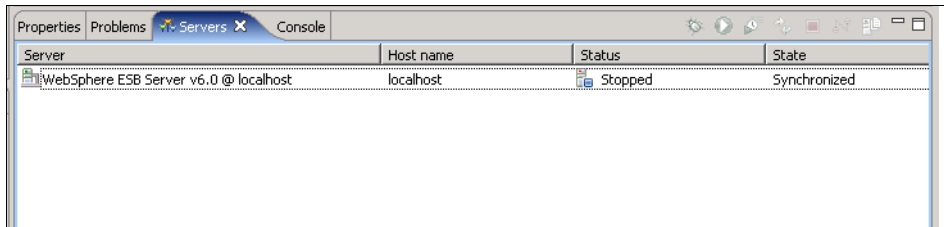
Figure 9-17   Servers tab window

5. Upon successful startup, in the Console tab you will see the message:

`WSVR0001I: Sever <server-name> open for e-business`
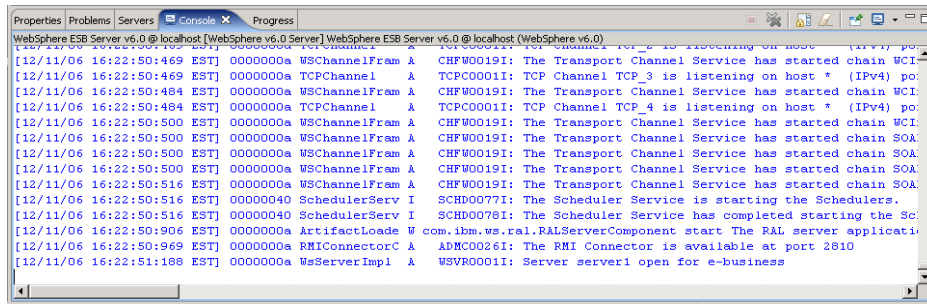
Figure 9-18   Console X window

9.4.2 Runtime artifacts

The project interchange files (Table 9-1) that cover the four scenarios that are discussed in this book are provided.

Table 9-1   Scenario and project interchange files

Scenario	Project Interchange
WebSphere ESB → WebSphere Message Broker using SOAP over HTTP	WESB_2_WMB_SOAP_HTTP.zip
WebSphere ESB → WebSphere Message Broker using MQJMS	WESB_2_WMB_MQJMS.zip
WebSphere ESB → WebSphere Message Broker using MQXML	WESB_2_WMB_MQXML.zip
WebSphere ESB → WebSphere Message Broker using SOAP over HTTP	WMB_2_WESB_SOAP_HTTP.zip

Please review Appendix C, "Additional material" on page 377, for instructions on how to acquire the files mentioned in the project interchange files.

To build the runtime artifacts we need to import the Project Interchange files.

1. Start the WebSphere Integration Developer Toolkit.

2. Import the project interchange into the Eclipse environment:

   a. Click **File** → **Import**.
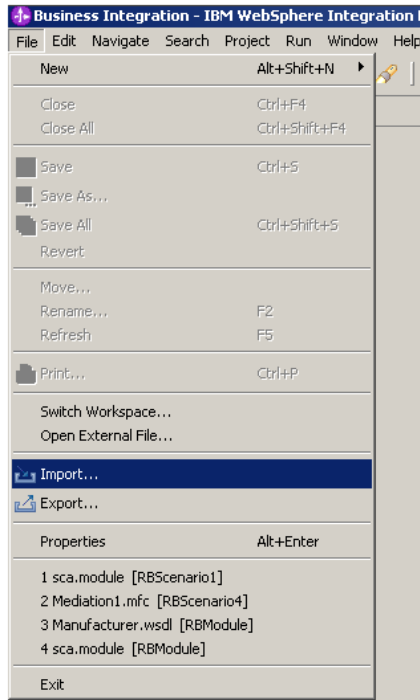
Figure 9-19   Select Import window

b. Select **Project Interchange** and click **Next**.
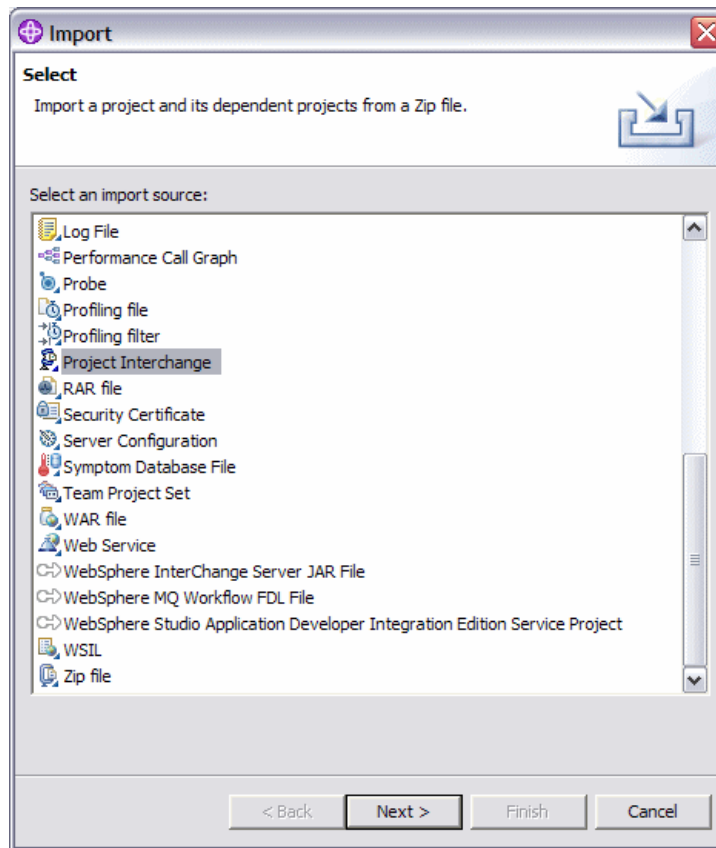
Figure 9-20   Import Project Interchange window example

c. Select the appropriate project interchange from the list and click **Open**. In our example, we selected sw-610_wmb6.0.2_releaseCandidate.zip.
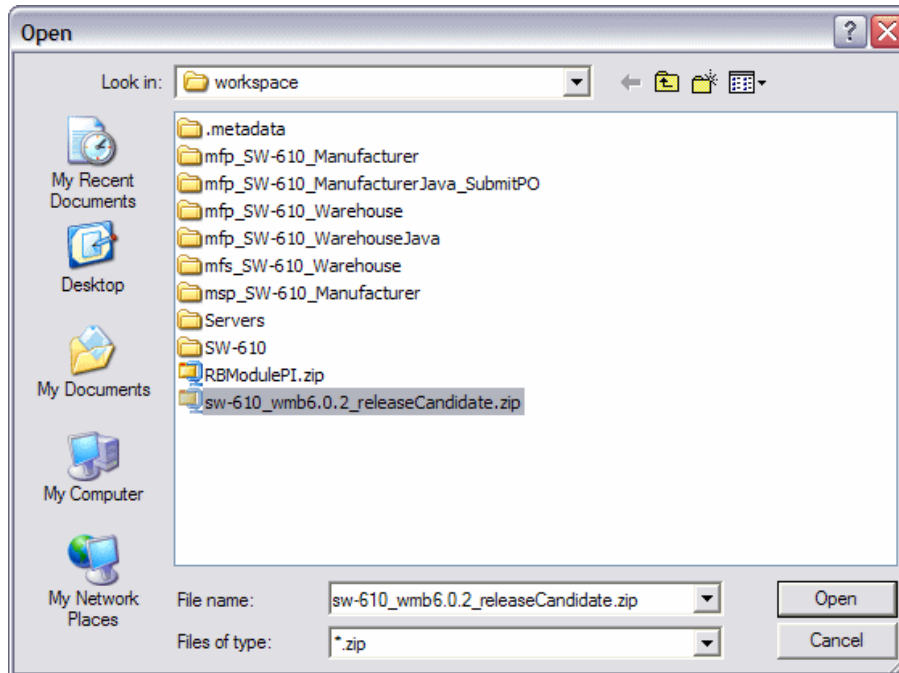
Figure 9-21   Project Interchange window example

d. Click **Select All**, and click **Finish** (Figure 9-22).
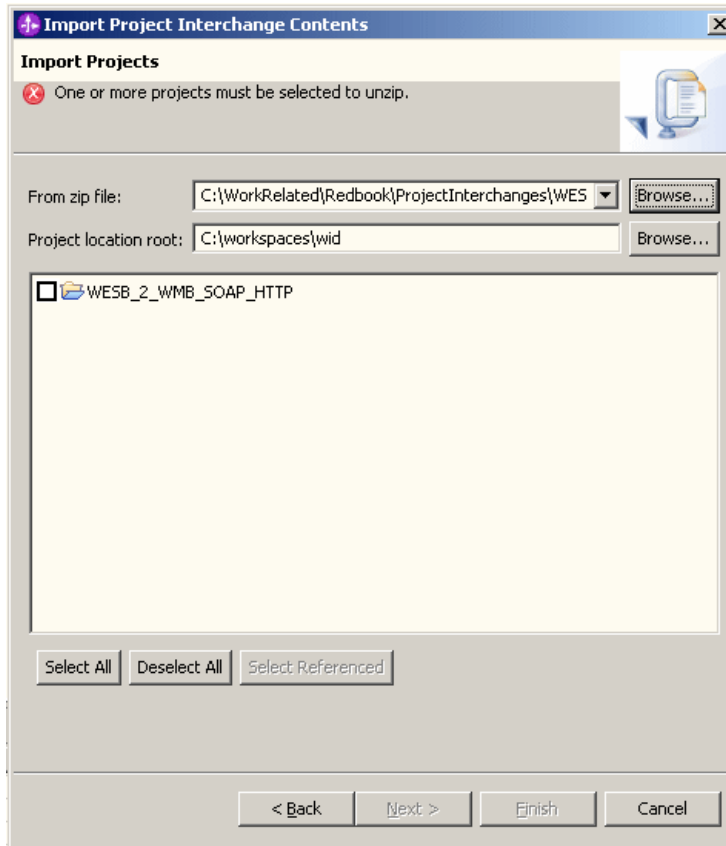
Figure 9-22   Import Projects window

This will import the mediation flow into the WebSphere Enterprise Service Bus server.
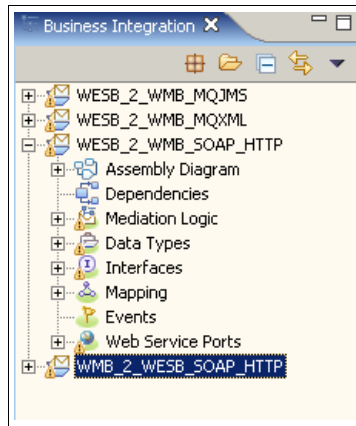
Figure 9-23   Business Integration window

9.5 Scenario 1: WebSphere ESB to WebSphere Message Broker interaction using SOAP over HTTP

This scenario demonstrates connectivity to WebSphere Enterprise Service Bus (retailer), sending a submitPO type message to the WebSphere Message Broker (manufacturer) using SOAP over HTTP. The manufacturer application responds to the request with an ackPO response. The following sections describe the mediation flow details.
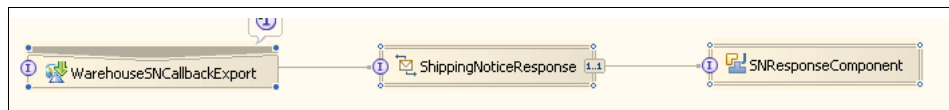
Assembly diagram

Figure 9-24   Assembly diagram

Component overview

Table 9-2   Exports

Display name	Name	Binding
ManufactureExport	ManufactureExport	

Table 9-3   Mediation flows

Display name	Name	Binding
SubmitPO	SubmitPO	Mediation1.mfc

Table 9-4   Exports

Display name	Name	Binding
ManufactureSOAPSvc	ManufactureSOAPSvc	WebService Binding

Table 9-5   Component wires

Source			Target	
Component name	Reference name	Component name	Interface name
SubmitPO	ManufacturePortType Partner	ManufactureSOAP Svc	ManufacturePorthType(www.ws-i.org/Sa mpleApplications/SupplyChainManage ment/2002-10/Manufacturer.wsdl

Table 9-6   Export wires

Export	Target	Interface name
ManufactureExport	SubmitPO	ManufacturerPortType

Mediation overview

Figure 9-25   SC1_AssemblyDiagram

Source interfaces

The following are the source interfaces used in the mediation flows.

ManufacturerPortTypeInterface

NameSpace:
http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Man ufacturer.wsdl

Table 9-7   ManufacturerPortTypeInterface

Operation name	Property	Reference	Operation
submitPO	request/response	ManufacturerPortTypePartner	submitPO

References

The following references are used in the mediation flows.

Table 9-8   ManufacturerPortTypePartner

Operation	Property	Interface	Operation
submitPO	request/response	`http://www.ws-i.org/SampleApplications/Suppl yChainManagement/2002-10/Manufacturer.wsdl`	submitPO

Request
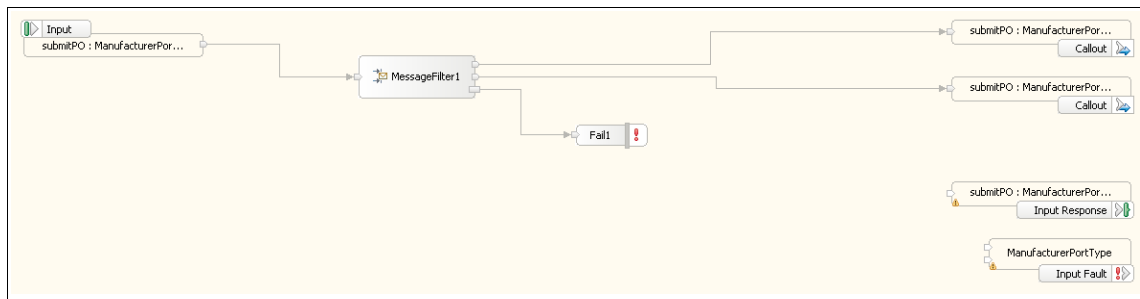
Figure 9-26   SC1_MF_Request

Input submitPO:ManufacturerPortType

The input node is the starting point for the request flow. It sends the message from the source operation into the request flow (ManufacturerPortType_submitPO_Input) and propagates the message to the out terminal.

Input response submitPO:ManufacturerPortType

The input response node is the endpoint in the request response flows. It returns the processed message as a response to the source operation.

Input fault ManufacturerPortType

The input fault node is an endpoint in the request flow. It has an input terminal for each fault message type defined in the source operation. Any message propagated to an input fault terminal will result in a WSDL fault on the source operation.

Callout submitPO:ManufacturerPortType

A callout node is an endpoint in the request flow. It sends the processed message to the target operation.
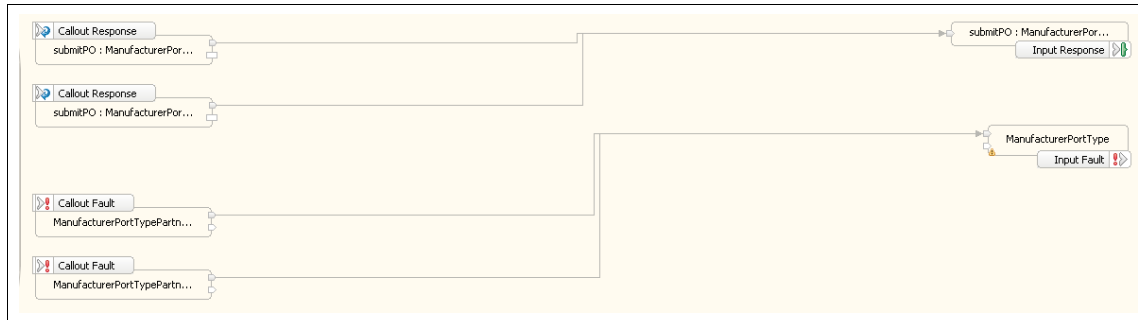
Response

Figure 9-27   SC1_MF_Response

Callout Response submitPO:ManufacturerPortTypePartner

A callout response node is a starting point for the response flow. It forwards the message received from the target operation into the response flow.

Outputs

Propagates the message to the primitive or node to which it is wired.

Fails

Propagates a generic message of the original message to the fail terminal if an exception that has not been explicitly described in the WSDL used in the callout occurs while invoking the callout. Exception information will be added to the transient header of the SMO. By default, a generic message will be propagated to the out terminal.

Callout fault ManufacturerPortTypePartner

A callout fault node is a starting point for the response flow. It has an output terminal for each fault message type defined in the target operation. When a WSDL fault occurs, the callout fault node propagates the message to the primitive or node to which it is wired.

Outputs

ConfigurationFaultMessage or submitPOFault based on the success or failure of the message.

Input response submitPO : ManufacturerPortType

The input response node is an endpoint in the request and response flows. It returns the processed message as a response to the source operation.

Inputs

Receives the processed message at the end of the flow.

Input fault ManufacturerPortType

The input fault node is an endpoint in the request flow. It has an input terminal for each fault message type defined in the source operation. Any message propagated to an input fault terminal will result in a WSDL fault of the source operation.

Message logger MessageLogger

Logger primitive that logs the response to a database.

Business objects

Business objects used in the mediation flows are listed below.
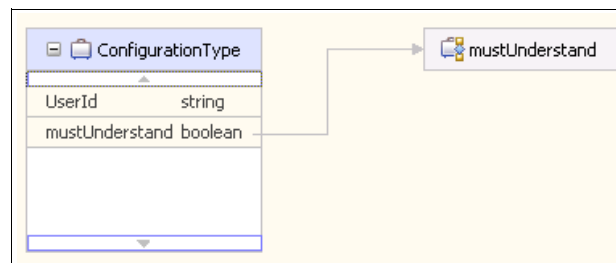
Figure 9-28   ConfigurationType
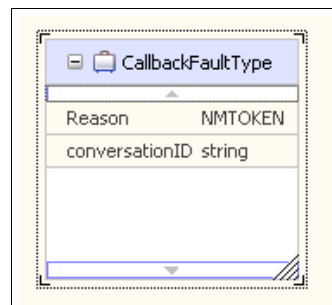
Figure 9-29   CallbackFaultType
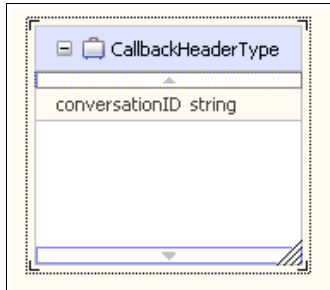
Figure 9-30   CallbackHeaderType
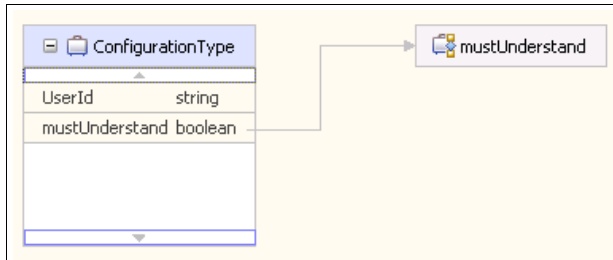
Figure 9-31   CustomerReferenceType
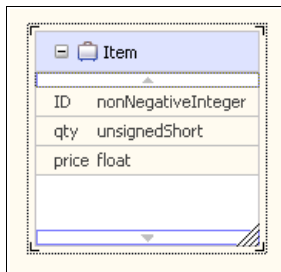
Figure 9-32   Item
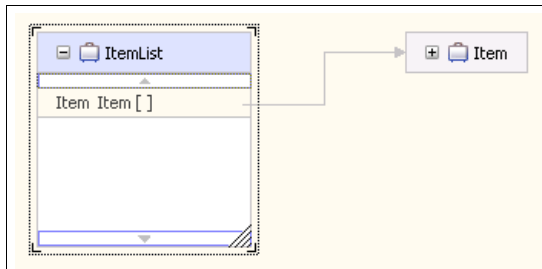
Figure 9-33   ItemList

