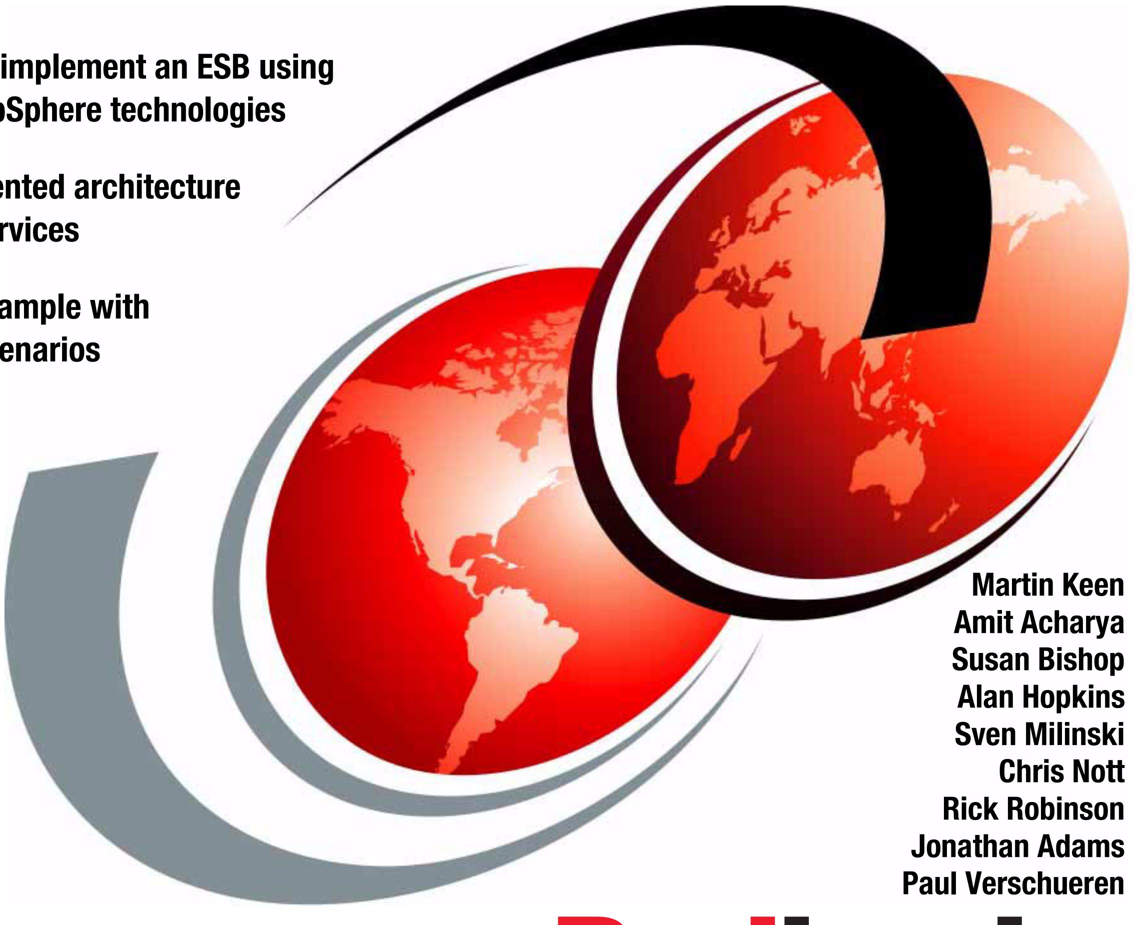
WebSphere® software

IBM

Patterns: Implementing an SOA Using an Enterprise Service Bus



Design and implement an ESB using current WebSphere technologies

Service-oriented architecture and Web services

Learn by example with practical scenarios

Martin Keen
Amit Acharya
Susan Bishop
Alan Hopkins
Sven Milinski
Chris Nott
Rick Robinson
Jonathan Adams
Paul Verschueren

Redbooks

ibm.com/redbooks

International Technical Support Organization

Patterns: Implementing an SOA Using an Enterprise Service Bus

July 2004

SG24-6346-00

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

First Edition (July 2004)

This edition applies to WebSphere Application Server V5.1, WebSphere Business Integration Server Foundation V5.1, and WebSphere Business Integration Message Broker V5.0.

Contents

Notices

Trademarks

Preface

Many enterprises (large and small) are focused on increasing their business flexibility while simplifying their IT infrastructure in order to better meet their business objectives. The IBM® on demand Operating Environment defines a set of integration and infrastructure management capabilities that enterprises can use to achieve these challenging objectives. The on demand Operating Environment features of particular relevance to this book are the use of a service-oriented architecture (SOA) together with an Enterprise Service Bus. These are both necessary to achieve the goals of increased business flexibility and a simplified IT infrastructure. Many of these enterprises are determined to use proven architectures, designs, and product mappings in order to speed their implementation and minimize their risk.

The Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying e-business applications. This IBM Redbook focuses on how the SOA profile of the Process Integration patterns can be used to start implementing SOA using an Enterprise Service Bus.

Part 1 presents a description of SOA and how it applies to Web services and e-business on demand™. Emerging SOA trends are also discussed.

Part 2 provides a detailed description of the Enterprise Service Bus (ESB) concept and how this fits with the Patterns for e-business. Common usage scenarios, a minimum capability ESB, and SOA patterns are described. IBM product mappings are then applied to the SOA patterns.

Part 3 guides you through the process of implementing an Enterprise Service Bus using current IBM technologies. Router and Broker interactions within an Enterprise Service Bus are covered, along with off-the-bus service choreography and the Exposed ESB Gateway to enable interaction in an inter-enterprise environment.

The team that wrote this redbook

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

Figure 1 Back row (left to right): Martin, Chris, Alan, Amit, Sven; front row: Susan and Rick

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

Amit Acharya is a Software Engineer with WebSphere Quality Center of Competence Organization, in Research Triangle Park, North Carolina. He has two years of experience in Enterprise Application Development (EAD) using J2EE, Web services, and WebSphere Studio. His areas of expertise include simulating customer environments for the latest releases of WebSphere Application Server, and scalability and performance of WebSphere Application Server. He has also actively contributed to the IBM patent portfolio. Before joining the WebSphere Application Server group, Amit worked with the WebSphere Edge Server performance group in IBM Pittsburgh Lab. He holds a Masters of

Science degree in Electrical and Computer Engineering from Purdue University in Indiana.

Susan Bishop is an accredited Senior Technical IT Specialist with IBM Software Group in Brisbane, Australia. She has eight years of experience in software development, consulting, technical sales, and integration architecture. Her areas of expertise include WebSphere technologies, EDI, J2EE, and customer solutions analysis. She received a BSc degree in Computer Information Systems from DeVry Institute of Technology in Calgary, Canada.

Alan Hopkins is a Senior IT Specialist with IBM Software Group Services, based at the Hursley Laboratory in the UK. He has more than 15 years of experience in technologies related to IBM middleware and e-business. Currently, he is focused on the application of these technologies to the integration of business processes. Alan has a Ph.D in computational statistical mechanics from the University of Bradford in the UK.

Sven Milinski is an IT Specialist with IBM Global Services, Germany, and has four years of experience in the Enterprise Application Development (EAD) and Enterprise Application Integration (EAI) fields using J2EE and messaging technologies. He architected and developed several Web services–based integration solutions and contributed to Web services–related publications external to IBM. Sven holds a Bachelor's Degree in Information Technology from the Berufsakademie (University of Cooperative Education) in Mannheim, Germany.

Chris Nott is a Consulting IT Specialist with IBM Software Group in the UK. He has 14 years of IT experience in software development, technical pre-sales consulting, and solution architecture. His areas of expertise include Enterprise Application Integration (EAI) using application connectivity and process integration approaches, and he has a strong background in relational systems design and database technology. He holds a Bachelor of Science degree in Mathematics from the University of Durham and is registered with the Engineering Council in the UK as a Chartered Engineer.

Rick Robinson is an Advisory IT Architect in IBM Software Group Services, based at the Hursley Laboratory in the UK. He has seven years of experience in IT, and his roles have included solution architecture, design, and development. Rick has a PhD in the Physics of Superconducting devices from the University of Birmingham in the UK. His areas of expertise include distributed systems design, the WebSphere platform, and service-oriented architecture, and he has written and spoken extensively on these subjects.

Jonathan Adams is an IBM Distinguished Engineer. He has been an IT Architect with IBM for 36 years. For the past 10 years, he has focused on helping IBM deliver reuseable end-to-end middleware solutions to its customers. Since

September 1998, he has worked in the Software Group Technical Strategy organization leading the definition and development of the Patterns for e-business. These patterns have been built by teaming across all major IBM divisions. The resultant patterns are being used by IBMers, customers and Business Partners to help reduce risk and increase speed to market on many e-business solution developments.

Paul Verschueren is a Consulting e-business Architect in the IBM Architecture Services group based at Hursley Laboratories, U.K. He has more than 20 years of experience in large-scale architecture and design problems, with a particular focus on business intelligence, business process management, identity management, and business Process Integration. He has most recently been leading work in IBM to revise and extend the application of IBM Patterns for e-business to the domain of Process Integration.

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

Eugene Kharlamov
WebSphere Studio Development, Toronto

Helen Wylie
Consulting IT Architect, Hursley

Graham Winn
Hursley Web Services Service Team

Adrian Spender, Adrian Warman
Web Services Development, UK

Marc-Thomas Schmidt
Distinguished Engineer, Enterprise Service Bus & Meta-information Management, Somers, NY

Shankar S Kalyana
Executive Architect, Atlanta

Stuart Jones
Worldwide Technical Sales, Chicago

Olaf Zimmermann
Senior IT Architect Enterprise Integration, Heidelberg, Germany

Frank Mueller
IBM Global Services, Germany

Beth Hutchison
Senior Technical Staff Member and Web Services Architect, Hursley

Rachel Reinitz
Web Services Senior Consultant and Senior Certified IT Specialist, California

Mark Colan
Evangelist for SOA & Web Services, Cambridge

Keith Jones
Development Manager & Senior Consulting IT Architect, Colorado

Dave Clarke
WebSphere Business Integration Marketing, Hursley

Kareem Yusuf
Program Director, WebSphere Product Management & Marketing, New York

Sarah Hill
Senior IT Specialist, Software Group, UK

Betsy Thaggard
Editor, ITSO, Austin Center

Finally, thanks to Mark Endrei and the redbook team of *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

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

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

  `ibm.com/redbooks`

- Send your comments in an e-mail to:

  `redbook@us.ibm.com`

- Mail your comments to:

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

Part 1

Patterns for e-business and SOA

Part one introduces the IBM Patterns for e-business, e-business on demand, and service-oriented architecture.

1

Introduction to Patterns for e-business

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

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

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

1.1 The Patterns for e-business layered asset model

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

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

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

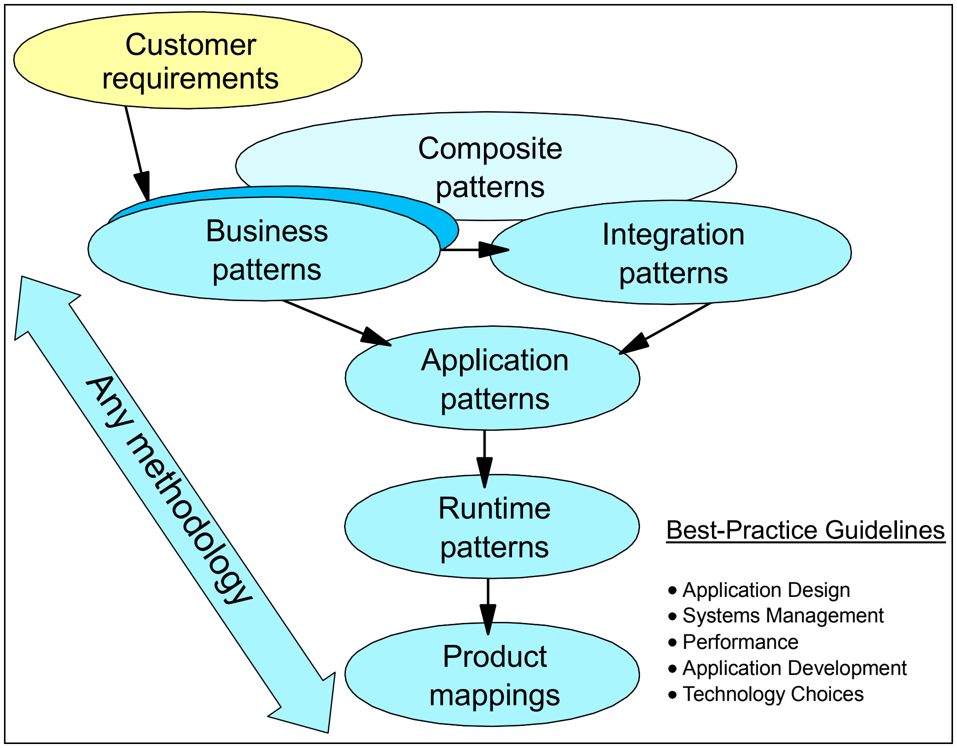


Figure 1-1 The Patterns for e-business layered asset model

Patterns for e-business Web site

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

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

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

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

1.2 How to use the Patterns for e-business

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

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

1.2.1 Select a pattern or Custom design

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

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

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

Business patterns

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

There are four primary Business patterns, as shown in Figure 1-2.

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

Figure 1-2 The four primary Business patterns

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

Integration patterns

Integration patterns enable us to tie together multiple Business patterns to solve a business problem. Figure 1-3 outlines the Integration patterns.

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

Figure 1-3 Integration patterns

These Business and Integration patterns can be combined to implement installation-specific business solutions. We call this a Custom design.

Custom design

We can illustrate the use of a Custom design to address a business problem through an iconic representation, as shown in Figure 1-4.

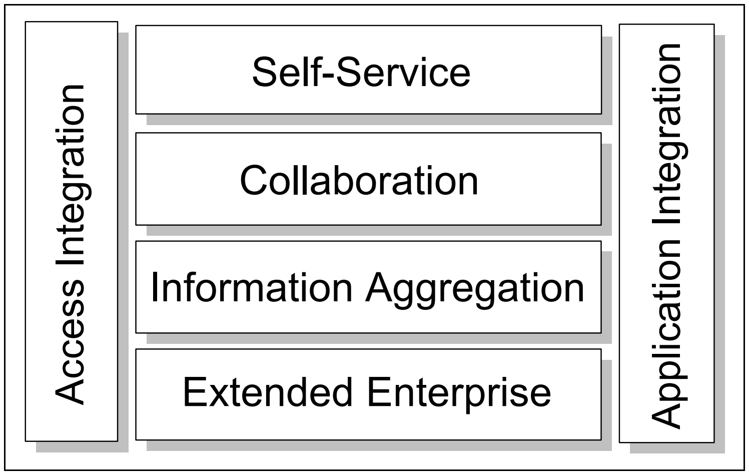


Figure 1-4 Patterns representing a Custom design

When illustrating a Custom design, we can show any unused Business or Integration patterns as lighter blocks than those that are used. For example, Figure 1-5 shows a Custom design that does not have a Collaboration business pattern or an Extended Enterprise business pattern for a business problem.

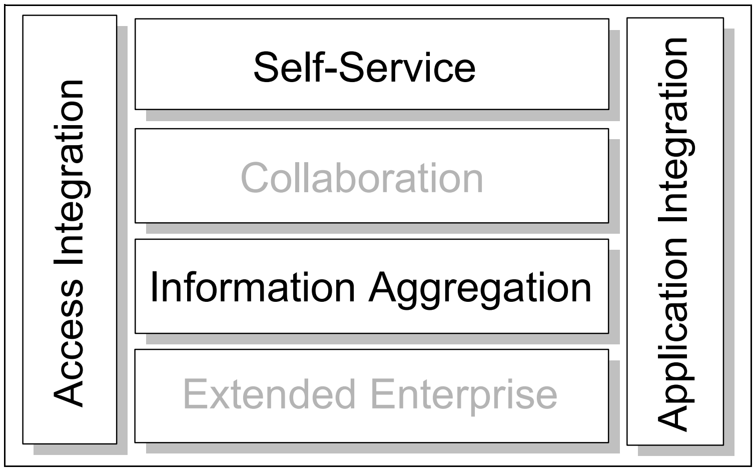


Figure 1-5 Custom design with Self-Service, Information Aggregation, Access Integration, and Application Integration

A Custom design may also be a Composite pattern if it recurs many times across domains with similar business problems. For example, the iconic view of a Custom design in Figure 1-5 can also describe a Sell-Side Hub composite pattern.

Composite patterns

Several common uses of Business and Integration patterns have been identified and formalized into Composite patterns. Figure 1-6 shows the identified Composite patterns.

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

Figure 1-6 Composite patterns

The makeup of these patterns is variable in that there will be basic patterns present for each type, but the Composite can be extended easily to meet additional criteria. For more information about Composite patterns, refer to *Patterns for e-business: A Strategy for Reuse* by Jonathan Adams et al.

1.2.2 Selecting Application patterns

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

Application patterns define the application by its most basic conceptual components to identify the goal of the application. In our example, the application falls into the Self-Service business pattern and the goal is to build a simple application that enables users to access back-end information. The Self-Service::Directly Integrated Single Channel application pattern shown in Figure 1-7 fulfills this requirement.

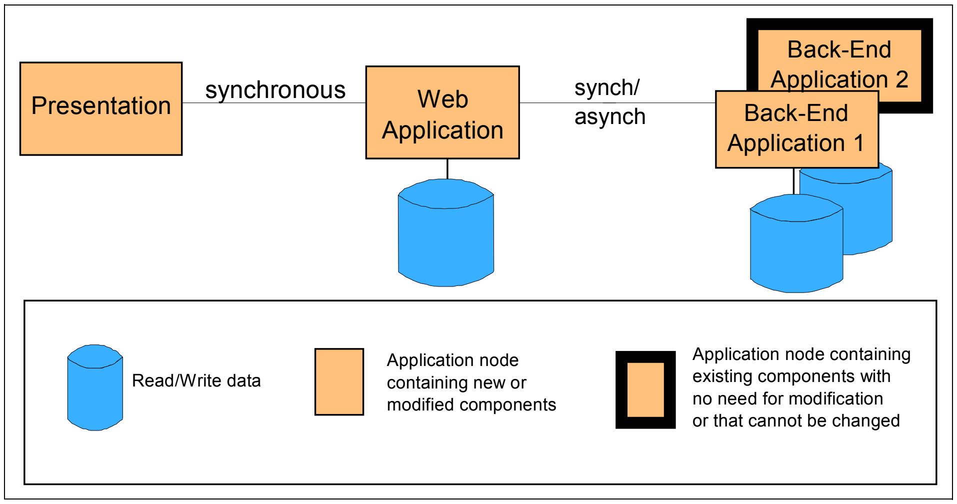

Presentation
synchronous
Web Application
synch/ asynch
Back-End Application 2
Back-End Application 1
Read/Write data
Application node containing new or modified components
Application node containing existing components with no need for modification or that cannot be changed

Figure 1-7 Self-Service::Directly Integrated Single Channel

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

Suppose that the situation is a little more complicated than that. For example, if the automobile policies and the homeowner policies are kept in two separate and dissimilar databases, the user request would actually need data from multiple, disparate back-end systems. In this case there is a need to divide the request into multiple requests (*decompose* the request) to be sent to the two different back-end databases, then to gather the information that is sent back from the requests and put it into the form of a response (*recompose*). In this case the Self-Service::Decomposition application pattern shown in Figure 1-8 would be more appropriate.

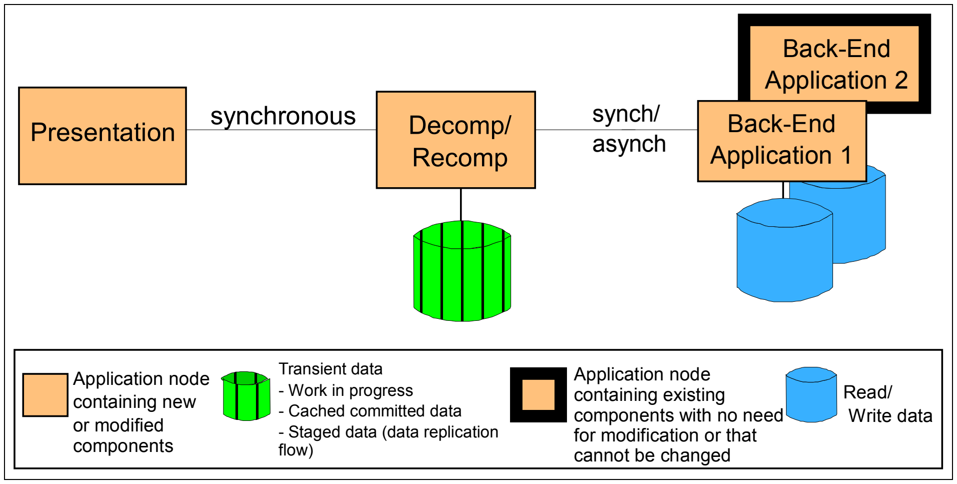

Back-End Application 2
Presentation
synchronous
Decomp/ Recomp
synch/ asynch
Back-End Application 1
Application node containing new or modified components
Transient data
- Work in progress
- Cached committed data
- Staged data (data replication flow)
Application node containing existing components with no need for modification or that cannot be changed
Read/ Write data

Figure 1-8 Self-Service::Decomposition

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

1.2.3 Review Runtime patterns

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

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

They know that users on the Internet will access their business data and will therefore require a measure of security. Security can be implemented at various layers of the application, but the first line of defense is almost always one or more firewalls that define who and what can cross the physical network boundaries into their company network.

They also need to determine the functional nodes that are required to implement the application and security measures. The Runtime pattern shown in Figure 1-9 is one of their options.

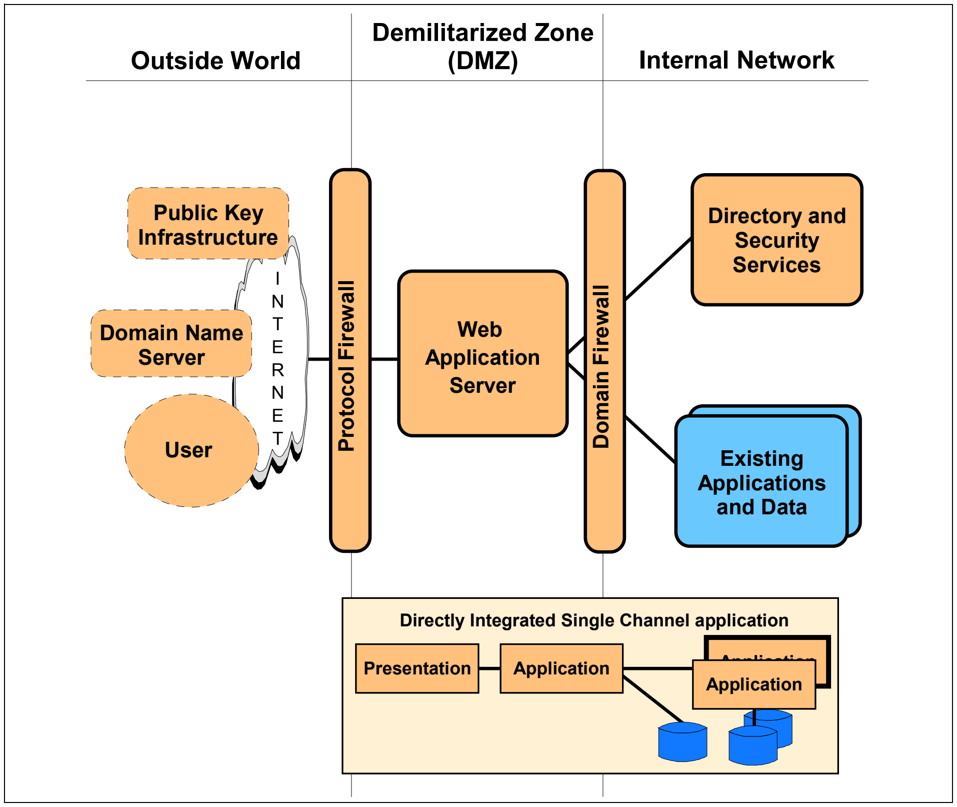


Figure 1-9 Directly Integrated Single Channel application pattern::Runtime pattern

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

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

A characteristic that makes this Runtime pattern different from others is the placement of the Web application server between the two firewalls. The Runtime pattern shown in Figure 1-10 is a variation on this. It splits the Web application server into two functional nodes by separating the HTTP server function from the application server. The HTTP server (Web server redirector) serves static Web pages and redirects other requests to the application server. It moves the application server function behind the second firewall, adding further security.

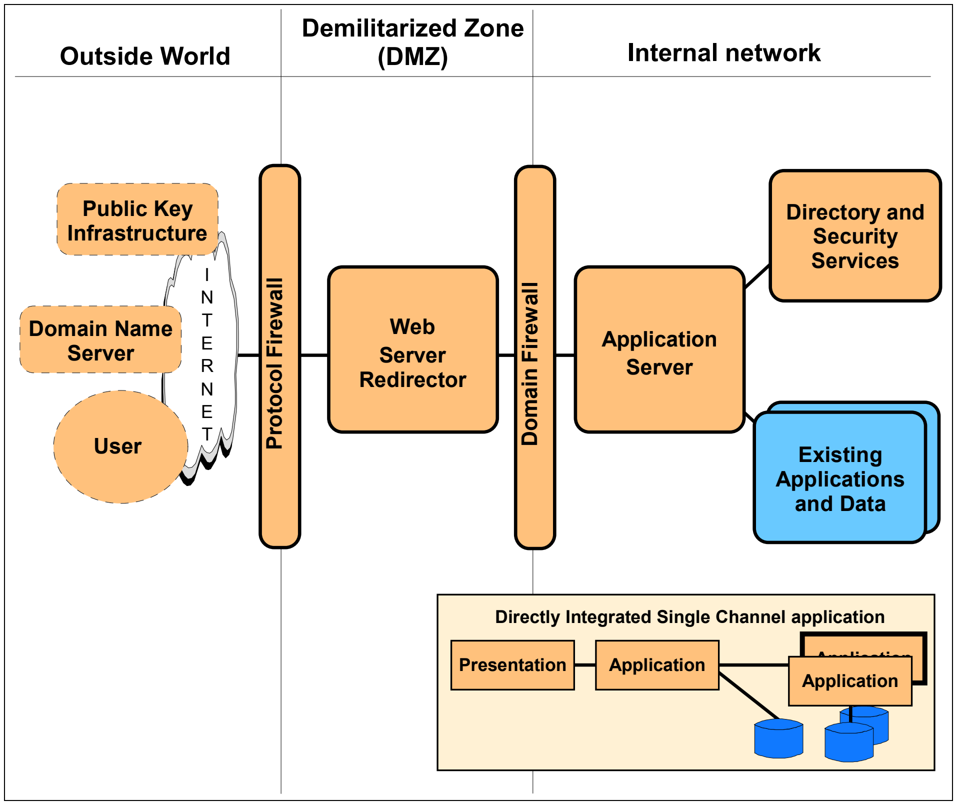


Figure 1-10 Directly Integrated Single Channel application pattern::Runtime pattern: Variation 1

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

1.2.4 Review Product mappings

The last step in defining the network structure for the application is to correlate real products with one or more runtime nodes. The Patterns Web site shows each Runtime pattern with products that have been tested in that capacity. The

Product mappings are oriented toward a particular platform, though more likely the customer has a variety of platforms involved in the network. In this case, it is simply a matter of mix and match.

For example, the runtime variation in Figure 1-10 on page 14 could be implemented using the product set depicted in Figure 1-11.