Figure 9-34   ManufacturerPortType
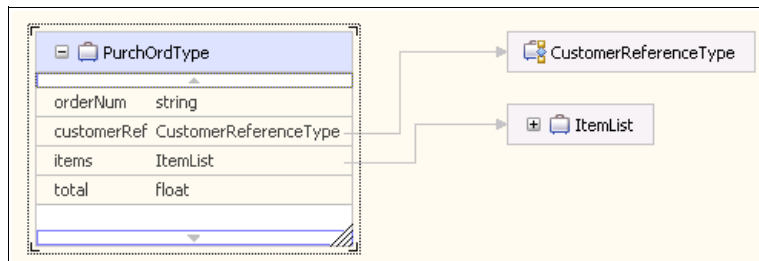
Figure 9-35   PurchOrdType
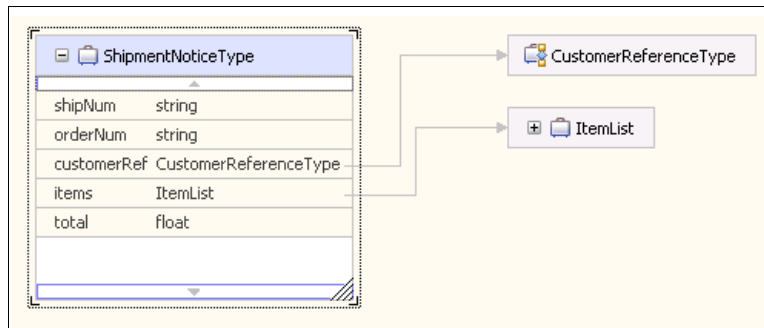
Figure 9-36   ShipmentNoticeType
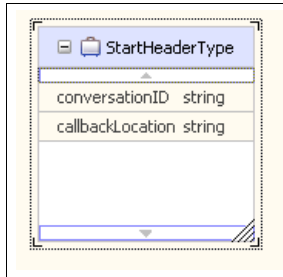
Figure 9-37   StartHeaderType
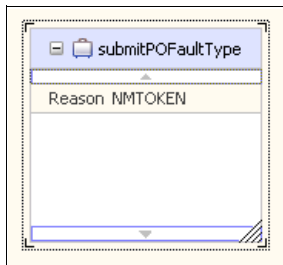
Figure 9-38   SubmitPOFaultType
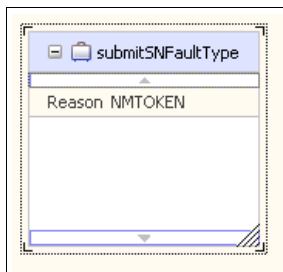
Figure 9-39   SubmitSNFaultType

Figure 9-40   WarehouseCallbackPortType

9.6 Scenario 2: WebSphere ESB to WebSphere Message Broker interaction using MQJMS

This scenario demonstrates connectivity of WebSphere ESB (retailer) sending a submitPO type message to WebSphere Message Broker (manufacturer) using MQ JMS. The manufacturer application responds to the request with an ackPO response. The following sections describe the mediation flow details.
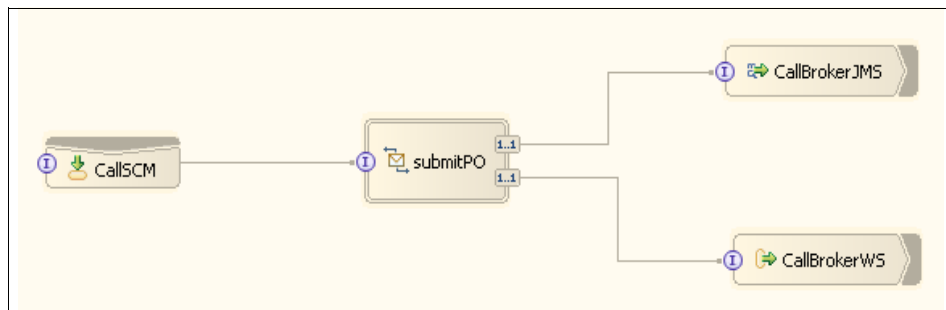
Assembly diagram

Figure 9-41   Assembly diagram

Component overview

The following tables list the overview of the components used in the mediation flows.

Table 9-9   Exports

Display name	Name	Binding
CallSCM	CallSCM	

Table 9-10   Mediation flows

Display name	Name	Binding
SubmitPO	SubmitPO	Mediation1.mfc

Table 9-11   Exports

Display name	Name	Binding
CallBrokerJMS	CallBrokerJMS	
CallBrokerWS	CallBrokerWS	

Table 9-12   Component wires

Source			Target
Component name	Reference name	Component name	Interface name
SubmitPO	ManufacturePortTypePartner1	CallBrokerJMS	ManufacturePortType(www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl
SubmitPO	ManufacturePortTypePartner2	CallBrokerWS	ManufacturePortType(www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl

Table 9-13   Export wires

Export	Target	Interface name
CallSCM	SubmitPO	ManufacturerPortType

Mediation overview
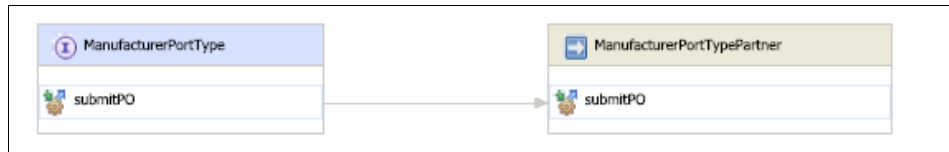
Figure 9-42   Mediation overview

Source interfaces

The following are the source interfaces used in the mediation flows.

ManufacturerPortTypeInterface

NameSpace:
http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl

Table 9-14   ManufacturerPortTypeInterface

Operation name	Property	Reference	Operation
submitPO	request/response	ManufacturerPortTypePartner1	submitPO
submitPO	request/response	ManufacturerPortTypePartner2	submitPO

References

The following are the tables describe the mediation references.

Table 9-15   ManufacturerPortTypePartner1

Operation	Property	Interface	Operation
submitPO	request/response	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl	submitPO

Table 9-16   ManufacturerPortTypePartner2

Operation	Property	Interface	Operation
submitPO	request/response	ManufacturerPortType	submitPO
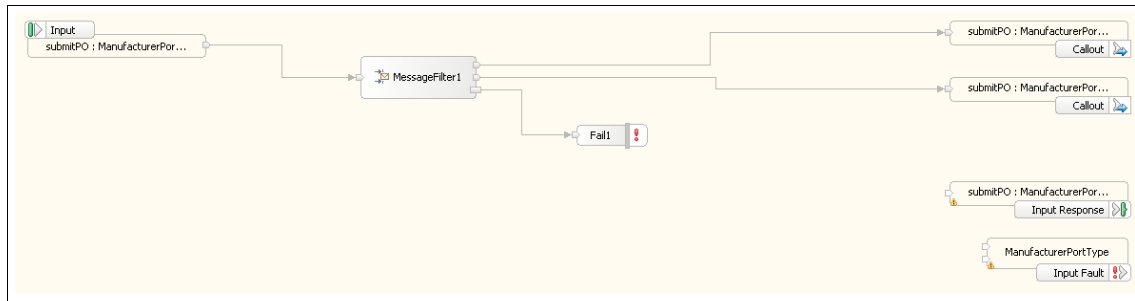
Request flow

Figure 9-43   Request flow

Input submitPO:ManufacturerPortType

The input node is the starting point for the request flow. It sends the message from the source operation into the request flow (ManufacturerPortType_submitPO_Input) and propagates the message to the out terminal.

Input response submitPO:ManufacturerPortType

The input response node is the endpoint in the request response flows. It returns the processed message as a response to the source operation.

Input fault ManufacturerPortType

The input fault node is an endpoint in the request flow. It has an input terminal for each fault message type defined in the source operation. Any message propagated to an input fault terminal will result in a WSDL fault on the source operation.

Callout submitPO:ManufacturerPortType1

A callout node is an endpoint in the request flow. It sends the processed message to the target operation.

Message filter MessageFilter1

The filter primitive filters (Table 9-17) the incoming message based on the SMO values and compares the message against a list of configured expressions.

Table 9-17   Message filter MessageFilter1

Pattern	Terminal Name
/body/PurchaseOrder/customerRe[self::node()="DirectOrder"]	match1

Callout submitPO:ManufacturerPortType2

A callout node is an endpoint in the request flow. It sends the processed message to the target operation.

Fail Fail1

Enables users to throw an explicit exception within the mediation flow. Flow execution is terminated and global transactions are rolled back.
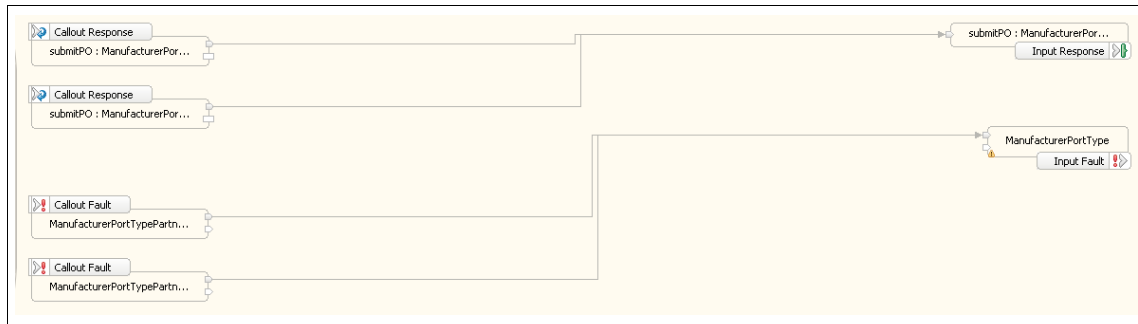
Response

Figure 9-44   Response

Callout response submitPO:ManufacturerPortTypePartner1

A callout response node is a starting point for the response flow. It forwards the message received from the target operation into the response flow.

Outputs

Propagates the message to the primitive or node to which it is wired.

Fails

Propagates a generic message of the original message to the fail terminal if an exception that has not been explicitly described in the WSDL used in the callout occurs while invoking the callout. Exception information will be added to the transient header of the SMO. By default, a generic message will be propagated to the out terminal.

Callout fault ManufacturerPortTypePartner1

A callout fault node is a starting point for the response flow. It has an output terminal for each fault message type defined in the target operation. When a WSDL fault occurs, the callout fault node propagates the message to the primitive or node to which it is wired.

Outputs

ConfigurationFaultMessage or submitPOFault based on the success or failure of the message.

Input response submitPO : ManufacturerPortType

The input response node is an endpoint in the request and response flows. It returns the processed message as a response to the source operation.

Inputs

Receives the processed message at the end of the flow.

Input fault manufacturerPortType

The input fault node is an endpoint in the request flow. It has an input terminal for each fault message type defined in the source operation. Any message propagated to an input fault terminal will result in a WSDL fault of the source operation.

Callout response submitPO:ManufacturerPortTypePartner2

A callout response node is a starting point for the response flow. It forwards the message received from the target operation into the response flow.

Outputs

Propagates the message to the primitive or node to which it is wired.

Fails

Propagates a generic message of the original message to the fail terminal if an exception that has not been explicitly described in the WSDL used in the callout occurs while invoking the callout. Exception information will be added to the transient header of the SMO. By default, a generic message will be propagated to the out terminal.

Callout fault ManufacturerPortTypePartner2

A callout fault node is a starting point for the response flow. It has an output terminal for each fault message type defined in the target operation. When a WSDL fault occurs, the callout fault node propagates the message to the primitive or node to which it is wired.

Business objects

The following are the business objects that are used in the mediation flows.

Figure 9-45   ConfigurationType
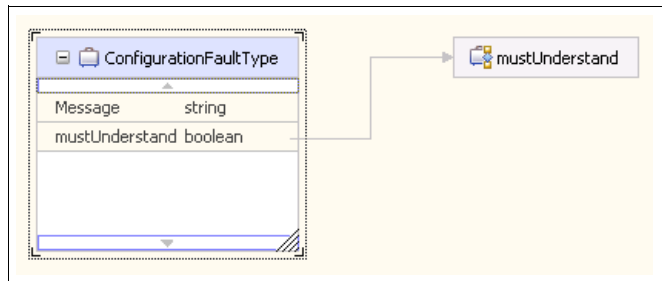
Figure 9-46   ConfigurationFaultType
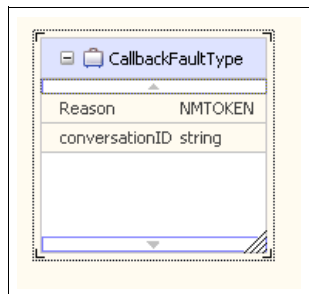
Figure 9-47   CallbackFaultType
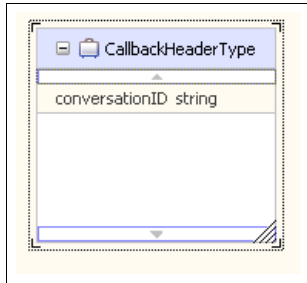
Figure 9-48   CallbackHeaderType
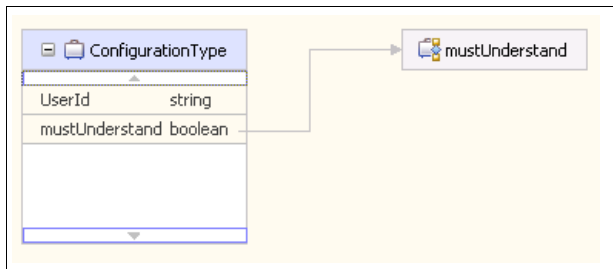
Figure 9-49   CustomerReferenceType
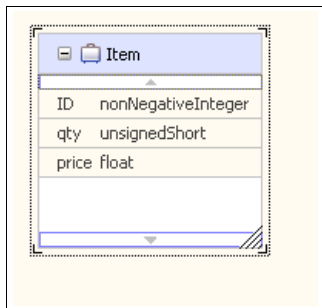
Figure 9-50   Item
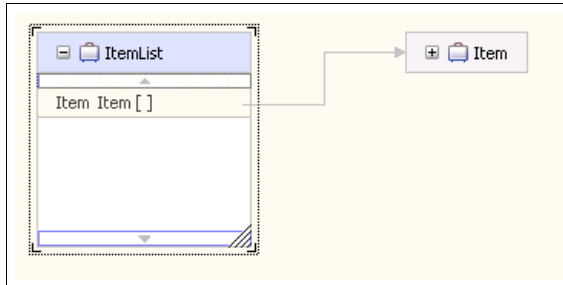
Figure 9-51   ItemList
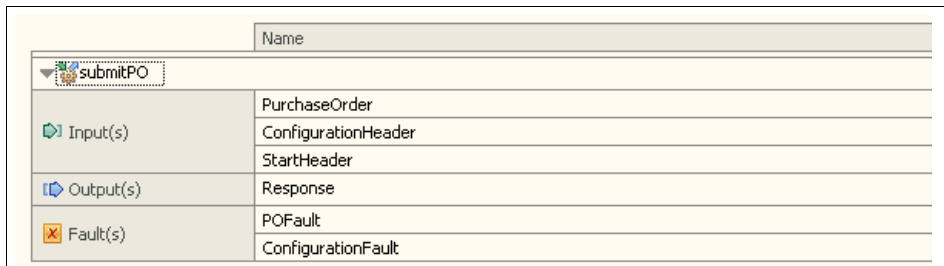
Figure 9-52   ManufacturerPortType
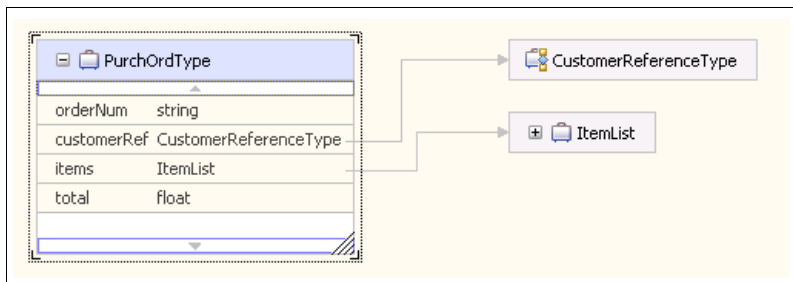
Figure 9-53   PurchOrdType
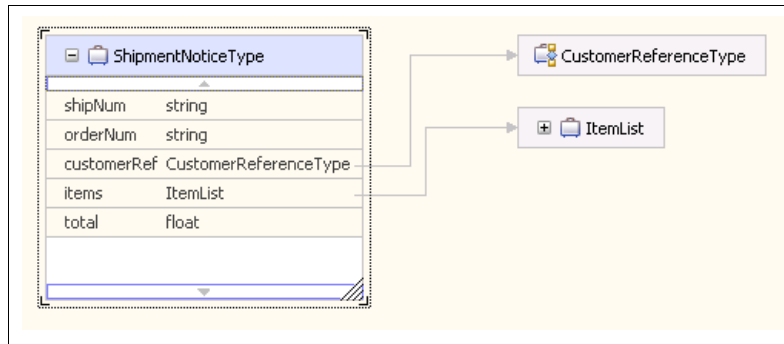
Figure 9-54   ShipmentNoticeType
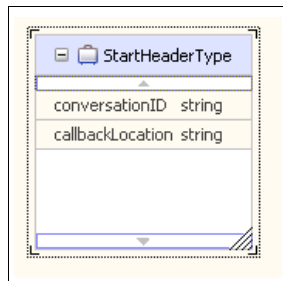
Figure 9-55   StartHeaderType
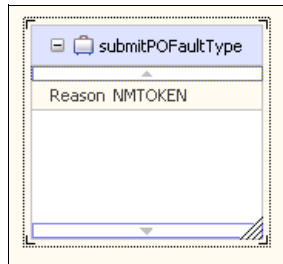
Figure 9-56   SubmitPOFaultType
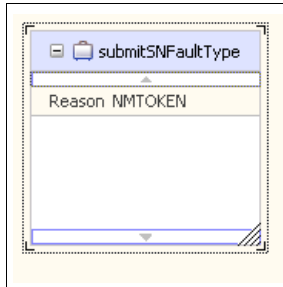
Figure 9-57   SubmitSNFaultType

Figure 9-58   WarehouseCallbackPortType

9.7 Scenario 3: WebSphere ESB to WebSphere Message Broker interaction using MQ XML

This scenario demonstrates connectivity of WebSphere ESB (retailer), sending a submitPO type message to WebSphere Message Broker (manufacturer) using MQ XML. The manufacturer application responds to the request with an ackPO response. The following sections describe the mediation flow details.
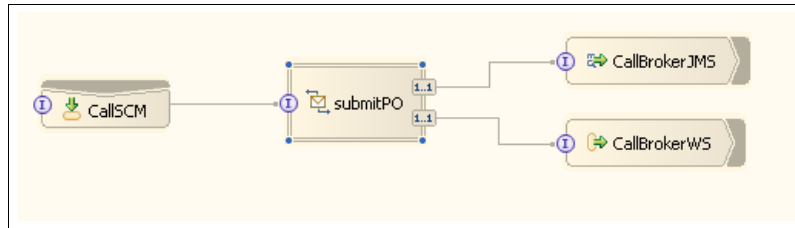
Assembly diagram

Figure 9-59   Assembly diagram

Component overview

Table 9-18   Exports

Display name	Name	Binding
CallSCM	CallSCM	

Table 9-19   Mediation flows

Display name	Name	Binding
SubmitPO	SubmitPO	Mediation1.mfc

Table 9-20   Exports

Display name	Name	Binding
CallBrokerJMS	CallBrokerJMS	
CallBrokerWS	CallBrokerWS	

Table 9-21   Component wires

Source			Target
Component name	**Reference name**	**Component name**	**Interface name**
SubmitPO	ManufacturePortTypePartner1	CallBrokerJMS	ManufacturePortType(www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl
SubmitPO	ManufacturePortTypePartner2	CallBrokerWS	ManufacturePortType(www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl

Table 9-22   Export wires

Export	Target	Interface Name
CallSCM	SubmitPO	ManufacturerPortType

Mediation overview
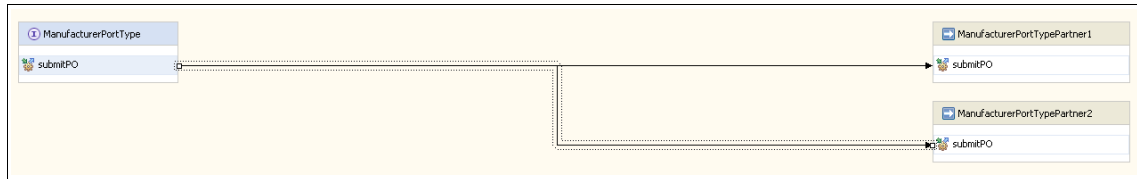
Figure 9-60   Mediation overview

Source interfaces

The following are the interfaces used in mediation flows.