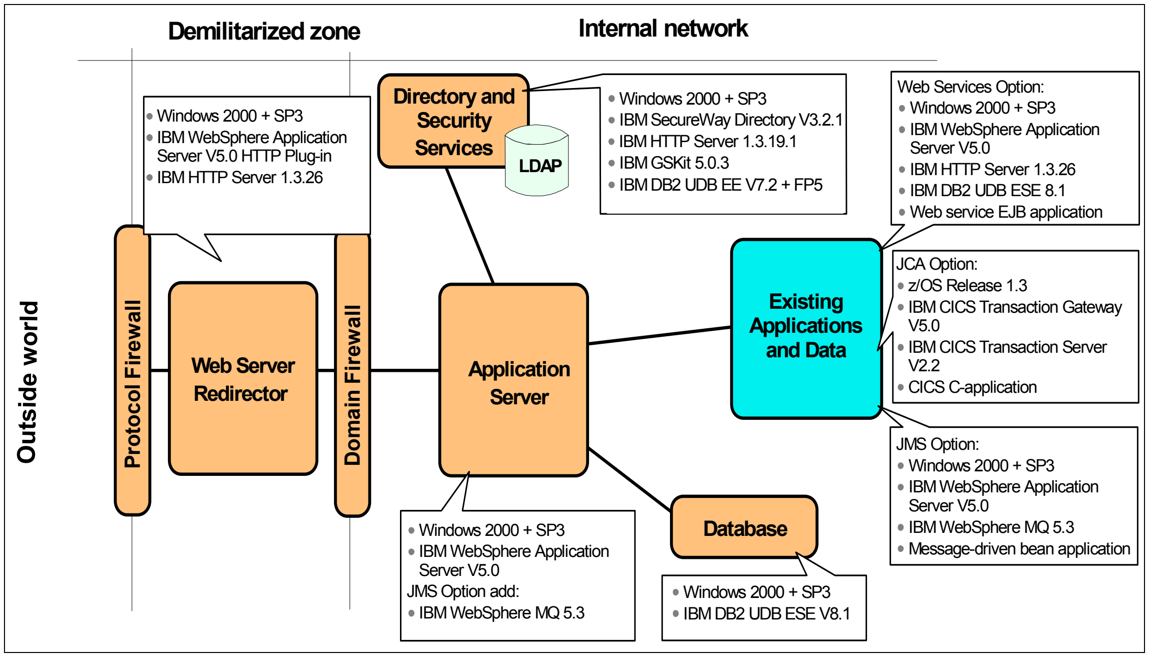


Figure 1-11 Directly Integrated Single Channel application pattern: Windows 2000 Product mapping

1.2.5 Review guidelines and related links

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

- *Design* guidelines instruct you on tips and techniques for designing the applications.
- *Development* guidelines take you through the process of building the application, from the requirements phase all the way through the testing and rollout phases.

- *System management* guidelines address the day-to-day operational concerns, including security, backup and recovery, application management, and so forth.
- *Performance* guidelines give information about improving the application and system performance.

1.3 Summary

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

2

e-business on demand and service-oriented architecture

Increasing consideration is being given to the strategic initiative of e-business on demand. This chapter provides an overview of the on demand concepts and discusses the correlation with the service-oriented architecture (SOA). Attention is given to how these concepts are achieved using Web services.

The chapter includes:

- A brief summary of the business drivers and technical capabilities of e-business on demand
- The correlation between e-business on demand and SOA
- IBM on demand Operating Environment and the Enterprise Service Bus

2.1 Overview of e-business on demand

The IBM vision of e-business on demand is to enable customers to succeed in an environment with an unprecedented rate of change.

Businesses want to focus on core competencies, reduce spending, and reuse existing information in new ways without a major overhaul of their existing infrastructure. There exists a constant pressure to juggle the often conflicting demands to provide flexibility, cost savings, and efficiency. The following sections outline the key business and technical attributes that provide the basis for the on demand message.

Figure 2-1 identifies the key components of e-business on demand, which are discussed in detail in the sections that follow.

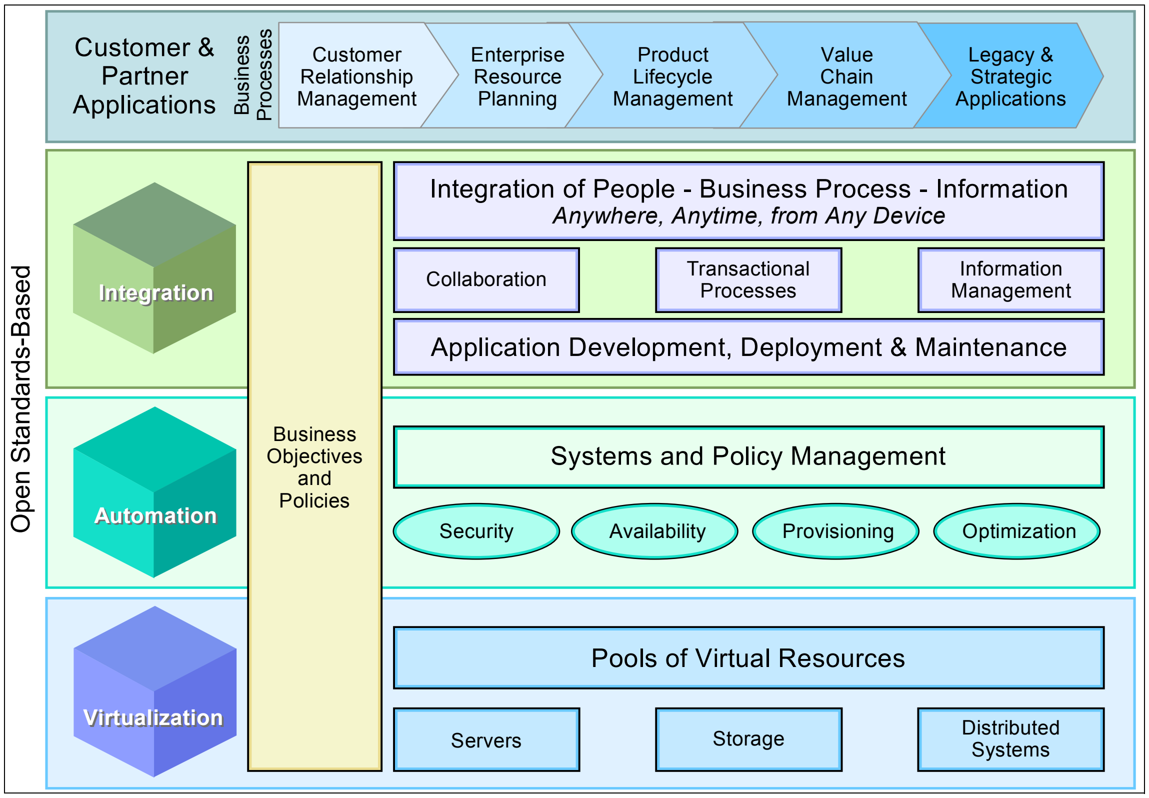

Open Standards-Based
Customer & Partner Applications
Business Processes
Customer Relationship Management
Enterprise Resource Planning
Product Lifecycle Management
Value Chain Management
Legacy & Strategic Applications
Integration
Automation
Virtualization
Business Objectives and Policies
Integration of People - Business Process - Information
Anywhere, Anytime, from Any Device
Collaboration
Transactional Processes
Information Management
Application Development, Deployment & Maintenance
Systems and Policy Management
Security
Availability
Provisioning
Optimization
Pools of Virtual Resources
Servers
Storage
Distributed Systems

Figure 2-1 e-business on demand overview diagram

2.1.1 Key business attributes

From a business perspective, e-business on demand is about providing a way for companies to realign their business and technology environment to match the request for reusable business functionality.

Business drivers can be summarized with the following key elements:

- **Focused**

  Enabling the enterprise to focus on their core competencies; what makes them successful and what makes them unique. Strategic alliances are formed to provide needs external to these core competencies.

- **Responsive**

  The ability to respond with agility to customer demands, market opportunities, or external threats. These decisions are guided through insight-driven decision management features.

- **Variable**

  To achieve operational and business process flexibility. To adapt variable cost structures (fixed to variable) to provide a high level of operational efficiency.

- **Resilient**

  Capability and robustness to respond to changes in both business and technical environments. Manage changes and threats with predictable outcomes.

Companies can achieve these business imperatives by exploiting current technological developments while drawing on experiences that have been learned from past architectural constructs.

2.1.2 Key technology attributes

The business drivers of e-business on demand must be supported by a well-defined technical infrastructure.

These key technological attributes deliver the flexibility, responsiveness, and efficiency that on demand organizations require:

- Integration
- Virtualization
- Automation
- Open standards

Figure 2-2 on page 20 provides a high-level overview of the range of each e-business on demand attribute.

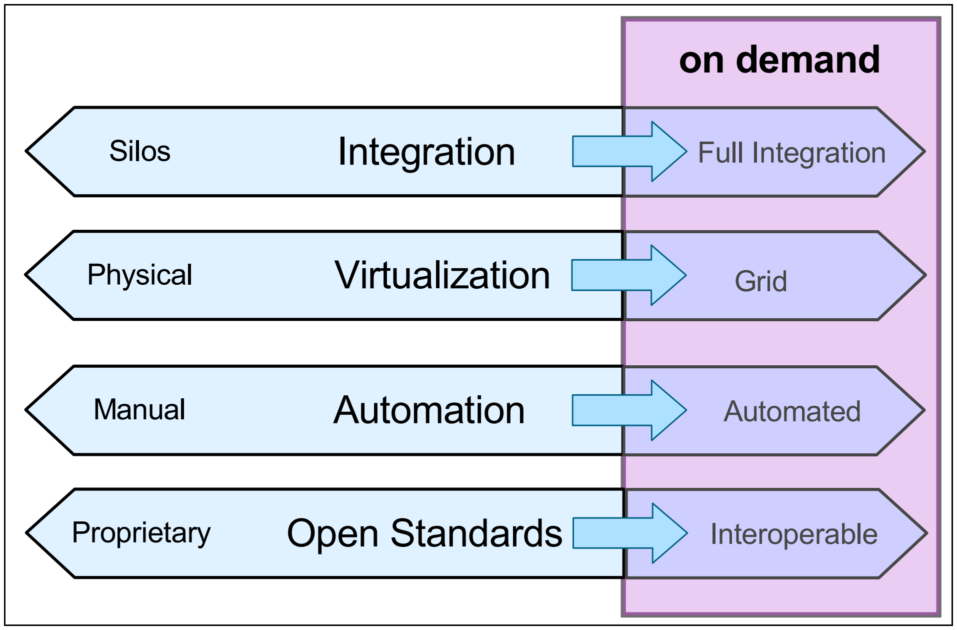


Figure 2-2 Four key technology attributes of e-business on demand

In the sections that follow, these four key elements are described as they apply to e-business on demand. They are then expanded to demonstrate the correlation of e-business on demand and the SOA.

Integration

The fundamental component of on demand infrastructure is integration:

In 2002, Sam Palmisano, Chief Executive Officer of IBM, defined on demand in the following way: “An on demand business is an enterprise whose business processes, integrated end-to-end with key partners, suppliers, and customers, can rapidly respond to any customer demand, market opportunity, or external threat.”

Integration can occur at various levels:

- People

  To function at an on demand operating level, human-to-human and human-to-process interaction requires integration throughout the various levels not limited to end users. Business partners, customers, and employees are all important resources to the value chain provided by on demand. For example, integration can occur for developers through open tooling paradigms based on open standards, for business partners by the creation of horizontal processes and employees through collaboration.

- Process

  Recurring elements (security, service level, monitoring, and so on) can be shared across applications to provide horizontal services to decouple these reusable application components. The use of SOA and Web services to implement these processes, including the emerging Business Process Execution Language for Web Services (BPEL4WS), will facilitate more rapid changes in these processes, enabling the business to respond with agility to changing market conditions.

- Applications

  Organizations have invested enormous resources and capital into custom-designed and off-the shelf applications. The application integration goal is to leverage, rather than replace, these assets by providing ways of connecting, routing, and transforming the data that is stored or shared among them. Applications sit on disparate systems in an enterprise or across many enterprises.

- Systems

  Systems manage, process, and deliver data to the people and applications in the solution environment. An on demand Operating Environment requires the system to be transparent to the elements that interact with it.

- Data

  Data is the primary business element of a system. The data is the source of the information and can more easily be shared through the adoption of standards specifications.

Virtualization

Various areas of technology in our lives exploit virtualization concepts, including cell phones, PDAs, wireless connectivity, printers, and so forth. Aspects of virtualization draw on widely adopted architectural concepts, including object oriented design and development, Web services, and XML.

There is a spectrum of virtualization that begins at independent stand-alone systems on one side (a large mainframe system, perhaps) and grid computing on the other. In the middle are varying degrees of client-server implementations.

A grid paradigm, an absolute example of on-demand virtualization, is a collection of distributed computing resources that are available over a local or wide area network and that appear to an end user or application as one large virtual computing system.

The Internet, the most widely recognized example of virtualization, provides a virtual network that supplies access to content and applications.

The vision is to create virtual dynamic organizations through secure, coordinated resource sharing among individuals, institutions, and resources. Grid computing is an approach to distributed computing that spans locations, organizations, machine architectures, and software boundaries.

Figure 2-1 on page 18 depicts virtualization as a set of virtualized resource pools based on:

- Servers

  This could include partioning, hypervisors, VM OS, emulators, I/O virtualization, virtual Ethernet, and so forth.

- Storage

  Here, the focus is on the addition of intelligence and value in the network.

- Distributed systems

  This includes Web services, scheduling, provisioning, workload management, billing/metering, and transaction management.

The goal of grid computing, and thus on demand virtualization, is to provide unlimited power, collaboration, and information access to everyone connected to a grid.

Note: Open Grid Services Architecture (OGSA) is an important starting point for grid enablement. For more information about OGSA, refer to the article at:

http://www-106.ibm.com/developerworks/grid/library/gr-visual/

Automation

Autonomic computing addresses an organization's need to limit the amount of time and cost that occurs as a result of:

- Overprovisioning
- High cost of new applications and highly skilled labor
- Amount of time spent on disparate technology platforms even within one organization
- IT budget spent on maintenance, not problem resolution
- Complexities in operating heterogeneous systems

So how can organizations begin to address these common concerns using an on demand Operating Environment? This is where autonomic computing comes in. Autonomic computing can be summarized using the four key components:

- Self-healing

  A system's ability to keep functioning. In order to achieve this, the system must detect, prevent, and recover from disruptions with minimal or no human intervention. This requirement is directly proportional to increased business dependence on technical infrastructures. The need for self-healing is directly proportional to the organization's availability requirement.

- Self-configuring

  The ability to adapt dynamically to changing environments, add and remove components to and from the systems, and change the environment to adapt to variable workloads.

- Self-optimization

  Configuration that maximizes operational efficiency including resource tuning and workload management. This alleviates the constant drain on resources to perform routine tasks. The goal is to tune systems to respond to the workload changes. Systems have to monitor and self-tune continuously, adapting and learning from the environment around them.

- Self-protecting

  Security is one of the inhibitors of the adoption of SOAs as organizations prepare themselves to share data externally. Self-protection requires the system to provide safe alternatives to secure information and data. Self-protecting automation works by anticipating, detecting, identifying, and protecting systems from external or internal threats.

Open standards

While described as an attribute on its own, open standards affects the on demand Operating Environment across the previously defined levels including automation, integration, and virtualization. Each of these elements leverage open standards specifications in order to achieve their objectives. Open standards are the key element of flexibility and interoperability across heterogeneous systems.

The global adoption of a standard specification enables the disparate systems to interact with each other. The underlying platforms may be completely different and independent but open standards enable processes to be built despite (or because of) these differences.

Open standards provide the e-business on demand Operating Environment with a standard, open mechanism to invoke system services.

Shortly, we will discuss the open standards that are involved in providing the level of interoperability that is required to create an SOA.

2.1.3 Key requirements for integration flexibility

In order to enable the business integration that is required by an on demand business while maintaining the maximum flexibility of implementation, we need to meet the requirements shown in Figure 2-3.

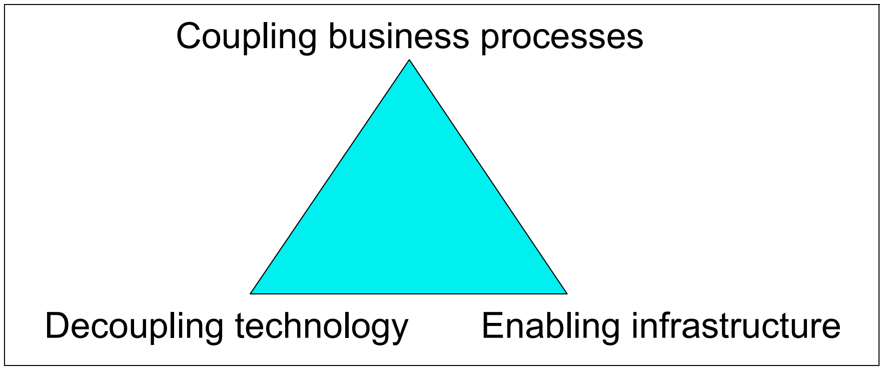


Figure 2-3 On demand key requirements for integration flexibility

Each requirement poses several questions:

- Coupling business processes
  - How do we model the business?
  - How do we refactor the business into processes, components, and services that can interact dynamically and change in an agile manner?
- Decoupling technology
  - How do we support business behavior with systems that can interact without joining them too tightly?
  - How can we change and evolve the systems and interactions on the timescales required by the business?
- Enabling infrastructure
  - How do we build the technical infrastructure to support, execute, manage, and measure these interactions, services, components, and processes?

2.2 e-business on demand and the service-oriented architecture

SOA, as described in 3.2, “Introduction to service-oriented architecture” on page 37, is an approach to defining integration architectures based on the concept of a service. The business and infrastructure functions that are required to make an effective on demand environment are provided as services. These services are the building blocks of the system.

Services can be invoked independently by either external or internal service requesters to process simple functions, or can work together by choreographic implementations to quickly devise new functionality to existing processes.

SOAs may use Web services as a set of flexible and interoperable standards for distributed systems. There is a strong complimentary nature between SOA and Web services as described in Chapter 3, “Web services and service-oriented architecture” on page 33.

SOA touches on the four key elements of e-business on demand in the following way:

- Open standards
  - SOA provides a standard method of invoking Web services (business logic and functionality) for disparate organizations to share across network boundaries.
  - Web services use open standards to allow inter-enterprise connectivity across networks and the Internet:
    - Messaging protocols (SOAP).
    - Transport protocols (including HTTP, HTTPS, JMS).
    - Security can be handled at both the transport level (HTTPS) and/or at a protocol level (WS-Security).
  - WSDL allows Web services to be self-describing for a loosely coupled architecture.
  - Standards bodies, including WS-I, W3C and OASIS exist using technologists from industry leading software vendors (IBM, BEA, Oracle, Microsoft® and so forth) to accelerate and guide open standards creation and adoption.
- Integration
  - Interfaces are provided to wrap service endpoints to provide a system-independent architecture to promote cross-industry communication.

  - SOAs can provide dynamic service discovery and binding, which means that service integration can occur on demand.
- Virtualization
  - A key principle of SOA is that services should be invoked by service requesters that are oblivious to service implementation details, including location, platform, and if appropriate to the business scenario, even the identity of the service provider.
  - Grid services and the very framework it all rests on is very much like object-oriented programming.
- Automation
  - Grid technologies are applying SOA principles to implementing infrastructure services that will provide an evolutionary approach to increased automation.

Further information

For more information about the topics that are covered in this section, visit:

- IBM Web Services

  `http://www.ibm.com/webservices`
- IBM on demand Operating Environment

  `http://www-3.ibm.com/software/info/openenvironment/`
- IBM developerWorks®: SOA and Web services zone

  `http://www.ibm.com/developerworks/webservices`

2.3 The on demand Operating Environment and the ESB

The Enterprise Service Bus (ESB) is to SOA as SOA is to e-business on demand. In this section, we explain that statement.

In order to create a truly successful e-business on demand, one must embrace the SOA. which helps businesses wrap functions (services) to provide loosely coupled accessibility to functions, flows, and applications.

So how does the Enterprise Service Bus address the IBM vision of an on demand business? This section aims to describe the way that the Enterprise Service Bus can help businesses create processes that meet the objectives of the capabilities of an on-demand environment.

2.3.1 The on demand Operating Environment

Figure 2-4 shows the on demand Operating Environment based on the SOA.

.

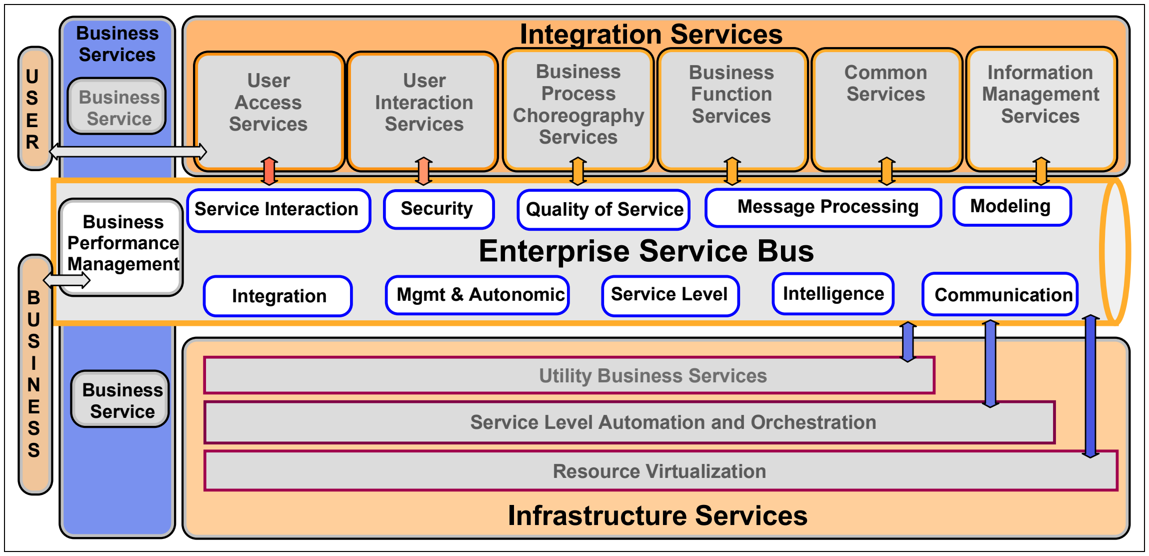


Figure 2-4 On demand Operating Environment architecture

The three core components of the on demand Operating Environment, including integration services, Enterprise Service Bus, and infrastructure services, work together to provide the capability to meet defined business objectives.

Business services leverage the application and infrastructure services, which are mediated by the Enterprise Service Bus, to provide real business processes to end users including customers, employees, and business partners.

Business service management incorporates the policies and goals of the organization, such as service levels, metrics, and other measurable business guidelines.

Enterprise Service Bus

The Enterprise Service Bus is emerging as a service-oriented infrastructure component that makes large-scale implementation of the SOA principles manageable in a heterogeneous world.

On demand applications are business services built from services that provide a set of capabilities that are worth advertising for use by other services. Typically, a business service relies on many other services in its implementation. Services interact via the Enterprise Service Bus, which facilitates mediated interactions between service endpoints. The Enterprise Service Bus supports event-based interactions as well as message exchange for service request handling. One innovation of the Enterprise Service Bus is a common model for messages and events. All messages can become events if deploying the service binds the message to a topic in the event space.

For both events and messages, mediations can be used to facilitate interactions (for example, to find services that provide capabilities that a requestor is asking for or to take care of interface mismatches between requesters and providers that are compatible in terms of their capabilities). In this context, we use the term *service* in a very general sense, and it might be worth noting that although from the perspective of the bus all application components can be specified through WS-* standards (because it requires a normalized representation for efficient mediated, capability-based matchmaking), this does not imply that they all communicate with SOAP or WS-* protocol standards.The Enterprise Service Bus supports a broad spectrum of ways to get on and off the bus, including on ramps for legacy applications or business connections that enable external partners in B2B interaction scenarios to participate in the service interaction game.

Although they all look the same from the perspective of the Enterprise Service Bus, services implement different facets of an overall on demand application, including:

- Realize interactions with people involved in the underlying business process.
- Provide adapters to existing applications that have to be integrated.
- Choreograph the interaction of several services to achieve a business goal.
- Watch for potential problems in the execution of the process, ready to take action to fix them if they occur.
- Manage resources that are needed to perform required business functions.

Therefore, in addition to providing the basic infrastructure for service interactions, the on demand Operating Environment identifies a set of common patterns for construction of on demand applications and provides specific capabilities to support realization of distinct service categories that play particular roles in those patterns. The two distinct service categories are integration and infrastructure service kinds.

The capabilities that are provided by the Enterprise Service Bus (including service level, service interface, quality of service, intelligence, communication, security, message management, modeling, management/automation, and integration) that facilitate the interactions between the levels in the on demand Operating Environment are discussed in detail in 4.3, “A capability model for the Enterprise Service Bus” on page 82.

Enterprise Service Bus scenarios are discussed and implemented in Part 3, “Scenario implementation” on page 167.

Integration services

The programming model for on demand business services is based on application development using component (service) assembly. The services in the integration category are used by on demand application builders to create new business services; they include services that facilitate integration as well as services that provide business functions to be integrated:

- User access services

  Handle adaptation from three orthogonal perspectives:

  - Endpoint form factor such as display size, memory, and processor limitations (ranging from desktop down to pervasive devices)
  - Modes of interaction including conventional display/keyboard interactions, as well as speech-based interactions and combinations (multi-modality)
  - Connection types such as peer-to-peer or client/server across a range of connection reliability including fully disconnected operations

- User interaction services

  Handle direct interactions with people involved in the business process; for example, processing work items that are spawned by choreography or collaborative process elements.

- Business process choreography services

  Support the execution of other services that express their behavior using process flow or rule technology. Process flows, for example, are used to describe the interaction of other services (nearly any of the integration kinds, including other process flow services) to perform the tasks required to realize the functions offered by the new (combined or aggregated) business service.

- Business function services

  Provide the atomic business functions (those that are not composed from other services) that are required by the overall business service; this includes adapters to packaged or existing custom applications as well as brand new application components created to realize a functional need that is not already covered by existing applications.

- Common services

  Implement useful features, or helper functions, that are designed to be used by many business services. Examples include services implementing personalization of user access and user interaction services, or for reporting status and progress of business services.

- Information management services

  Help to integrate information hosted in a variety of data sources such as databases or legacy applications, to access (query, update, and search) that information, to analyze information from those sources in business intelligence scenarios, or taking care of metadata about information and services used and provided by the business services living in the on demand Operating Environment.

Integration services are hosted by application services that provide container facilities to simplify their participation in interactions with other integration services and with on demand Operating Environment infrastructure services. On demand integration service developers focus on realizing the business logic that they care about, assembling integration services that provide required business function and declaring expected quality of service.

Programmers and administrators annotate their applications and services with policy declarations that specify quality of service. The application container (and the Enterprise Service Bus) automates the interactions with infrastructure services to achieve the expressed policies. An application container also provides generic facilities such as taking care of security or transaction management requirements for the services that it hosts, as well as kind-specific facilities such as generating events reporting status and progress of business process choreography services.

Infrastructure services

The services in the infrastructure category provide and manage the infrastructure into which business services and their constituents are deployed. These include:

- Utility business services

  Support functions such as billing, metering, rating, peering, and settlement; commonly used, for example, when hosting on demand business services or their components.

- Service level automation and orchestration

  Provide services that facilitate translation into reality of quality of service policy declarations that are associated with business services. This is achieved by services that implement autonomic managers, which monitor the execution of services (more precisely, services instrumented to be managed elements) in the on demand Operating Environment according to the policy declarations they receive. They then analyze their behavior, and if the analysis indicates problems, plan a meaningful reaction to that problem and initiate execution of that plan. This closed feedback loop is called an *M-A-P-E* (Monitor, Analyze, Plan, Execute) *loop*. Several specializations of such services focus on managing, for example, availability, configuration or workload for the managed elements, provisioning resources, performing problem management, handling end-to-end security for on demand Operating Environment services, or managing data placement.

- Resource virtualization services

  Provide the instrumentation of server-, storage-, network-, and other resources, including structured (relational) and unstructured information content that is held in a variety of data sources, to enable management and virtualization of those resources under the control of on demand Operating Environment resource managers. Virtualization services include mapping requirements of business services and their components to available resources based on quality of service declarations of the service and knowledge about the current utilization of available resources.

Besides the fact that they implement very different capabilities that support a variety of on demand Operating Environment patterns, the main difference between the two categories of services is which user roles build and use them. Infrastructure elements are built by middleware providers and ISVs while integration elements are built by on demand infrastructure and application builders.

One of the most important insights of the on demand Operating Environment is that a common pattern supports both application services and infrastructure services. For example:

- Adapters enable integration of existing infrastructure components into the Enterprise Service Bus.
- Service choreography is often used for scripting of M-A-P-E execution plans.
- The Enterprise Service Bus provides the infrastructure for exchange of events between managed elements and autonomic managers.
- End users interact with infrastructure services through the portal user interaction services.

3

Web services and service-oriented architecture

This chapter provides an introduction to service-oriented architecture (SOA). It also introduces Web services as an implementation of SOA. The primary goal is to be explicit concerning the design principles in order to assist architects and designers in creating SOAs that are likely to achieve the promoted benefits. In this chapter, we discuss the following topics:

- Drivers for Web services and SOA
- An overview of SOA
- An overview of Web services architecture
- The combined benefits of Web services and SOA
- Where to find more information

3.1 Drivers for Web services and SOA

The implementation of SOA using Web services technologies is the current state of the art in systems integration. Both topics are covered extensively in industry literature (see, for example, the sources listed in 3.7, “Further information” on page 68), but there is some variation in their description, so an introduction is provided here to place the remaining content of this redbook in context.