ManufacturerPortTypeInterface

NameSpace:
http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl

Table 9-23   ManufacturerPortTypePartner1

Operation name	Property	Reference	Operation
submitPO	request/response	ManufacturerPortTypePartner1	submitPO
submitPO	request/response	ManufacturerPortTypePartner2	submitPO

References

The following are the references used in the mediation flow.

Table 9-24   ManufacturerPortTypePartner1

Operation	Property	Interface	Operation
submitPO	request/response	http://www.ws-i.org/Sample Applications/SupplyChainM anagement/2002-10/Manufa cturer.wsdl	submitPO

Table 9-25   ManufacturerPortTypePartner2

Operation	Property	Interface	Operation
submitPO	request/response	ManufacturerPortType	submitPO
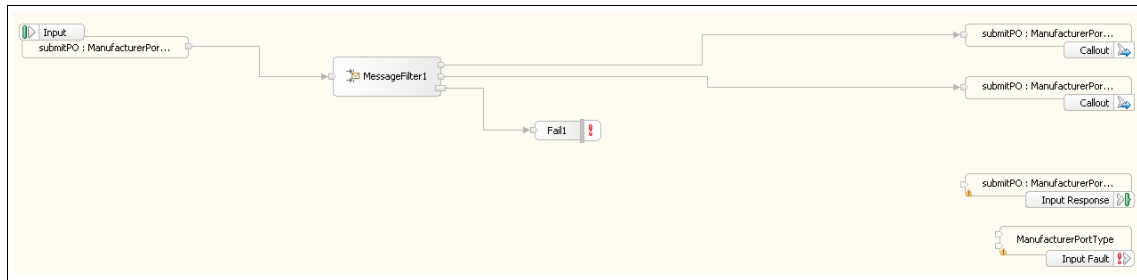
Request flow

Figure 9-61   Request flow

Input submitPO:ManufacturerPortType

The input node is the starting point for the request flow. It sends the message from the source operation into the request flow (ManufacturerPortType_submitPO_Input) and propagates the message to the out terminal.

Input response submitPO:ManufacturerPortType

The input response node is the endpoint in the request response flows. It returns the processed message as a response to the source operation.

Input fault manufacturerPortType

The input fault node is an endpoint in the request flow. It has an input terminal for each fault message type defined in the source operation. Any message propagated to an input fault terminal will result in a WSDL fault on the source operation.

Callout submitPO:ManufacturerPortType1

A callout node is an endpoint in the request flow. It sends the processed message to the target operation.

Message filter MessageFilter1

The filter primitive filters the incoming message based on the SMO values and compares the message against a list of configured expressions.

Table 9-26   Message filter MessageFilter1

Pattern	Terminal name
/body/PurchaseOrder/customerRe[self::node()="DirectOrder"]	match1

Callout submitPO:ManufacturerPortType2

A callout node is an endpoint in the request flow. It sends the processed message to the target operation.

Fail Fail1

Enables users to throw an explicit exception within the mediation flow. Flow execution is terminated and global transactions are rolled back.
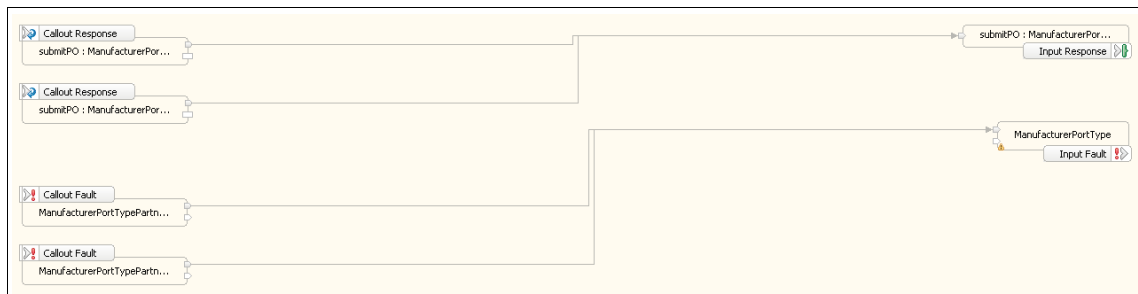
Response

Figure 9-62   Response

Callout response submitPO:ManufacturerPortTypePartner1

A callout response node is a starting point for the response flow. It forwards the message received from the target operation into the response flow.

Outputs

Propagates the message to the primitive or node to which it is wired.

Fails

Propagates a generic message of the original message to the fail terminal if an exception that has not been explicitly described in the WSDL used in the callout occurs while invoking the callout. Exception information will be added to the transient header of the SMO. By default, a generic message will be propagated to the out terminal.

Callout fault ManufacturerPortTypePartner1

A callout fault node is a starting point for the response flow. It has an output terminal for each fault message type defined in the target operation. When a

WSDL fault occurs, the callout fault node propagates the message to the primitive or node to which it is wired.

Outputs

ConfigurationFaultMessage or submitPOFault based on the success or failure of the message.

Input Response submitPO : ManufacturerPortType

The input response node is an endpoint in the request and response flows. It returns the processed message as a response to the source operation.

Inputs

Receives the processed message at the end of the flow.

Input fault ManufacturerPortType

The input fault node is an endpoint in the request flow. It has an input terminal for each fault message type defined in the source operation. Any message propagated to an input fault terminal will result in a WSDL fault of the source operation.

Callout response submitPO:ManufacturerPortTypePartner2

A callout response node is a starting point for the response flow. It forwards the message received from the target operation into the response flow.

Outputs

Propagates the message to the primitive or node to which it is wired.

Fails

Propagates a generic message of the original message to the fail terminal if an exception that has not been explicitly described in the WSDL used in the callout occurs while invoking the callout. Exception information will be added to the transient header of the SMO. By default, a generic message will be propagated to the out terminal.

Callout fault ManufacturerPortTypePartner2

A callout fault node is a starting point for the response flow. It has an output terminal for each fault message type defined in the target operation. When a WSDL fault occurs, the callout fault node propagates the message to the primitive or node to which it is wired.

Business objects

The following are the business objects used in the mediation flow.

Figure 9-63   ConfigurationType
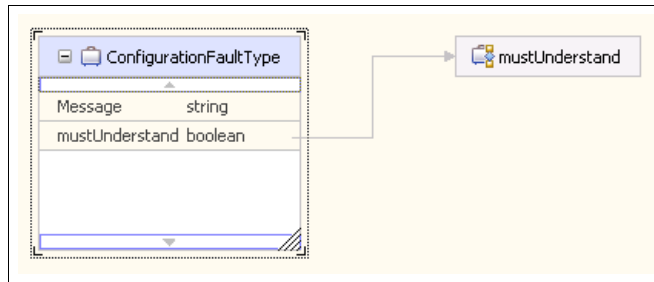
Figure 9-64   ConfigurationFaultType
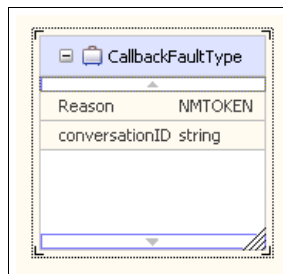
Figure 9-65   CallbackFaultType
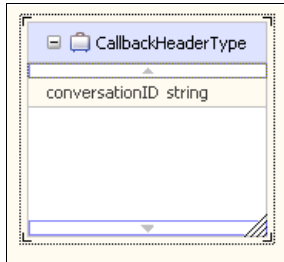
Figure 9-66   CallbackHeaderType
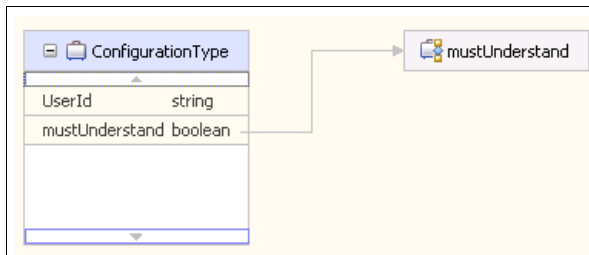
Figure 9-67   CustomerReferenceType
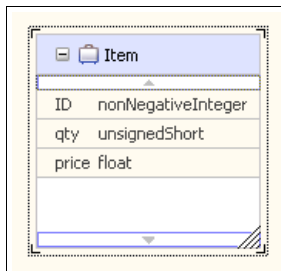
Figure 9-68   Item
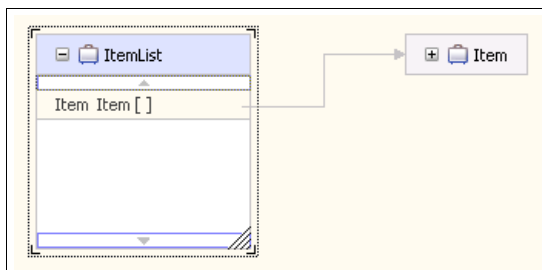
Figure 9-69   ItemList

Figure 9-70   ManufacturerPortType
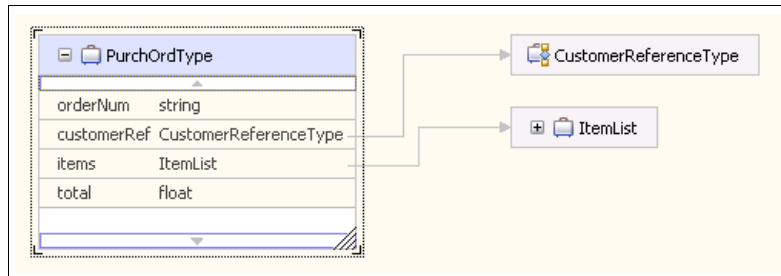
Figure 9-71   PurchOrdType
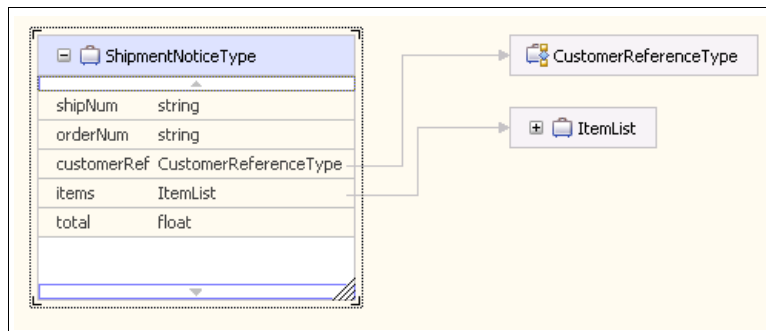
Figure 9-72   ShipmentNoticeType
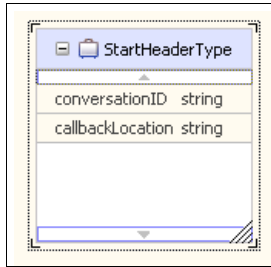
Figure 9-73   StartHeaderType
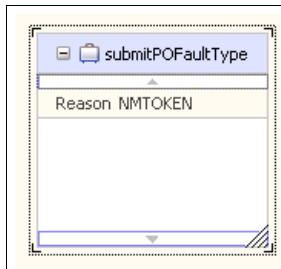
Figure 9-74   SubmitPOFaultType
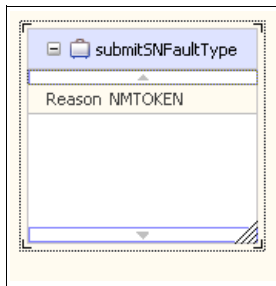
Figure 9-75   SubmitSNFaultType

Figure 9-76   WarehouseCallbackPortType

9.8 Scenario 4: WebSphere Message Broker to WebSphere ESB interaction using MQ XML

This scenario demonstrates connectivity WebSphere Message Broker (manufacturer) sending a submitSB type message to WebSphere ESB (retailer) using MQ XML. The retailer application has a Java component that responds to the request with an boolean response. The following sections describe the mediation flow details.
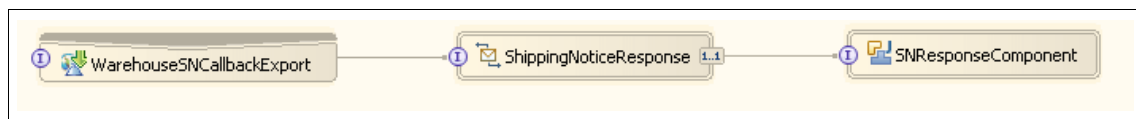
Assembly diagram

Figure 9-77   Assembly diagram

Mediation overview
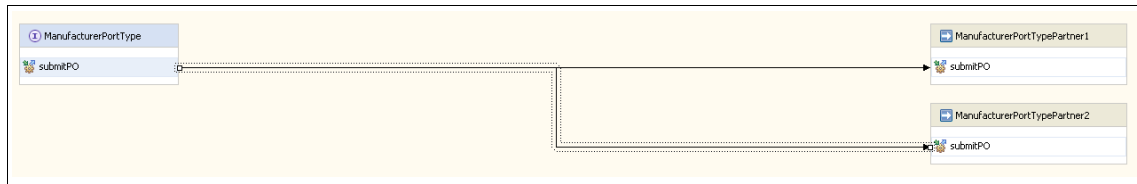
Figure 9-78   Mediation overview

Source interfaces

The following are the source interfaces used in the mediation flow.

WarehouseCallbackPortType

NameSpace:
http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl

Table 9-27   WarehouseCallbackPortType

Operation name	Property	Reference	Operation
submitSN	request/response	WarehouseCallbackPortType	submitSN

References

The following references are used in the mediation flow.

Table 9-28   WarehouseCallbackPortTypePartner

Operation	Property	Interface	Operation
submitSN	request/response	WarehouseCallbackPortType http://www.ws-i.org/Sample Applications/SupplyChainM anagement/2002-10/Manufacturer.wsdl	submitSN

Request flow
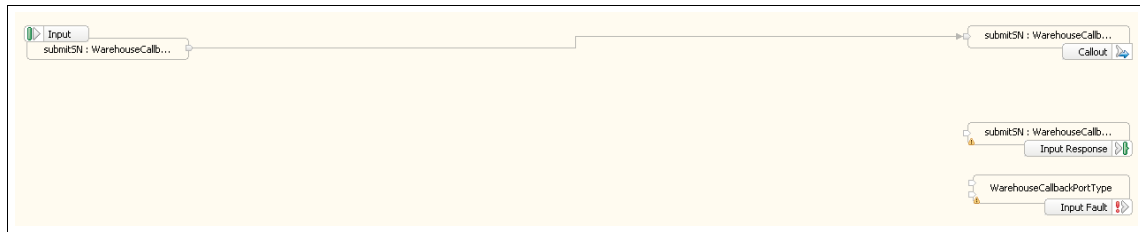
Input submitSN:WarehouseCallbackPortType

The input node is the starting point for the request flow. It sends the message from the source operation into the request flow (WarehouseCallbackPortType_submitSN_Input) and propagates the message to the out terminal.

Input response submitSN:WarehouseCallbackPortType

The input response node is the endpoint in the request response flows. It returns the processed message as a response to the source operation.

Callout submitSN:WarehouseCallbackPortTypePartner

A callout node is an endpoint in the request flow. It sends the processed message to the target operation.

Response

Figure 9-79   Response

Callout response submitSN:WarehouseCallbackPortTypePartner

A callout response node is a starting point for the response flow. It forwards the message received from the target operation into the response flow.

Outputs

Propagates the message to the primitive or node to which it is wired.

Fails

Propagates a generic message of the original message to the fail terminal if an exception that has not been explicitly described in the WSDL used in the callout occurs while invoking the callout. Exception information will be added to the transient header of the SMO. By default, a generic message will be propagated to the out terminal.

Callout fault WarehouseCallbackPortTypePartner

A callout fault node is a starting point for the response flow. It has an output terminal for each fault message type defined in the target operation. When a WSDL fault occurs, the callout fault node propagates the message to the primitive or node to which it is wired.

Outputs

ConfigurationFaultMessage or submitPOFault based on the success or failure of the message.

Input response submitSN:WarehouseCallbackPortType

The input response node is an endpoint in the request and response flows. It returns the processed message as a response to the source operation.

Inputs

Receives the processed message at the end of the flow.

Input fault WarehouseCallbackPortType

The input fault node is an endpoint in the request flow. It has an input terminal for each fault message type defined in the source operation. Any message propagated to an input fault terminal will result in a WSDL fault of the source operation.

SNResponseComponent

```
package sca.component.java.impl;

import commonj.sdo.DataObject;
import com.ibm.websphere.sca.ServiceManager;

public class Component1Impl {
        /**
         * Default constructor.
         */
        public Component1Impl() {
                super();
        }                                                              .

        /**
         * Return a reference to the component service instance for this implementation
         * class.  This method should be used when passing this service to a partner reference
         * or if you want to invoke this component service asynchronously.
         *
         * @generated (com.ibm.wbit.java)
         */
        private Object getMyService() {
                return (Object) ServiceManager.INSTANCE.locateService("self");
        }

        /**
         * Method generated to support implemention of operation "submitSN" defined for WSDL
port type
         * named "interface.WarehouseCallbackPortType".
         *
         * The presence of commonj.sdo.DataObject as the return type and/or as a parameter
         * type conveys that its a complex type. Please refer to the WSDL Definition for more
information
         * on the type of input, output and fault(s).
         */
        public boolean submitSN(DataObject shipmentNotice,
                        DataObject configurationHeader, DataObject callbackHeader) {
                //TODO Needs to be implemented.


                return true;
        }
        /**
         * Method generated to support implemention of operation "errorPO" defined for WSDL
port type
         * named "interface.WarehouseCallbackPortType".
         *
         * The presence of commonj.sdo.DataObject as the return type and/or as a parameter
         * type conveys that its a complex type. Please refer to the WSDL Definition for more
information
         * on the type of input, output and fault(s).
         */
        public boolean errorPO(DataObject processPOFault,
                        DataObject configurationHeader, DataObject callbackHeader) {
                //TODO Needs to be implemented.
                return false;
        }

}
```