For some time, the vision of much of the IT industry has been to achieve rapid, flexible integration of IT systems across all elements of the business cycle. The drivers behind this vision include:

- Increasing the speed at which businesses can implement new products and processes or change existing ones
- Reducing implementation and ownership costs
- Enabling flexible pricing models by outsourcing elements of the business or moving from fixed to variable pricing, based on transaction volumes
- Simplifying the integration work that is required by mergers and acquisitions
- Achieving better IT utilization and return on investment
- Simplifying the enterprise architecture and computing model

Really achieving these goals affects the entire scope of a business’s processes and IT systems, as depicted in Figure 3-1 on page 35. Such pictures should be familiar to anyone with an interest in Enterprise Application Integration, Business-to-Business, or Portal technologies, but it is fair to say that, perhaps until recently, the industry has lacked a consistent and comprehensive approach to technology and architecture on this scale. Although several systems that cover some elements of this scope have been implemented, there has not been a single, broadly accepted approach.

The combination of Service Oriented Architecture, an approach that draws together proven techniques from several proceeding architecture and design styles, with new open standards and integration technologies has the potential to provide such a consistent approach.

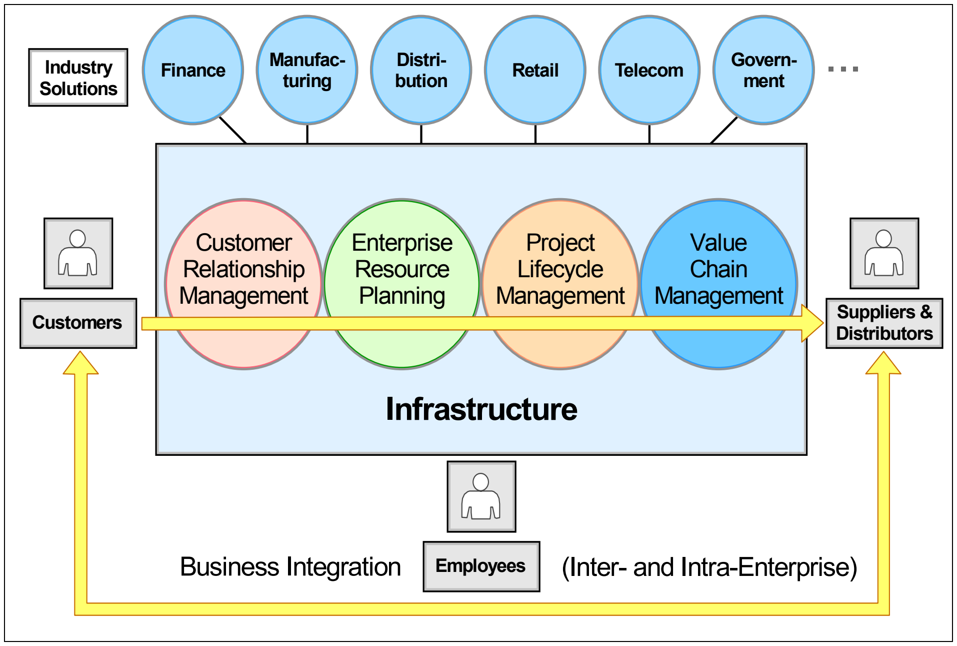


Figure 3-1 Integration across the value chain

In order to describe why both Web services and SOA are necessary to achieve these goals, it is informative to consider the specific technical issues that arise in any attempt to flexibly integrate systems on the scale that we are discussing:

- Business systems are implemented using a multitude of technologies and platforms.
- Business processes are implemented by a mixture of people practices, application code, and interactions between people and systems or systems and systems.
- Changes to one system tend to imply ripples of change at many levels to many other systems.
- No single, fully functional integration solution will talk to all of the systems in the enterprise.
- Deployment of any single, proprietary integration solution across the enterprise is complex, costly, and time-consuming.
- All issues that are involved in internal integration are encountered again when integrating with partners and their systems.
- There is no single data, business, or process model across or beyond the enterprise.

- Not all integration technologies work as well across a wide area network or the Internet as they do across a local area network, perhaps due to:
  - The use of exotic protocols.
  - Constraints imposed by security technologies, including firewalls.
  - Constraints imposed by network bandwidth.

As we discuss Web services and SOA in this section, we see how those issues are addressed; particularly, it is only the appropriate combination of both the Web services technology and the SOA approach that enables us to address them all on the broadest scales. In that vein, we should take stock briefly of what both Web services and SOA have achieved separately to date:

- Most significant SOAs are proprietary or customized implementations based on reliable messaging and Enterprise Application Integration middleware (for example WebSphere Business Integration) and do not use Web services technologies. They have, however, demonstrated the benefits of SOA, usually within a single enterprise.
- 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, and they use HTTP (an unreliable transport) as the communication mechanism. They have, however, demonstrated the efficacy of the Web services technologies in integrating heterogeneous systems both within and among organizations.
- There are several more ambitious efforts underway using both Web services and SOA. However, many of these efforts are building significant customized infrastructure function in addition to using off-the-shelf products and technologies.

It is also worth noting that as we are dealing with enterprise integration and implementation here, we have to be aware of all of the usual requirements for enterprise class systems to, for example:

- Leverage existing assets.
- Support both customized systems and commercial off-the-shelf (COTS) packages.
- Support incremental adoption and implementation.
- Provide for loose coupling between systems.
- Incorporate synchronous and asynchronous communication and transaction models.
- Be secure.
- Support multiple programming languages and platforms.

- Handle high volumes and transaction rates that exhibit peaked behavior.
- Support global deployment, including multiple languages, currency independence, and 24/7 operations.

Finally, we should be clear that there are is no magic for achieving this. We contend that all of this is possible with Web services and SOA, but you cannot just “deploy” a “Web services SOA” and switch it on. Instead, we describe an incremental approach to designing and deploying what can become an enterprise-class SOA using Web services over an appropriate timescale.

3.2 Introduction to service-oriented architecture

Service-oriented architecture is an approach to defining integration architectures based on the concept of a *service*. It applies successful concepts proved by Object Oriented development, Component Based Design, and Enterprise Application Integration technology. The goal of SOA can be described as bringing the benefits of loose coupling and encapsulation to integration at an enterprise level.

In order to describe SOA, it is first necessary to define what we understand by a “service” in this context. This is key as, unless we are confident that the services that we define really are *well designed*, we cannot be sure to achieve the promoted benefits of SOA. The most commonly agreed-on aspects of the definition of a service in SOA are:

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

The use of explicit interfaces to define and encapsulate services function is of particular importance and is illustrated in Figure 3-2 on page 38. Note how the interface encapsulates those aspects of process and behavior that are common to an interaction between two systems, while hiding the specifics of each implementation. The use of interfaces to define and mediate various aspects of service interactions is discussed in 3.2.1, “Coupling and decoupling of aspects of service interactions” on page 39. By explicitly defining the interaction in this way, those aspects of either system (for example the platform they are based on) that are not part of the interaction are free to change without affecting the other system.

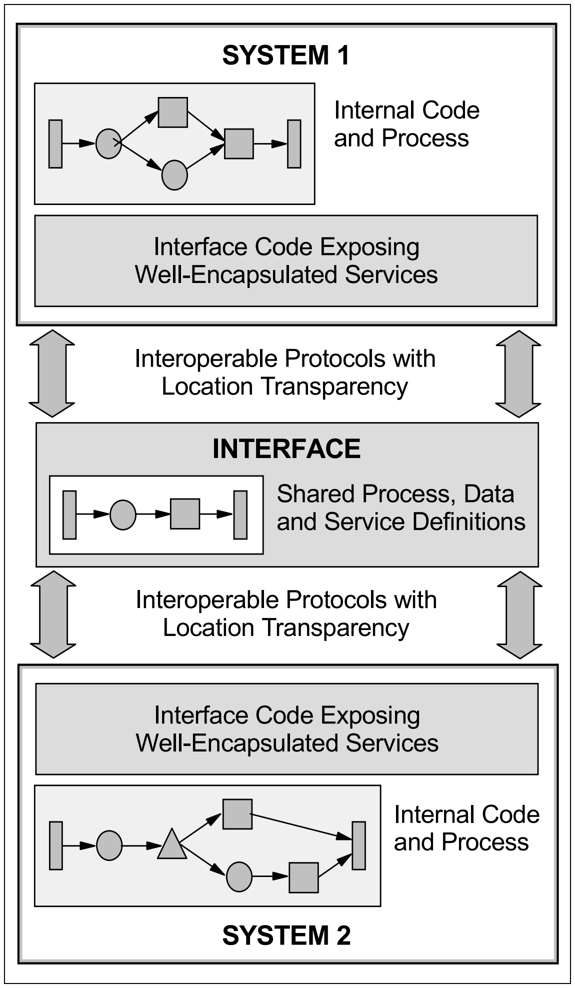

SYSTEM 1
Internal Code and Process
Interface Code Exposing Well-Encapsulated Services
Interoperable Protocols with Location Transparency
INTERFACE
Shared Process, Data and Service Definitions
Interoperable Protocols with Location Transparency
Interface Code Exposing Well-Encapsulated Services
Internal Code and Process
SYSTEM 2

Figure 3-2 The key concepts of SOA

After the function has been encapsulated and defined as a service in an SOA, it can be used and reused by one or more systems that participate in the architecture. For example, when the reuse of a Java™ logging API could be described as “design time” (when a decision is made to reuse an available package and bind it into application code), the intention of SOA is to achieve the reuse of services at:

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

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

Note that in contrast to reusing service implementations at runtime, the encapsulation of functions as services and their definition using interfaces also enables the substitution of one service implementation for another. For example, the same service might be provided by multiple providers (such as a car insurance quote service, which might be provided by multiple insurance companies), and individual service requesters might be routed to individual service providers through some intermediary agent.

The encapsulation of services by interfaces and their invocation through location-transparent, interoperable protocols are the basic means by which SOA enables increased flexibility and reusability. In order to really understand how these benefits can be achieved we delve a little further into the detail of good service design by considering these topics:

- Coupling and decoupling of aspects of service interactions
- Designing connectionless services
- Service granularity and choreography

3.2.1 Coupling and decoupling of aspects of service interactions

A basic tenet of SOA is that the use of explicit service interfaces and interoperable, location-transparent communication protocols means that services are *loosely coupled* with each other. To understand how this is implemented in practice, and how it enables the benefits of SOA, we explore the meaning of loose coupling in more detail.

By loosely coupling services, we mean restricting the number of things that the requester application code and the provider application code know about each other. If a change is made to any aspect of a service that is coupled, then either the requester or the provider application code (or, more likely, both) will have to change. If a change is made by any party (the requester, provider, or mediating infrastructure) to any aspect of a service that is decoupled, then there should be no need to make subsequent changes in the other parties.

Notice that we are no longer discussing loosely coupled services, but coupled and decoupled *aspects* of services. We can also ask whether coupled and decoupled are the only two relationships that can exist for an aspect of a service between the requester and the provider. For example, the business behavior (the function and data model) obviously must be coupled in order for the requester and provider to interact. In order to flexibly integrate systems in a heterogeneous environment, it is best that the requester and provider platforms (for example AIX® or Windows®) be *decoupled*.

However, in a realistic situation, the interactions between requester and provider must also be secured, and the relationship between their transactional models will have to be understood in order to define how failures will be handled. These and other characteristics fall somewhere between coupled and decoupled the sense that those terms are used here. (We would rather not have to include complex security and transactional function in the application code of either the requester or provider, but neither can we afford for them to be entirely independent.)

As a working framework, we define the following relationship styles for service aspects among requesters and providers:

- An aspect is *coupled* if changes to the aspect by one party in the interaction (requester, provider, or mediating infrastructure) require corresponding changes by the other parties.
- An aspect is *declared* if its behavior is specified in the interface to the service, and service requesters and providers can only interact if they have matching declared behavior, and this behavior is consistent with the capabilities of the intermediary infrastructure supporting the interaction.

  There are two variations of declared behavior that provide some additional levels of flexibility:

  – An aspect is *transformed* if it is declared by both service requesters and service providers, but the infrastructure provides some transformation capability to enable interactions between service requesters and providers that declare mismatched behavior.

  – An aspect is *negotiated* if both requester and provider declare a spectrum of behaviors that they are able to support, and if the intermediary infrastructure is capable of negotiating an agreed behavior between them for each interaction.
- An aspect is *decoupled* if changes to the aspect by one party in the interaction *do not* require corresponding changes by the other parties.

In order to clarify these ideas, it is useful to consider an example of each type of coupling:

- Business data models are usually coupled between service requesters and providers; the application code of each must understand the information that is required to describe (for example) a Customer, Account, or Order.
- Communication protocols can be declared in the service interface. In practice, this requires that applications code to a protocol-independent service API, such as a suitable implementation of JAX-RPC. If this is the case, then the

protocol binding in the service interface definition can be changed (for example, from SOAP/HTTP to SOAP/JMS). This does not require changes to the application code, but it affects the behavior of the service API implementation, which will execute the service interaction through a different protocol.

- Data formats are often transformed: It is very common, for example, to convert legacy formats, such as COBOL copybooks, to XML formats when enabling service interfaces to legacy systems. Alternatively, different XML schema may be used by different systems in an SOA to describe the same data models. In either case, the supported format is, or can be, defined in a service interface, and middleware transformation capabilities can be used in the service infrastructure to perform the required transformations without affecting application code or behavior.
- The identity of a service provider might be negotiated through a third-party broker component. The broker might use geographical location, client identify, membership scheme information, transaction value, or several other criteria to match the service requester with a suitable service provider.
- The implementation platform is often decoupled; if two systems interact through interoperable protocols such as SOAP/HTTP or messaging middleware, then neither is aware in any way of the hardware, operating system, or perhaps even the application server platform supporting the other; either party can change any or all of these aspects without affecting the other.

We can apply these relationships to various aspects of service that can be identified in a SOA. For some aspects, SOA or other design principles specify the desired style of relationship; for other aspects, several relationships might be appropriate depending on specific scenarios. For each aspect, different techniques can be applied to implement the desired relationship. Table 3-1 on page 42 identifies some service aspects, relationships, and techniques. It should be noted, however, that this area is the subject of ongoing debate and evolution, and the table should not be taken as definitive. Similarly, the available technologies are evolving rapidly (for example, the emerging WS-Policy specification will affect this area of design deeply in coming years.

Table 3-1 Service aspects, relationships and implementation techniques in SOA

Aspect	SOA intention	Techniques
Semantic interface	Coupled	► Business systems must share an understanding of the tasks and data that are processed by the service. ► Shared business object libraries or XML schemas can be exploited. ► Some transformations, aggregations or enrichments of data might be consistent with the interface semantics and implemented by the bus infrastructure; more likely such transformations would be related to service granularity and choreography, as described in “Service granularity and choreography” on page 47.
Language	Decoupled	► Language and platform-independent interface definition such as IDL, WSDL, XSD. ► Language and platform-independent data formats such as XML. ► Language and platform-independent communication protocols such as IIOP, SOAP, WebSphere MQ. ► Invocation APIs (for example, JAX-RPC), adapters, or ESB infrastructure to integrate applications to the interface definitions and data formats.
Platform	Decoupled	
Data format	Declared or Transformed	► Language and platform-independent data formats such as XML. ► Adapters, XSL style sheets, or bus infrastructure required to support transformations between data formats, such as between COBOL copybooks and XML. ► Application development tool wizards can create language-specific representations of some data formats, particularly XML. ► Other aspects of data format that are critical to real-world SOA implementations are data encoding, code pages, and data compression, including XML compression techniques.

Aspect	SOA intention	Techniques
Protocol	Declared or Transformed	► Service invocation mechanisms for service requesters and providers that do not specify service protocol or locations; for example, an implementation of JAX-RPC with support for multiple protocols. ► Adapters or ESB infrastructure can perform service routing and protocol transformation.
Location	Decoupled	
Service provider identity or implementation	Declared or Negotiated	► Service invocation mechanisms that enable service substitution, for example JAX-RPC. ► Adapters or ESB infrastructure can perform service routing to different providers. ► Directory (for example, UDDI) or broker intermediary to decide who fulfills the service each time. ► An ESB might identify a suitable service provider based on WS-Policy, for example by selecting the cheapest or most-responsive provider available at the time.
Time	Declared or Negotiated	► As IT systems show many differing planned and unplanned availability characteristics (such as 24/7 versus working hours), service interactions will sometimes have to span systems with different characteristics. ► Declared by WSDL or negotiated through WS-Policy. ► Use of asynchronous transport protocols, for example WebSphere MQ, WS-ReliableMessaging. ► ESB or intermediary store and forward capability for asynchronous request / response, message correlation, and so forth. ► Message correlation and transaction identifiers used to associate individual service interactions with longer ongoing business process interactions.

Aspect	SOA intention	Techniques
Delivery assurance, integrity, and error handling	Declared or Negotiated	► Assured delivery communication protocols; for example WebSphere MQ, WS-ReliableMessaging. ► Error and exception handling processes, for example for SOAP faults. ► Use the features and deployment descriptors of containers, such as J2EE, in service implementations. ► Advanced WS standards, for example WS-ReliableMessaging and WS-Transaction. ► Negotiated through WS-Policy by the ESB. ► In order to provide a consistent end-to-end approach to delivery assurance, integrity, and error handling for a chain of service interactions, it will often be necessary to combine several techniques that are used for individual interactions. These techniques might include handling communication failures, the use of synchronous two-phase commit, the ability to handle duplicate messages, and compensation schemes.
Security	Declared or Negotiated	► Declared by WS-Security or negotiated through WS-Policy. ► Point-to-point or communication-based security and trust models. ► Implemented through applications or through third-party or intermediary components in the SOA architecture.
Service version	Declared or Negotiated	► Service naming standards. ► Version-based routing in the bus infrastructure. ► Service request / provider tolerance of changes in optional data attributes.

Aspect	SOA intention	Techniques
Interaction state	Declared	► Matching of messages or events to long-lived processes by explict process or transaction IDs in semantic interface, or by application data (for example, customer ID). ► Service Choreography technology may provide some facility to use a variety of input data to associate messages with specific instances of processes. ► Primary key matching technology such as provided by WebSphere InterChange Server. ► The emerging WS-ResourceFramework provides a standard model for associating services with stateful resources. ► Enterprise Application Integration middleware support for message aggregation and correlation. ► Customized solutions involving custom message headers.

It is interesting to note in the table that the only aspect of the service that is specified as coupled is the business behavior. By specifying other aspects to be declared, transformed, negotiated, or decoupled, the intention is to build the maximum possible flexibility into the architecture, enabling other aspects of service implementation and interaction to vary as freely as possible.

In combination with the flexibility of business behavior that is achieved by encapsulating well-designed business function as services, SOA attempts to maximize the overall flexibility of integrated business systems. The next two sections discuss some aspects of what is meant by encapsulating well-designed business function as services, in order to ensure that the flexibility of behavior really is achieved.

3.2.2 Designing connectionless services

The question of whether services should be stateful or stateless has been discussed frequently in relation to SOA. However, the issue is complicated by whether it really is possible to draw a clear line between state and business data. Many service interactions must be stateful in order to play a role in ongoing business processes or interactions; the issue is how we should design such services so as to maximize the flexibility of the architecture and the processes it supports.

To answer this issue, we return to the key to SOA: defining behavior in the interface. In doing so, we do away with considering stateful and stateless services, and instead declare that services, whether they deal with stateful business behavior (for example, the renewal process for an insurance policy) or stateless business behavior (such as performing an exchange rate calculation) should be *connectionless*.

Connectionless services are those that do not allow or require a service requester and a specific, executable instance of the service provider to maintain a relationship between service invocations. The successful implementation of connectionless services depends on two considerations:

- The use of technology that prevents handles being retained to specific executable instances.
- The design of service interfaces that do not depend on implicit, shared knowledge created through a sequence of interactions between a specific requester and provider.

The first consideration is relatively easy to address: When stateless protocols such as asynchronous messaging are used to invoke services, this criterion is fulfilled. When technologies that are capable of supporting stateful behavior are used, the features of the technology that manipulate state (for example, HTTPSession or cookies) should not be used. Meeting this criterion might imply the use or assessment of specific communication technologies, or the application of design and development guidelines to the implementation of systems that participate in the SOA.

The second criterion can only be fulfilled through its application as a design principle to the design of the individual service interface. Table 3-2 shows an example of both a connected and connectionless design for two services. In this example, a store system in a consumer electronics shop is trying to charge the cost of an expensive television to a card account that belongs to Bruce; the cost is high enough that the store system must explicitly authorize the transaction with the card supplier.

Table 3-2 Connected and connectionless service interactions

Connectionless	**Connected**
Service Client: Can Bruce pay $1000 for a new television? Service Provider: Yes Service Client: Charge *Bruce* $1000 for a new television Service Provider: OK	Service Client: Can Bruce pay $1000 for a new television? Service Provider: Yes Service Client: Charge *him* $1000 for a new television Service Provider: OK
All of the business data is defined in the interface	Part of the business data (the fact that we are dealing with Bruce) is implied in the sequence

The Connectionless example in the table shows that the interface to each call specifies all of the data that is required to perform the service, other than business information owned by the service provider. For example, Bruce's card balance and credit limit are not part of the service interface because they are business information owned by the card provider. However, the fact that Bruce is the owner of the account that is relevant to this specific transaction is a part of the interface, because both the service to authorize the payment and the service to make the payment must relate to the same cardholder. If this information were not part of the service interface (as in the Connected example), then the specific, executable instances of the service client and the service provider would have to maintain a reference to each other in order to identify the correct context.

This could cause difficulty if, for example, the physical server that supports the instance of the service provider crashed; in such as case, what happens to the instance of the service that remembers that it was dealing with Bruce? Was the state of that instance safely stored somewhere prior to the failure, perhaps in a database? If so, how does the service requester then connect to another executable instance of the service that somehow knows which information to read from the database? All of these issues should be familiar to anyone who has developed stateful distributed applications, such as J2EE or WebSphere applications that make use of the Java HTTPSession object.

The principle that services should be designed to be connectionless is really saying that for a shared sequence of activity, each instance of that activity should be identified uniquely (for example, through a transaction ID or, in this case through a customer ID, Bruce), and that the identity should form part of all service calls. Even better would be to explicitly define and share the process definition, as the emerging BPEL4WS standard could do.

3.2.3 Service granularity and choreography

Many descriptions of SOA also refer to the use of “large-grained” services. However, some powerful counterexamples of successful, reusable, fine-grained services exist. For example, getBalance is a very useful service, but hardly large-grained.

More realistically, there will be many useful levels of service granularity in most SOAs; for example:

- Technical functions (such as logging)
- Business functions (such as getBalance)
- Business transactions (such as openAccount)
- Business processes (such as applyForMortgage)

Some degree of choreography or aggregation is required between each granularity level. It is unlikely that all organizations will share identical definitions of granularity, but each will undoubtedly find it beneficial to define their own. At each level of granularity, it is important that service definitions encapsulate function well enough that it is reusable. Figure 3-3 shows an example of service granularities and choreographies between them.

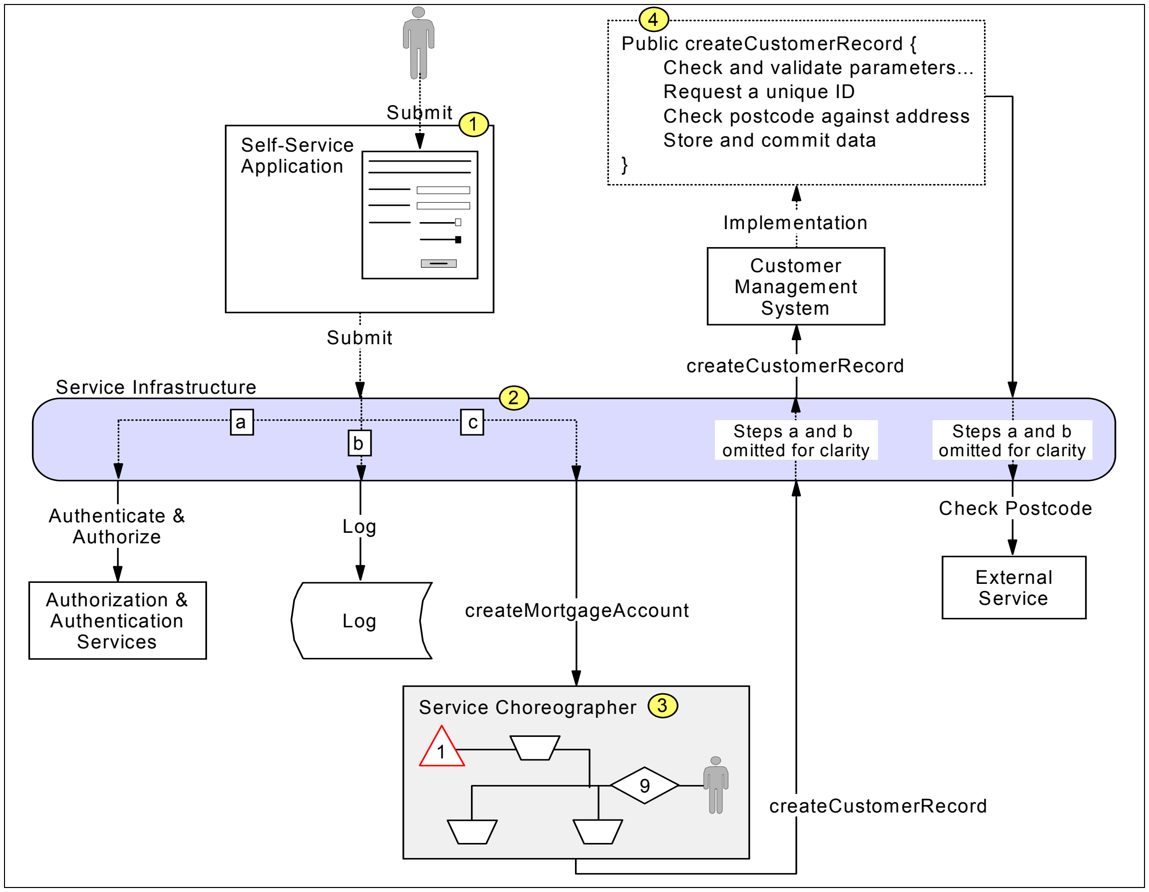


Figure 3-3 Service granularity and choreography

Figure 3-3 describes these interactions among services of various granularities:

1. A user submits a request to a self-service application to create a mortgage account. The self-service application submits the business process service request createMortgageAccount through the service infrastructure to a service choreographer component, whose purpose is to choreograph business transaction services into business process services.
2. On receiving the request for the createMortgageAccount business process service, the service infrastructure first invokes *authentication* and *authorization* technical function services to ensure that the request is valid,

then a *log* technical function service before finally invoking the createMortgageAccount business process service in the service choreographer.

3. The service choreographer executes the createMortgageAccount business process service. If the request is valid, then when the other process elements are complete the choreographer invokes the createCustomerRecord business transaction service through the service infrastructure to store the details of the new customer. (Before doing this, it may already have invoked storeMortgageDetails.)
4. In the implementation of the Customer Management System createCustomerRecord business transaction service, it is necessary to validate the information for the new customer. Part of this validation is checking whether the post code and address match. In order to do this, a CheckPostCode business function service is invoked through the service infrastructure.

To summarize, three aggregations or choreographies are performed by distinct components for distinct granularity levels:

Service choreographer	Choreographs business transaction services into higher level business process services.
Service Infrastructure	(may be an Enterprise Service Bus) Choreographs technical function services to control the invocation of business process services, business transaction services, and business function services.
Individual application components	Responsible for invoking business function services where they are required in order to implement business transaction services.

Of course, this is just one hypothetical example. Real organizations must formulate their own definitions.

Large-grained interfaces simplify coupling between processes

A recurring issue in system design and implementation is building interactions between two systems that have to share the execution of a process, in such a way that the process is flexible and can enable other systems to participate. A good example is in the design of Web browser applications, where the process of naviagating through browser screens is matched by the application code to some elements of a business process. A typical example of the impact when this matching is not performed well is what happens when a new interface with a different sequence of screens, such as a mobile phone interface, is added. All too often, significant changes are required to the application business logic.

Given that the business process did not change, it would be better if changes to business logic were not required.

The use of large-grained services can help to match such processes in a more flexible manner. In order to illustrate this, consider the two examples in Table 3-3.

Table 3-3 Different interaction styles in applying for a mortgage

Service-like	**API-like**
By Post: Client requests application form. Provider sends it. Client fills it in and returns it. Provider says `yes` or `no`.	By Telephone: Client calls provider. Provider says "Hello, how can we help?" "I'd like a mortgage, please." "What's your name?" "John Smith." "What's the property address?" "27 ..." ...and so forth... "... Your mortgage agreement number is 12345, I'll post the rest of the details."

The second approach, By Telephone, consists of a large number of fine-grained interactions. Both parties, the applicant and the worker in the call center, maintain an implicit knowledge of where they are in a conversation. If, for example, the caller must be away from the telephone for a period of time (perhaps to answer the door), then either the call center worker must hold the line (which is not good for productivity), or the caller will have to call back later, when another worker will have to scan records for the previous interaction and determine the point to resume the process.

The first approach, By Post, is much simpler. At the start of the interaction, the mortgage provider publishes the information that is required to process a mortgage application. The client may collect that information through any process they prefer, over any period of time, without further involving the mortgage provider’s resources. When ready, the client may return the form by post, visit the nearest office, or telephone a call center and read the information over the telephone. Either way, much more flexibility of behavior is allowed, because a single, large-grained entity was published as the interface to a service with a clear business purpose. In contrast, By Telephone requires many individual interactions, and is concerned with many separate elements of data with no individual significance to the business process, and which are not explicitly defined in an open manner. Even if the call center worker follows a script, that script is available only to one party.

In this sense, large-grained services tend to be more flexible because they reflect the underlying business process and changes in state of business data rather than the specifics of any one interaction style. Similarly, a failure to focus on interactions that are meaningful to business processes (rather than those that are specific to individual pages in a Web application, for example) are part of the difficulty that is experienced in opening several early Web applications to additional channels, such as mobile phones and PDAs, with very different interface devices that require very different screen arrangements.

In the By Telephone example, the mortgage application is in the same business state — incomplete — for most of the interactions. It is in different states only at the very start and very end: "new application" and "complete application." These are the two most obvious services that are required to implement the interaction, and they map well to the description of the By Post example.

In a pragmatic sense, any system that implements this also requires the ability to make general updates to a transaction that is in the "incomplete application" state, so we will need additional services. For those services to reflect the business process rather than the characteristics (sequence of screens) of any one application, we should design a general "update application" service, rather than specific services such as "update customer name," "update property address," and so on.

Large-grained interfaces can be tolerant to certain changes

It is often asserted that the use of large-grained service definitions inherently leads to more flexible systems, or that the use of simple interfaces leads to more stable interfaces. It is worth examining why this may be so.

For example: An organization that has grown by merger and acquisition stores data describing some customers in one database and data describing other customers in another. The organization might implement an "update customer details" service that provides common access to both systems, perhaps with some routing capability with knowledge of which customers are stored in which database, perhaps based on different formats of the Customer ID field.

Now consider that a new company is acquired and brings with it a third customer database. This new customer database has a marital status field that the first two do not. Therefore, the "update customer" interface to this database includes a field that the other databases do not have.

If the "update customer details" service had originally been defined as accepting first name, second name, address line 1, address line 2, date of birth... (each individual attribute of a customer), we would now have a problem: the third database requires a new attribute, and so requires a different service interface, so there are now two distinct update customer details services.

We could have designed the “update customer details” service to accept an XML data type that is defined through an XML schema as a Customer type. When the third customer database is added, the marital status field could be added to the XML schema as an optional attribute. The same service interface definition can then be used to define the update customer details interfaces to all three databases without requiring changes to the existing two systems.

Of course, in practice it may not be that easy. Applications that are used to capture changes to customer data may have to be changed to manipulate the extra field. Or if an update request that includes a marital status field is sent to a customer database that doesn’t recognize it, the database may ignore the additional field or be unable to process the request. Is it meaningful to the business to ignore data to this way, and will it match customer expectations?

Generally, there are potential uses for such flexibility, but it will not be appropriate in all cases and may in fact be difficult for some applications and some technologies to implement. Overall, it is more important that the granularity of the service definition results in the encapsulation of reusable function that makes sense to service clients than for it simply to be large. And there is no escaping the need to define a governance process and technical means to support the evolution and versioning of service interfaces over time.

3.2.4 Implications of service-oriented architecture

The encapsulation of reusable business function, the achievement of loose coupling, the definition of appropriate levels of granularity, and so forth are analysis issues as much as a technology issues. They are difficult issues to grasp, so SOA cannot be successful without skilled architects and designers who understand and are able to articulate them. It is easy to see these concerns becoming hostage to time, skill, and cost issues, leading to another generation of isolated systems that will require integration.

Widespread implementation of an SOA and infrastructure is a long-term endeavour that involves all of the usual hard business decisions, questions of data, and process ownership. It requires serious, long-term commitment by business and by the IT organization that supports it. It may involve upfront costs, centralized costs, and many other challenges:

- No specific technologies are ruled in or ruled out.
- Legacy implementations are possible (for example, CICS® Transaction Server “super router” transactions with simplified, text-based interfaces)
- EAI implementations are commonplace (for example, XML over MQ / WebSphere Business Integration Message Broker)
- Web services are potentially a very good fit, but are still maturing.

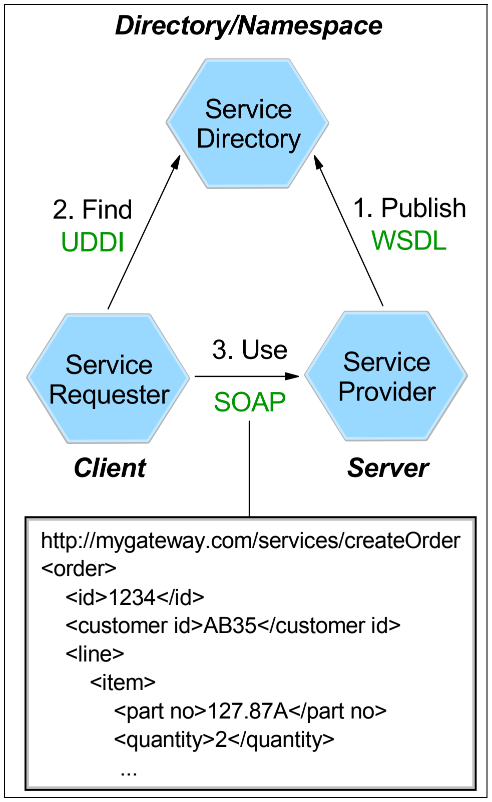
3.3 Web services architecture

Web services are a recent set of technology specifications that leverage existing proven open standards such as XML, URL, and HTTP to provide a new system-to-system communication standard. Based on this communication model, additional higher-level Web services standards have also been defined to address transactions, security, business processes, and so forth: the higher-order functions that are required to get systems, applications, and processes (rather than objects and components) talking to each other.

Web services learn from the way the Web revolutionized how people talk to systems: new customers, new business models, extensions of opportunity, new transparency and improved collaboration between employees and employers, and in some cases reductions in infrastructure costs and complexity. The key to these successes was a universal server-to-client model that is consistent with a highly distributed environment, based on simple open standards and industry support.

Web services promises to do the same thing for the way systems talk to systems: integrating one business directly with another so that the process doesn't have to wait for people to provide the glue, get your own business talking to itself or your partners to provide integrated IT systems, and again the potential for dramatic reductions in infrastructure costs and complexity. Once again, the key is a universal program-to-program communication model based on simple open standards and industry support.

Figure 3-4 on page 54 shows the basic interaction model supported by Web services.



```
http://mygateway.com/services/createOrder
<order>
   <id>1234</id>
   <customer id>AB35</customer id>
   <line>
      <item>
         <part no>127.87A</part no>
         <quantity>2</quantity>
          ...
```

Figure 3-4 Basic Web services

Basic Web services define interactions among Service Requesters, Service Providers, and Service Directories as follows:

Service Requesters find Web services in a UDDI Service Directory. They retrieve WSDL descriptions of Web services offered by Service Providers, who previously published those descriptions to the Service Directory. After the WSDL has been retrieved, the Service Requester binds to the Service Provider by invoking the service through SOAP.

The basic Web services are often described in terms of SOAP, WSDL, and UDDI, each of which we define and discuss. However, it should be noted that each of these standards can be used in isolation, and there are many successful implementations of SOAP alone, or SOAP and WSDL, in particular.

SOAP

SOAP is an XML messaging protocol that is independent of any specific transport protocol. SOAP defines a framework within which messages contain headers, which are used to control the behavior of SOAP-enabled middleware, and a message body. As SOAP is an XML format, and as XML is text-based, SOAP is supportable in the vast majority of existing and new technical environments and can be transported over a vast variety of protocols.

In practice, SOAP is most often communicated over HTTP, although this is likely to evolve rapidly because HTTP is an unreliable protocol. (For instance, it is already possible to send SOAP messages through JMS implementations such as WebSphere MQ.) Basic SOAP also makes no reference to characteristics of interactions such as security and transactionality. However, as SOAP headers provide an extensible model, these aspects are being added gradually to the Web services specifications as extensibility elements, as we describe further in the next section. The use of SOAP over specific protocols, such as HTTP, is usually written as SOAP/HTTP, SOAP/JMS, and so forth.

The SOAP V1.2 specification is available from the World Wide Web Consortium, and deliberately does not define a meaning for SOAP as an acronym. (SOAP is sometimes referred to as Service Oriented Architecture Protocol, or by its definition in the more widely supported SOAP V1.1 specification, Simple Object Access Protocol.)

WSDL: Web Services Description Language

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

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

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

UDDI: Universal Description, Discovery, Integration

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

A public UDDI directory is provided by IBM, Microsoft, and SAP, each of whom runs a mirror of the same directory of public services. However, there are many patterns of use that involve private registries; see Steve Graham's articles:

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

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

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

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

SOAP/HTTP uses existing namespaces and infrastructure

One of the important potential benefits of Web services is to reduce the reliance of integration on specific integration technologies that require heavyweight deployment where such deployment would be problematic or impossible — typically, in business-to-business interactions, particularly as they become more widespread and dynamic.

The specific use of Web services with HTTP as a communication protocol has some extraordinary benefits in this area. Because SOAP/HTTP uses HTTP as a communication protocol and URL as the addressing format, the entire global network of distributed, resilient routing and communications infrastructure that the Internet provides can be used. Allowances must be made for the unreliable nature of HTTP, but the advantages of a service communication protocol that is already deployed and globally pervasive should not be underestimated.

3.3.1 Web services interoperability

A unique feature of Web services is that it is a relatively high-level integration protocol with near-ubiquitous support in the IT industry; this alone is an important reason for its success and is behind why many individual projects have used the Web services standards to perform integrations between different platforms.

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

The WS-I published the first profile for interaction, the Basic Profile, in July 2003, and many technology vendors provide product implementations of Web services that are compliant with this profile, which is described further in "WS-I Basic Profile V1.0" on page 57. The WS-I is creating a Basic Security Profile to describe interoperability using the Web services security (WS-Security) standards. A draft specification of this profile was published in February 2004.

Of course, interoperability can be achieved using Web services where WS-I profiles do not exist; however, it may be more limited or require additional work to achieve. Therefore, the WS-I is an important mechanism for assuring and simplifying interoperability between implementations of Web services standards as those standards mature and evolve.

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

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

WS-I Basic Profile V1.0

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

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

- SOAP V1.1
- WSDL V1.1
- UDDI V2.0
- XML V1.0 (Second Edition)
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes
- RFC2246: The Transport Layer Security Protocol Version 1.0
- RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile
- RFC2616: HyperText Transfer Protocol V1.1
- RFC2818: HTTP over TLS
- RFC2965: HTTP State Management Mechanism
- The Secure Sockets Layer Protocol Version 3.0

The WS-I Basic Profile 1.0 - Usage Scenario consists of three usage scenarios, where a usage scenario is a design pattern of interacting entities including actor, roles, and message exchange patterns:

- One-way usage scenario
  - Simplest usage scenario in which the message exchange is one-way, with a consumer sending a request to a provider.
  - Should be used only when loss of information can be tolerated.
- Synchronous request/response usage scenario
  - Most commonly used usage scenario: A consumer sends a request to a provider, who processes the request and sends back a response.
- Basic callback usage scenario
  - This is used to simulate an asynchronous operation using synchronous operations.
  - Composed of two synchronous request/response usage scenarios, one initiated by a consumer and the other by a producer.

The WS-I Supply Chain Management sample application depicts an application for a fictitious consumer electronics retailer. This sample application is the basis of the scenarios in this Redbook, and is described in Chapter 7, “The business scenario used in this book” on page 169.

See also the following IBM developerWorks articles:

- First look at the WS-I Basic Profile 1.0

  `http://www.ibm.com/developerworks/webservices/library/ws-basicprof.html`
- First look at the WS-I Usage Scenarios

  `http://www.ibm.com/developerworks/webservices/library/ws-iuse/`
- Preview of WS-I sample application

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

3.3.2 Advanced and future Web services standards

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

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

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

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

Web services security

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

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

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

Support for the basic WS-Security standards is available in existing products and can be used to implement secure Web Services solutions. As described in "Security issues affecting the Enterprise Service Bus" on page 89, understanding the security requirements of specific SOA situations and selecting appropriate technologies, include those compliant with the WS-Security standards, is a key decision in SOA implementation.

Further information

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

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

- Web Services Security: Moving up the stack

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

WS-ReliableMessaging and SOAP/JMS

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

The WS-ReliableMessaging specification defines a protocol for reliable communication (including SOAP messages) that use a variety of communication technologies, which may themselves be less reliable. An updated specification was published in March 2004, but production support is not yet available in middleware products.

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

Further information

- Updated: Web Services Reliable Messaging: A new protocol for reliable delivery between distributed applications

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

- Implementation Strategies for WS-ReliableMessaging: How WS-ReliableMessaging can interact with other middleware communication systems

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

Business Process Execution Language for Web Services

As the encapsulation and exposure of business functions as services in an SOA enables the definition of processes consisting of those services, the Business Process Execution Language for Web Services (BPEL4WS) provides a standard, XML language for expressing business processes consisting of functions that are defined through WSDL interfaces. BPEL4WS supports both short-lived processes and long-lived processes (processes that must wait at certain points until some event occurs, such as the receipt of an event).

As with WSDL, BPEL4WS has both design time and runtime uses. At design time, development or modeling tools can use, import, or export BPEL4WS to enable business analysts to specify processes and developers to refine them and bind process steps to specific service implementations. However, runtime choreography and workflow engines can use BPEL4WS to control the execution of process and invoke the services that are required to implement them.

Although BPEL4WS is a relatively new standard, product support such as WebSphere Business Integration Server Foundation V5.1 is available. This provides additional facilities to compensate failed processes (a proprietary equivalent to the WS-BusinessActivity standard described in the next section, “Web services transactions”) and provide a user workflow interface to enable human actions to fulfill WSDL-defined steps in a BPEL4WS process.

Further information

- BPEL4WS specification

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

- Business Process with BPEL4WS, a series of introductory articles and references

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

- BPEL4WS support in WebSphere Business Integration Server Foundation

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

- BPEL4WS support in WebSphere Studio Application Developer Integration Edition

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

Web services transactions

Although WS-ReliableMessaging will provide a means to assure the delivery of individual communications in a Web services interaction, a means is also required to control the integrity of business transactions in an SOA that consist of one or more Web services invocations or interactions.

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

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

Further information

- WS-AtomicTransaction specification

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

- WS-BusinessActivity specification

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

- Transactions in the world of Web Services, part 1 and part 2

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

- WS-Coordination specification

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

Web Services Policy Framework (WS-Policy)

The Web Services Policy Framework is intended to provide a set of languages by which service requesters and providers can express their requirements and capabilities concerning QoS of service interactions, such as security, transactionality, and communication reliability. Eventually a framework of such languages, supported by Enterprise Service Bus middleware, will enable open-standard implementations of negotiated coupling between various aspects of service interactions. (See 3.2.1, “Coupling and decoupling of aspects of service interactions” on page 39.)

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

Further information

- WS-Policy framework specification

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

- Web Services Policy Framework: New specifications improve WS-Security

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

Web Services Resource Framework (WS-ResourceFramework)

As we write this book, WS-ResourceFramework is an architectural proposal rather than a standard, but it relates to some of the aspects of SOA that we have only touched on in the discussion of Web services (namely the design, rather than technical, characterization of services). In 3.2.2, “Designing connectionless services” on page 45, we discussed the relationship between service definitions, processes and stateful behavior, and suggested some techniques for designing flexible services to participate in stateful interactions.

In order to enable middleware to provide increasingly sophisticated support for such stateful interactions, the Web Services Resource Framework provides a model for associating Web services with stateful resources (for example data, such as rows in a database), as opposed to stateful processes (as can be accomplished with BPEL4WS), which is essentially a model for making Web services middleware and infrastructure aware of stateful identifiers such as transaction IDs.

Further information

- WS-ResourceFramework overview

  `http://www.ibm.com/developerworks/webservices/library/ws-resource/ws-wsrfpaper.html`

3.4 Emerging infrastructure components for Web services and SOA

In addition to the ongoing evolution of the basic SOA concepts and Web services technologies, architectures and infrastructures for SOA are evolving some common components. These components are increasingly forming the basis for packaged technologies that are offered by IT vendors. In this section, we discuss a few of the more obvious or important infrastructure and architecture components.

Enterprise Service Bus

Although the basic Web services technologies, particularly SOAP/HTTP, can provide a certain quality of service to SOA by simply using existing Internet and intranet infrastructures, many enterprise requirements demand higher qualities of service, which require a dedicated infrastructure. This infrastructure must support both the established basic Web services technologies, established middleware communication technologies such as WebSphere MQ, and, eventually, emerging standards such as WS-ReliableMessaging.

Similarly, by just enabling or adding Web service interactions that use existing Internet and intranet infrastructure between systems in an architecture, many individual point-to-point integrations can be created. This has long been known to be difficult to maintain and evolve, hence the broad use of Enterprise Application Integration middleware supporting hub-and-spoke integration patterns.

These requirements to provide an appropriately capable and manageable infrastructure for Web services and SOA are coalescing into the concept of the

Enterprise Service Bus, which will be the subject of much of this redbook, from Chapter 4, “Enterprise Service Bus and SOA patterns” on page 73 onward.

Service directories and brokers

Although the UDDI directory specification was one of the earlier Web services specifications, it is implemented in only a small number of Web services implementations. Many Web service or SOA implementations use either simpler, design-time directories (perhaps based around collaboration technology) or customized service directories (often using database technology).

However, a key benefit of SOA, particularly in enabling on demand solutions, is a more flexible selection of service providers. This can give businesses the choice of either providing their own service implementations to support their business processes or selecting from services provided by partner organizations. Some service providers may implement the service themselves, and others might be brokers for several end-service providers.

Similarly, as Web services standards mature to support increased security, trust models, and declarative policies, the use of services that are discovered dynamically in public directories may become a more attractive option.

Several SOA implementations have already implemented some or all of these ideas, often with significant customized development. In time, products, standards, and architecture and design patterns are likely to evolve toward a number of well-defined intermediary models, including directories and brokers.

Service choreography

The desire to explicitly model and execute business processes is nothing new: Business analysis tools have been available for many years, and workflow function has been implemented in many technologies, from legacy systems to packaged applications to collaboration and groupware technology.

However, the emergence of service-oriented architecture and the Web services standards is opening new opportunities in this area. The SOA principles provide guidelines for defining services and processes that are likely to be more flexible and can be implemented in existing systems. The Web services technologies provide new, standard means of exposing and defining those services, and choreographing them into business processes.

Although this is still a new area, the appropriateness of SOA and Web services to long-standing requirements to model and automate processes is strong enough that this trend is likely to grow swiftly over the next few years.

User access to services

Service-oriented architecture specifies the use of interfaces to define encapsulated, reusable business function: in part, those interfaces identify a business function and specify the data required to interact with it. This is precisely the purpose of many application user interfaces: to enable users to identify a function, collect the data required to invoke it, and return the outcome to the user.

This correspondence has led to several interesting patterns emerging in providing user access to services and Web services:

- Portal technologies, such as WebSphere Portal Server, offer the capability to automatically present some Web services as portlets; however, this is often dependent on the addition of specific display-related information to the Web services description.
- The Oasis Remote Portlet Web Services specification provides an open standards means to exposed Web services in a manner that is suitable for display by portal technology, but the standard is relatively recent so full product support may take some time to emerge.
- The World Wide Web Consortium recently published the xForms specification for device-independent description of the data model for user interfaces. The content of xForms descriptions bears several similarities with that of WSDL descriptions of the data that is required by and returned from a service. If a service interface can be transformed manually or programmatically into an xForms definition, then xForms UI generators can be used to generate a variety of Web-based, desktop, or other UIs. The xForms specification can be found at:

  `http://www.w3.org/TR/xforms/`

3.5 Web services and SOA together

The link between Web services and SOA is threefold:

- Web services provide an open standard and machine-readable model (WSDL) 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 BPEL4WS, document-style SOAP, and WSDL, and emerging technologies such as WS-ResourceFramework to support the technical implementation of well-designed services that encapsulate and model reusable function in a flexible manner.

Together, Web services and SOA have the potential to address the technical issues that we introduced at the start of this section:

- (WS) A multitude of technologies and platforms support your business systems.

  Web services are a set of open-standard technologies that are supported by most of the IT industry and by the Web Services Interoperability organization. Their basis in simple, text-based, and open-standard technologies such as XML and HTTP, and the fact that they can leverage more sophisticated interoperable technologies such as asynchronous messaging, means that they can be supported in the vast majority of IT environments. Increasing ubiquity and maturity of product support means that implementing and integrating Web services will become increasingly efficient.

- (SOA) Business process models are a mixture of people practices, application code, and interactions among people and systems or systems and systems.

  Although SOA is an approach to architecture that must be applied to systems and integrations, it specifies a set of principles and techniques that encourage the encapsulation and modeling of reusable business functions and processes. Recent and emerging trends in Web services, such as BPEL4WS and WS-ResourceFramework, will increasingly support the modeling concepts of SOA.

- (SOA) Changes to one system tend to imply ripples of change at many levels to many other systems.

  SOA specifies several principles and techniques for achieving the encapsulation of service function and the loose coupling of service interactions. These techniques minimize the cases where change to one part of a system implies changes to other parts to those cases where the implied changes are necessary to support the underlying changes to the way the system supports the business.

- (WS) No single, fully functional integration solution will talk to them all.

  At one level, the use of widely available and interoperable basic Web services open standards such as SOAP/HTTP with existing Internet and intranet infrastructure provide an integration solution that already has impressive reach, and it will become increasingly ubiquitous. Where increased manageability and qualities of service are required, emerging Enterprise Service Bus middleware, which combine Web services and SOA concepts with the power of traditional Enterprise Application Integration middleware technology, will provide a sophisticated and widely interoperable integration infrastructure.

- (WS) Deployment of any single, proprietary integration solution across the enterprise is complex, costly, and time consuming.

  Where basic Web services that utilize existing infrastructure are appropriate, deployment costs and efforts are minimal. The increasing availability of Web services support in Enterprise Application Integration middleware also enables the integration of different middleware infrastructures. Similarly, emerging Enterprise Service Bus technologies will interact with existing integration infrastructure rather than automatically replace it. So, although SOA and Web services cannot remove the cost and effort of deploying integration infrastructure, they offer several characteristics to minimize it.

- (WS) Assuming that you get past that, will your integration solution talk to your partners? Your future partners?

  The Web services technologies have proven effective in many B2B integrations, where their open standards basis and use of simple, existing infrastructure and protocols makes them particularly effective. Recent and emerging standards such as WS-Security add to the sophistication of interaction that is possible when using Web services in this model.

- (SOA) There is no single data, business, or process model across (or beyond) the enterprise.

  Although they are not a magic solution, the SOA principles define an approach that enables organizations to progressively expose functions across their business as services and to combine those services into process. Over time, businesses that take this approach will improve the consistency of their business and process models, and will leverage the use of business process modeling and automation technology to more explicitly control and monitor their execution of processes.

- (WS) Not all integration technologies work as well across a wide area network or the Internet as they do across a local area network

  The Web services technologies support multiple protocols, so they can use the simplest protocols available, such as HTTP when that offers an advantage, or leverage other infrastructures such as WebSphere MQ when that is more appropriate.

3.6 Conclusion

We began this chapter by discussing the factors that drive businesses to consider service-oriented architecture and Web services, and we elaborated on the technical issues that make it hard (in practice) to achieve goals of flexible integration and process automation that are commensurate with those factors.

We have discussed how the key features of SOA and Web services enable us to address those technical issues, and we have offered new opportunities for more flexible, rapid, and widespread integration, in a model that is consistent with the exposure of business function as services, and the choreography of those services into processes that can be modeled, executed, and monitored:

- *Service-oriented architecture* defines concepts and general techniques for designing, encapsulating, and invoking reusable business functions through loosely bound service interactions. Most of the techniques have been proven individually in previous technologies or design styles. SOA unites them in an approach intended to bring encapsulation and re-use to the enterprise level.
- *Web services* provide an emerging set of open-standard technologies that can be combined with proven existing technologies to implement the concepts and techniques of SOA.
- *Industry support* for Web services standards, interoperability among different implementations of Web services, and the infrastructure technology that is required to support an SOA give technology customers increasingly mature and sophisticated technologies that are suitable for SOA implementation.

These techniques and technologies give companies the tools that are required to implement flexible SOAs and evolve toward an on demand business model. However, at the current time and for some time to come, the technologies will be evolving rather than mature and stable. Therefore, individual SOA solutions must make carefully balanced solutions among customized, proprietary, and open-standard technologies, which characteristics and components of SOA to implement, and which areas of business function and process to apply them to. Of course, these decisions will be balanced between business benefits, technology maturity, and implementation or maintenance efforts.

3.7 Further information

Consult these sources for more information:

- The IBM Redbook *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303
- "Web Service Oriented Architecture - The Best Solution to Business Integration," by Annrai O'Toole, Cape Clear Software CEO, at:

  http://www.capeclear.com/clear_thinking1.shtml
- "SOA - Save Our Assets," by Lawrence Wilkes, CBDI Forum (subscription required) at:

  http://www.cbdiforum.com/report_summary.php3?topic_id=2&report=623&start_rec=0

- The IBM series of articles “Migrating to a Service Oriented Architecture” by Kishore Channabasavaiah, Kerrie Holley, and Edward M. Tuggle Jr., at:

  http://www.ibm.com/developerworks/library/ws-migratesoa/
  http://www.ibm.com/developerworks/webservices/library/ws-migratesoa2/

- “Service-Oriented Architecture expands the vision of Web services”

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

- “Coarse-Grained Interfaces Enable Service Composition in SOA,” by Jeff Hanson at:

  http://builder.com.com/5100-6386-5064520.html

- Olaf Zimmermann, Mark Tomlinson, Stefan Peuser, *Perspectives on Web Services,* Springer, 2003, ISBN 3-540-00914-0

Part 2

Enterprise Service Bus

This section defines the SOA patterns, including the Enterprise Service Bus (ESB) pattern, that extend the scope of the Patterns for e-business. A top-down approach is used to help you model an ESB architecture for your given business scenario:

- Common business scenarios are described in 4.5, “Common ESB scenarios” on page 112. Each business scenario lists the SOA patterns that are relevant to it.
- The ESB subset of the SOA patterns are described in 4.4, “SOA profile of the Application Integration patterns” on page 90.
- The SOA patterns and components have Product mappings defined, as shown in 5.2, “SOA component product mappings” on page 140.

Implementations of the Products mappings are described in Part 3 of this redbook.

4

Enterprise Service Bus and SOA patterns

This chapter describes the Enterprise Service Bus (ESB) and its role within a service-oriented architecture (SOA). It discusses:

- The role of the Enterprise Service Bus in SOA
- A model for analyzing ESB requirements and capabilities
- Common scenarios for ESB requirements
- SOA patterns (including ESB patterns) that are based on the Patterns for e-business Process Integration patterns

4.1 Introducing the Enterprise Service Bus

Chapter 3, “Web services and service-oriented architecture” on page 33, described the principles of SOA by defining the characteristics of services and their interactions. The redbook *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303, describes the implementation of such service interactions using the Direct Connection pattern in the Patterns for e-business Process Integration patterns.

However, enterprises that wish to implement an SOA need a more sophisticated, manageable infrastructure that can support high volumes of individual interactions. Additionally, any such infrastructure should support more established integration styles, such as message-oriented and event-driven integration, or at least integrate with existing infrastructures. Such an infrastructure should support enterprise-level qualities of service. The Enterprise Service Bus is emerging as the unifying concept for such infrastructure, and it is the subject of this chapter and much of the rest of this book.

The Process Integration patterns define Broker, Serial Process, and Parallel Process as interaction styles that can be mediated by infrastructure components, in addition to the Direct Connection pattern. This chapter demonstrates that the Enterprise Service Bus is consistent with support for the Broker pattern and its Router variation, whereas Serial Process, Parallel Process, and their Workflow variations are the subject of other components of SOA.

The concept of an Enterprise Service Bus was first described as “a new architecture that exploits Web services, messaging middleware, intelligent routing, and transformation” by Roy Schulte of Gartner, in the paper “Predicts 2003: Enterprise Service Buses Emerge” in December 2002. Since then, the ESB has been the subject of much debate in the SOA and Web services community. Some of this debate has focused specifically on Java and Web services technologies to such an extent that Andrew Binstock (Integration Watch) remarked in December 2003:

> *“Am I the only one who thinks this [ESB] emperor has no clothes? The only thing this vision of the ESB offers that cannot be found in IBM’s WebSphere MQ and in TIBCO’s various products is that the latter are not inherently based on Java specifications... ESBs have no defining advantage over products such as WebSphere MQ, and they lack its credentials.”*

In this chapter we attempt to resolve issues that concern the nature and benefits of the ESB by analyzing its capabilities and indicating ways in which an ESB can be built. To do so, we combine SOA design principles, Web services technologies and existing features of Enterprise Application Integration middleware, consistent with both the original Gartner vision and an appreciation of the value of mature middleware technologies.