Figure 9-80   sca.component.java.impl window

Business objects

The following are the business objects that are used in the mediation flows.
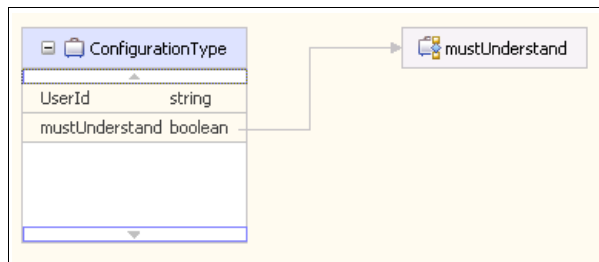
Figure 9-81   ConfigurationType
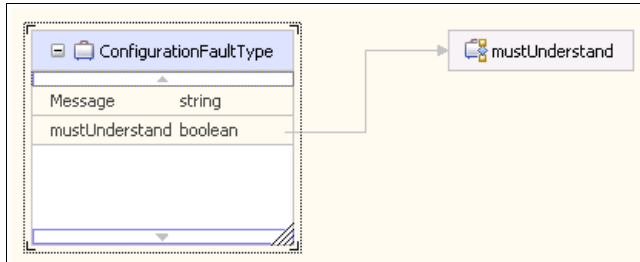
Figure 9-82   ConfigurationFaultType
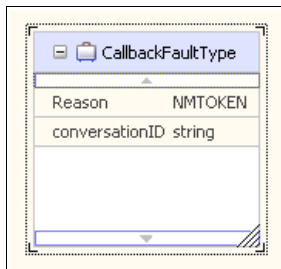
Figure 9-83   CallbackFaultType
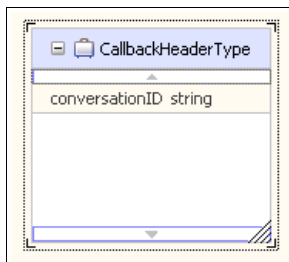
Figure 9-84   CallbackHeaderType
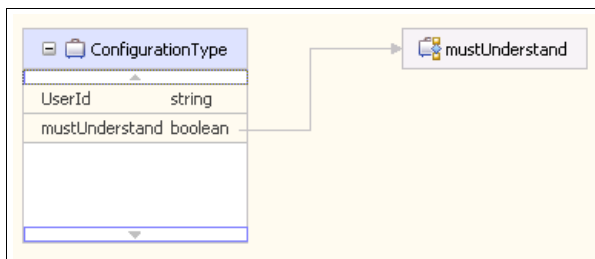
Figure 9-85   CustomerReferenceType
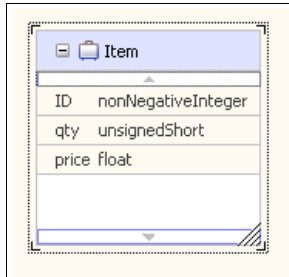
Figure 9-86   Item
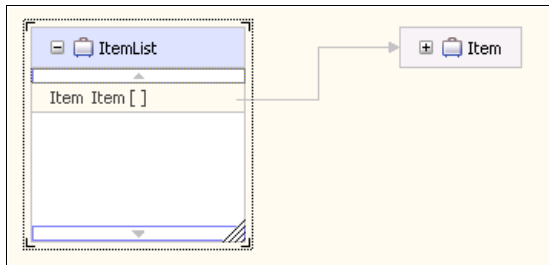
Figure 9-87   ItemList
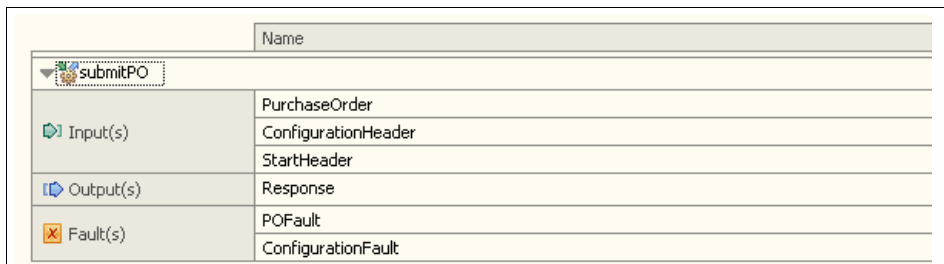
Figure 9-88   ManufacturerPortType
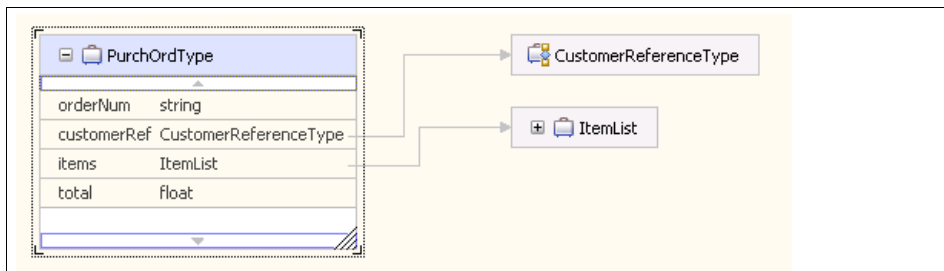
Figure 9-89   PurchOrdType
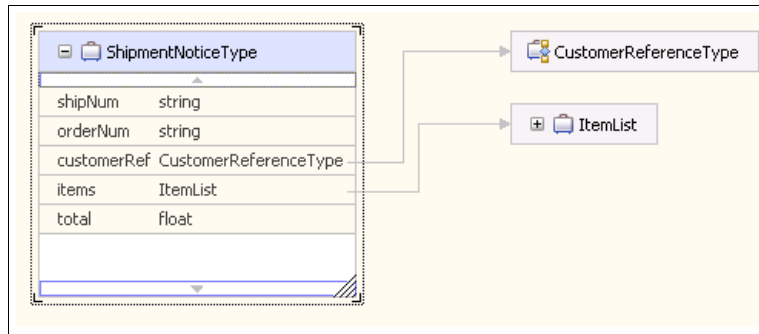
Figure 9-90   ShipmentNoticeType
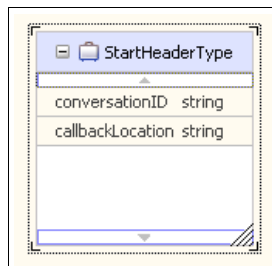
Figure 9-91   StartHeader Type
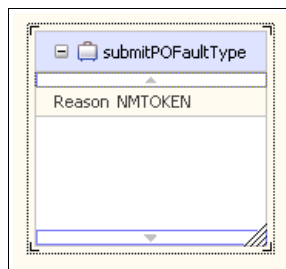
Figure 9-92   SubmitPOFaultType
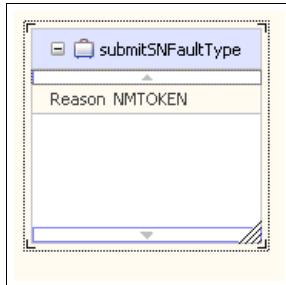
Figure 9-93   SubmitSNFaultType

Figure 9-94   WarehouseCallbackPortType window

9.9 Testing the scenarios

Using the steps below, you can test each scenario:

1. In the Business Integration Perspective, right-click the appropriate project and select **Test Module** (Figure 9-95).
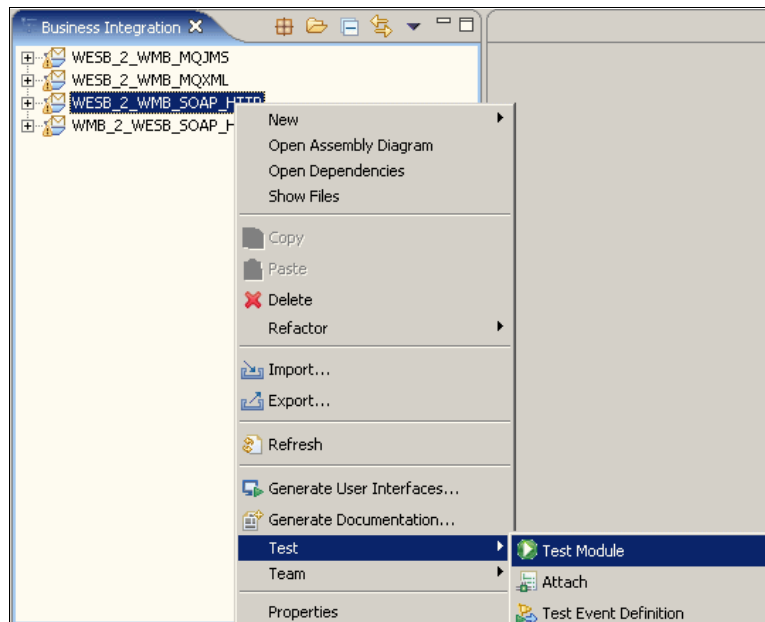
Figure 9-95   Business Integration window

2. In the Events window, click **Configurations**. Select **Emulators** and click **Remove** (Figure 9-96).
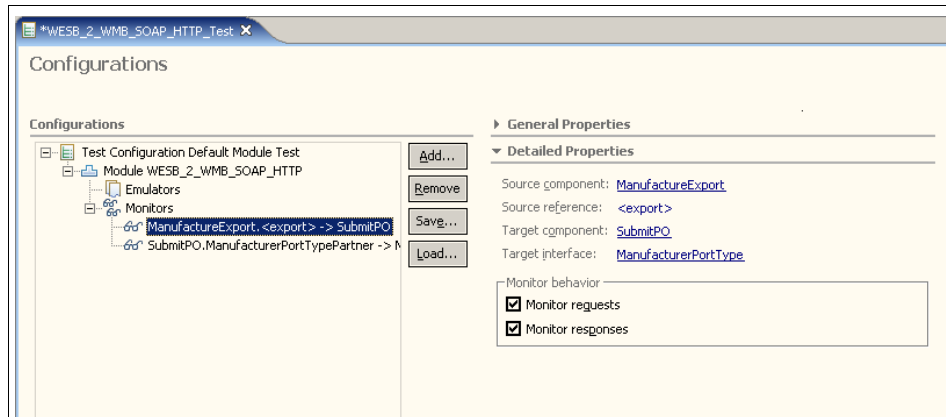
Figure 9-96   Configurations

3. In the Events window, fill in the values for the PurchaseOrder (Figure 9-97).
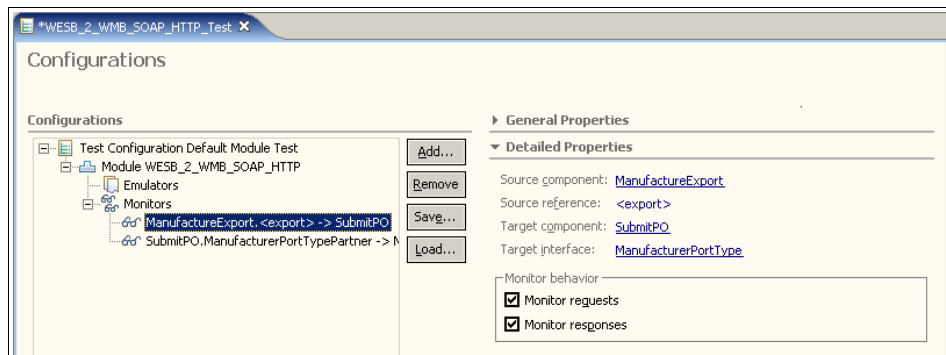
Figure 9-97   Configuarations

4. Click **Continue**. This will send the message to the Message Broker, and when the response is received from the warehouse this can be viewed in the Events Window.

9.10 Runtime guidelines for back-end existing manufacturer applications

The manufacturer application is a stand-alone application that uses the WebSphere MQ API to communicate with the outer world.

The same application code is designed to be run as a triggered process under WebSphere MQ. We have defined three local queue: SAQ610.INITQ, SAW610.POREQ.OUT, and SAW610.POREQ.IN. The queue attributes for the local queue SAW610.POREQ.IN are triggered (trigger type of every, initq of SAQ610.INITQ, and process as SAW610.RESPONSE). The process definition SAQ610.RESPOSE is a Windows NT® application calling a command file BackEnd.cmd, which is supplied in the resource samples file for this book. After defining the queues, process, channels, and service (as described in 9.3.1, "Configure WebSphere MQ environment" on page 261), you need to verify that the Trigger monitor is active. Open the WebSphere MQ explorer and display the services or check the task manager. See Figure 9-98.
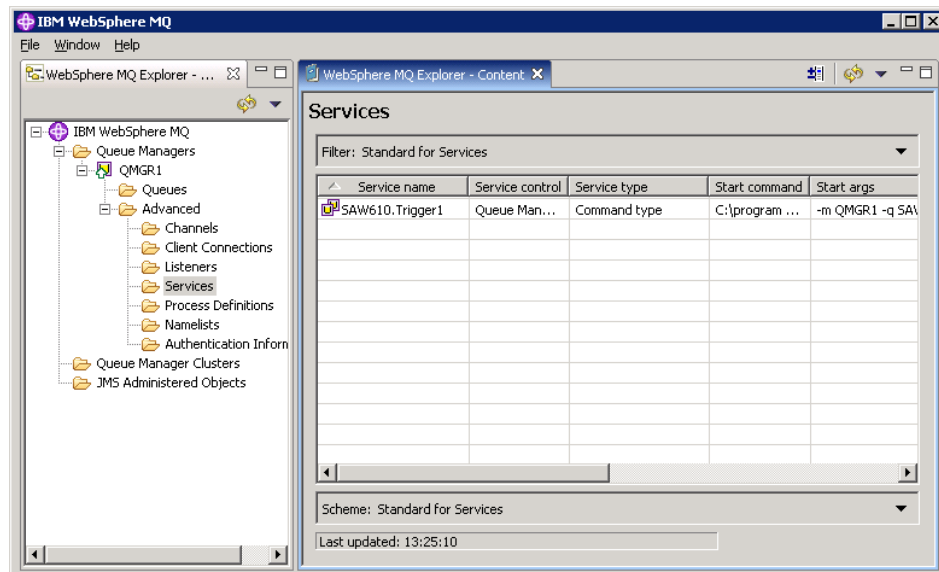
Figure 9-98   WebSphere MQ Explorer - Services window
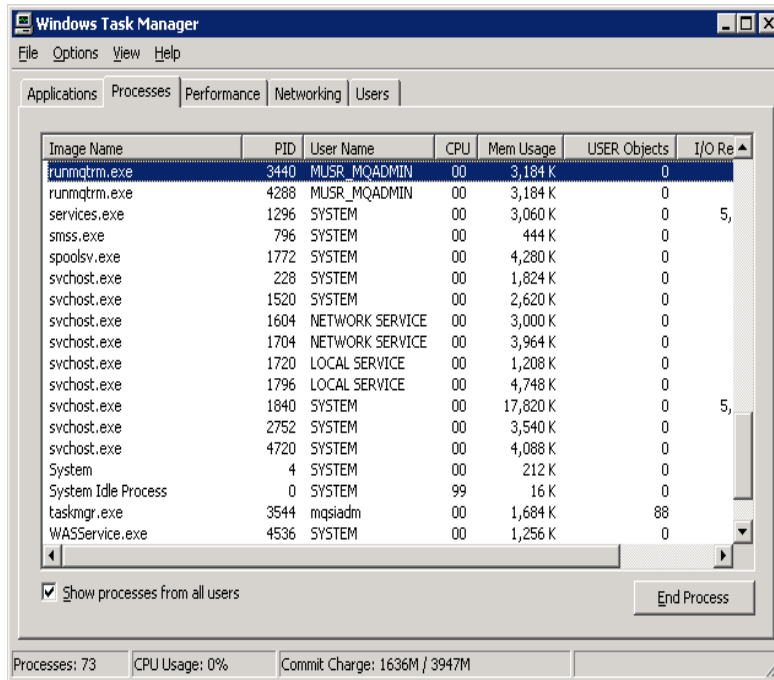
Figure 9-99   Processes window

Example 9-6   BackEnd.CMD - Back-end manufacture application simulator

```
set mqserver=SW610.CLNT/TCP/9.42.170.160(1414)
C:\SAW610\q -oSAW610.POREQ.OUT -FC:\SAW610\po_soap_legacy_ex.xml
C:\SAW610\q -ISAW610PO.REQ.TEST -dd3
C:\SAW610\q -ISAW610.INITQ -dd3
```

9.11 Testing the application

To test the mediation flow into the message flow, ensure that the following things are running:

► The WebSphere Message Broker server with the ESB configured as described in 9.3, "Runtime guidelines for ESB based on WebSphere Message Broker" on page 261.

► The WebSphere Enterprise Service Bus server instance with the ESB configured as described in 9.3, "Runtime guidelines for ESB based on WebSphere Message Broker" on page 261.

► The back end manufacturer applications are started as described in 9.10, "Runtime guidelines for back-end existing manufacturer applications" on page 315.

We provide two methods of verification testing. The first method is the use of the RFHUTILC utility from the IH03 supportpac. The second method is the use of the Test Component feature under the WebSphere Integration Developer Eclipse console.

The system verification test will route a purchase order request from the WebSphere Enterprise Service Bus retail system to the WebSphere Message Broker ESB warehouse system. We simulate a test from a MQ client with a SOAP/MQ message. To test the sample application, we use the Test Component feature under the WebSphere Integration Developer Eclipse console. Each method inserts a test message into the purchase order system.

Test SOAP/MQ with RFHUTILC

To test the SOAP/MQ messaging, on your workstation:

1. Set the system environment variable MQSERVER=SAW610.CLNT/TCP/ipaddress.

2. Open the RFHUTILC utility provided by the IH03 Supportpac (Figure 9-100).
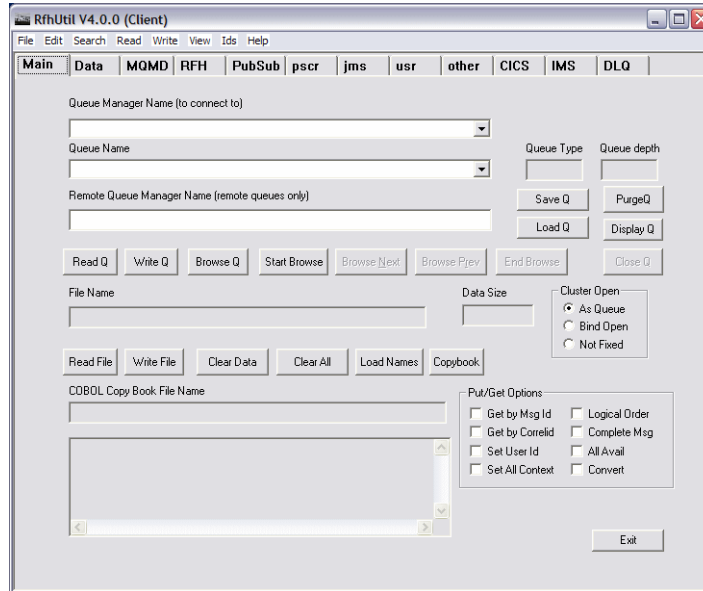
Figure 9-100   RfhUtil window

3. Enter `QMGR1` in the Queue Manager Name field and SAW610.POREQ.IN in the Queue name field. Click **Read File**.
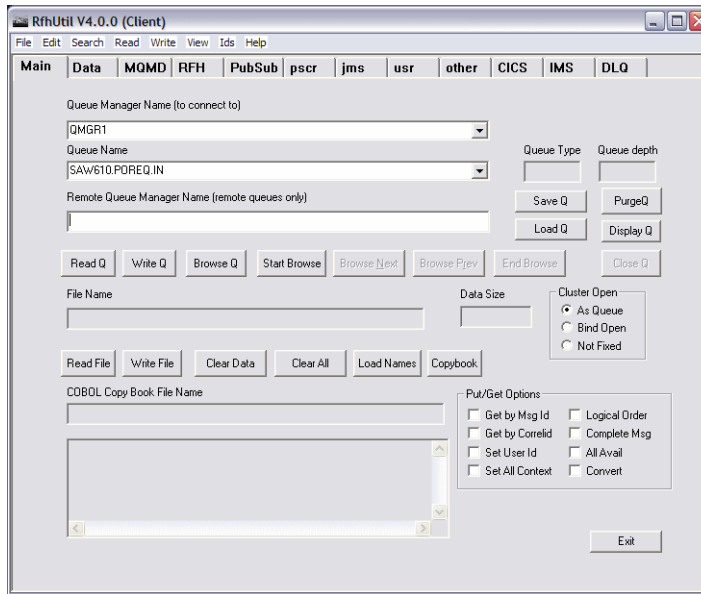
Figure 9-101   RfhUtil window

4. Select **PO_SOAP_EX.XML** and click **Open** (Figure 9-102).
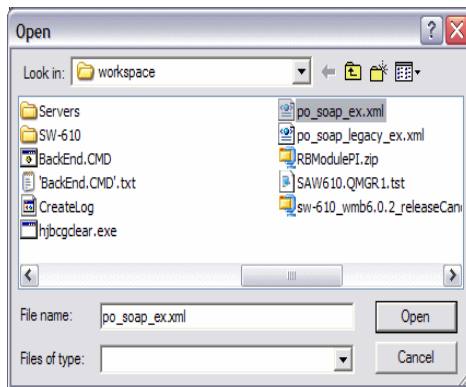
Figure 9-102   Select OP_SOAP_EX.XML window

5. Click **Write Q** to sent the message to the PO Submit message flow (Figure 9-103).
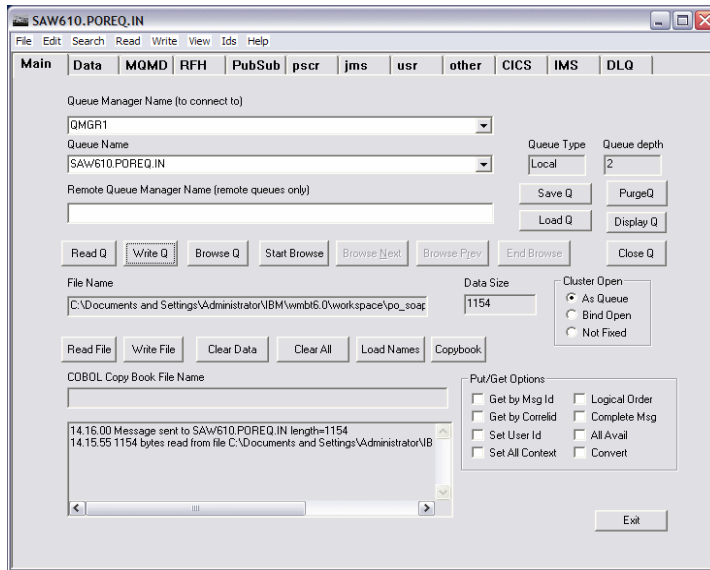
Figure 9-103   SAW610.PORLQ.IN window

6. Change the Queue from field to SAW610.POREQ.OUT and click **Read Q** (Figure 9-104).
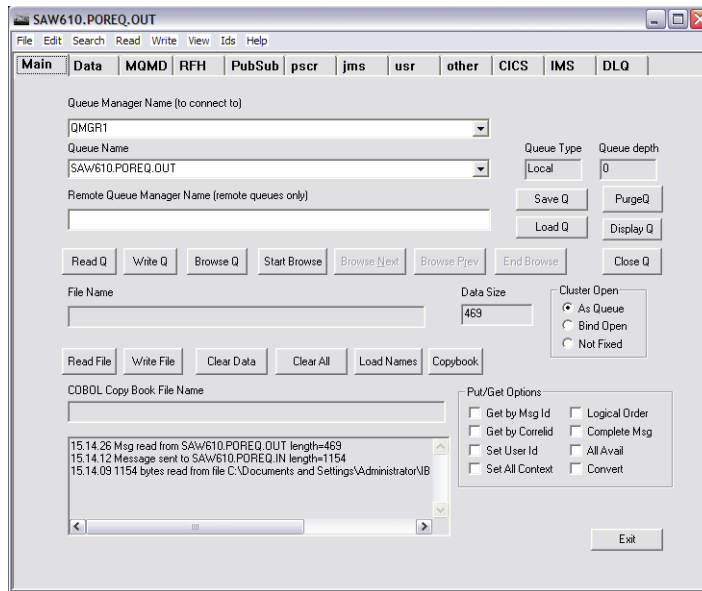
Figure 9-104   SAW610.POREQ.OUT window

7. To display the response detail, click the **Data** tab and then select **XML** for the data format (Figure 9-105).
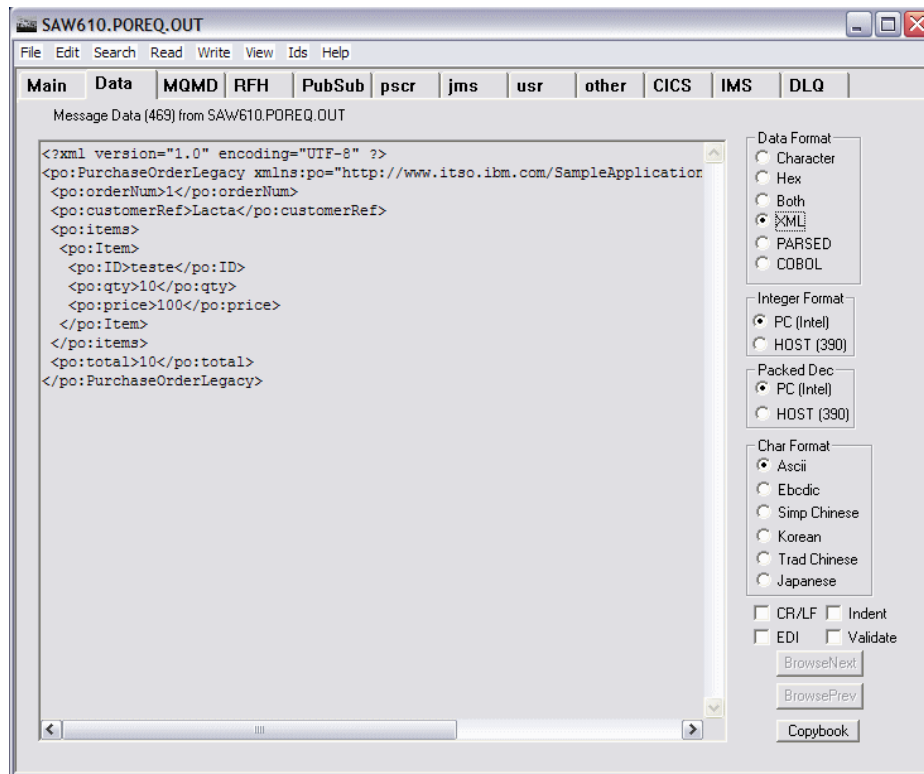
Figure 9-105   SAW610.PORLQ.OUT window
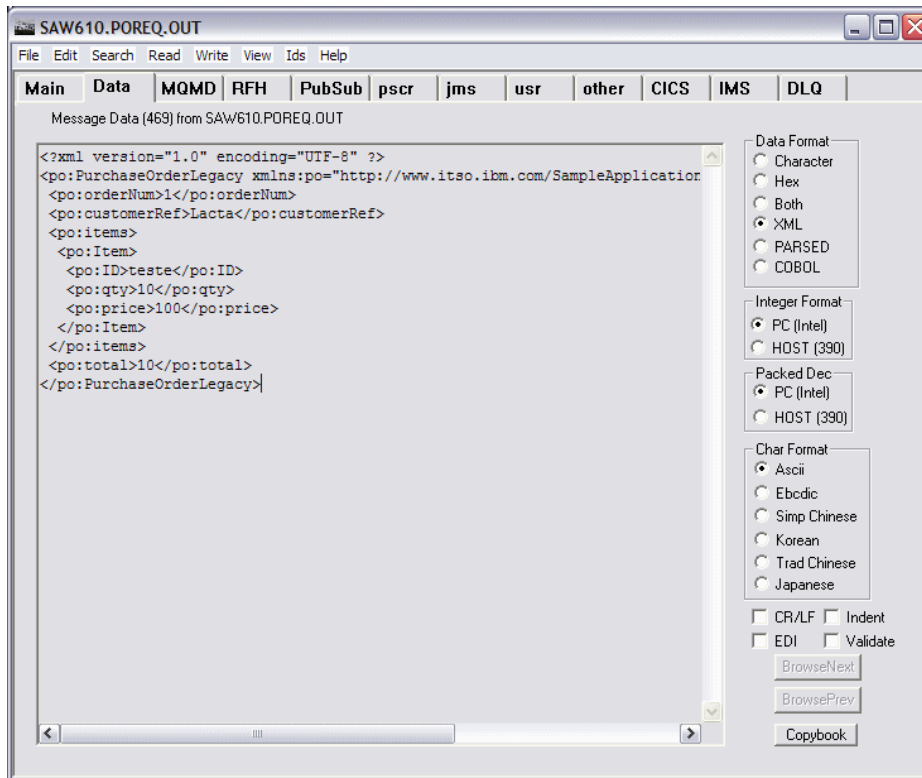
Figure 9-106 SAW610.POREQ.OUT window

You have sent a SOAP/MQ message to the WebSphere Message Broker warehouse system message flow and received a reply.

10

Scenario: DataPower in an SOA

In this chapter we discuss the scenario regarding the Web Service gateway using DataPower. In the scenario we use the Web Service endpoint GetLocationInformationByAddress. This function returns summary information about a location via an address. In the second scenario we include a basic authentication mechanism provided by DataPower.

For more information regarding DataPower refer to Chapter 7, "WebSphere DataPower appliances in SOA" on page 165.

10.1 Scenario 1: Build Web Service gateway using DataPower

First we need to check the availability of the Web service. To do this we use the curl tool (a command-line tool) for transferring files with URL syntax, supporting FTP, FTPS, TFTP, HTTP, and so on. For our example, we use the cendata.xml file. This file includes the SOAP request that is sent to the Web service. In return, the Web service successfully sends a SOAP response back to the client. Afterwards, we continue with the building of the WS-proxy gateway. In order to build the WS-proxy gateway, we need to obtain a WSDL file for the Web service. The WSDL file specifies what a Web service can do (interface specification), how to invoke it (binding specification), and where it resides (service specification). It also has a Types section, which corresponds to the type definitions, such as XML schema.

Figure 10-1 illustrates the Web service gateway using DataPower to access an external Web Service.

The content for dynamically generated XML SOAP documents from the application server can be retrieved from the Web Service Proxy.

Web Service Proxy

External Web Service

**Transformations
XML Schema validation
WS-Security**

Web Service Clients
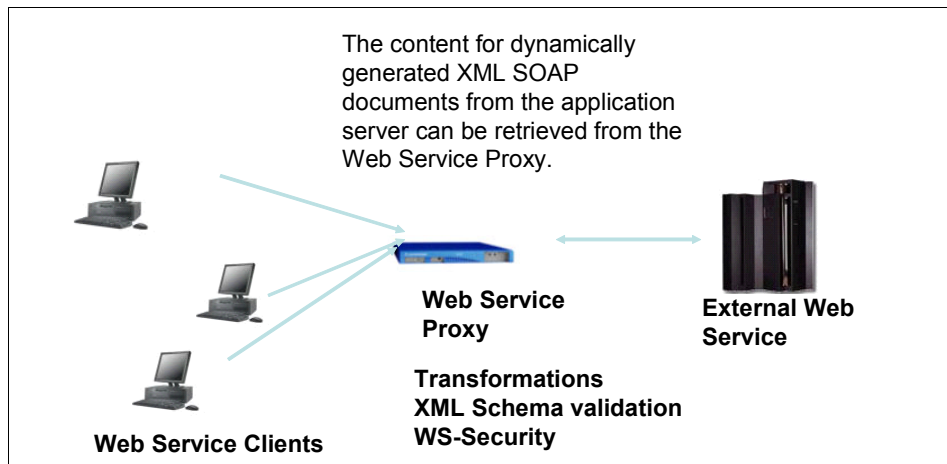
Figure 10-1   Machine topology

Perform the following steps to test and then build the Web Service gateway:

1. Test connectivity to the existing Web services using the curl tool. The curl tool can be obtained from the following Web site:

   http://curl.haxx.se/download.html

2. Use the cendata.xml file to initiate our SOAP request executed by the curl tool. Place the cendata.xml in the directory where curl is installed.

> **Note:** The cendata.xml can be retrieved from the Additional Material folder from the book link. This SOAP request xml file can be generated by tools such as Web Service Explorer in Rational Application Development, Oxegen, and so on. It has been provided here to help streamline your efforts in testing your Web services connectivity.

3. Go to the command prompt and execute the following command:

```
type cendata.xml
```

You will see a window similar to Figure 10-2. Note the content of the SOAP test message.

Figure 10-2   Viewing and executing the cendata.xml file

4. Execute at the command prompt:

```
curl --data-binary @cendata.xml -i
http://ws.cdyne.com/DemographixWS/DemographixQuery.asmx -H
"Content-type: text/xml" -H u
http://ws.cdyne.com/DemographixWS/GetLocationInformationByAddress"
```

5. After the successful Web services testing, you will obtain a WSDL file. In our example, we used the demografix.wsdl file and downloaded it from:

`http://ws.cdyne.com/DemographixWS/DemographixQuery.asmx?wsdl`

6. After downloading the WSDL to your local file system, you need to upload it to the DataPower using the File management icon. From your Web browser, access the DataPower Control Panel:

`https://<dpip>:9090`

You will see a window similar to Figure 10-3.
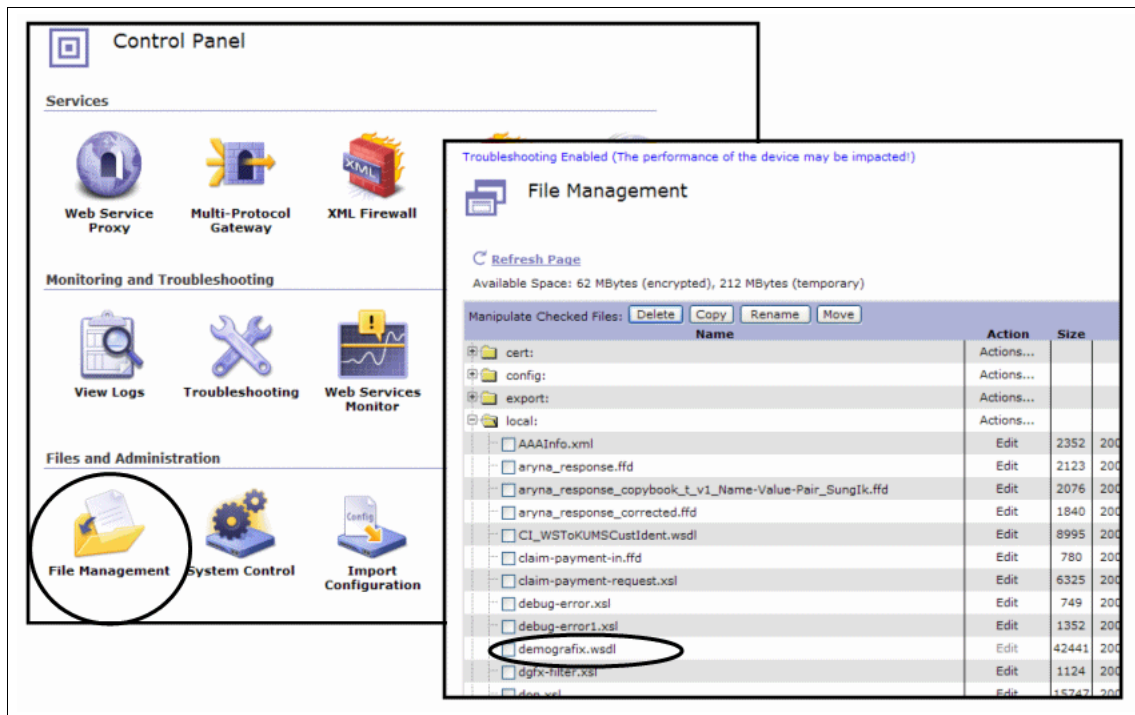
Figure 10-3   Control Panel and File Management windows

7. From the Control Panel, double-click the **File Management** icon. You will be directed to another window where you are able to upload the file to the DataPower. In Figure 10-3, the File Management window displays the file updated to the DataPower.

8. Now we are ready to build the WS Proxy. Return to the DataPower Control panel and click the **Web Service Proxy** icon (Figure 10-4).
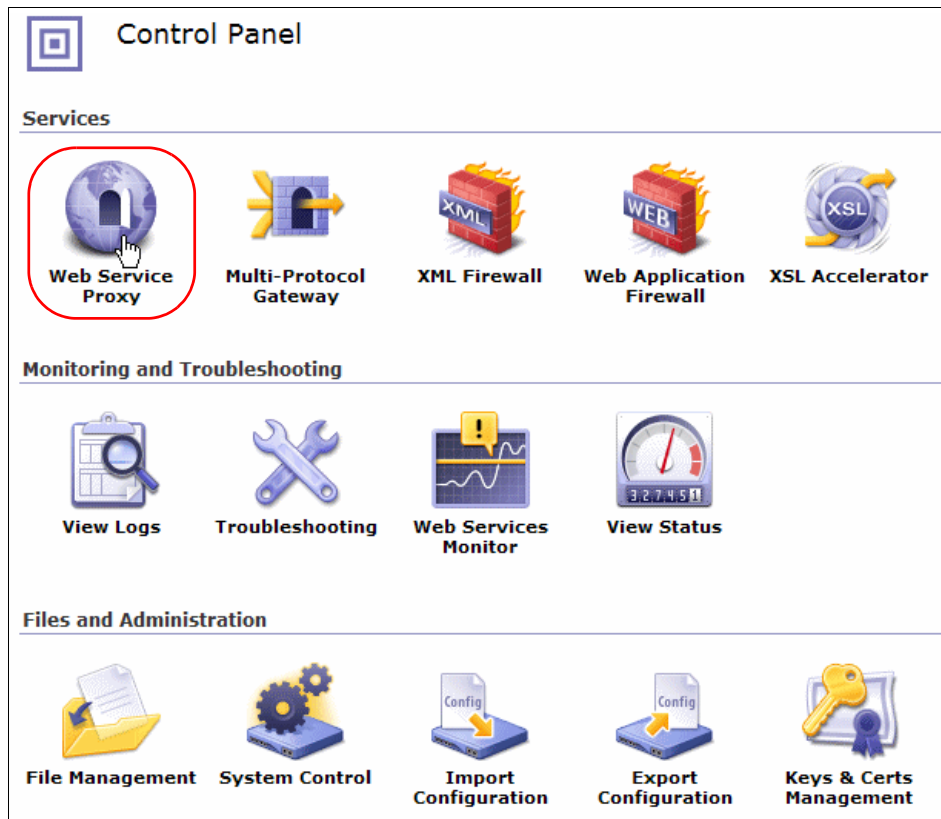
Figure 10-4   Control Panel window

9. From the Configure Web Service Proxy window (Figure 10-5), click **Add**.

Figure 10-5   Configure Web Service Proxy window

10. Enter a Web Service Proxy name and select the WSDL file from the menu (Figure 10-6).

Figure 10-6   Configure Web Service Proxy window

11.Create a local HTTP Front handler. Under Local Endpoint Handler
(Figure 10-7), click **Create a New** → **HTTP Front Side Handler**.

Figure 10-7   Configure Web Service Proxy window

12.From the Configure HTTP Front Side Handler window, enter the name and port number. In our example, we use WS_proxy-demo and 3380. See Figure 10-8.

Figure 10-8   HTTP Front Side Handler window

13.Click **Apply** (Figure 10-9).
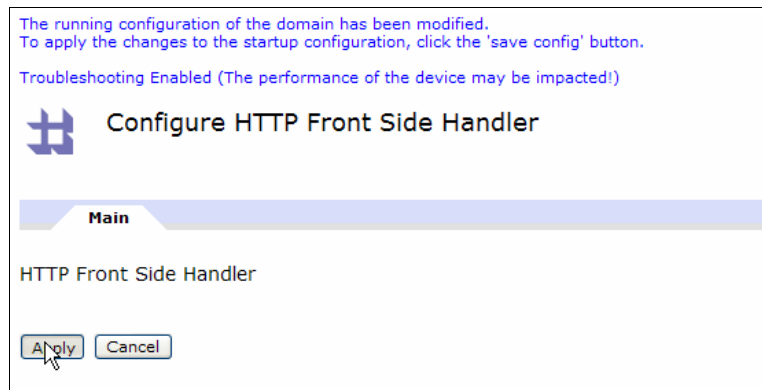
Figure 10-9   Configure HTTP Front Side Handler window

14.Click **Apply** → **Save Config**. See Figure 10-10.

Figure 10-10   Configure Web Service Proxy window

15. At this time we are ready to test the Web service using DataPower. From the command prompt, type:

```
curl --data-binary @cendata.xml -i
http://isswdatapower2.rtp.raleigh.ibm.com:3380/DemographixWS/Demogra
phixQuery.asmx -H "Content-type: text/xml" -H "SOAPAction:
http://ws.cdyne.com/DemographixWS/GetLocationInformationByAddress"
```

See Figure 10-11.