Figure 4-1 maps the SOA design method to the components that were defined in 2.1.3, “Key requirements for integration flexibility” on page 24. It shows that the SOA design method is unified by emerging methods and techniques for SOA design that combine the technologies and disciplines of Web services and the ESB.

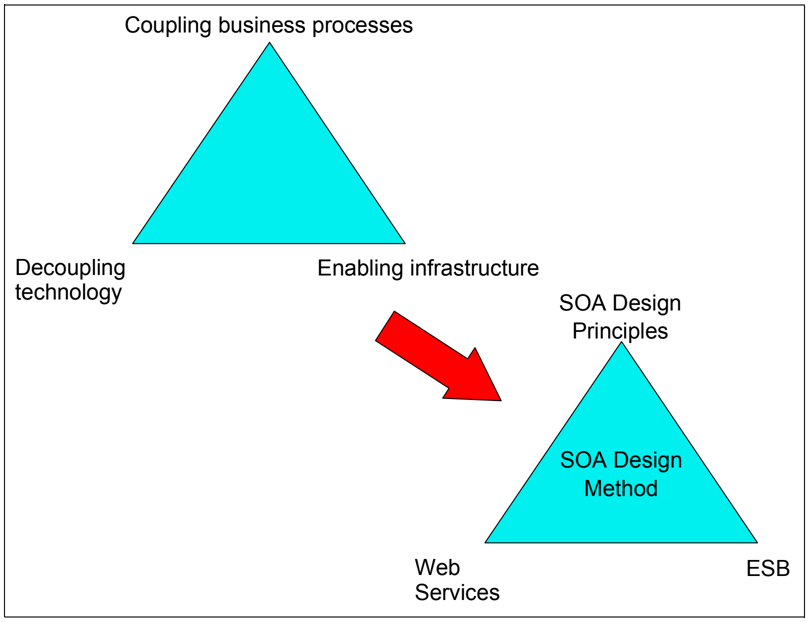


Figure 4-1 On demand linchpins

We describe the Enterprise Service Bus as providing a set of infrastructure capabilities, implemented by middleware technology, that enable the integration of services in an SOA. A variety of ESB capabilities have been identified and are summarized here, but not all of them are required in every situation in which some form of bus can deliver value. We also define a set of minimum capabilities that fulfill the most basic needs for an Enterprise Service Bus that are consistent with the principles of SOA.

Identifying these minimum capabilities enables us to identify which existing technologies can be used to implement an ESB. By considering how the requirements of a specific situation indicate the need for additional capabilities, we can choose the most appropriate implementation technology for that situation.

4.2 The role of the ESB in SOA

In order to implement an SOA, both applications and infrastructure must support the SOA principles. Enabling applications involves the creation of service interfaces to existing or new functions, either directly or through the use of adapters. Enabling the infrastructure at the most basic level involves the provision of capability to route and transport service requests to the correct service provider. The role of the Enterprise Service Bus is, in part, simply to enable the infrastructure in this way.

The true value of the Enterprise Service Bus concept, however, is to enable the infrastructure for SOA in a way that reflects the needs of today's enterprise: to provide suitable service levels and manageability, and to operate and integrate in a heterogeneous environment. The implications of these requirements go beyond basic routing and transport capability, and they are described in The Enterprise Service Bus Capability Model in 4.3, "A capability model for the Enterprise Service Bus" on page 82.

The ESB should enable the substitution of one service implementation by another with no effect to the clients of that service. This requires both the service interfaces that are specified by SOA and that the ESB allows client code to invoke services in a manner that is independent of the service location and communication protocol that is involved.

The ESB supports multiple integration paradigms

In order to fully support the variety of interaction patterns that are required in a comprehensive SOA (for example, request / response, publish / subscribe, events), the Enterprise Service Bus must support in one infrastructure the three major styles of Enterprise Integration:

- *Service-oriented architectures* in which applications communicate through reusable services with well-defined, explicit interfaces. Service-oriented interactions leverage underlying messaging and event communication models.
- *Message-driven architectures* in which applications send messages through the ESB to receiving applications.
- *Event-driven architectures* in which applications generate and consume messages independently of one another.

The ESB does this while providing additional capabilities to mediate or transform service messages and interactions, enabling a wide variety of behaviors and supporting the various models of coupling interaction that are described in 3.2.1, "Coupling and decoupling of aspects of service interactions" on page 39. These capabilities are discussed in more detail in 4.3, “A capability model for the Enterprise Service Bus” on page 82.

Figure 4-2 shows a high-level view of the Enterprise Service Bus.

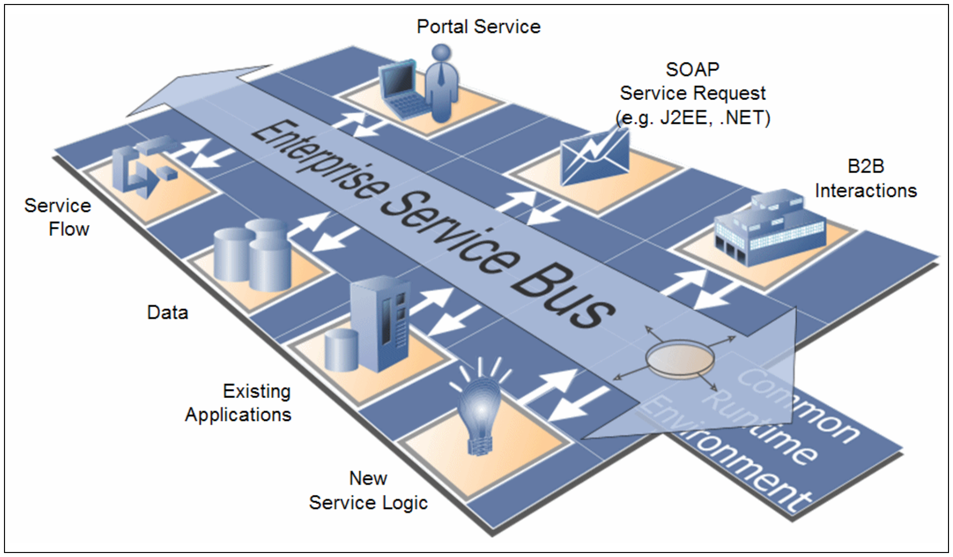


Figure 4-2 The Enterprise Service Bus

The ESB centralizes control and distributes processing

The ESB is sometimes described as a distributed infrastructure and is contrasted with solutions (such as broker technologies) that are commonly described as *hub-and-spoke*. Figure 4-3 illustrates this common depiction of the ESB. However, this view of the ESB is not very helpful in describing how the ESB is physically implemented. For example, what infrastructure components implement the ESB, which is depicted as a line in this diagram?

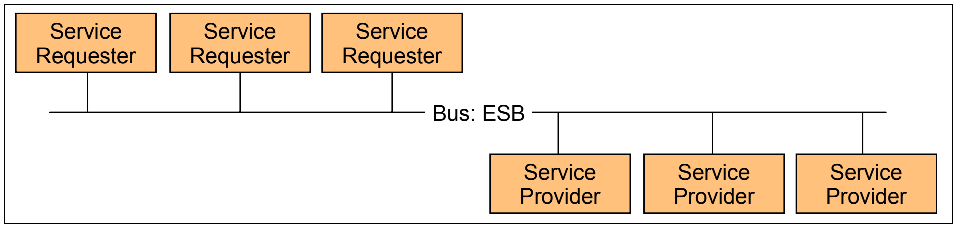


Figure 4-3 The Enterprise Service Bus as a physical infrastructure

In contrast, hub-and-spoke integration solutions (Figure 4-4) seek to centralize control of configuration: routing information, service naming, and so forth.

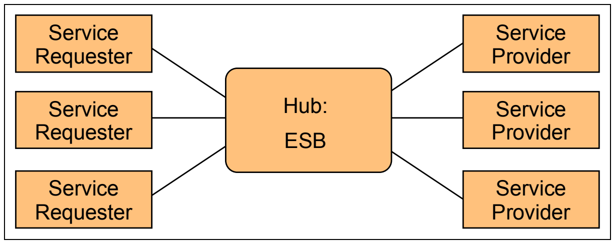


Figure 4-4 Hub-and-spoke integration

In the Patterns for e-business Process Integration patterns an ESB is classified as a type of bus, which in turn is classified as a type of hub, as shown in Figure 4-5 and Figure 4-6 on page 79.

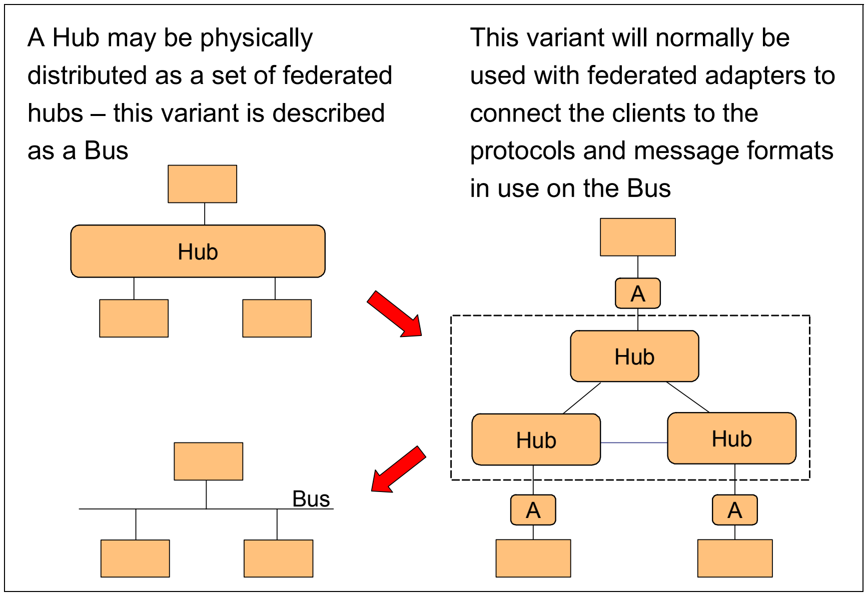

A Hub may be physically distributed as a set of federated hubs – this variant is described as a Bus
This variant will normally be used with federated adapters to connect the clients to the protocols and message formats in use on the Bus
Hub
A
Hub
Hub
Hub
A
A
Bus

Figure 4-5 Hub variation: Bus

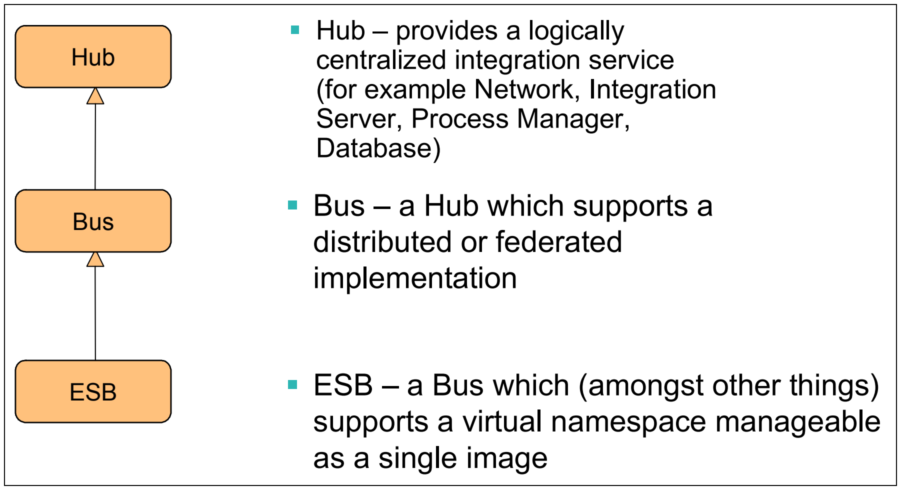

Hub
Bus
ESB
Hub – provides a logically centralized integration service (for example Network, Integration Server, Process Manager, Database)
Bus – a Hub which supports a distributed or federated implementation
ESB – a Bus which (amongst other things) supports a virtual namespace manageable as a single image

Figure 4-6 Hub, Bus, and ESB relationship

The distinction between distributed bus and centralized hub-and-spoke solutions is really a false one. Two different issues are being addressed here: the centralization of control and the distribution of infrastructure. In initial or small-scale implementations of integration solutions, the physical infrastructure is likely to be centralized: concentrated on a single cluster, or hub, of servers. However, as the implementation evolves, the infrastructure may become more physically distributed, as a bus, while retaining at least logically the central control over configuration. Figure 4-7 shows the resulting implementation of an ESB. (The Configuration and Control Services node is shown dotted to illustrate that it is a logical construct.)

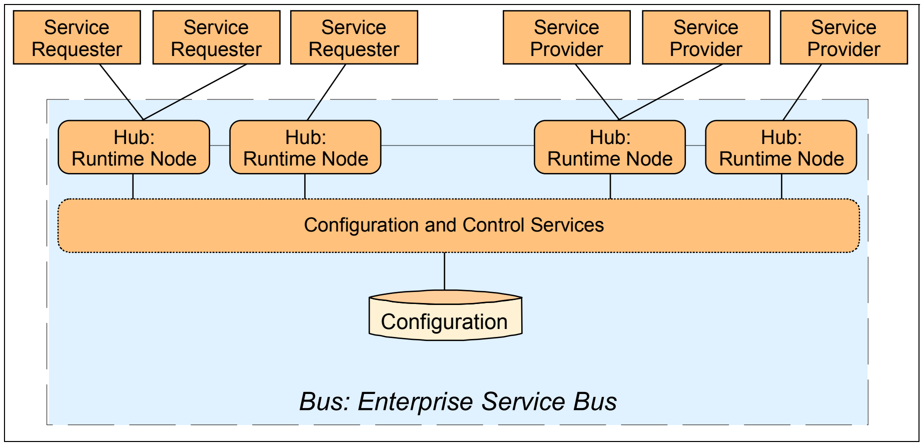


Figure 4-7 The ESB as a distributed infrastructure with centralized control

Of course, this wide distribution of broker technology in a bus pattern is dependent on the capabilities of specific technologies to support such distribution patterns. Equally important from the perspective of incremental implementation and deployment of ESB technology is the ability to extend existing deployments by adding further distributed processing capacity without affecting the existing infrastructure.

The role of the ESB and other SOA components

The ESB is not the only infrastructure component in a SOA. Although individual scenarios vary, there are other commonly occurring components whose role we should position relative to the ESB:

- The Business Service Directory, which provides a taxonomy and details of available services to systems that participate in an SOA.
- The Business Service Choreography, which is used to orchestrate sequences of service interactions into short or long-lived business processes.
- The ESB Gateway, which is used to provide a controlled point of external access to services where the ESB does not provide this natively. Larger organizations are likely to keep the ESB Gateway as a separate component. An ESB Gateway can also be used to federate ESBs within an enterprise.

Figure 4-8 on page 81 illustrates these components interacting with the ESB in an SOA.

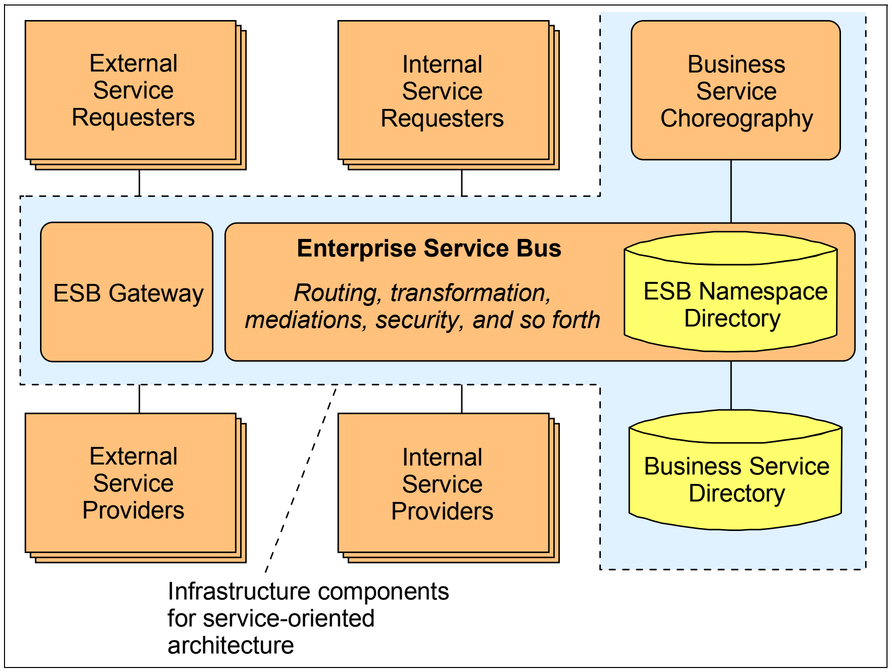

External Service Requesters
Internal Service Requesters
Business Service Choreography
Enterprise Service Bus
ESB Gateway
Routing, transformation, mediations, security, and so forth
ESB Namespace Directory
External Service Providers
Internal Service Providers
Business Service Directory
Infrastructure components for service-oriented architecture

Figure 4-8 The role of the Enterprise Service Bus in a service-oriented architecture

Business Service Directory

In order to perform routing of service interactions, the ESB obviously requires at least basic routing information, which might be provided by an ESB Namespace Directory, or by more simple means such as a routing table. However, this routing information is not necessarily the same as the business service directory SOA component; the role of the business service directory is to provide details of services that are available to perform business functions that are identified within a taxonomy. The business service directory might be an open-standard UDDI directory, or more basic forms might be implemented as a design-time service catalogue, perhaps using collaboration technology. Such catalogues can achieve one of the primary goals of a business service directory: to publish the availability of services and encourage their reuse across the development activity of an enterprise.

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

Business Service Choreography

The role of the Business Service Choreography and its relationship to the granularity or type of services that comprise an SOA was described in 3.2.3, “Service granularity and choreography” on page 47. The role of the choreographer is usually to define and execute business processes, whose configuration and flows are determined by business logic. Such behavior does not usually have a place in an infrastructure component such as the Enterprise Service Bus.

However, the ESB is responsible for infrastructure functions that involve service sequencing, aggregations and mediations (such as splitting and recombining messages), or invoking technical infrastructure services around invocations of business services. The dividing line between business and technical logic controlling such sequencing and mediations is an indistinct one, so in some cases logic that could be considered as business logic to some extent may in fact be implemented in the ESB. Additionally, many of the capabilities that are required to sequence technical functions and mediations in the ESB are similar to those that are required to choreograph services in the Business Service Choreography, so it might be that in some cases the same technology is used to implement both.

ESB Gateway

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

4.3 A capability model for the Enterprise Service Bus

Table 4-1 on page 83 summarizes and categorizes some of the Enterprise Service Bus capabilities that are identified in existing literature. Although some are quite basic, some (such as autonomic or intelligent capabilities) represent significant steps toward an on demand Operating Environment. These capabilities are used to assess the suitability of various existing technologies for implementing the ESB.

Table 4-1 Categorized Enterprise Service Bus capabilities

Communication	**Service interaction**
► Routing ► Addressing ► Protocols and standards (HTTP, HTTPS) ► Publish / subscribe ► Response / request ► Fire & forget, events ► Synchronous and asynchronous messaging	► Service interface definition (WSDL) ► Substitution of service implementation ► Service messaging models required for communication and integration (SOAP, XML, or proprietary Enterprise Application Integration models) ► Service directory and discovery
Integration	**Quality of service**
► Database ► Legacy and application adapters ► Connectivity to enterprise application integration middleware ► Service mapping ► Protocol transformation ► Data enrichment ► Application server environments (J2EE and .Net) ► Language interfaces for service invocation (Java, C/C++/C#)	► Transactions (atomic transactions, compensation, WS-Transaction) ► Various assured delivery paradigms (WS-ReliableMessaging or support for Enterprise Application Integration middleware)
Security	**Service level**
► Authentication ► Authorization ► Non-repudiation ► Confidentiality ► Security standards (Kerberos, WS-Security)	► Performance ► Throughput ► Availability ► Other continuous measures that might form the basis of contracts or agreements
Message processing	**Management and autonomic**
► Encoded logic ► Content-based logic ► Message and data transformations ► Message / service aggregation and correlation ► Validation ► Intermediaries ► Object identity mapping ► Service / message aggregation ► Store and forward	► Administration capability ► Service provisioning and registration ► Logging ► Metering ► Monitoring ► Integration to systems management and administration tooling ► Self-monitoring and self-management

Modeling	Infrastructure Intelligence
► Object modeling ► Common business object models ► Data format libraries ► Public versus private models for business-to-business integration ► Development and deployment tooling	► Business rules ► Policy-driven behavior, particularly for service level, security and quality of service capabilities (WS-Policy) ► Pattern recognition

Many of these capabilities can be implemented either using proprietary technologies or through the use of open standards. This is similar to the different techniques that can be applied to coupling the various aspects of service interactions as described in "Coupling and decoupling of aspects of service interactions" on page 39. Indeed, many of the ESB capabilities are concerned with supporting those coupling and decoupling techniques.

Decisions about whether any capability should be implemented through a customized, proprietary, or open-standard implementation are among the key decisions when designing SOA architectures. The trade-offs involve:

- Whether an open-standards approach offers specific interoperability benefits through "out-of-box" support for multiple technologies, and whether interoperability will actually be achieved without significant work. (For example, do the WS-I interoperability profiles cover those standards?)
- What service level requirements around performance, scalability, and delivery assurance are required, how do these match the capabilities and maturity of proprietary versus open-standards technologies, and how do these contrast with what could be achieved through a customized approach?
- What combination of capabilities, whether customized, proprietary, or open-standard, is required, and which technologies support implementation of that combination?

We do not discuss each capability or category in great detail in this redbook; instead, we focus on those that are of the most interest in deciding how to implement an ESB, and which technologies are available to implement them.

4.3.1 The minimum capability ESB implementation

In 3.2.1, "Coupling and decoupling of aspects of service interactions" on page 39, we discussed the characteristics that interactions should have in order to be designated as services, as this helps us to design architectures that will indeed achieve the promoted benefits of the SOA approach. In a similar way, we will try to define some minimum or mandatory characteristics that should be exhibited by an Enterprise Service Bus. Our goal in doing this is to provide a means by which to assess whether a given proposed ESB architecture or design is likely to achieve the projected benefits of both SOA and ESB.

In order to do this, we start from the most commonly agreed elements of the ESB definition:

- The ESB is a logical architectural component that provides an integration infrastructure consistent with the principles of SOA.
- SOAs consist of services that are defined by explicit, implementation-independent interfaces. They are loosely bound and invoked through communication protocols that stress location transparency and interoperability. Services encapsulate reusable business function.
- The ESB may be implemented as a distributed, heterogeneous infrastructure.
- The ESB provides the means to manage the service infrastructure and the capability to operate in today's distributed, heterogeneous environment.

We can then define the minimum capabilities that an ESB should have in order to provide an infrastructure consistent with those principles, and hence consistent with the promoted benefits of SOA and the Enterprise Service Bus (Table 4-2).

Table 4-2 Minimum capabilities for the Enterprise Service Bus

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

We are now in a position to assess the suitability of individual technologies or products for implementing the Enterprise Service Bus; although the minimum capabilities can help us define which technologies are candidates, the detailed requirements of any particular scenario drive additional ESB capabilities that can then be used to select specific appropriate products. Note that at this stage we do not require the use of any technologies such as Web Services, J2EE, or even XML. The use of those technologies is very likely as they fit these requirements well, but it is not mandatory. This relationship is similar to that between SOA and the Web services technologies.

Basic SOAP/HTTP and WSDL are not an ESB

Given both the prominence of Web services technologies in current discussions of SOA and the Enterprise Service Bus, and the fact that many successful implementations of Web services technologies exist, it is interesting to analyze what the use of basic Web services technologies (WSDL and SOAP/HTTP) achieves against the minimum ESB capabilities (Figure 4-9).

- URL addressing and the existing HTTP and DNS infrastructure provide a bus with routing services and location transparency.
- SOAP/HTTP supports the request/response messaging paradigm.
- The HTTP transport protocol is widely available.
- SOAP and WSDL are an open, implementation-independent messaging and interfacing model.

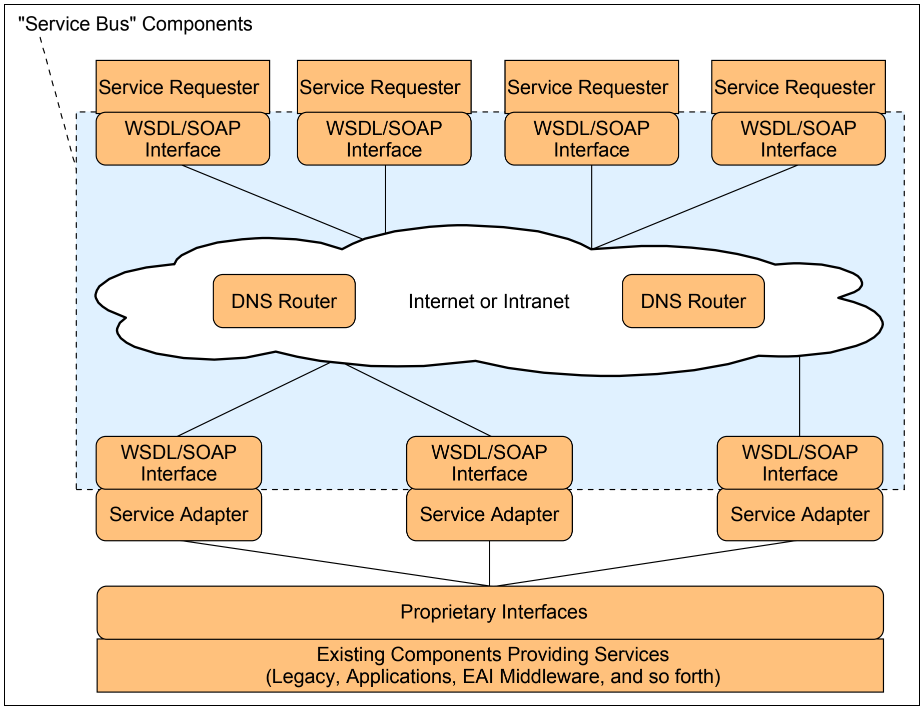


Figure 4-9 SOAP/HTTP and WSDL through the Internet do not provide an ESB

Although the use of SOAP/HTTP and WSDL in this way has many advantages, there are some important ways in which this scenario falls short of the capabilities of the Enterprise Service Bus:

- The scenario relies on the provision of interoperable SOAP/HTTP enablement of each participating system. As the Web services standards are still maturing, there are many systems for which this will not be feasible. An Enterprise Service Bus should provide some form of support for alternative integration techniques.
- Control over service addressing and routing is dispersed between client invocation code, adapter configurations, and the DNS infrastructure. There is no single point of infrastructure control — in other words, this is a point-to-point integration style.
- Vitally, there is no capability to substitute the one implementation of a service provider for another without changing the service requesters; clients and provider code tend to be bound to service invocations over specific protocols and to specific addresses.

Finally, note that the use of adapters to integrate using SOAP/JMS shares many of the same characteristics of the SOAP/HTTP scenario.

Requirements for ESB capabilities beyond the minimum

As just noted, the specific requirements of any SOA or ESB scenario for capabilities beyond the minimum ESB set can be used to identify appropriate implementation technologies. The remainder of this section, and Chapter 5, “ESB and SOA component implementations” on page 133, discuss how scenarios might be analyzed in order to determine the required capabilities and possible implementation technologies.

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

- Quality of service and service-level capabilities.
- Higher-level SOA concepts, such as a service directory, and transformations.
- IBM on demand Operating Environment demands, such as management and autonomic capabilities and intelligent capabilities.
- Truly heterogeneous operation across multiple networks, multiple protocols, multiple domains of disparate ownership.

4.3.2 The Enterprise Service Bus is an infrastructure component

In “The role of the ESB and other SOA components” on page 80, the ESB is positioned as an *infrastructure* component, and as such as a component that does not host or execute business logic. This is in contrast to components such as service requesters, service providers, and the Business Service Choreography whose role is to handle business logic.

However, it is often difficult to draw a clear line between what is *business logic* and what is *infrastructure function*. As increasingly intelligent infrastructures are designed and implemented during the move to an on demand Operating Environment, we will often confront the dilemma as to what is or is not business logic, and what function should be implemented in the Enterprise Service Bus or elsewhere.

Some examples of logic that might be difficult to categorize in this way are:

- Sequencing calls to legacy “screen-scraping” adapters to construct one business transaction from several fine-grained legacy transactions.
- Disaggregating and aggregating messages when broking interactions between a single requester and multiple providers, particularly where aggregation involves combining results to a query from multiple sources.
- Sequencing logging, auditing, and metering service invocations around business service invocations; auditing is a business requirement, so it could be argued that this sequencing incorporates business logic. As an infrastructure evolves with on demand capability, do metering services become business rather than infrastructure function when they link directly to payment services, for example?
- Content-based routing, where the infrastructure inspects the content of messages before deciding which service provider to route them to.

From another perspective, many technologies that provide ESB capability offer a *mediations* facility between service requesters and providers. Examples in current technology include WebSphere Business Integration Message Broker message flows or JAX-RPC handlers. However, all of these are firmly aimed at technical or infrastructure rather than business function (even when they are intended, for example, to perform per-transaction billing functions in an on demand environment – that is viewed as an infrastructure service). So, the ESB is still consistent with the Broker and Router patterns.

In individual architectures, it is more important that each type of logic is clearly assigned to one or more components and technologies than that any categoric distinction is made between off-ESB business logic and on-ESB infrastructure logic.

4.3.3 Security issues affecting the Enterprise Service Bus

The extensive, multi-system integration that is involved in SOAs gives rise to requirements to provide security across a scope that has not been the focus of many existing security technologies. We will not cover security issues for SOA and the ESB in depth, but we will highlight those that may have a particular influence on ESB design or technology selection.

Point-to-point or end-to-end security

Securing individual interactions in isolation is relatively straightforward, as only the technologies and requirements of the two interacting parties need to be considered in designing the security solution. However, if this interaction is actually part of a multi-step interaction (perhaps between a service requester, a service broker, and a service provider), things become more complicated:

- Can a “trust” model be used to view the entire interaction as secure, as long as each constituent interaction is secured? Can this “trust” simply be agreed in principle, or does it depend on the use of specific security solutions (for example, authentication using certificates that are provided by a certificate authority) for the individual interactions?
- Is the infrastructure required to provide an end-to-end security model, for example, to propagate the service requester identity through the infrastructure, including brokers or other intermediaries, and use it to authenticate to the end service provider?
- Is a more sophisticated hybrid model required (for example, enable service requesters to authenticate to a service broker)? Enable only the broker to authenticate to and invoke services from service providers. Hide personal information concerning the clients from the broker, but make it available to the service provider.

These examples are not intended to be an exhaustive description of security requirement possibilities, but they indicate some of the complexity that SOA can drive.

Implementation techniques for ESB security

Whatever the requirements, there are several means to address them:

- Can specific security technologies, such as the Tivoli® security products, be used to support the security requirements of the infrastructure?
- Can communication-level security techniques (for example, SSL encryption or certificate authentication) be used to secure individual interactions within a trust model?

- Can support for open standards (for example, WS-Security, SAML, Kerberos, LDAP, x509 certificates) in the infrastructure and other participating systems be used?
- Can support of open standards or a proprietary security model be added using extension points to the infrastructure, such as a handler or intermediary programming models?
- Can some aspects of the security model, such as client identity and password, be processed as part of the application and data design (for example, include identity and authorization information in each service message and process them using application code)? Does such an approach require specific security applications or components (authentication servers or providers of security tickets) to be included in the architecture?

All of these approaches are possible, but the industry direction is toward standards-compliant (such as WS-Security) security features supported by infrastructure and middleware. However, these standards are relatively recent, and product support for them is emerging rather than established, particularly where interoperability is concerned. Therefore, a priority of any ESB architecture should be to establish the security requirements as early as possible so that they can be included in the choice of implementation technology.

For more information, consult the following resources:

- Securing Web services tutorial

  `http://www.ibm.com/developerworks/webservices/edu/ws-dw-ws-secws-i.html`

- Best Practices for Web services: Web services security

  `http://www.ibm.com/developerworks/webservices/library/ws-best11/`

4.4 SOA profile of the Application Integration patterns

In this section we will use the Patterns for e-business process-focused Application Integration patterns (also referred to as Process Integration patterns) to define a way of modeling the Enterprise Service Bus, and define some SOA patterns that can be used to design and describe ESB components in an SOA infrastructure.

For more information about the process-focused Application Integration patterns, see the modular redpaper *e-business Patterns for Application Integration*, REDP3837, which is part of the redbook *Patterns: Broker Interactions for Intra- and Inter-Enterprise*, SG24-6075.

4.4.1 Summary of Process Integration patterns

The Process Integration patterns define a set of patterns and a methodology for using them within the Application Integration patterns in the Patterns for e-business. The methodology defines a process for analyzing collaboration and interaction requirements in order to specify requirements for any infrastructure to support them.

In the most general sense, a collaboration denotes N-to-N activities between subsystems within a distributed system. Complex collaborations between subsystems can be broken down into more basic interactions. An interaction focuses on 1-to-1 or 1-to-N activities originating from a single subsystem. In this way, complex collaborations involving many subsystems can be decomposed into simpler interactions that are easier to analyze.

Collaboration patterns

The Process Integration patterns provide a set of Collaboration patterns that are used to design or describe broad organizational relationships between applications, and a set of Interaction patterns that are used to describe required behavior in greater detail.

The Process Integration::Collaboration patterns are shown in Figure 4-10 on page 92 and can be summarized as follows:

- *Composed Service* pattern: Represents applications that compose function from other applications.
- *Zone* pattern: Is widely applicable and can represent network zones, such as intranets or demilitarized zones, application server containers, and so on. It is described in "The Process Integration::Zone pattern" on page 95.
- *Hub* patterns: Represents dedicated integration infrastructure providing either basic connection services (a *Network Hub*), or more advanced services such as process management, or data integration services (a *Collaboration Hub*). Its relevance to the Enterprise Service Bus is described in "The ESB centralizes control and distributes processing" on page 77.

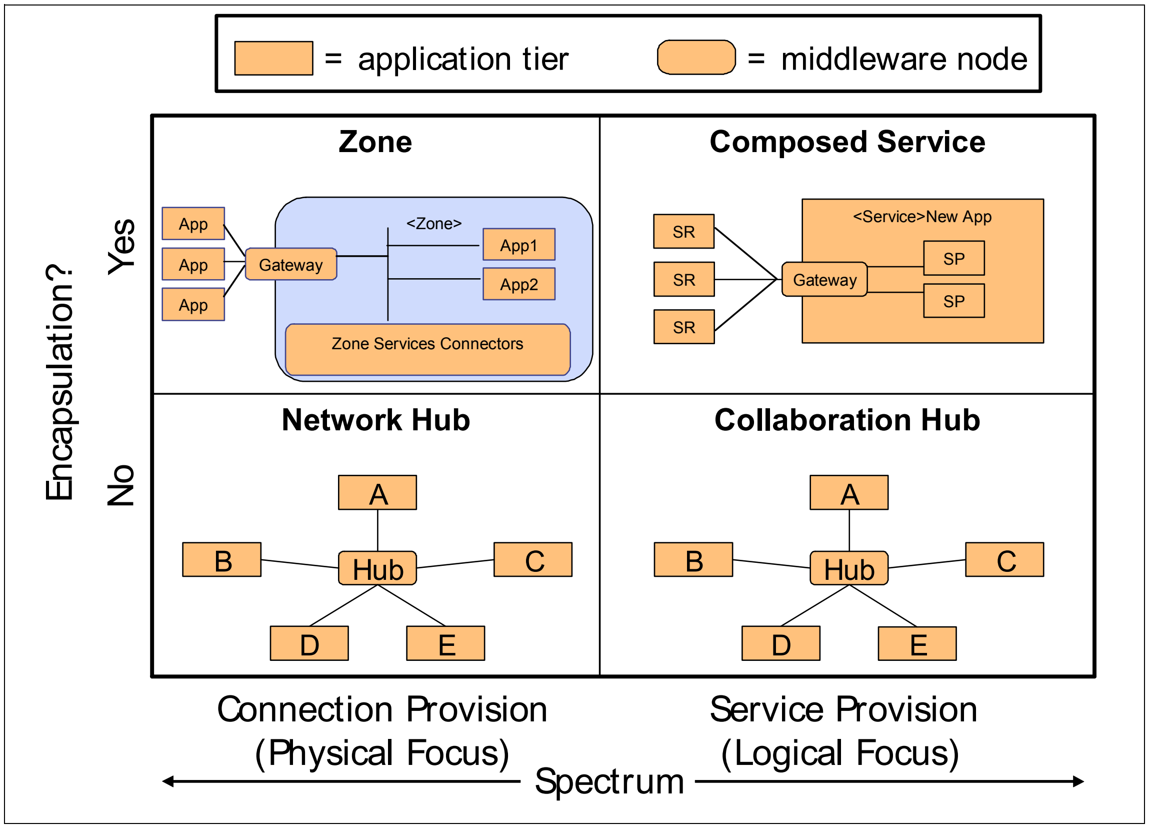


Figure 4-10 Process Integration::Collaboration patterns

Figure 4-11 on page 93 shows the Bus pattern, which is a variation of the Hub pattern. Note that orientation is not significant.

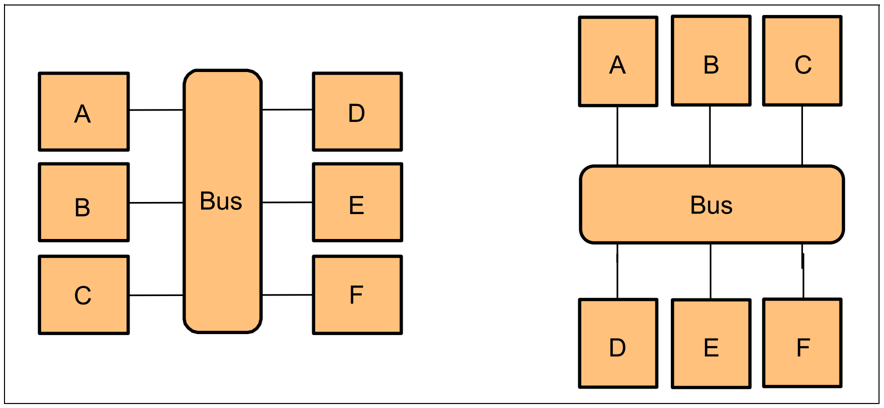


Figure 4-11 Bus pattern as a variation of the Hub pattern

Interaction patterns

The Process Integration::Integration patterns define four styles of interaction (and several variations) that are listed here in order of increasing flexibility and sophistication. As the Application patterns build on each other, their capabilities and reliance on middleware increase, and they require less application development effort:

- Direct Connection application pattern

  The simplest interaction type, which is based on a 1-to-1 topology. It enables a pair of applications within the organization to directly communicate with each other.

  - Message Connection variation

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

  - Call Connection variation

    Applies to solutions where the business process depends on the target application processing a request and returning a response within the scope of the interaction.

- Broker application pattern

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

  - Router variation

    Applies to solutions where the source application initiates an interaction that is forwarded to at most one of multiple target applications.

  - Pub/Sub variation

    In the Pub/Sub variation, the Broker provides an additional Direct Connection interaction. This provides the capability to dynamically update the distribution rules governing the Broker. By executing this Direct Connection interaction, interested parties can add themselves to the list of targets for a particular Broker Interaction.

- Serial Process application pattern

  Extends the 1-to-N topology provided by the Broker application pattern by facilitating the sequential execution of business services that are hosted by several target applications. Thus it enables the orchestration of a serial business process in response to an interaction initiated by the source application.

  - Serial Process Workflow variation

    Extends the basic serial process orchestration capability by supporting human interaction for completing certain process steps.

- Parallel Process application pattern

  Extends the basic serial process orchestration capability provided by the Serial Process application pattern by supporting parallel (concurrent) execution of the subprocesses.

  - Parallel Process Workflow variation

    Extends the basic parallel process orchestration capability by supporting human interaction for completing certain process steps.

Figure 4-12 on page 95 summarizes these four Process Integration::Interaction patterns. One dimension shows support for concurrent interactions to multiple target applications in parallel. The other dimension shows support for non-concurrent interactions to multiple targets in series.

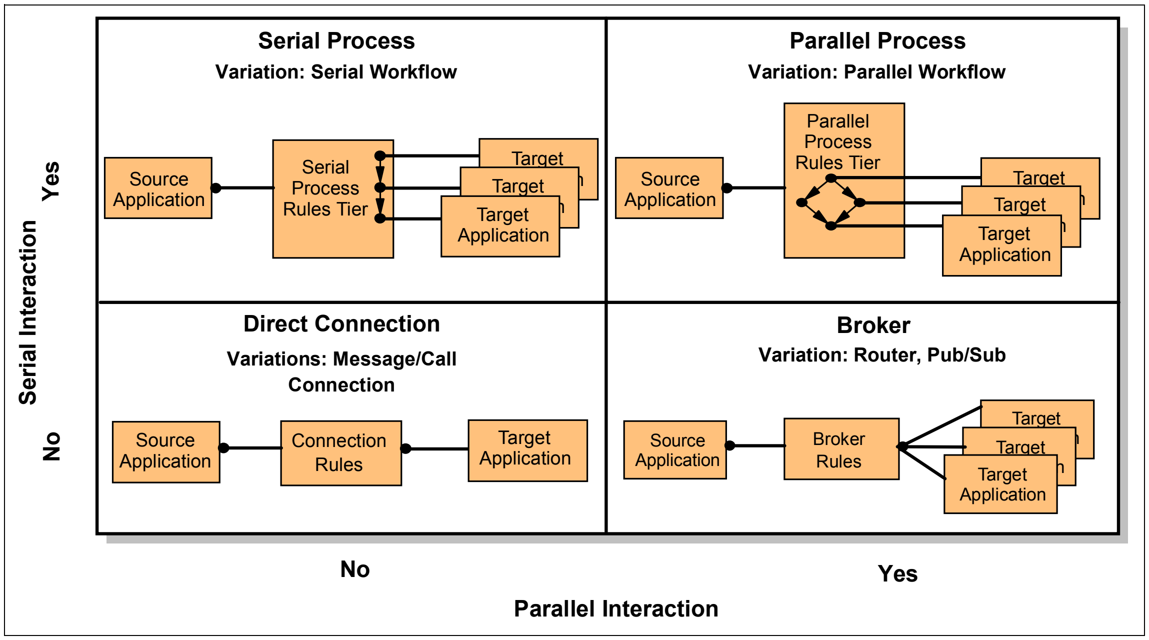

Serial Process
Variation: Serial Workflow
Source Application
Serial Process Rules Tier
Target Application
Parallel Process
Variation: Parallel Workflow
Parallel Process Rules Tier
Direct Connection
Variations: Message/Call Connection
Connection Rules
Broker
Variation: Router, Pub/Sub
Broker Rules
Serial Interaction
Yes
No
Parallel Interaction

Figure 4-12 Process Integration::Interaction patterns

Levels of decomposition

In discussing integration problems and solutions we often have need to show components of the solution at different levels of detail. In the Process Integration patterns we call these *levels of decomposition*. When a component is shown without any internal detail, we call this a level 0 representation. Subsequent decompositions may be labelled level 1, level 2 decompositions and so forth. This is purely intended as a suggestive notational convenience; the different level numbers are helpful in sequencing diagrams but they have no absolute meaning.

Process Integration pattern profiles

The Process Integration patterns define extensions to the basic Process Integration patterns as *profiles*. For example, the Extended Enterprise profile extends the applicability of the patterns to encompass enterprise-to-enterprise scenarios, including the Internet.

The patterns described in this chapter form part of a new *SOA* profile.

The Process Integration::Zone pattern

The Patterns for e-business Process Integration patterns describe a Zone as an area in which a specific set of services is available. A Zone is accessed through one or more Gateways, which are also referred to as Ports in the SOA profile.

Examples of a Zone and Gateway might be:

- An intranet, accessed via a firewall and proxy server
- A J2EE Web container accessed through an HTTP listener
- A message broker accessed through a queue

In Figure 4-13 the Process Integration::Zone pattern is specialized using a Port as a Gateway in order to connect to an ESB.

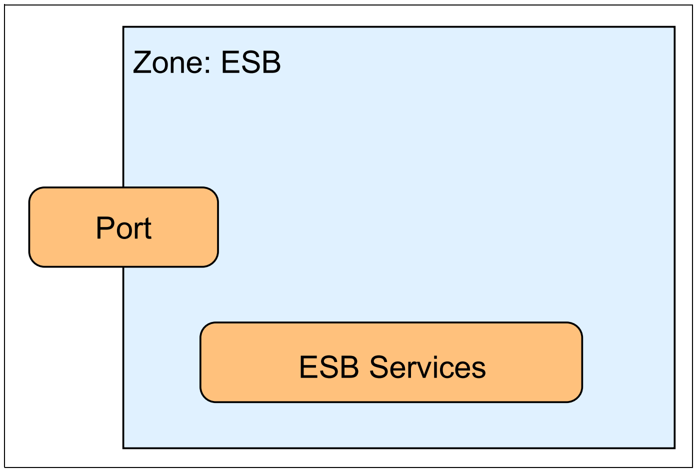


Figure 4-13 A specialization of the Process Integration::Zone pattern

4.4.2 The Enterprise Service Bus pattern

The Enterprise Service Bus pattern is part of the SOA profile in the Patterns for e-business Process Integration patterns. They provide a set of names for composite Process Integration patterns that are particularly useful in the SOA domain.

"The role of the ESB in SOA" on page 76 and "A capability model for the Enterprise Service Bus" on page 82 describe the ESB as providing a set of capabilities, including a point of control over service addressing and naming. Service requestors access the ESB by invoking services with specific addresses through specific protocols. The ESB integrates with service providers by supporting several integration mechanisms including services, but all of which could be described as invoking services through specific addresses and protocols — even if in some cases the "address" is the name of a CICS transaction and the "protocol" is a J2EE resource adapter integrating with the CICS Transaction Gateway.

This enables us to define the ESB in terms of the Process Integration::Zone pattern:

- The Enterprise Service Bus is a Zone in which the ESB capabilities are made available to service interactions.
- The ESB exposes a set of Ports to service requesters. Each Port is identified with a specific protocol and set of addresses through which it provides access to the ESB.
- The ESB uses a set of Ports to integrate to service providers. Each Port supports a specific protocol and a set of addresses that are specific to the services it provides access to.
- The ESB will contain processing units (Hubs) that apply the ESB capabilities to service interactions between service requester ports and service provider ports. These processing units may be distributed in any physical pattern, but all share a common administration infrastructure and configuration. The configuration may form part of the deployed ESB infrastructure or may be separate from it.

Note: Inbound and outbound ports are defined by the following:

- A protocol
- One or more addresses
- A specific way of handling service interaction characteristics such as transactionality, security, and so forth.

An inbound port can listen for a specific address over a specific protocol. However, multiple outbound ports can invoke the same address over the same protocol.

This modeling of the ESB has the effect of defining the *scope* of a single ESB:

- A single ESB provides access to a well-defined set of services implemented by one or more providers. It controls the invocation protocols and addresses made available to service requesters through which to invoke those services.
- A single ESB is controlled by a single administration infrastructure.

And note some of the implications of that definition:

- A single ESB may be associated with any number of protocols. In fact, although it is common to discuss ideas such as the HTTP Service Bus, the ESB is associated with a set of service implementations first, a set of protocols second.
- The ESB infrastructure may be physically deployed in any central, clustered, or distributed pattern.

- A single administration infrastructure is specified to provide a single point of control. We will consider this infrastructure to be part of the ESB for the purposes of our initial discussions.

Figure 4-14 illustrates the ESB pattern. This level 1 decomposition is based on the specialized Process Integration::Zone pattern (Figure 4-13 on page 96) where a Port is a Gateway, specialized for connection to an ESB.

This model depicts the ESB as belonging *within* an Enterprise (as implied by the name Enterprise Service Bus). We will use this definition in this redbook but treat an Enterprise rather flexibly to mean an organization of governance. The ellipses in this diagram represent context. For example, they show a shared namespace controlled by the ESB.

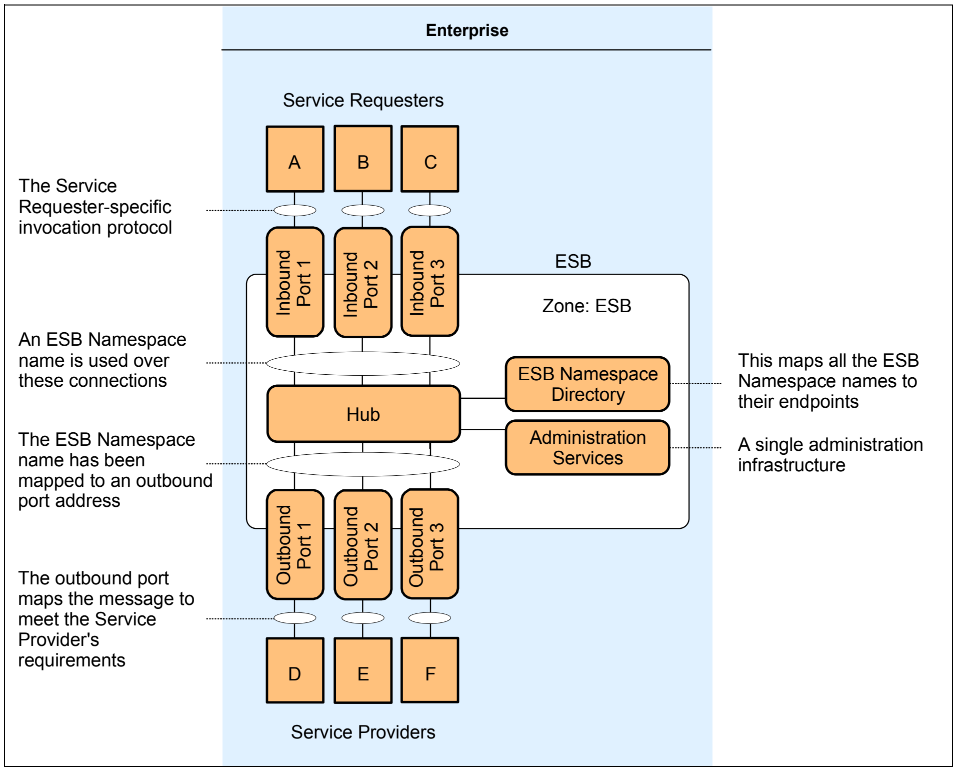

Enterprise
Service Requesters
A
B
C
The Service Requester-specific invocation protocol
Inbound Port 1
Inbound Port 2
Inbound Port 3
ESB
Zone: ESB
An ESB Namespace name is used over these connections
ESB Namespace Directory
This maps all the ESB Namespace names to their endpoints
Hub
Administration Services
A single administration infrastructure
The ESB Namespace name has been mapped to an outbound port address
Outbound Port 1
Outbound Port 2
Outbound Port 3
The outbound port maps the message to meet the Service Provider's requirements
D
E
F
Service Providers

Figure 4-14 ESB pattern: level 1 decomposition

By applying the ESB capabilities from 4.3, “A capability model for the Enterprise Service Bus” on page 82 to this model, we can identify that the Hub execution units within the ESB should support interactions that fit the Process Integration Broker pattern and its variation, Router. The distinction between the two patterns is encapsulated in the ESB capability *message processing - message / service* *aggregation and correlation*: If the ESB is required to support Broker interactions as opposed to Router interactions, it will need this capability.

The role of the ESB as an infrastructure component, as opposed to a component such as Business Service Choreography that processes business logic, means that the ESB is less appropriate to the Serial Process and Parallel Process patterns and particularly their Workflow variations; however, as discussed in 4.3.2, "The Enterprise Service Bus is an infrastructure component" on page 87, such distinctions are not always clear, and the growth of intermediary capabilities in ESB technologies may mean that increasing Serial Process and Parallel Process interactions are supported, but for supporting infrastructure function rather than business logic.

Although we should not prejudge a proper analysis of solution patterns based on requirements, we can also begin to see the suitability of some technologies to implementing the ESB pattern:

- The Web Services Gateway component of WebSphere Application Server Network Deployment can be used to implement an Enterprise Service Bus managing a group of services available through SOAP/HTTP or SOAP/JMS and fulfilled through SOAP/HTTP, SOAP/JMS, RMI/IIOP, J2C, and so on.
- WebSphere Business Integration Message Broker can be used to implement an Enterprise Service Bus managing a group of services available through SOAP/HTTP or SOAP/JMS and fulfilled through SOAP/HTTP, SOAP/JMS, RMI/IIOP, J2C, EDI, JDBC, and so on, while providing additional message processing, modeling, and service level capabilities.

We can use this analysis to indicate how the capability to control service addressing, routing, and protocol transformations supports the vital concept of service implementation substitution: If each inbound and outbound port is associated with a specific service address and protocol, and the Hub contains service routing and mapping and protocol transformation capability, then the outbound port can be replaced without affecting the inbound port, simply by changing the intermediary processing within the Hub. Figure 4-15 on page 100 illustrates this process.

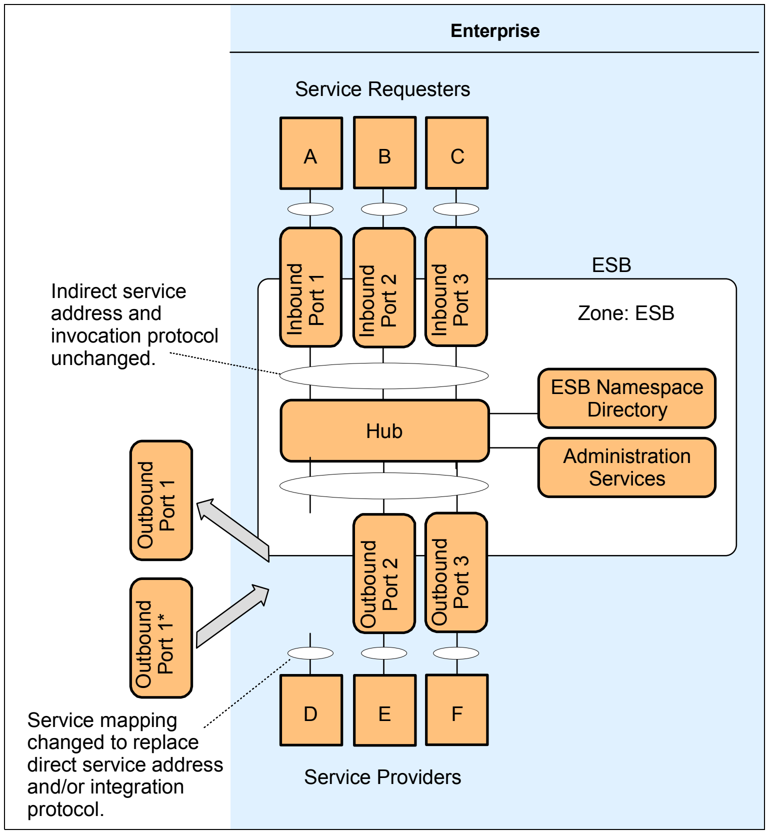


Figure 4-15 The ESB enables service provider substitution

Business drivers for the ESB pattern

Although the majority of this section has been concerned with defining the ESB in relation to business and IT drivers for SOA and the ESB, we should summarize those drivers as part of the formal pattern definition of the ESB:

- Provide a robust, manageable, distributed integration infrastructure consistent with the principles of SOA.
- Enable interaction through services that are defined by explicit implementation-independent interfaces, are loosely bound and invoked through communication protocols that stress location transparency and interoperability, and encapsulate reusable business function.
- Provide an integrated infrastructure capability to support SOA, message-oriented middleware, and event-driven integration.
- Support service routing and substitution, protocol transformations, and other message processing.

- Support both Web Services and traditional EAI communication standards and technologies.

Implementing SOA coupling styles with the ESB

The discussion so far has concentrated on characteristics of service interactions that were described in “The minimum capability ESB implementation” on page 84. However, in many ESB scenarios, the infrastructure should support SOA coupling styles for more sophisticated aspects of service interaction as described in “Coupling and decoupling of aspects of service interactions” on page 39, including security and transactionality.

Table 4-3 indicates why SOA requires the infrastructure capabilities that are provided by the ESB: they are fundamental to providing the loose coupling between applications. Each capability implemented in the bus is a capability that does not have to be implemented by, and matched between, applications.