Figure 10-11   Execute curl tool to test successful build of WS-proxy gateway using DataPower

10.2 Scenario 2: Basic authentication mechanism provided by DataPower

In this section we implement the DataPower Authentication, Authorization, and Auditing (AAA) Framework features that enable the appliance to integrate flexibly with all types of access control architectures. In our example, we only use DataPower Authentication to simplify the scenario. Perform the following steps:

1. From the DataPower Control Panel, click the **Web Service Proxy** icon (Figure 10-12).
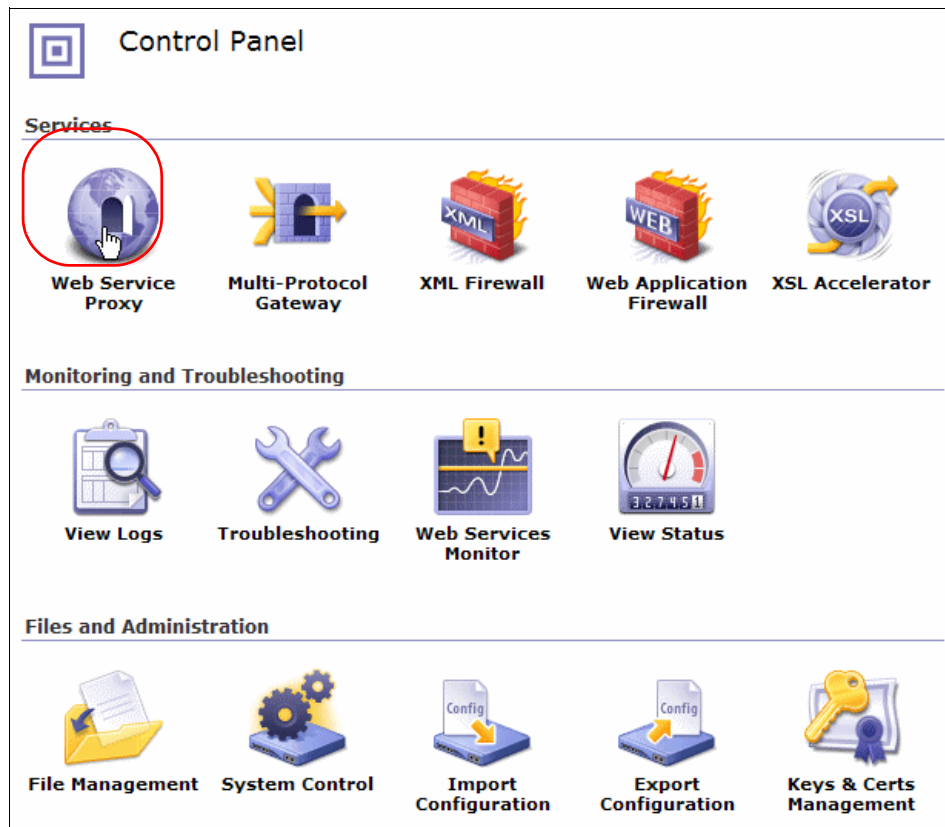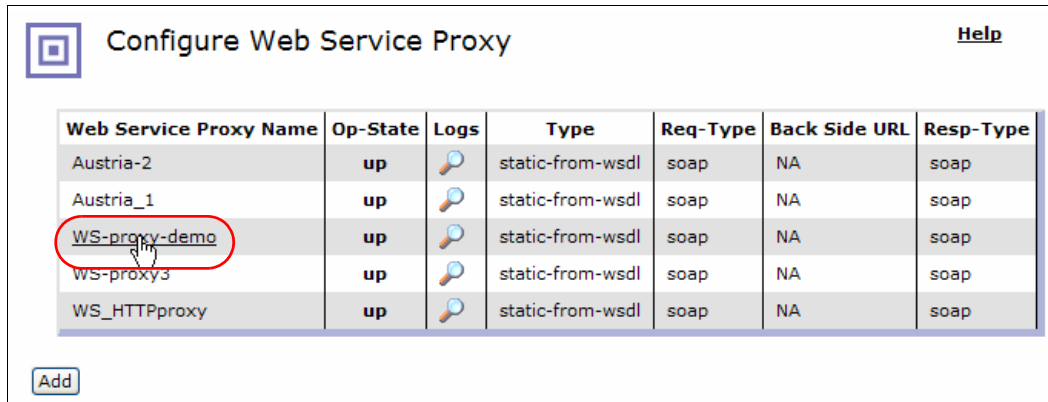
Figure 10-12   DataPower Control Panel window

2. Click the WS-proxy link that you created in the previous section. In our example, the link is WS-proxy-demo (Figure 10-13).
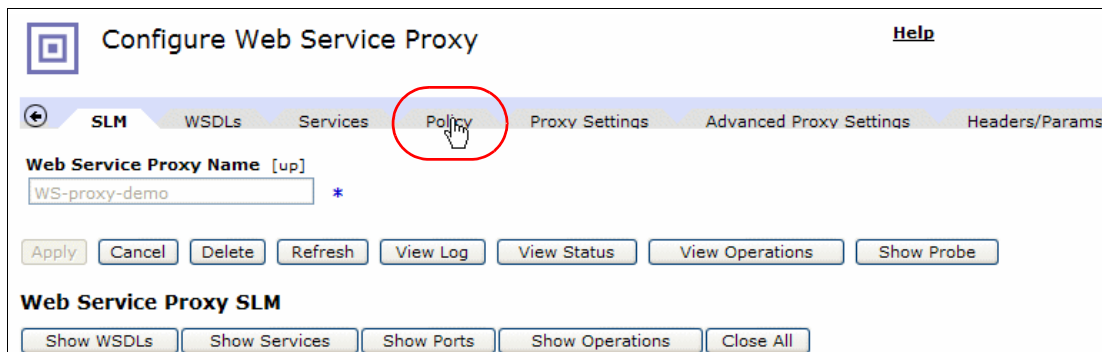
Figure 10-13   Configure Web Service Proxy window

3. Click the **Policy** tab (Figure 10-14).

Figure 10-14   Configure Web Service Proxy window

4. Click the **WS-proxy-demo default request-rule(request-rule)** link (Figure 10-15).

Figure 10-15   Web Service Proxy Policy window

5. Drag the AAA icon between ⬦ and ⬜ (Figure 10-16).
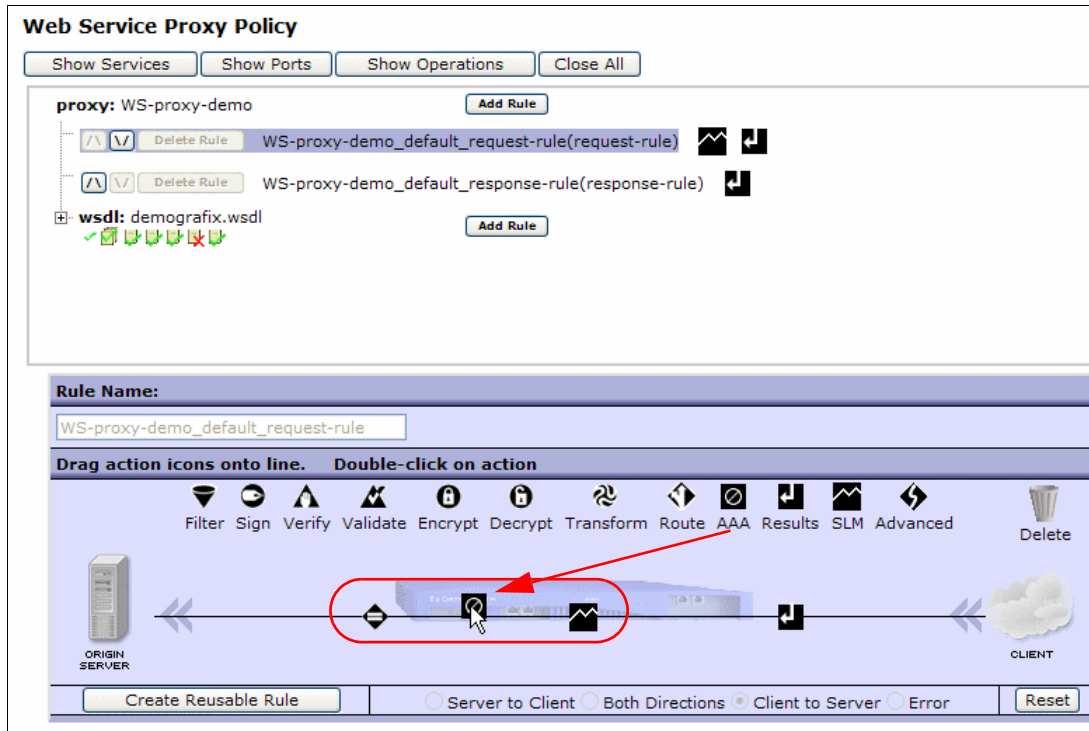
Figure 10-16   Web Service Proxy Policy window

6. You will see the AAA icon surrounded by yellow (Figure 10-17). Double-click it.
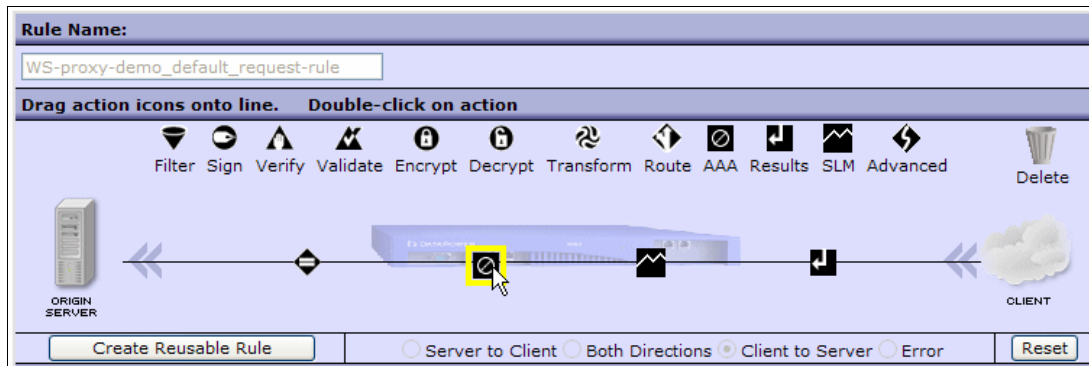
Figure 10-17   Web Service Proxy Policy window

7. The Configure AAA Action window appears. In the Input field, change (auto) to INPUT. In the AAA Policy field, change to AAAFile (Figure 10-18).
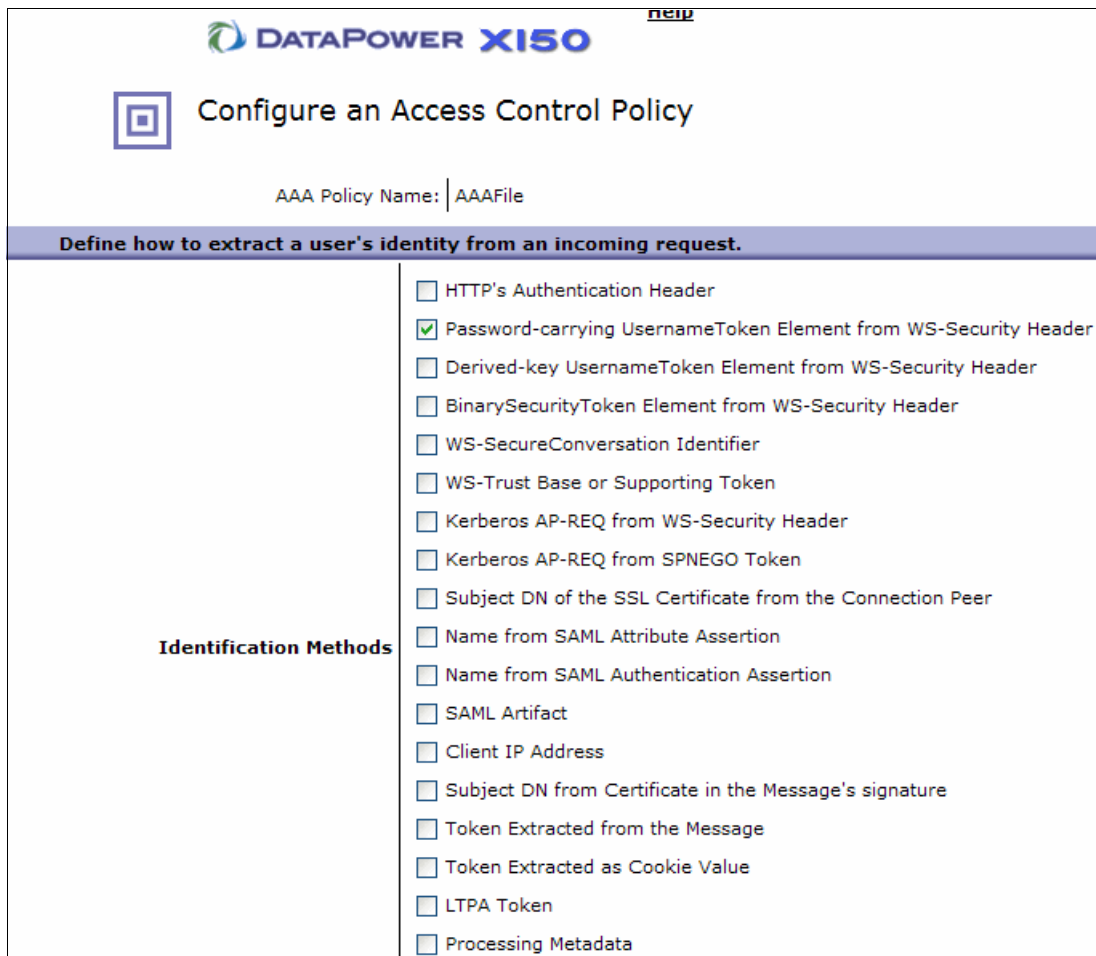
Figure 10-18   Configure AAA Action window

8. Click the **...** button next to AAAFile (Figure 10-19).

Figure 10-19   Configure AAA Action window

9. The Configure an Access Control Policy window appears. Check **Password-carrying UsernameToken Element from WS-Security Header**. Scroll down and click **Next** (Figure 10-20).

Figure 10-20   Configure an Access Control Policy window

10. Click **Next** three times and then click **Commit** (Figure 10-21).

Figure 10-21   Configure an Access Control Policy window

11. Click **Done** (Figure 10-22).

Figure 10-22   Successful creation of Access Control Policy AAAFile window

12. From your Configure AAA Action window (Figure 10-23), click **Done**.

Figure 10-23   Configure AAA Action window

13. Click **Apply** → **Save Config** (Figure 10-24).

Figure 10-24   Web Service Proxy window

14. We now modify the AAAFile to insert a valid user ID and password. From the DataPower Control Panel (Figure 10-25), click the **File Management** icon.
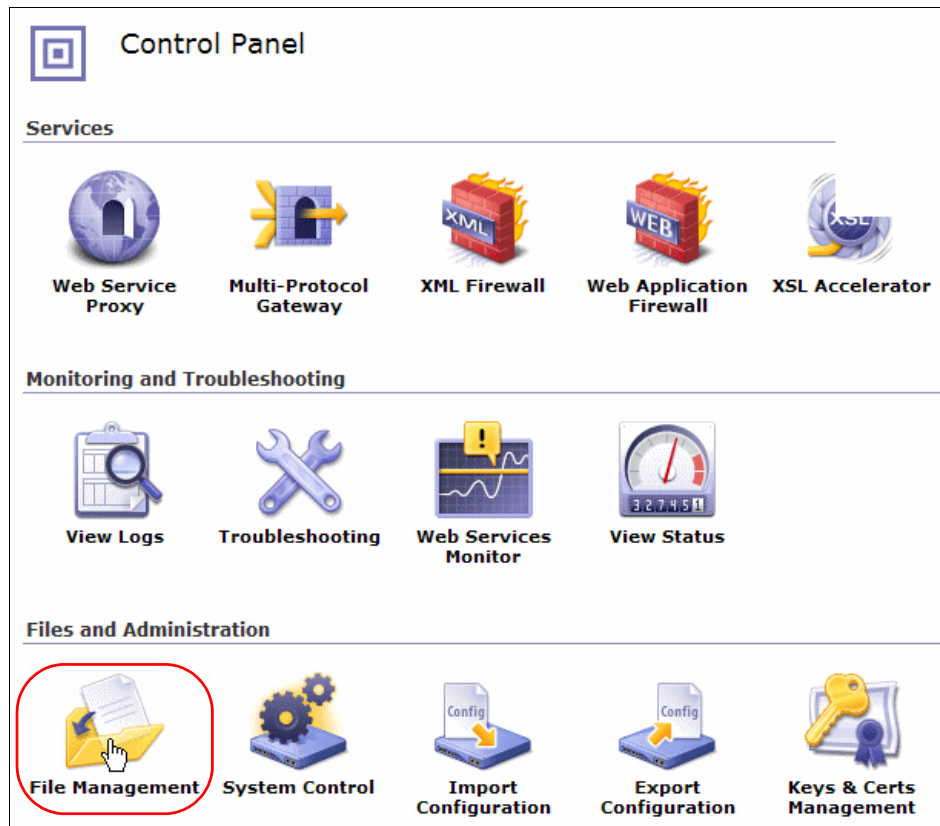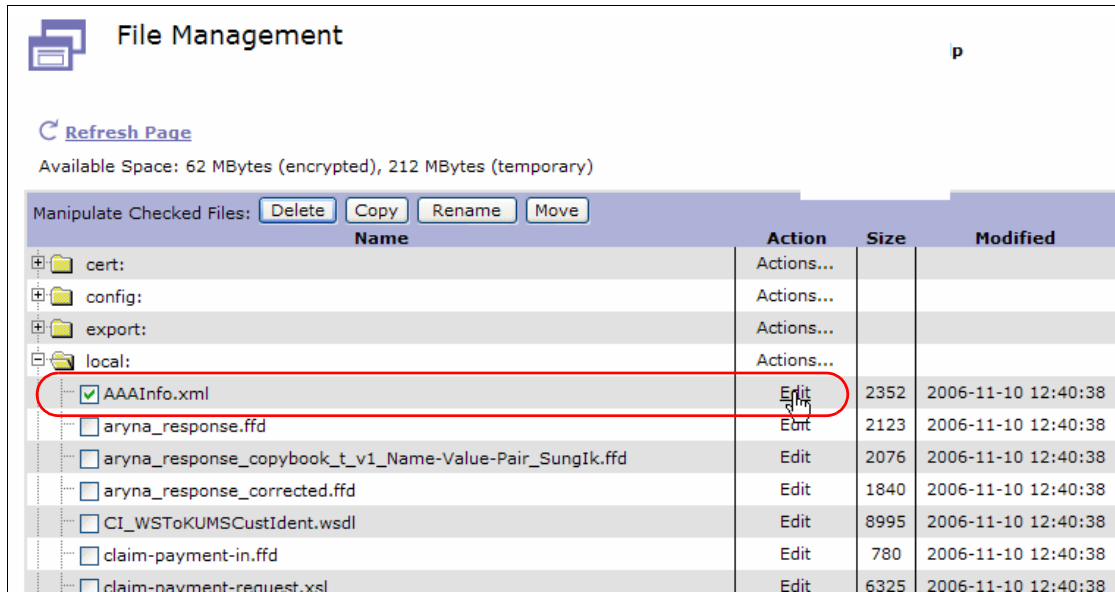
Figure 10-25   DataPower Control Panel window

15.Navigate to **local** → **AAAInfo.xml** and then click **Edit** (Figure 10-26).