Table 4-3 Service aspects, relationships, and implementation using ESB capabilities

Aspect	SOA intention	Relevant ESB capabilities
Semantic Interface	Coupled	► Service interaction: service interface definition ► Modeling: data format libraries ► Message processing: message and data transformations, service / message aggregation
Language	Decoupled	► Communication: protocols and standards ► Integration: connectivity to EAI middleware, protocol transformation, application server environments, language interfaces for service invocation ► Service interaction: service interface definition, service messaging model
Platform	Decoupled	
Data Format	Declared or Transformed	► Service interaction: service messaging model ► Integration: legacy and application adapters ► Message processing: message and data transformations
Protocol	Declared or Transformed	► Communication: routing, addressing, protocols and standards ► Service interaction: substitution of service implementation ► Integration: service mapping, protocol transformation, language interfaces for service invocation
Location	Decoupled	

Aspect	SOA intention	Relevant ESB capabilities
Service Provider Identity or Implementation	Declared or Negotiated	► Service interaction: substitution of service implementation, service directory and discovery ► Integration: service mapping, protocol transformation, language interfaces for service invocation ► Infrastructure intelligence: policy driven behavior
Time	Declared or Negotiated	► Communication: synchronous and asynchronous messaging ► Integration: protocol transformation ► Infrastructure intelligence: policy driven behavior ► Message processing: message service aggregation and correlation, store and forward ► Quality of service: assured delivery paradigms
Delivery Assurance, Integrity, and Error Handling	Declared or Negotiated	► Quality of service: all ► Infrastructure intelligence: policy driven behavior
Security	Declared or Negotiated	► Security: all ► Infrastructure intelligence: policy driven behavior
Service Version	Declared or Negotiated	► Communication: routing, addressing ► Integration: service mapping ► Message processing: content-based logic ► Infrastructure intelligence: policy driven behavior

Aspect	SOA intention	Relevant ESB capabilities
Interaction State	Declared	► Matching of messages or events to long-lived processes by explicit process or transaction IDs in semantic interface, or by application data (for example, customer ID). ► Business Service Choreography technology may provide some facility to use a variety of input data to associate messages with specific instances of processes. ► Primary key matching technology such as provided by WebSphere Interchange Server. ► The emerging WS-ResourceFramework provides a standard model for associating services with stateful resources. ► Enterprise Application Integration middleware support for message aggregation and correlation. ► Customized solutions involving custom message headers.

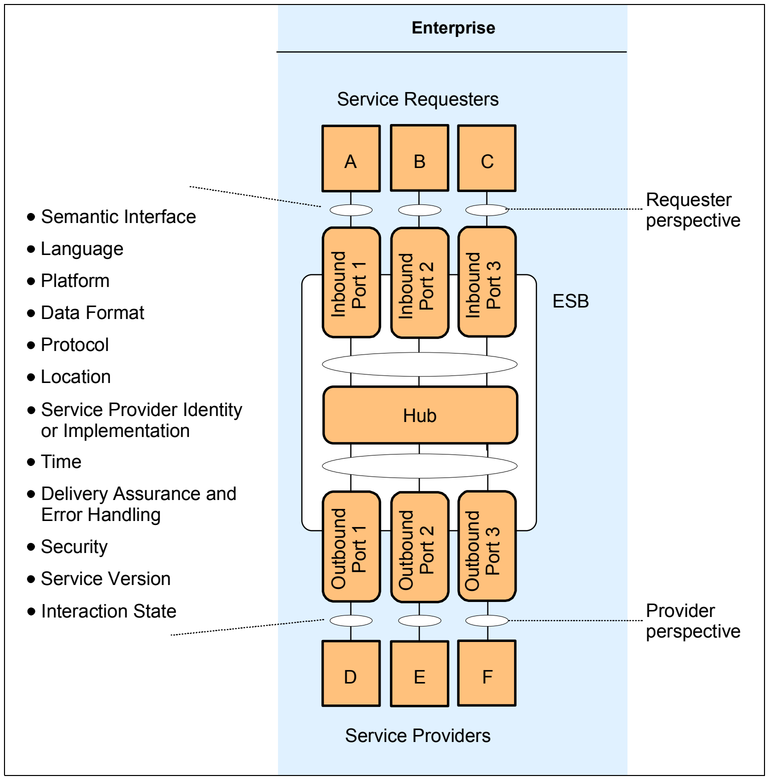


Figure 4-16 Coupling service interaction aspects between service requesters and service providers through the ESB

Note that Figure 4-16 shows a single ellipse to illustrate the service requester perspective of service interactions through the ESB. This should not imply that all interactions share exactly the same characteristics. In a realistic SOA, it is likely that several styles of service interaction will be required; an obvious example would be the use of two protocols, such as SOAP/HTTP or WebSphere MQ. The intention of the diagram is to emphasize the need to limit and control the number of service interaction styles to a manageable number, through the use of an Enterprise Service Bus. This shields service requesters from dealing with the complexity and variety of different interaction styles that might be necessary to deal with all of the service providers (such as Web Services applications, legacy systems, and packaged applications) in a heterogeneous environment.

In “Modeling additional SOA components” on page 106, we combine the Enterprise Service Bus pattern with other Patterns for e-business Process Integration patterns. To simplify the depiction of these combinations, we will introduce the Enterprise Service Bus pattern at a level 0 decomposition, as shown in Figure 4-17 on page 105. Note that orientation is not significant.

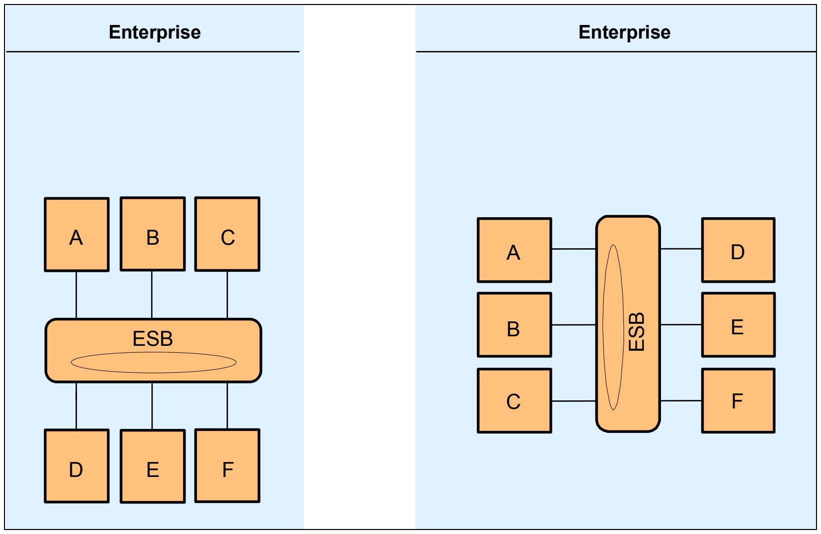


Figure 4-17 ESB pattern: level 0 decomposition

Directionality of service interactions

The preceding discussion has described the roles of service requester and service provider as separate; of course, in many situations the same application or infrastructure component performs both roles by offering its functions as services while invoking services from other applications. Of course, the Enterprise Service Bus itself both provides services to service requesters and requests services from service providers, as do Business Service Choreography components.

However, this distinction between the roles of service requesters and service providers is still a useful one; a full analysis of the various possible interaction styles (for example request / response, publish / subscribe, fire and forget, events) is beyond the scope of our discussion, but it is generally true that:

- Service requesters can contact service providers using specific named services.
- Service providers must either maintain contact with the service requester that invoked them in order to respond, retain a locating handle, or rely on the service infrastructure to do this for them.
- Interaction models that involve pushing messages from service providers to service requesters (for example publish / subscribe or event models) require the service provider to maintain contact with the service requester or retain a locating handle, or require the service requester to offer a callback service.

In this way, the roles of service requester and service provider are rather different, and so it is worth distinguishing between them, even when both are performed by the same application or component.

4.4.3 Other SOA patterns

An ESB Gateway node makes the services of one organization available to others, and vice versa, in a controlled and secure manner. The ESB Gateway can be used to federate ESBs across an organization, for example. If the ESB Gateway is exposed, it can be used in an inter-enterprise environment.

Figure 4-18 shows the ESB Gateway pattern.

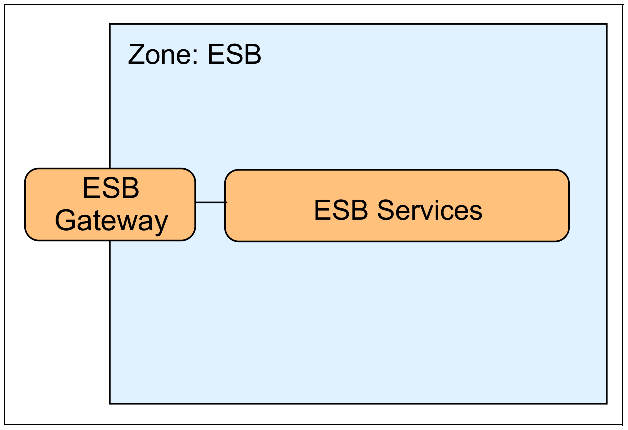


Figure 4-18 ESB Gateway pattern

4.4.4 Modeling additional SOA components

As the SOA pattern that is introduced here is a composite of the Process Integration patterns, it can be combined with the base Process Integration patterns in modeling entire infrastructures and architectures.

For example, the ESB pattern can be used to replace a Direct Connection pattern, such as a point-to-point interaction using a connector to reach an application. Figure 4-19 on page 107 illustrates this idea. Alternatively, for connectors that are closely integrated with the ESB, these might be modeled as specific Ports that provide access to the Zone representing the Enterprise Service Bus. Such a pattern might apply where a particular service integration requires capabilities that are not required for other service interactions that the ESB supports. (Typical examples are Legacy or Application Adapter capabilities.)

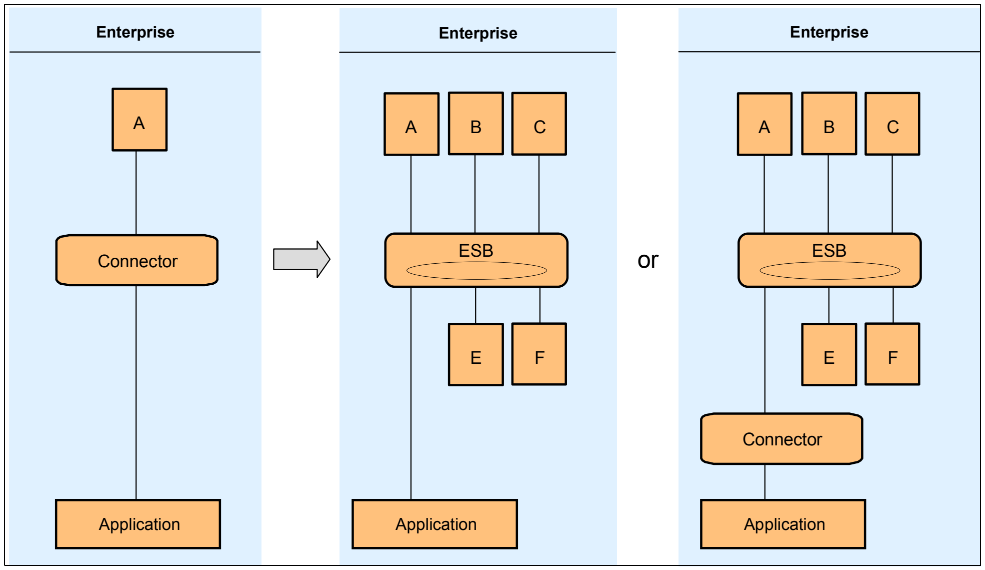


Figure 4-19 Replacing the Direct Connection pattern with a composite Process Integration pattern

Other applications and components can also be modeled, such as Business Service Choreography shown in Figure 4-20.

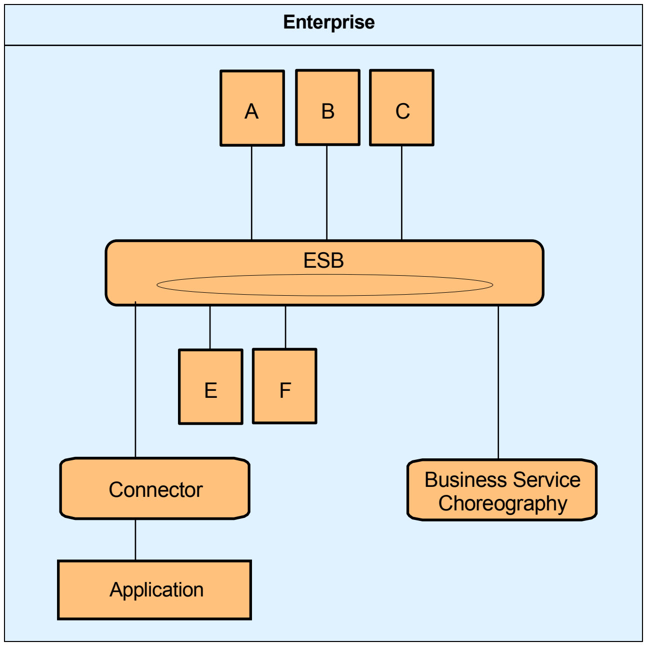


Figure 4-20 Business Service Choreography as connected to the ESB

4.4.5 Extended Enterprise SOA patterns

This section describes how to use an ESB to communicate with service requesters and providers in external enterprises. The exposed patterns do not define the service requesters and providers that the patterns are interacting with; these service providers may or may not be other ESBs.

Product mappings for these patterns are described in 5.2.3, “SOA product mappings” on page 142.

Exposed ESB pattern

The Exposed ESB pattern extends access to all or a subset of services to external clients, and enables the ESB to access external resources as services. This pattern is concerned with identifying the basic capabilities that are required to provide such access, rather than modeling specific external components, services, or interactions.

Drivers

- Provide controlled external access to a subset of services that are available through an ESB.
- Enable an ESB to access external services and resources.
- Reuse ESB infrastructure rather than implementing a separate ESB Gateway.

Description

This pattern represents direct exposure of the ESB to external service requesters and providers. The ESB implementation will have to provide appropriate routing, security, and protocol transformation capabilities.

Figure 4-21 on page 109 illustrates the Exposed ESB pattern. Note that the diagram assumes that the Internet is used as the communication transport for interaction with other enterprises. This is the most common scenario, but it could be replaced equally by a Value Added Network (VAN).

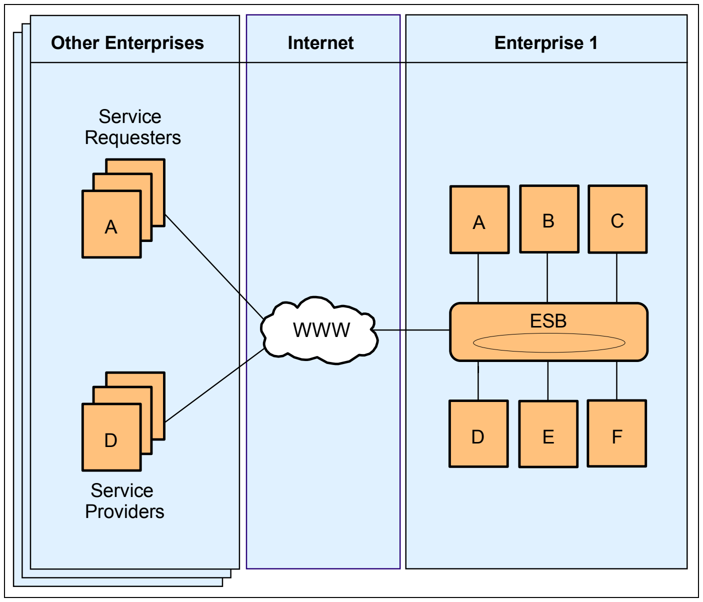


Figure 4-21 The Exposed ESB pattern

Relevant ESB capabilities

The following ESB capabilities (from those listed in 4.3, “A capability model for the Enterprise Service Bus” on page 82) are relevant to the Exposed ESB pattern. You must consider these capabilities when deciding on product mappings to implement this pattern.

- Communications
  - Routing
  - Addressing
  - Protocols and standards
- Service interaction (all)
- Integration
  - Service mapping
  - Protocol transformation
- Quality of service (all)
- Security (all)
- Service level (all)

- Message processing
  - Message and data transformations
  - Validation
  - Intermediaries
- Management and autonomic:
  - Service provisioning and registration
  - Logging, metering, and monitoring
  - Discovery
- Modeling
  - Public versus private models

Exposed ESB Gateway composite pattern

The Exposed ESB Gateway pattern represents a composite of the ESB pattern and the ESB Gateway pattern for inter-enterprise access. The Exposed ESB Gateway pattern is implemented in Chapter 11, “Exposed ESB Gateway composite pattern” on page 299.

Drivers

- Provide controlled external access to a subset of services available through an ESB.
- Enable an ESB to access external services and resources.
- Provide additional or alternative security, routing, logging, and other capabilities to external service interactions that are relative to those that are available internally.
- Provide additional protection from external access to internal services.
- Provide additional features to manage and provision external service partners.

Description

A more sophisticated scenario introduces an explicit service gateway. We can model this as an Exposed ESB Gateway pattern. The Exposed ESB Gateway pattern connects to service requesters or service providers through the Internet. This pattern can enable more sophisticated control over external service access by providing additional security models, buffering of service requests to protect internal system capacity, and additional transformations of protocol, data formats, or models of delivery assurance.

The potential disadvantage of this model, aside from the extra infrastructure it requires, is the need for an additional point of administration between internal and external systems participating in service requests. (Both the ESB and the Exposed ESB Gateway must now be administered.)

In order to provide a consistent view of service interactions to both internal and external service requesters and providers, the features of the ESB and the Exposed ESB Gateway that are used to support service interactions, as described in “Implementing SOA coupling styles with the ESB” on page 101, will have to be matched.

Figure 4-22 illustrates the Exposed ESB Gateway pattern.

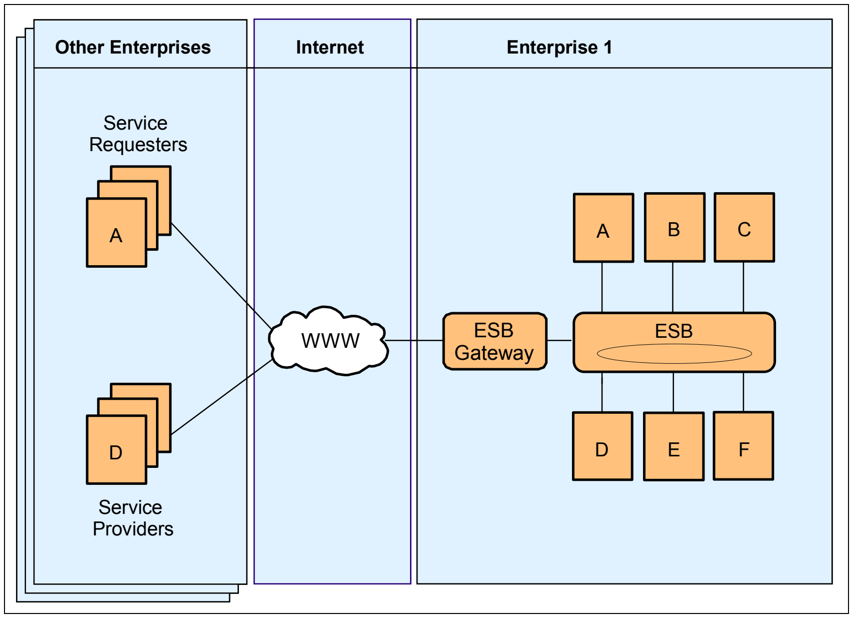


Figure 4-22 The Exposed ESB Gateway composite pattern

Relevant ESB capabilities

The following ESB capabilities (from those listed in 4.3, “A capability model for the Enterprise Service Bus” on page 82) are relevant to the Exposed ESB Gateway pattern. You must consider these capabilities when deciding on product mappings to implement this pattern.

- Communications
  - Routing
  - Addressing
  - Protocols and standards

- Service interaction (all)
  - Integration
  - Service mapping
  - Protocol transformation
- Quality of service (all)
- Security (all)
- Service level (all)
- Message processing
  - Message and data transformations
  - Validation
  - Intermediaries
- Management and autonomic
  - Service provisioning and registration
  - Logging, metering, and monitoring
  - Discovery
- Modeling
  - Public versus private models

4.5 Common ESB scenarios

In the following scenarios, we describe common starting points for the implementation of SOA with or without an ESB. The scenarios are intended to describe *individual* situations to which SOA with or without an ESB might be appropriate. They are not intended to describe the SOA or ESB requirements of an entire organization. Indeed, in some cases SOA or an Enterprise Service Bus might be strategic goals driven by business requirements across an organization, but in many more cases SOA or an ESB are suggested by IT organizations in response to increasing demands across a business for more flexible and widespread integration in order to support several projects.

Our reason for focusing on individual situations is that they characterize individual requirements that can be analyzed in order to identify suitable approaches and technologies. In some cases, it will be most appropriate to implement simple solutions to each individual scenario; in other situations it may be more advantageous to combine requirements across scenarios and implement a common, more advanced solution. In other cases it may be better to take a mixed approach, and implement individual solutions, but with characteristics that enable future integration across the solutions in an evolutionary manner.

The following sections are useful references:

- The scenarios are described with reference to the ESB capabilities described in 4.3, “A capability model for the Enterprise Service Bus” on page 82.
- The scenarios reference the SOA patterns and technologies that are described in 4.4, “SOA profile of the Application Integration patterns” on page 90.

Each common ESB scenario describes:

- Drivers: business and technical factors leading to the scenario.
- Scenario discussion: a brief discussion of the scenario.
- Candidate technologies: the generic technology types that are often considered in this scenario.
- Relevant issues: links to specific relevant issues that are further considered in 4.5.8, “Architecture decision questions” on page 125.
- Relevant patterns: links to specific SOA patterns that form candidate solutions.
- ESB capability requirements: those aspects of ESB capability as described in 4.3, “A capability model for the Enterprise Service Bus” on page 82 that are most relevant to the scenario. The capabilities might be implemented in any aspect of the solution (for example, using the ESB pattern or perhaps in adapters that connect to it). Note that these lists are not intended to be exhaustive; further capabilities are likely to be driven by the requirements of specific situations, or by the issues that are discussed in 4.5.8, “Architecture decision questions” on page 125.

4.5.1 Basic integration of two systems

This section describes the drivers, technologies, and issues of the basic integration of two systems. It then highlights the relevant SOA patterns and ESB capabilities that apply to this scenario.

Drivers

- A need to integrate two and only two systems that are implemented in different technologies (for example, J2EE and .Net)
- A lack of knowledge as to more general integration requirements
- A need to implement a quick, tactical integration

- A desire to loosely couple the integration so as to later replace it with a more strategic solution

Scenario discussion

Some interoperable standard likely to be under consideration. Whichever technology is chosen will have to be supported to some extent in the environments of both applications. This might be true particularly when the two systems are owned by different parties or where only limited network access is allowed between the systems.

Candidate technologies

- Web services
- Messaging technology
- Adapters or connectors

Relevant issues

- 1. Function and Data Interfaces
- 3. Technologies for Interoperability
- 4. Advanced Interaction Characteristics
- 6. Technology Support in Existing Systems
- 10. Consistent and Controlled Service Enablement
- 11. External Access to Services
- 13. Service Level Requirements
- 14. Security Requirements

Relevant SOA patterns

- Direct Connection: Basic integration within or between organizations using adapters, interruptible protocols, and so forth. Described in *Patterns: Service-Oriented Architecture and Web Services,* SG24-6303.
- ESB pattern: Use of infrastructure technology to implement an Enterprise Service Bus.
- Extended Enterprise SOA patterns: Application of the SOA patterns for extended enterprise.

ESB capability requirements

- Communications (all)
- Service interaction:
  - Service interface definition
  - Service messaging models

- Integration:
  - Database
  - Legacy and application adapters
  - Application server environments
  - Language interfaces for service invocation
- Quality of service (all)
- Security (all)

4.5.2 Enable wider connectivity to one or more applications

This section describes the drivers, technologies, and issues of enabling wider connectivity to one or more applications. It then highlights the relevant SOA patterns and ESB capabilities that apply to this scenario.

Drivers

- Enable service access to packaged applications (for example CRM, ERP).
- Enable service access to existing or new customized applications, perhaps implemented in J2EE or other application server environments.

Scenario discussion

There is value in exposing these functions as services either to enable the applications to interoperate with each other, or to provide access to new channels or clients. The use of open standards communication and service protocols seems the best way forward.

Candidate technologies

- Web services
- Asynchronous messaging
- Enterprise Application Integration middleware, message broking technology, Enterprise Service Bus

Relevant issues

- 1. Function and Data Interfaces
- 2. Common Business Data Model
- 3. Technologies for Interoperability
- 4. Advanced Interaction Characteristics
- 6. Technology Support in Existing Systems
- 8. Availability of Services in EAI Infrastructure
- 9. Service Provider Protection

- 10. Consistent and Controlled Service Enablement
- 11. External Access to Services
- 12. Business Service Choreography
- 13. Service Level Requirements
- 14. Security Requirements

Relevant SOA patterns

- Direct Connection: Basic integration within or between organizations using adapters, interruptible protocols and so forth. Described in *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303.
- ESB pattern: Use of infrastructure technology to implement an Enterprise Service Bus.
- Extended Enterprise SOA patterns: Application of the SOA patterns for extended enterprise.
- Business Service Choreography: If business service choreography is also required, implement the Serial Process or Parallel Process patterns as described in *Patterns: Serial and Parallel Processes for Process Choreography and Workflow*, SG24-6306.

ESB capability requirements

- Communications (all)
- Service interaction:
  - Service interface definition
  - Substitution of service implementation
  - Service messaging models
  - Service directory and discovery
- Integration:
  - Database
  - Legacy and application adapters
  - Service mapping
  - Protocol transformation
  - Application server environments
  - Language interfaces for service invocation
- Quality of service (all)
- Security (all)
- Service level (all)

- Management and autonomic:
  - Service provisioning and registration
  - Logging, metering, and monitoring

4.5.3 Enable wider connectivity to legacy systems

This section describes the drivers, technologies, and issues of enabling wider connectivity to legacy systems. It then highlights the relevant SOA patterns and ESB capabilities that apply to this scenario.

Drivers

Business value in exposing core business transactions in legacy systems to new channels.

Scenario discussion

Many organizations have an enormous investment in legacy technologies (such as CICS Transaction Server and IMS™ Transaction Server) that support applications that provide their core business transactions and data access. There may be significant value in providing open standard, service-based access to those transactions (for example, transactions that query account balance, create orders, schedule or track deliveries, and query stock levels).

Candidate technologies

- Adapters and connectors
- Legacy integration technology
- Asynchronous messaging
- Web services

Relevant issues

- 1. Function and Data Interfaces
- 2. Common Business Data Model
- 3. Technologies for Interoperability
- 4. Advanced Interaction Characteristics
- 7. Legacy XML Support and Processing
- 8. Availability of Services in EAI Infrastructure
- 9. Service Provider Protection
- 10. Consistent and Controlled Service Enablement
- 11. External Access to Services
- 13. Service Level Requirements
- 14. Security Requirements

Relevant SOA patterns

- Direct Connection: Basic integration within or between organizations using adapters, interruptible protocols, and so forth. Described in *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303.
- ESB pattern: Use of infrastructure technology to implement an Enterprise Service Bus.
- Extended Enterprise SOA patterns: Application of the SOA patterns for extended enterprise.

ESB capability requirements

- Communications (all)
- Service interaction:
  - Service interface definition
  - Service messaging models
- Integration:
  - Database
  - Legacy and application adapters
- Quality of service (all)
- Security (all)
- Service level (all)
- Message processing
  - Message / service aggregation and correlation
  - Message and data transformations
- Management and autonomic:
  - Service provisioning and registration
  - Logging, metering, and monitoring

4.5.4 Enable wider connectivity to an EAI infrastructure

This section describes the drivers, technologies, and issues of enabling wider connectivity to EAI infrastructure. It then highlights the relevant SOA patterns and ESB capabilities that apply to this scenario.

Drivers

Business value in extending access to services that are available through Enterprise Application Integration (EAI) infrastructure.

Scenario discussion

Existing functions of application and legacy systems or both have already been exposed through EAI infrastructure. However, additional connections through additional channels can drive further value, but these channels may not be in reach of the EAI infrastructure, either because they are deployed beyond it or they support incompatible communication mechanisms.

Candidate technologies

- Customized or proprietary HTTP protocols
- Web services
- Additional EAI middleware

Relevant issues

- 3. Technologies for Interoperability
- 4. Advanced Interaction Characteristics
- 5. Adoption of Standards at the Edge or in the Heart of Infrastructure
- 8. Availability of Services in EAI Infrastructure
- 9. Service Provider Protection
- 11. External Access to Services
- 13. Service Level Requirements
- 14. Security Requirements

Relevant SOA patterns

- ESB pattern: Use of infrastructure technology to implement an Enterprise Service Bus
- Extended Enterprise SOA patterns: Application of the SOA patterns for extended enterprise

ESB capability requirements

- Communications (all)
- Service interaction:
  - Service interface definition
  - Substitution of service implementation
  - Service messaging models
- Integration:
  - Connectivity to Enterprise Application Integration middleware
  - Service mapping
  - Protocol transformation
  - Application server environments

  - Language interfaces for service invocation
- Quality of service (all)
- Security (all)
- Service level (all)
- Management and autonomic:
  - Service provisioning and registration
  - Logging, metering, and monitoring

4.5.5 Implement controlled integration between organizations

This section describes the drivers, technologies, and issues of implementing controlled integration across organizations. It then highlights the relevant SOA patterns and ESB capabilities that apply to this scenario.

Drivers

- Enable customers, suppliers, or other partners to integrate directly with functions provided by one or more applications (legacy or otherwise).
- Provide a point of control to secure and manage external access to services.

Scenario discussion

The use of open standards is preferred as the organization has no direct control over the technologies that are used by its partners. Note that this scenario might apply either between separate organizations or between units of a larger distributed organization.