Figure 10-26   File Management window

16. Add the following lines into the AAAInfo.xml file (Figure 10-27):

```
<Authenticate>
    <Username>sungik</Username>
    <Password>purdue</Password>
    <OutputCredential>admin</OutputCredential>
</Authenticate>
```

17. Click **Submit**.
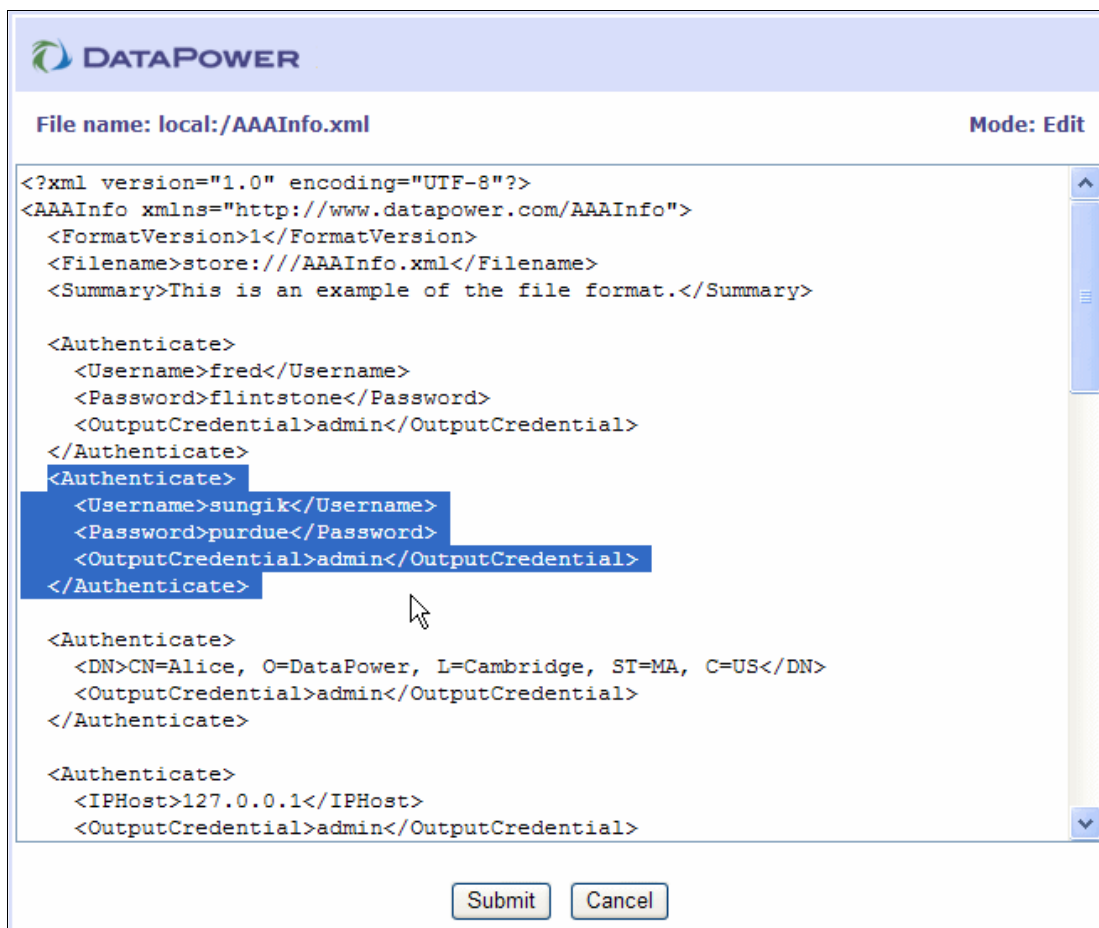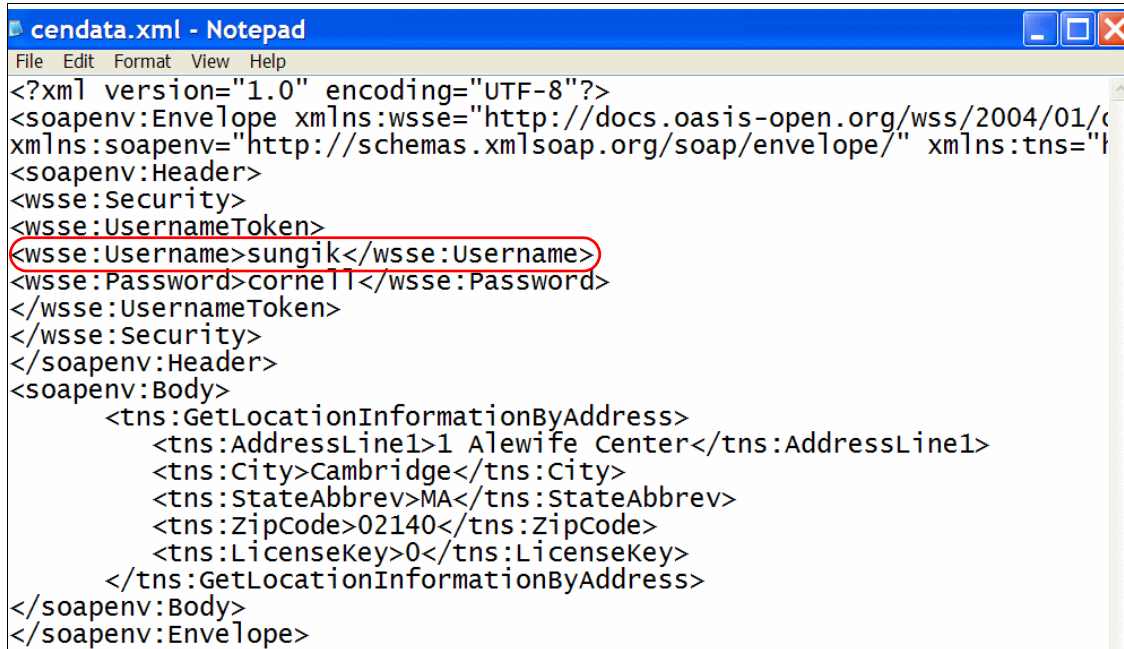
Figure 10-27   File name: local:/AAAInfo.xml window

18. Modify the password value of the cendata.xml. Change to the value other than purdue (Figure 10-28). In our example, we changed to cornell.

Figure 10-28   cendata.xml window

19. Test using curl. From a command prompt, type and execute the following command:

```
curl --data-binary @cendata.xml -i
http://isswdatapower2.rtp.raleigh.ibm.com:3380/DemographixWS/Demogra
phixQuery.asmx -H "Content-type: text/xml" -H "SOAPAction:
http://ws.cdyne.com/DemographixWS/GetLocationInformationByAddress"
```

We see that the request was rejected by the AAA policy (Figure 10-29).

Figure 10-29   First curl test - failure window

20. Change back to purdue for the password. Test it again using curl (Figure 10-30). You will notice that curl executed successfully.

Figure 10-30   Second curl test - successful window

10.3 How to create a Domain

In this section we discuss how to create a domain called LocalDomain1 on DataPower.

1. Login to the DataPower X150 console.

Figure 10-31   DataPower X150 Console window

2. From the Control Panel, click **Administration** → **Configuration** → **Application Domain** (Figure 10-32).
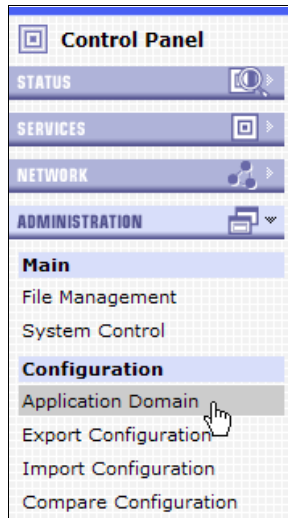
Figure 10-32   Control Panel window

3. Click **Add**.

4. Enter domain name `LocalDomain1` and comments `Local Domain`. Click **Apply** (Figure 10-33).

Figure 10-33   Application Domain window

5. Click **Save Config** (Figure 10-34).

Figure 10-34   Dialog window

Now you will see the newly created domain (Figure 10-35).

| LocalDomain1 | saved | **up** | 🔍 | enabled | Local Domain |
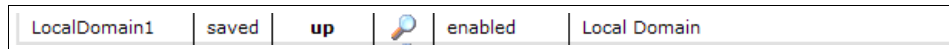
Figure 10-35   Domain created successfully window

Figure 10-36   DataPower X150 Console window

A

Java node source code

Example: A-1 WarehouseSubmitPOTransform.java

```java
package com.ibm.itsoral.scenario;

import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbOutputTerminal;
import com.ibm.broker.plugin.MbRFH2C;
import com.ibm.itsoral.scenario.po.SOAPMessageHelper;

/**
 * <p> Transform the PO request message. </p>
 *
 * <p> Creation date Dec 5, 2006
 * @author Joao Batista De Los Rios
 * @version SAW610 1.0
 */
public class WarehouseSubmitPOTransform extends MbJavaComputeNode
{

    public void evaluate(MbMessageAssembly inAssembly) throws
MbException
```

353

```
    {
        MbOutputTerminal out = getOutputTerminal("out");
      MbOutputTerminal alt = getOutputTerminal("alternate");

      MbMessage msgEnv= new
MbMessage(inAssembly.getLocalEnvironment());
      MbMessage inMessage = inAssembly.getMessage();

      // create a empty message //
      MbMessage outMessage = new MbMessage();
      try
      {
          // Copy the headers //
          SOAPMessageHelper.copyHeaders(inMessage, outMessage);

          // Define the element variables to be used in this message //
          MbElement envRoot =
inAssembly.getGlobalEnvironment().getRootElement();
          MbElement inRoot  = inMessage.getRootElement();
          MbElement outRoot = outMessage.getRootElement();


          // Save the Reply destination information so that it
          // can be reinstated by the PurchaseOrderResponse
         // message flow once the Legacy Manufacturer has
          // responded to the WBIMB ESB. Only used with MQ requests
          MbElement mqmd = inRoot.getFirstElementByPath("MQMD");
          // Verifies the MQMD headers for dynamic routing later on //
          if (SOAPMessageHelper.verifyMQMDHeaders(mqmd, outRoot)) //
Could be a message from a HTTP request
          {
              MbElement reply2Queue =
mqmd.getFirstElementByPath("ReplyToQ");
              MbElement reply2QM   =
mqmd.getFirstElementByPath("ReplyToQMgr");

              MbElement usr = outRoot.getLastChild()
              .createElementAfter(MbRFH2C.PARSER_NAME)
                .createElementAsFirstChild(MbElement.TYPE_NAME, "usr",
null);
              usr.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
"FinalReplyToQ", reply2Queue.getValue());
              usr.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
"FinalReplyToQMgr", reply2QM.getValue());
          }
```

```
            else // HTTP
            // Save the HTTP request in the global env //
               SOAPMessageHelper.saveHTTPRequest(inRoot, envRoot);

            // Saves the SOAP Operations rotines if they dont exist //
            SOAPMessageHelper.resetSoapHeader(envRoot);
            // Save some SOAP call references that can be used later on
            SOAPMessageHelper.initService(envRoot,
                         "Intermediary",

"http://www.ws-i.org/SampleApplications/SupplyChainManagement/",
                         "PurchaseOrderRequest");
            // Save the valid headers to limit the operations later on
            SOAPMessageHelper.setValidHeaders(envRoot, "PurchaseOrder");
            SOAPMessageHelper.setValidHeaders(envRoot, "Configuration");
            SOAPMessageHelper.setValidHeaders(envRoot, "StartHeader");
            // Decode the SOAP message to be used by the PO Legacy system

SOAPMessageHelper.decodeSOAPMessage(inRoot.getFirstElementByPath("MRM")
,
                              outRoot, envRoot, "PurchaseOrder");
            ////////////////////////////////

            // Create the out Assembly //
            MbMessageAssembly outAssembly = new
MbMessageAssembly(inAssembly, outMessage);
            // Propagate the message //
            out.propagate(outAssembly);
        }
        finally
        {
            // clear the outMessage
            outMessage.clearMessage();
        }
    }
}
```

Example: A-2   POFormatSOAPMessage.java

```java
/*
 * Created on Dec 5, 2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.ibm.itsoral.scenario;

import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbOutputTerminal;
import com.ibm.itsoral.scenario.po.SOAPMessageHelper;

/**
 * <p> Create the PO message to be processed by the legacy system. </p>
 * <p> Pure XML messages. </p>
 *
 * <p> Creation date Dec 5, 2006
 * @author Joao Batista De Los Rios
 * @version SAW610 1.0
 */
public class POFormatSOAPMessage extends MbJavaComputeNode
{
    /**
     * <p> The QUEUE is hardcoded, should be loaded dinamically in a
real world. </p>
     */
    protected static final String QUEUE_DEST = "SAW610PO.REQ.TEST";

    /**
     * @see
com.ibm.broker.javacompute.MbJavaComputeNode#evaluate(com.ibm.broker.pl
ugin.MbMessageAssembly)
     */
    public void evaluate(MbMessageAssembly inAssembly) throws
MbException
    {
        MbOutputTerminal out = getOutputTerminal("out");
       MbOutputTerminal alt = getOutputTerminal("alternate");
```

```java
        MbMessage inMessage = inAssembly.getMessage();

    // create a empty message //
    MbMessage outMessage = new MbMessage();
    try
    {
        MbElement inRoot = inMessage.getRootElement();
        MbElement localEnv =
inAssembly.getLocalEnvironment().getRootElement();
        // Copy the headers //
        SOAPMessageHelper.copyHeaders(inMessage, outMessage);

        // Even now the SOAP has been removed, the
        // message needs transforming into the format expected
       // by the Legacy Manufacturer
        MbElement outRoot = outMessage.getRootElement();
        MbElement mrm = outRoot.createElementAsLastChild("MRM");
        MbElement mrmIn = inRoot.getLastChild();
        MbElement sdata = mrmIn.getFirstChild(); // ServiceData
        MbElement po= sdata.getFirstChild(); // PurchaseOrder
        boolean end=false;
        do
        {
            String poName = po.getName();
            if (po!=null && (end=(poName.equals("PurchaseOrder"))))
            {
               // Create the elements //
               MbElement poL =
                   mrm.createElementAsLastChild(MbElement.TYPE_NAME,
"PurchaseOrderLegacy", null);// Top level
               poL.setNamespace(m_po_ns);

               MbElement innerElements;
               // Creates the Message Parts //
               if ((innerElements = po.getFirstChild())!=null)
               {
                   MbElement _e =
poL.createElementAsLastChild(MbElement.TYPE_NAME, "orderNum",
innerElements.getValue());
                   _e.setNamespace(m_po_ns);
               }
               if
((innerElements=innerElements.getNextSibling())!=null)
               {
```

```
                        MbElement _e =
poL.createElementAsLastChild(MbElement.TYPE_NAME, "customerRef",
innerElements.getValue());
                        _e.setNamespace(m_po_ns);
                    }
                    if
((innerElements=innerElements.getNextSibling())!=null)
                    {
                        MbElement _e =
poL.createElementAsLastChild(MbElement.TYPE_NAME, "items", null);
                        _e.copyElementTree(innerElements);
                        _e.setNamespace(m_po_ns);
                    }
                    if
((innerElements=innerElements.getNextSibling())!=null);
                    {
                        MbElement _e =
poL.createElementAsLastChild(MbElement.TYPE_NAME, "total",
innerElements.getValue());
                        _e.setNamespace(m_po_ns);
                    }
                }
            }
        while ((po=sdata.getNextSibling())!=null&&!end);

        // TODO: Examine the Service Directory, to locate which
Manufacturer
        // and set the destination list accordingly.
        // The role of a naming directory for the WBIMB ESB can be
assumed by a DB2
        // Reset the message name now that the SOAP Envelope has been
removed
        MbElement props =
outRoot.getFirstElementByPath("Properties");
        props.getFirstElementByPath("MessageFormat").setValue("MRM");
// ms_Manufacturer

props.getFirstElementByPath("MessageSet").setValue("G71ASSS002001"); //
ms_Manufacturer

props.getFirstElementByPath("MessageType").setValue("PurchaseOrderLegac
y"); // XSD Formatter
        props.getFirstElementByPath("MessageFormat").setValue("XML");

        // Create the "out" Assembly //
```

```
            MbMessageAssembly outAssembly = new
MbMessageAssembly(inAssembly, outMessage);
            // Propagate the message //
            out.propagate(outAssembly);
        }
        finally
        {
            // clear the outMessage
            outMessage.clearMessage();
        }
    }

    final private String m_man_ns =
"http://www.itso.ibm.com/SampleApplications/Manufacturer/Manufacturer.x
sd";
    final private String m_po_ns  =
"http://www.itso.ibm.com/SampleApplications/Manufacturer/ManufacturerPO
_Legacy.xsd";
}
```

Example: A-3   SOAPFaultHandler.java

```
package com.ibm.itsoral.scenario;

import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbOutputTerminal;
import com.ibm.itsoral.scenario.po.SOAPMessageHelper;

/**
 * <p> Create the error in a SOAP envelope format. </p>
 *
 * <p> Creation Date Dec 7, 2006
 * @author Joao Batista De Los Rios
 * @version 1.0
 */
public class SOAPFaultHandler extends MbJavaComputeNode
{

    /**
     * @see
com.ibm.broker.javacompute.MbJavaComputeNode#evaluate(com.ibm.broker.pl
ugin.MbMessageAssembly)
     */
    public void evaluate(MbMessageAssembly inAssembly) throws
MbException
    {
        MbOutputTerminal out = getOutputTerminal("out");
      MbOutputTerminal alt = getOutputTerminal("alternate");

      MbMessage msgEnv= new
MbMessage(inAssembly.getLocalEnvironment());
      MbMessage msgEx= new MbMessage(inAssembly.getExceptionList());
      MbMessage inMessage = inAssembly.getMessage();

      // create a empty message //
      MbMessage outMessage = new MbMessage();
      try
      {
          MbElement env    = msgEnv.getRootElement();
          MbElement exRoot  = msgEx.getRootElement(); // Exception list
          MbElement inRoot  = inMessage.getRootElement();
```

```
            MbElement outRoot = outMessage.getRootElement();

            String name = exRoot.getFirstChild().getName();
            String val =  (String) exRoot.getFirstChild().getValue();
            // Add the HTTP support if it was a HTTP request //
            if (SOAPMessageHelper.isHTTPRequest(inRoot))
            {
            SOAPMessageHelper.copyHeaders(inMessage, outMessage, new
String[]{"MQMD"});
            SOAPMessageHelper.addHTTPRequest(outRoot, env);
            }
            else SOAPMessageHelper.copyHeaders(inMessage, outMessage);

            // Creates the message structure //
            MbElement mrm = outRoot.getFirstElementByPath("MRM");
            if (mrm==null) mrm=outRoot.createElementAsLastChild("MRM");

            // PO Response ... as simple as that! :-)
            String msgErr = "MalformedOrder";
            // Create the Soap FAULT Codes //
            SOAPMessageHelper.formatPOFaultMessage(mrm, env, exRoot,
msgErr);

            // Defines the parsing properties of the message //
            MbElement props = outRoot.getFirstChild(); // Properties

props.getFirstElementByPath("MessageSet").setValue("G71ASSS002001");

props.getFirstElementByPath("MessageType").setValue("Envelope");
        props.getFirstElementByPath("MessageFormat").setValue("XML");

            // Create the "out" Assembly //
            MbMessageAssembly outAssembly = new
MbMessageAssembly(inAssembly, outMessage);
        // Propagate the message to the right place - Dinamically
route
        MbOutputTerminal terminal =
            SOAPMessageHelper.selectOutputTerminal(inRoot, out, alt);
        terminal.propagate(outAssembly);
    }
    finally
    {
       // clear the outMessage
       outMessage.clearMessage();
    }
```

```
        }

}
```

Example: A-4   POTransformReply.java

```
package com.ibm.itsoral.scenario;

import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbOutputTerminal;
import com.ibm.itsoral.scenario.po.SOAPMessageHelper;

/**
 * <p> Transform the message to the SOAP format again, to be retorned
to the requester. </p>
 *
 * <p> Creation date Dec 5, 2006
 * @author Joao Batista De Los Rios
 * @version SAW610 1.0
 */
public class POTransformReply extends MbJavaComputeNode
{

    /**
     * @see
com.ibm.broker.javacompute.MbJavaComputeNode#evaluate(com.ibm.broker.pl
ugin.MbMessageAssembly)
     */
    public void evaluate(MbMessageAssembly inAssembly) throws
MbException
    {
         MbOutputTerminal out = getOutputTerminal("out");
       MbOutputTerminal alt = getOutputTerminal("alternate");

       MbMessage msgEnv= new
MbMessage(inAssembly.getGlobalEnvironment());
       MbMessage inMessage = inAssembly.getMessage();

       // create a empty message //
```

```
      MbMessage outMessage = new MbMessage();
      try
      {
         MbElement env    = msgEnv.getRootElement();
          MbElement inRoot  = inMessage.getRootElement();
          MbElement outRoot = outMessage.getRootElement();

          // Add the HTTP support for the message //
          if (SOAPMessageHelper.isHTTPRequest(inRoot)) // The request
can from HTTP?
          {
              // Dont copy the MQMD header for the SOAP over HTTP reply
          SOAPMessageHelper.copyHeaders(inMessage, outMessage, new
String[]{"MQMD"});
          SOAPMessageHelper.addHTTPRequest(outRoot, env); // Adds the
HTTP Reply
          }
          else SOAPMessageHelper.copyHeaders(inMessage, outMessage); //
Just copy the headers :-)

          // Creates the acknowledge PO SOAP Message //
          SOAPMessageHelper.encodeAckPOSOAPMessage(inRoot, outRoot,
env);
          // Define the MessageSet to create the message in a SOAP
envelope format
          MbElement props =
outRoot.getFirstElementByPath("Properties");

props.getFirstElementByPath("MessageSet").setValue("G71ASSS002001");

props.getFirstElementByPath("MessageType").setValue("Envelope");// If
not using SOAP: ackPO
          props.getFirstElementByPath("MessageFormat").setValue("XML");

          // Create the "out" Assembly and get out of here //
          MbMessageAssembly outAssembly = new
MbMessageAssembly(inAssembly, outMessage);

          // Propagate the message to the right place (HTTP or MQ) -
Dinamic routing //
          MbOutputTerminal terminal =
SOAPMessageHelper.selectOutputTerminal(inRoot, alt, out);
          terminal.propagate(outAssembly);
      }
      finally
```

```
      {
          // clear the outMessage
          outMessage.clearMessage();
      }
    }

}
```

Example: A-5   POLogEvent.java

```
/*
 * Created on Dec 5, 2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.ibm.itsoral.scenario;

import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbOutputTerminal;
import com.ibm.itsoral.scenario.po.SOAPMessageHelper;

/**
 * <p> TODO: Implement Comment here. </p>
 *
 * <p> Creation date Dec 5, 2006
 * @author root
 * @version SAW610 1.0
 */
public class POLogEvent extends MbJavaComputeNode
{

    /**
     * @see
com.ibm.broker.javacompute.MbJavaComputeNode#evaluate(com.ibm.broker.pl
ugin.MbMessageAssembly)
     */
    public void evaluate(MbMessageAssembly inAssembly) throws
MbException
```

```
    {
        MbOutputTerminal out = getOutputTerminal("out");
      MbOutputTerminal alt = getOutputTerminal("alternate");

      MbMessage msgEnv= new
MbMessage(inAssembly.getLocalEnvironment());
      MbMessage inMessage = inAssembly.getMessage();

      // create a empty message //
      MbMessage outMessage = new MbMessage();
      try
      {
          SOAPMessageHelper.copyHeaders(inMessage, outMessage);

          MbElement env     = msgEnv.getRootElement();
          MbElement inRoot  = inMessage.getRootElement();
          MbElement outRoot = outMessage.getRootElement();

          // Clear the service parameters from any previous requests
          SOAPMessageHelper.resetSoapHeader(env);
          // Takes references to InputRoot and constructs
          // an OutputRoot with SOAP envelope
          SOAPMessageHelper.encodeLogSOAPMessage(inRoot, outRoot,

inAssembly.getGlobalEnvironment().getRootElement());

          // Defines the parsing properties of the message //
          MbElement props = outRoot.getFirstChild(); // Properties
          String name = props.getName();

props.getFirstElementByPath("MessageSet").setValue("G71ASSS002001"); //
ms_Manufacturer (HGDCJ2C002001)

props.getFirstElementByPath("MessageType").setValue("Envelope");
        props.getFirstElementByPath("MessageFormat").setValue("XML");

          // Create the "out" Assembly //
          MbMessageAssembly outAssembly =
              new MbMessageAssembly(inAssembly, outMessage);
          // Propagate the message //
          out.propagate(outAssembly);
      }
      finally
      {
          // clear the outMessage
```

```
            outMessage.clearMessage();
        }
    }

}
```

Example: A-6   WarehouseSubmitSN.java

```
/*
 * Created on Dec 10, 2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.ibm.itsoral.scenario;

import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.*;

/**
 * <p> TODO: Implement Comment here. </p>
 *
 * <p> Creation date Dec 10, 2006
 * @author root
 * @version SAW610 1.0
 */
public class WarehouseSubmitSN extends MbJavaComputeNode
{
    public final String m_soap11ns =
"http://schemas.xmlsoap.org/soap/envelope/";
    public final String m_configns =
"http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/C
onfiguration.xsd";
    public final String m_callbackns =
"http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/M
anufacturer/CallBack";
    public final String m_shipns =
"http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/M
anufacturerSN.xsd";

    public void evaluate(MbMessageAssembly inAssembly) throws
MbException
    {
```

```java
        MbOutputTerminal out = getOutputTerminal("out");
        MbOutputTerminal alt = getOutputTerminal("alternate");

        MbMessage inMessage = inAssembly.getMessage();

        // create new message
        MbMessage outMessage = new MbMessage();
        MbMessageAssembly outAssembly =
            new MbMessageAssembly(inAssembly, outMessage);

        try
        {
            copyHeaders(inMessage, outMessage);

            // Get the elements //
            MbElement inRoot  = inMessage.getRootElement();
            MbElement outRoot = outMessage.getRootElement();

            // Creates the message parts //
            MbElement mrm = outRoot.createElementAsLastChild("MRM");

            /////////////////////////////////////////////
            // Creates the Header parts
            MbElement header =
mrm.createElementAsLastChild(MbElement.TYPE_NAME, "Header", null);
            header.setNamespace(m_soap11ns);
            // Callback
            MbElement callback =
header.createElementAsLastChild(MbElement.TYPE_NAME, "CallbackHeader",
null);
            callback.setNamespace(m_callbackns);
            MbElement convId =
                callback.createElementAsLastChild(MbElement.TYPE_NAME,
"conversationID", String.valueOf(hashCode()));
            convId.setNamespace(m_callbackns);
            // Configuration
            MbElement config =
header.createElementAsLastChild(MbElement.TYPE_NAME, "Configuration",
null);
            config.setNamespace(m_configns);
            // Defines the first attribute //
            MbElement mustUnderstand =

config.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
"mustUnderstand", "false");
```

```
                mustUnderstand.setNamespace(m_soap11ns);
                MbElement userId =
config.createElementAsLastChild(MbElement.TYPE_NAME, "UserId",
"Admin"); // User ID fixed

                ////////////////////////////////////////////
                // Creates the Body parts //
                MbElement body =
mrm.createElementAsLastChild(MbElement.TYPE_NAME, "Body", null);
                body.setNamespace(m_soap11ns);
                // Creates the body parts //
                MbElement mrmIn = inRoot.getLastChild(); // MRM
                transformMessageParts(mrmIn, body);

                // The following should only be changed
                // if not propagating message to the 'out' terminal
                out.propagate(outAssembly);

        }
        finally
        {
                // clear the outMessage
                outMessage.clearMessage();
        }
    }

    protected void transformMessageParts(MbElement mrmIn, MbElement
body)
    throws MbException
    {
        MbElement sn = mrmIn.getFirstChild();
        do
        {
            // Base tag
            if (sn!=null)
            {
                MbElement snOut =
body.createElementAsLastChild(MbElement.TYPE_NAME, sn.getName(), null);
                snOut.setNamespace(m_shipns);
                snOut.copyElementTree(sn);
                // Defines the NS for every element in the tree
                addNS(snOut , m_shipns);
            }
        }
        while ((sn=sn.getNextSibling())!=null);
```

```java
    }
    private void addNS(MbElement sn, String ns)
    throws MbException
    {
        // Just to remember: <xs:sequence>
        MbElement shipNum = sn.getFirstChild(); // shipNum
        if (shipNum!=null) shipNum.setNamespace(ns);
        MbElement orderNum = shipNum.getNextSibling(); // orderNum
        if (orderNum!=null) orderNum.setNamespace(ns);
        MbElement custRef = orderNum.getNextSibling(); // customerRef
        if (custRef!=null) custRef.setNamespace(ns);
        MbElement items = custRef.getNextSibling(); // items
        if (items!=null) items.setNamespace(ns);

        MbElement item = items.getFirstChild(); // Item
        boolean end=false;
        // Search inside every item //
        do
        {
            if (!(end=!(item!=null)))
            {
                item.setNamespace(ns);
                MbElement id = item.getFirstChild(); // ID
                if (id!=null) id.setNamespace(ns);
                MbElement qty = id.getNextSibling(); // qty
                if (qty!=null) qty.setNamespace(ns);
                MbElement price = qty.getNextSibling(); // price
                if (price!=null) price.setNamespace(ns);
            }
        }
        while (!end&&(item=item.getNextSibling())!=null);

        MbElement total = items.getNextSibling(); // items
        if (total!=null) total.setNamespace(ns);
    }

    /**
     * <p> Copy the entire header </p>
     * @param msgIn
     * @param msgOut
     * @throws MbException
     */
    protected void copyHeaders(MbMessage msgIn, MbMessage msgOut)
        throws MbException
    {
```

```
        MbElement headersIn = msgIn.getRootElement().getFirstChild();
        MbElement headerOut = msgOut.getRootElement();

    while (headersIn!=null && headersIn.getNextSibling()!=null)
    {
        headerOut.addAsLastChild(headersIn.copy());
        headersIn = headersIn.getNextSibling();
    }
  }
}
```

B

Sample instructions

In this appendix we provide sample instructions.

371

WebSphere Message Broker message flows

In this section we discuss WebSphere Message Broker flows.

Import of build time artifacts

To implement the WebSphere Message Broker runtime ESB, we import a project interchange consisting of the message set, message flows, and XML Schema Definitions Broker artifacts. The rest of this section provides instructions for the importation of the required build time resources to run the scenario. Descriptions of these message flows and the Java Compute code are shown later in this appendix.

Import the message flow and message set projects

To import the message flows and message set project, follow these steps:

1. Download the samples provided with *Patterns SOA Design using WebSphere Message Broker and WebSphere Enterprise Service Bus,* SG24-7369-00, and copy the sw-610_wmb6.0.2_releaseCandidate.zip files.

2. Paste the 610_wmb6.0.2_releaseCandidate.zip files into your Message Broker workspace. If you installed the WebSphere Message Broker Toolkit in the default location, this will be C:\Documents and Settings\<userid>\IBM\wmbt6.0\workspace.

3. Start the WebSphere Business Integration Message Broker Toolkit.

4. Import the project interchange into the Eclipse environment:

   a. Click **File → Import**.

   b. Select **Project Interchange** and click **Next**.
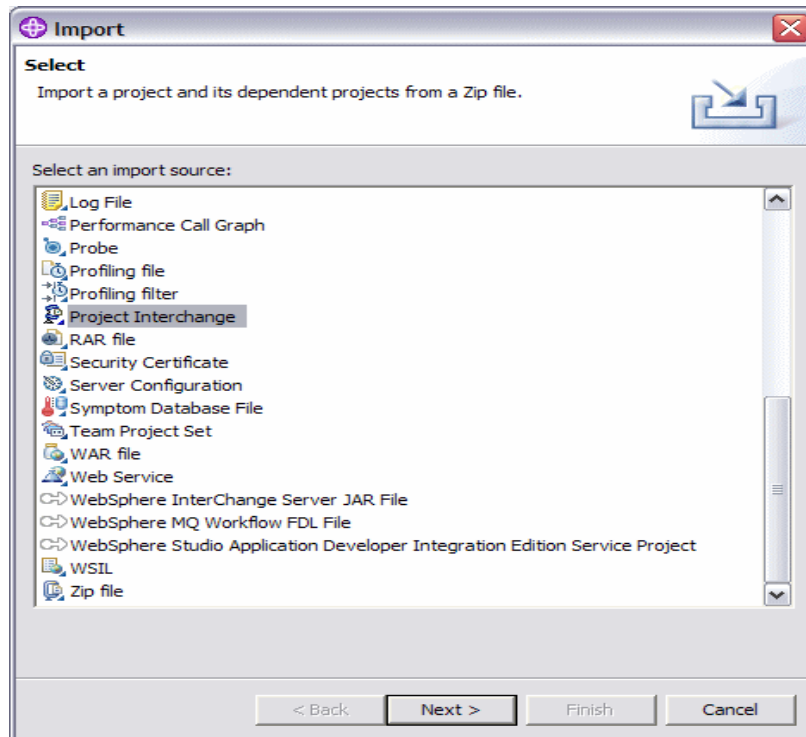
Figure B-1   Import Project Interchange window example

c. Select **sw-610_wmb6.0.2_releaseCandidate.zip** and click **Open**.
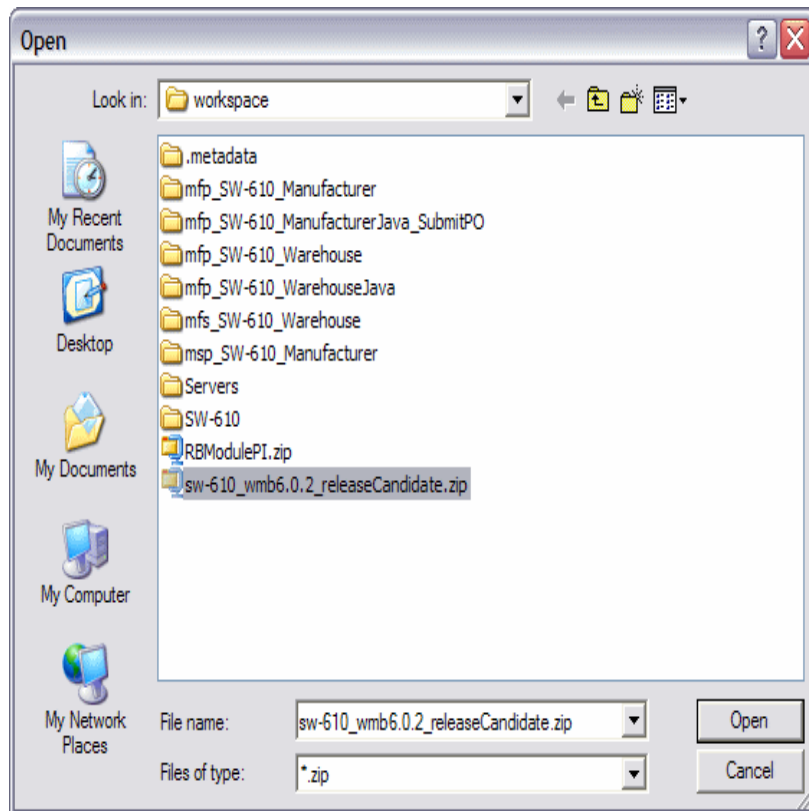
Figure B-2   Project Interchange window example

d. Click **Select ALL** to select all projects listed, and click **Finish**.
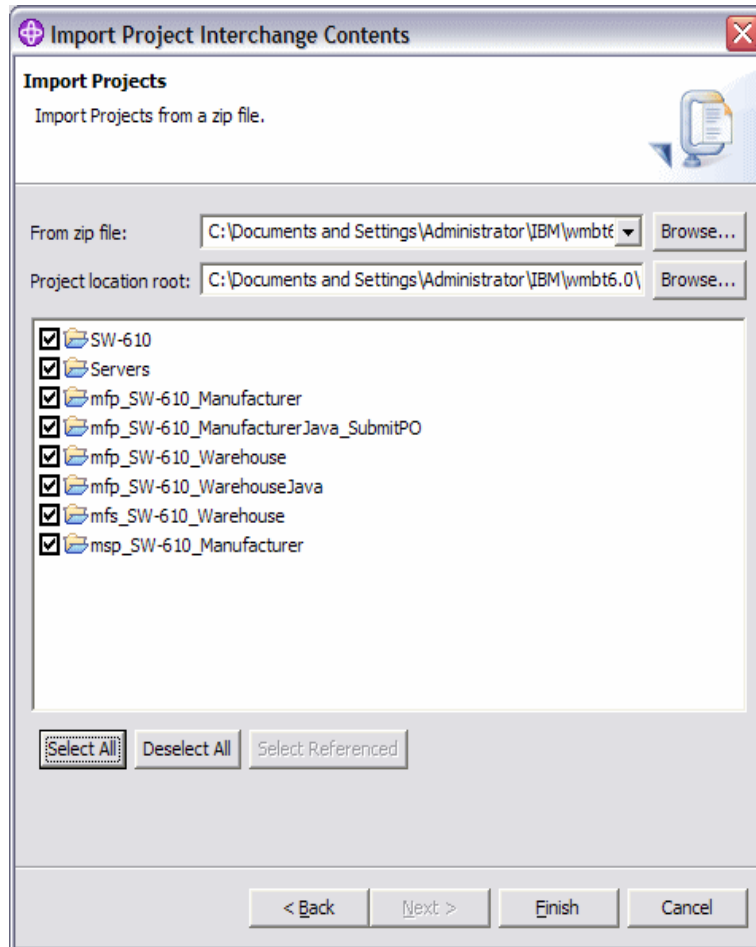
Figure B-3   Select All Projects panel example

At this time you have imported all of the message flows and messages sets required for the implementation. You may see several errors and warnings after these two projects have been imported. These can be ignored, and will be addressed when we do a clean of the message flows and message sets. The clean request will rebuild and resolve the errors.

Removing errors and 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 errors and warnings:

1. From the WebSphere Business Integration Message Broker Toolkit, click **Project** → **Clean**.

2. Select **Clean all projects** and click **OK**.
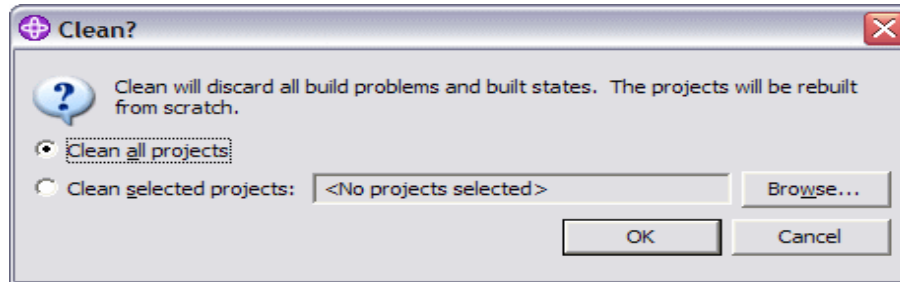
Figure B-4   Clean all projects window example

At this time all errors should have been resolved. You can ignore any remaining warnings because they will be resolved at run time.

C
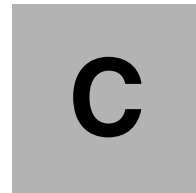
Additional material

This book 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/SG247369`

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 book form number, SG24-7369.

Using the Web material

The additional Web material that accompanies this book includes the following files:

File name   *Description*
SOAFiles.zip   Zipped book code samples

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 book.

IBM Redbooks

For information about ordering these publications, see "How to get IBM Redbooks" on page 379. Note that some of the documents referenced here may be available in softcopy only.

► *Patterns: SOA Foundation Service Connectivity Scenario*, SG24-7228

► *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212

► *WebSphere Message Broker Basics*, SG24-7137

► *Enabling SOA Using WebSphere Messaging*, SG24-7163

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

► *Patterns: Implementing and SOA Using and ESB,* SG24-6347

► *WebSphere Service Registry and Repository Handbook*, SG24-7386

► *WebSphere Message Broker V6, Best Practices Guide: Bullet Proofing Message Flows*, REDP-4043

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

Symbols
(Web Services Interoperability Organization   148

Numerics
69097
    Head 5
        FaultHandler JavaCompute node   255

A
Access Control List (ACL)   160
ACID properties   57
Activity Sessions   100
AggregateControl Node   128
AggregateReply Node   128
AggregateReply node   130
AggregateRequest Node   128
any-to-any transformation engine   171
application cluster   204
approach to creating service
    bottom-up   28
    top-down   28
ASN.1   172
Asynchronous reliability   101
authentication and authorization   170

B
backend resource gateway   172
Binding   95
biphttplistener process   232
BLOB   115, 250
BLOB domain   120
bottom-up   28
broker   107
broker domain   109
broker engine   106
Broker Topology   264
BuildReply node   130
business design   10
Business Process Execution Language for Web Services   29
business requirements   13
business scenario   245

Business Service Registry   176

C
C and COBOL data structures   112
centralized policy management   178
CICS EXCI   111
CICS Transaction Server   61
CICS transaction server   106
CICSRequest node   132
COBOL Copybook   172
COBOL Copybooks   175
COBOL copybooks   114
COBOL program   113
collaborative business   13
Commercial Off The Shelf (COTS)   40
Common Base Event   102
common event infrastructure   102
common event infrastructure cluster (CEI Cluster)   202
component based application   15
Component Business Model (CBM)   19
composite application   11
Compute node   116
Configuration manager   263
configuration manager   108
Configuration Manager Proxy (CMP)   135, 160
configuration manager proxy (CMP)   236
Configuration Manager Proxy API   135
consume Web service   28
create customer account service   109
create, read, update and delete (CRUD) operations   192
CSV   172
custom mediation support   109
Custom Wire Format (CWF)   135
customer balance query   109

D
data format transformations   181
data model and namespace   171
data parsing   56
DataGlue technology   172
DataPower   4, 168

IBM

Redbooks

Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB

Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB

IBM ®

Redbooks

ESB implementation options for maturing SOA

Enhance your knowledge of IBM ESB products

Learn how to enable your environment with ESB patterns

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 Redbooks publication focuses on the use of the WebSphere Enterprise Service Bus and WebSphere Message Broker together to form an enterprise service bus (ESB) implemented in a service-oriented architecture (SOA).

This book discusses patterns for integrating WebSphere Enterprise Service Bus and WebSphere Message Broker and includes a scenario to help you design, develop, and deploy these products.

This book is designed to assist customers that are approaching the use of both advanced and basic ESB products from typically messaging and J2EE worlds, but are not quite sure when each is appropriate.