Candidate technologies

- Web services
- ESB Gateways

Relevant issues

- 1. Function and Data Interfaces
- 2. Common Business Data Model
- 3. Technologies for Interoperability
- 4. Advanced Interaction Characteristics
- 6. Technology Support in Existing Systems
- 8. Availability of Services in EAI Infrastructure
- 9. Service Provider Protection
- 10. Consistent and Controlled Service Enablement
- 11. External Access to Services
- 13. Service Level Requirements

- 14. Security Requirements

Relevant SOA patterns

- ESB pattern: Use of infrastructure technology to implement an Enterprise Service Bus
- Extended Enterprise SOA patterns: Application of the SOA patterns for extended enterprise

ESB capability requirements

- Communications (all)
- Service interaction (all)
- Integration:
  - Legacy and application adapters
  - Connectivity to Enterprise Application Integration middleware
  - Service mapping
  - Protocol transformation
  - Application server environments
- Quality of service (all)
- Security (all)
- Service level (all)
- Message processing
  - Message and data transformations
  - Validation
  - Intermediaries
- Management and autonomic:
  - Service provisioning and registration
  - Logging, metering, and monitoring
  - Discovery
- Modeling
  - Data format libraries
  - Public versus private models

4.5.6 Automate processes by choreographing services

This section describes the drivers, technologies, and issues of automating systems by choreographing services. It then highlights the relevant SOA patterns and ESB capabilities that apply to this scenario.

Note: This scenario can be considered an evolution of the Enable Wider Connectivity to One or More Applications scenario

Drivers

- Model and automate business processes.
- Choreograph services into processes in a flexible manner.
- Enable monitoring and measurement of business processes.

Scenario discussion

Existing packaged applications (such as CRM and ERP) or customized applications, perhaps implemented in J2EE or other application server environments, provide functions that are useful beyond the applications themselves. These functions can be exposed as services using open-standard communication and service protocols so that the applications can interoperate. The interactions between the applications form business processes at some level, and these processes should be explicitly modeled and executed using appropriate modeling and process execution technology, preferably in compliance with open standards.

Candidate technologies

- Web Services Choreography
- Workflow

Relevant issues

- 1. Function and Data Interfaces
- 2. Common Business Data Model
- 3. Technologies for Interoperability
- 4. Advanced Interaction Characteristics
- 6. Technology Support in Existing Systems
- 10. Consistent and Controlled Service Enablement
- 11. External Access to Services
- 12. Business Service Choreography
- 13. Service Level Requirements
- 14. Security Requirements

Relevant SOA patterns

- Business Service Choreography: If business service choreography is also required, implement the Serial Process or Parallel Process patterns as described in *Patterns: Serial and Parallel Processes for Process Choreography and Workflow*, SG24-6306.
- Direct Connection: Linking the choreographer to service providers through basic integration using adapters, interruptible protocols, and so forth. Described in *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303.
- ESB pattern: Use of infrastructure technology to implement an Enterprise Service Bus.
- Extended Enterprise SOA patterns: Application of the SOA patterns for extended enterprise.

ESB capability requirements

- Communications (all)
- Service interaction (all)
- Integration (all)
- Quality of service (all)
- Security (all)
- Service level (all)
- Message processing (all)
- Modeling (all)

4.5.7 Implement a robust SOA with Web services support

This section describes the drivers, technologies, and issues of implementing a robost SOA with Web services support. It then highlights the relevant SOA patterns and ESB capabilities that apply to this scenario.

Drivers

- Enable widespread internal or external access to services provided by multiple applications (legacy or otherwise).
- Provide a general infrastructure for controlled and flexible integration in response to demands from across the business.
- Enable the widespread exposure of functions as services and their choreography into automated, measurable business processes.

Scenario discussion

This scenario is effectively a composite of the preceding scenarios. Various security, aggregation, transformation, routing, and business service choreography capabilities are required.

Candidate technologies

- Web services and Web services choreography
- Workflow
- Enterprise Application Integration middleware

Relevant issues

- 1. Function and Data Interfaces
- 2. Common Business Data Model
- 3. Technologies for Interoperability
- 4. Advanced Interaction Characteristics
- 5. Adoption of Standards at the Edge or in the Heart of Infrastructure
- 6. Technology Support in Existing Systems
- 7. Legacy XML Support and Processing
- 8. Availability of Services in EAI Infrastructure
- 9. Service Provider Protection
- 10. Consistent and Controlled Service Enablement
- 11. External Access to Services
- 12. Business Service Choreography
- 13. Service Level Requirements
- 14. Security Requirements

Relevant SOA patterns

- ESB pattern: Use of infrastructure technology to implement an Enterprise Service Bus.
- Extended Enterprise SOA patterns: Application of the SOA patterns for extended enterprise.
- Business Service Choreography: If business service choreography is also required, implement the Serial Process or Parallel Process patterns as described in *Patterns: Serial and Parallel Processes for Process Choreography and Workflow*, SG24-6306.

ESB capability requirements

- Communications (all)
- Service interaction (all)
- Integration (all)
- Quality of service (all)

- Security (all)
- Service level (all)
- Message processing (all)
- Management and autonomic:
  - Service provisioning and registration
  - Logging, metering, and monitoring
  - Discovery
- Modeling (all)
  - Object modeling
  - Common business object models
  - Data format libraries
  - Public versus private models
  - Development and deployment tooling

4.5.8 Architecture decision questions

This section presents several key architectural issues, referred to by the individual scenarios in 4.5, “Common ESB scenarios” on page 112. The discussion of each issue indicates additional ESB capabilities that may be required to address it. These capabilities can contribute to the selection of candidate implementation technologies by using the assessment of ESB capabilities in Chapter 5, “ESB and SOA component implementations” on page 133.

1. Function and Data Interfaces: Are the existing functions and their data interfaces good matches to the services you want to provide, or can appropriate modification or aggregation be performed in the applications?
   - If not, the following capabilities will be required either in adapters or the ESB infrastructure, or will have to be performed by service requesters:
     - Integration: service / message aggregation
     - Message processing: message and data transformations, message / service aggregation and correlation
2. Common Business Data Model: Should the services be exposed in the form of some common business data model? If so, do the systems that implement those services already support that model, or can they be made to do so?
   - If not, the following capabilities will be required either in adapters or the ESB infrastructure:
     - Integration: service / message aggregation

    - Message processing: message and data transformations

3. Technologies for Interoperability: Are open standards required, or can appropriate interoperability be achieved through Enterprise Application Integration middleware? If open standards are required, which ones are appropriate?
   - Although the use of open standards is one way to achieve interoperability, proprietary Enterprise Application Integration middleware is also highly interoperable, and often significantly more mature. Many organizations also have extensive existing infrastructures that can, in some scenarios, minimize the benefits of open standards.
   - In scenarios where open standards are required, Web services are perhaps the most obvious choice in this context. However, JMS, JDBC, basic XML, or several other technologies such as EDI or industry XML formats can also be applied.
   - In practice, interoperability between different implementations of the same standards cannot always be assumed, particularly if the standards are recent or emerging. In the case of Web services, the Web Services Interoperability Organization has recently published the Basic Profile for interoperability using SOAP and WSDL, and other profiles for more advanced standards will follow (including WS-Security and WS-Transaction). Until such profiles are comprehensive, established, and widely supported by products, the use of open standards will not guarantee, and may not always facilitate, interoperability.
   - Relevant capabilities:
     - Communications: communication technologies, protocols, and standards
     - Service interaction: service interface definition, service messaging models
     - Integration (all)
     - Message processing: message and data transformations
     - Quality of service (all)
     - Service levels (all)
4. Advanced Interaction Characteristics: Is support for basic communication protocols and standards (such as WebSphere MQ, SOAP, WSDL) required, or more sophisticated capabilities such as WS-Security and WS-Transaction?
   - Requirements to support more sophisticated standards will impose more stringent constraints on the options for implementation technologies and may imply the use of less mature technologies. Relevant capabilities are:
     - Communications (all)

    - Integration: protocol transformation
    - Quality of service (all)
    - Security (all)

5. Adoption of Standards at the Edge or in the Heart of Infrastructure: Where changes to the message formats and protocols that are used by an existing infrastructure are under consideration, including the adoption of open standards, are the changes required throughout the existing infrastructure, or can they be applied at the edges? If EAI technology is in use or under consideration, does that have its own internal format, or can it process open standards as an internal format?
   - Any use of open standards is likely to be driven by needs to extend access, so it is usually more important that they are available at the interfaces to existing infrastructure than that they are used internally.
   - If internal use of specific formats, technologies or standards is required, this will place constraints on the choice of implementation technology.
   - Relevant capabilities:
     - Communications: communication technologies, protocols, and standards
     - Service interaction: service interface definition, service messaging models
     - Integration (all)
     - Message processing: message and data transformations
     - Quality of service (all)
     - Service levels (all)
6. Technology Support in Existing Systems: Do the systems that implement functions that should be exposed as services support the required technologies or open standards such as SOAP, JMS, or XML?
   - If not, either the ESB infrastructure or adapters will need the capability to transform between the required open standards and the formats that are supported by the service providers:
     - Communications: communication technologies, protocols, and standards
     - Service interaction: service interface definition, service messaging models
     - Integration (all)
     - Message processing: message and data transformations

7. Legacy XML Support and Processing: Where access to legacy systems is required using more recent XML-based technologies (including SOAP, but also basic XML with Enterprise Application Integration middleware), is direct support (such as CICS SOAP support) available, or is a separate adapter required? Does the legacy platform support XML processing, and is such processing a sensible use of the platform capabilities?
   - If for any of these reasons a required SOAP or XML capability will not be made available on a legacy platform, appropriate transformation capability will be required either in adapters (such as JCA or WebSphere Business Integration Adapters), in an integration tier, or in the ESB infrastructure:
     - Integration: legacy and application adapters, protocol transformation
     - Message processing: message and data transformations
8. Availability of Services in EAI Infrastructure: If an EAI technology is already available, does it implement services as message flows with appropriate function and interface granularity, or can it be made to do so? What connectivity protocols does it support (for example JCA, SOAP, WebSphere MQ, RMI)?
   - If existing message flows do not provide the required services, then additional flows will be needed to perform transformations. If the EAI technology does not directly support the required standards, a gateway component can be added:
     - Communications: communication technologies, protocols, and standards
     - Service interaction: service interface definition, service messaging model
     - Integration: connectivity to EAI middleware, service mapping, protocol transformation
     - Message processing: message and data transformations, message / service aggregation and correlation
9. Service Provider Protection: What measure of protection should be afforded to the service provider systems from service requester channels in the form of workload buffering, security, logging, and so forth?
   - Such buffering will often be a role of the ESB infrastructure, and define some of the capabilities it requires. If specific service provider systems (such as legacy transactional systems) have additional needs for protection, a dedicated integration tier could be used:
     - Communications: synchronous and asynchronous messaging
     - Integration: legacy and application adapters, protocol transformation
     - Security (all)

    - Service level (all)
    - Message processing: store and forward
    - Management and autonomic: logging, metering, monitoring

10. Consistent and Controlled Service Enablement: How many services should be enabled? What aspects of enablement should be consistent across the services, and how can consistency be enforced, perhaps across multiple platforms and applications?

    – If very few services are involved, a simple point-to-point integration model may be appropriate. However, if more are involved or are likely to become so over time, the addition of a control point such as that provided by an ESB becomes increasingly beneficial:

    - Communications: routing, addressing
    - Service interaction: substitution of service implementation, service directory and discovery
    - Integration: service mapping, protocol transformation
    - Security (all)
    - Management and autonomic: service provisioning and registration, logging, metering, monitoring

11. External Access to Services: Are the service interactions contained within the organization or are some external?

    – If external access is required, a gateway component can be used to provide additional control. This is often the case in addition to an ESB infrastructure that is implemented within a single organization, as the requirements for security and service routing may differ for services made available externally:

    - Communications: routing, addressing
    - Service interaction: substitution of service implementation, service directory and discovery
    - Integration: service mapping, protocol transformation
    - Security (all)
    - Management and autonomic: service provisioning and registration, logging, metering, monitoring
    - Modeling: public versus private models

12. Business Service Choreography: Are there requirements for business service choreography, and do they involve short-lived or long-lived (stateful) processes, or both? Do they include manual activities?

    – Where these requirements constitute business function, the choreography should be implemented in a Business Service Choreography component separate from the ESB. Requirements to support long-lived stateful processes or manual activities will place constraints on the choice of implementation technology.
    – Long-lived services or event models may require message processing: Message / service aggregation and correlation capabilities.

13. Service Level Requirements: What service level requirements should the infrastructure support, such as service response time, throughput, and availability, and how is it required to scale over time?

    – Some of the candidate technologies for ESB implementation are relatively new and may only have been tested against limited service levels. Similarly, because the relevant open standards are either recent or emerging, support for them in more established products and technologies is also new.
    – For the foreseeable future, critical architectural decisions will be concerned with balancing the benefits of specific open standards supported by emerging or mature product technologies against service level requirements. These point-in-time decisions will have to recognize that some standards, and product support for them, are relatively mature (such as XML and SOAP), some (such as WS-Security) are newer, and some (such as WS-Transaction) are still emerging.
    – The trade-off between the benefits of standards and proven service-level characteristics often drive a mixed approach that combines standards-compliant technologies with proprietary or customized technologies in an ESB and SOA architecture.
    – Relevant capabilities:
        • Communication: communication technologies, standards, and protocols
        • Integration: connectivity to EAI middleware, protocol transformation
        • Quality of service (all)
        • Security (all)
        • Service level (all)
        • Message processing: message and data transformations

14. Security Requirements: Is a point-to-point or end-to-end security model required (should the ESB simply authorize service requests, or should it pass

the requestor identity or other credentials through to the service provider)? Is there a need to integrate the service security model with application or legacy security systems?

- If point-to-point security is acceptable, several existing solutions (such as SSL, J2EE security for database access, and adapter security models) can be applied. If end-to-end security is required, the WS-Security standard is a possibility if it is supported by all of the involved systems. Alternatively, a customized model using custom message headers or passing security information as application data could be used.
- Relevant capabilities:
  - Security (all)

4.6 Summary and next steps in the design process

In this chapter, we have introduced and analyzed the Enterprise Service Bus and its role in SOA. Through the use of a capability model and with reference to the various coupling styles applied to aspects of service interactions in an SOA, we were able to provide a model for analyzing common ESB and SOA scenarios to specify both appropriate implementation patterns and detailed capabilities that should be supported by the technologies that are used to implement those patterns.

The next step in the design process is to select specific technologies, standards, and other techniques to implement the chosen patterns. Where those patterns are variations of the Enterprise Service Bus as described in this chapter, the mapping of the patterns to runtime patterns and specific implementation technologies can be found in Chapter 5, “ESB and SOA component implementations” on page 133.

More detail concerning such implementations is the subject of the scenario implementation chapters in Part 3 of this redbook. Where implementation patterns are either simpler, such as Direct Connection, or refer to other SOA components, such as Serial Process or Parallel Process, we have provided references to further resources in this chapter.

5

ESB and SOA component implementations

In this chapter, we discuss technologies that can be used to implement the Enterprise Service Bus, and other related components of a service-oriented architecture. This chapter contains the following sections:

- Runtime product descriptions

  Provides a description of the products that are used throughout this book.

- SOA component product mappings

  Summarizes the SOA components we have discussed in the book and identifies product mappings for them.

- Product capabilities for the Enterprise Service Bus

  As the role of the ESB capabilities is so crucial to identifying appropriate implementation technologies, this section presents an analysis of product features against the ESB capability model.

5.1 Runtime product descriptions

This section describes IBM products that are discussed and used throughout this book. After we briefly describe each product, we illustrate how to choose which products to use to implement the various patterns that were described in the previous chapter.

5.1.1 IBM WebSphere Application Server V5.1

IBM WebSphere Application Server V5.1 represents a continuation of the evolution to a single, integrated, cost-effective, Web services–enabled, J2EE server foundation for applications that offers customers:

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

With IBM WebSphere Application Server V5.1, IBM enables customers to expand their business opportunities and productivity through a world-class infrastructure that is ready for e-business on demand.

IBM WebSphere Application Server base V5.1 provides a robust application deployment environment for single-server, light production situations.

It contains a base application server that supports the full J2EE 1.3 environment. It enables a full range of enterprise integration and offers enhanced security, performance, availability, connectivity, and scalability options. Administration is done through a Web-based interface or through a scripting tool.

It includes support for new Web services standards, including JAX-RPC and Web services for J2EE (both part of the J2EE 1.4 release), and for WS-Security. It also provides runtime support for SOAP messaging using WebSphere MQ as a transport.

More information about IBM WebSphere Application Server base V5.1 can be found at:

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

5.1.2 IBM WebSphere MQ V5.3

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

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

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

Dynamic workload distribution is another important feature of WebSphere MQ. This feature shares the workload among a group of queue managers that are part of the same cluster. This 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.

More information can be found at the IBM WebSphere MQ Web site:

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

5.1.3 IBM WebSphere Application Server Network Deployment V5.1

WebSphere Application Server Network Deployment is an extension to WebSphere Application Server base. It includes the Web Services Gateway component. Although it is shipped with WebSphere Application Server Network Deployment, the Web Services Gateway can also run in a standalone WebSphere Application Server instance.

The Web Services Gateway is a runtime component that provides configurable mapping between Web service requesters and providers. Services that are defined with WSDL can be mapped to available transport channels. The basic components of the Web Services Gateway are:

- Channels that define the entry points into the Web Services Gateway.

- Services that map to WSDL-described Web service implementations.
- UDDI references to manage the publishing of an exposed Web service to a private or public UDDI registry.
- Filters that are used to intercept service invocations that come into the Web Services Gateway and act on the services.
- JAX-RPC handlers that are used to intercept service requests between the service requester and the Web Services Gateway (inbound handler) and between the Web Services Gateway and the target service (outbound handler). JAX-RPC handlers can perform similar tasks as filters but provide an approach based on open and accepted standards.

More information about IBM WebSphere Application Server Network Deployment V5.1 can be found at:

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

5.1.4 IBM WebSphere Business Integration Message Broker V5.0

WebSphere Business Integration Message Broker V5.0 extends the messaging capabilities of WebSphere MQ by adding message routing, transformation, and publish/subscribe features. Message Broker provides a runtime environment that executes message flows, which consist of a graph of nodes that represent the processing that is needed for integrating applications. They can be designed to perform a wide variety of functions, including:

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

WebSphere Business Integration Message Broker also provides these features:

- Simplified integration of existing applications with Web services through the transformation and routing of SOAP messages, as well as logging of Web services transactions.
- Mediation between Web services and other integration models as both a service requester and a service provider.

- Compliance with standards such as Web Services Definition Language (WSDL), Simple Object Access Protocol (SOAP), and Hypertext Transfer Protocol (HTTP).
- Integrated WebSphere MQ transports for enterprise, mobile, real-time, multicast, and telemetry endpoints.
- Standards-based metadata including XML schema and WSDL.

More information about IBM WebSphere Business Integration Message Broker V5.0 can be found at:

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

5.1.5 IBM WebSphere Business Integration Server Foundation V5.1

WebSphere Business Integration Server Foundation V5.1 builds on WebSphere Application Server to provide a premier Java 2 Enterprise Edition (J2EE) and Web services technology-based application platform for deploying enterprise Web services solutions for dynamic e-business on demand.

It includes WebSphere Process Choreographer, which provides IBM WebSphere Application Server with the ability to choreograph intra-enterprise and inter-enterprise services into business processes that are described using the open-standard Business Process Execution Language for Web Services (BPEL4WS). Each activity in the business process is defined as a service using WSDL. The business process in itself is also exposed as a WSDL-defined Web service.

The business processes that are implemented in an enterprise typically require a mixture of human and IT resources, and these processes are supported by Process Choreographer. A process is a directed graph that starts with an Input node and ends with an Output node. A process itself is described in WSDL. Its input and output are described as WSDL messages.

A process can contain many activities. An activity can be the invocation of an EJB, a Java class, a service, or another process. A process can also be event driven. For example, it can be paused to wait for an event and then resumed when a message arrives.

WebSphere Process Choreographer supports processes that can be:

- Long-running (macro-flow) and interruptible (requiring human intervention)
- Short-running (micro-flow) and part of a one-business transaction

More information about IBM WebSphere Business Integration Server Foundation V5.1 can be found at:

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

5.1.6 IBM WebSphere InterChange Server V4.2

WebSphere InterChange Server is a integration process management broker that is used to integrate applications. It has a common business object model on which process logic (called a *collaboration*) executes, which is isolated from the endpoint applications. This facilitates the reuse of process integration logic and enables execution consistency and simplified ongoing maintenance.

WebSphere InterChange Server provides business object transformation, intelligent routing of messages, and a runtime container for business process integration logic. This means that it is easy to manage stateful interactions between multiple disparate integration endpoints.

WebSphere InterChange Server has a library of prebuilt integration processes and business objects. These describe commonly occurring business process functions that typically are used to integrate packaged (COTS) applications.

Using WebSphere InterChange Server with the WebSphere Business Integration Adapter for Web Services:

- Collaborations can be exposed as Web services.
- Collaborations can consume Web services using service calls.
- Mediation between service requesters and service providers can be implemented.
- SOAP messages can be processed using the SOAP Data Handler.

The capabilities of WebSphere InterChange Server should be considered in conjunction with the WebSphere Business Integration Adapter for Web Services (which includes the SOAP Data Handler):

- Routing can be performed in collaborations and polymorphic maps.
- Aggregation of calls to multiple service providers is implemented by:
  - Making service calls serially.
  - Using asynchronous service calls.
- Transformation is performed in maps.

Find more information about IBM WebSphere InterChange Server V4.2 at:

http://www.ibm.com/software/integration/wbiserver/ics/

5.1.7 IBM WebSphere MQ Workflow V3.5

IBM WebSphere MQ Workflow is aimed at helping organizations automate their business processes. It is best suited for automating *process-centric* business processes, as its strength is in people-based workflows. WebSphere MQ Workflow supports many different staff delegation algorithms, and it can drive system integration via the implementation of one or more User Program Execution Servers (UPES). A UPES activity within a process sends a WebSphere MQ XML-formatted message to a user-defined WebSphere MQ queue. The UPES is a custom program that receives this message and performs the request, constructs an XML response that the WebSphere MQ Workflow server will understand, and sends the message back to the server. Because a UPES request is delivered by WebSphere MQ, this means that the UPES can run on any of the many operating system platforms that WebSphere MQ runs on.

WebSphere MQ Workflow can be used with WebSphere Business Integration Workbench to provide real-time process tracking. Also, real production metrics from the audit trail can be used within WebSphere Business Integration Monitor to analyze an organization's processes. Using these products together provides any customer with the information that is needed to achieve continuous process improvement.

WebSphere MQ Workflow is built on proven IBM technology. The communication layer is built on WebSphere MQ, and the database can use DB2®. The servers can run on many OS platforms, including z/OS®, providing flexibility for an organization's environment. The operational model that can be developed for a WebSphere MQ Workflow implementation can be designed for highly available environments.

More information about IBM WebSphere MQ Workflow V3.5 can be found at:

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

5.1.8 IBM WebSphere Business Integration Connect V4.2.1

WebSphere Business Integration Connect is a B2B community management solution that provides extensive support for partner definition and management of document interactions. The Advanced and Enterprise editions provide SOAP/HTTP support as one of the transport solutions for trading partner integration in addition to other transport formats including FTP, SMTP, RosettaNet, cXML, and AS2. It runs on WebSphere Application Server and has three main components:

- Community Console

  This provides the administration interface for setting up the trading community and monitoring the flow of documents and processes within the community.

- Receiver

  This component handles secure and reliable receipt of documents, independent of the transport protocol from community participants over the Internet. It writes the documents to shared file service for the Document Manager to process.

  Documents can enter the system from within the enterprise via a directory or over HTTP, HTTPS, or JMS. The Document Manager detects the document and routes it to the community participant.

- Document Manager

  This component receives documents, performs any user-configured validation processing, and then delivers the document to its final destination. Subsystems can encrypt and decrypt the document, perform digital signature verification, transform and validate XML, and log entries about the processing of the document.

WebSphere Business Integration Connect provides SOAP bi-directional passthrough support between enterprise and trading partners. There is no introspection, parsing, or validation of the SOAP body during processing. It provides support for SOAP V1.1, WSDL V1.1, WS-I Basic Profile 1.0, and supports RPC-encoded, RPC-literal, and Document-literal binding styles.

Setting up a Web service in WebSphere Business Integration Connect involves uploading the WSDL definition file that is provided by the trading partner for the service and setting the target Web service public URL. The endpoint for the service is not delineated from WebSphere Business Integration Connect.

WebSphere Business Integration Connect provides support for SOAP/HTTP and SOAP/HTTPS. Currently, there is no support for SOAP/JMS or transport rebinding. Web services security is via SSL and basic authentication; at present no WS-Security integration is provided. Non-repudiation is provided using authentication and auditing.

More information about IBM WebSphere Business Integration Connect V4.2.1 can be found at:

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

5.2 SOA component product mappings

In the previous chapters, we identified and described the following components, or node types, that play a role in an SOA infrastructure:

- Direct Connection

- Service Directory
- Business Service Choreography
- Enterprise Service Bus patterns:
  - Enterprise Service Bus
  - Exposed ESB
  - Exposed ESB Gateway

In this book, we do not concentrate on the Direct Connection pattern, other than the discussion in Chapter 6, “Endpoint enablement roadmap” on page 153. For more information about using the Direct Connection pattern in an SOA, refer to *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303.

In the following sections, we discuss the remaining patterns and components, and indicate which technologies may be used to implement them.

5.2.1 Service Directory product mappings

The Service Directory component may exist either as part of the Enterprise Service Bus or as a component in its own right. Where this component exists as part of the Enterprise Service Bus, often it is implemented by the ESB technology. However, some implementations might involve linking the ESB to an external service directory.

In this section, we discuss only the cases where an explicit Service Directory is required, in which case Service Directory can be considered a node type.

Several approaches could be taken to implementing a Service Directory node:

- UDDI

  A dedicated UDDI technology, such as that available with WebSphere Application Server Network Deployment, to provide Service Directory based on Web services standards.
- LDAP server

  LDAP directory servers, such as IBM Tivoli Directory Server, are not specifically intended to provide directories of services but they do provide a flexible directory format that could be adapted to this requirement.
- Customized database

  Where neither UDDI or LDAP are appropriate, a customized directory can be implemented using a database. This is the approach that is taken in Chapter 5, “ESB and SOA component implementations” on page 133.

- Other customized solutions

  The preceding three options have concentrated on technologies that provide standards support, use supported products, and are suitable for dynamic service discovery. However, there are many other possibilities such as the provision of simple service directories using collaboration technologies such as Lotus Notes® and Domino, or through basic Web sites. Such directories may not be so suited to dynamic service discovery, but they do fulfill the basic goal of publishing information that describes available services in a SOA.

5.2.2 Business Service Choreography product mappings

Where a separate Business Service Choreography component is required to model and execute service choreographies that encapsulate business logic, a separate runtime component usually is provided. An alternative would be to use the same technology that is used to implement the Enterprise Service Bus (assuming that the technology has appropriate capabilities); however, this is unlikely to be a common scenario, as the intents of the two components are rather separate.

Candidate products for implementing the Business Service Choreography component include:

- WebSphere Business Integration Server Foundation
- WebSphere Business Integration InterChange Server
- WebSphere MQ Workflow

The ESB capabilities of WebSphere Business Integration Server Foundation V5.1 and WebSphere Business Integration Interchange Server are described in 5.3.1, “Assessment of ESB capabilities by product” on page 145. However, that section does not consider their suitability for implementation of the Business Service Choreography component.

Implementation of service choreography using WebSphere Business Integration Server Foundation V5.1 is described in Chapter 10, “Business Service Choreography” on page 271.

For more details about service choreography, consult *Patterns: Serial and Parallel Processes for Process Choreography and Workflow*, SG24-6306.

5.2.3 SOA product mappings

The previous chapters defined the following SOA patterns:

- ESB pattern
- ESB Gateway pattern

- Exposed ESB pattern
- Exposed ESB Gateway composite pattern

These are all examples of the Hub runtime pattern in the Patterns for e-business Process Integration patterns - they all represent an infrastructure component for mediating interactions. Additionally, they must support the ESB capabilities that are identified through the discussion of requirement scenarios, architectural issues, and solution patterns as presented in Chapter 4, “Enterprise Service Bus and SOA patterns” on page 73. In any individual situation, those capabilities will determine which technologies are appropriate for implementing an ESB, ESB Gateway, Exposed ESB, or Exposed ESB Gateway pattern.

Depending on the exact ESB capabilities that are required, the following approaches could be taken:

- Web Services Gateway (available with WebSphere Application Server Network Deployment)
- WebSphere Business Integration Message Broker
- WebSphere Business Integration Connect (although this is primarily suited to the Exposed ESB Gateway pattern)
- Customized implementation based on WebSphere Application Server
- Customized implementation based on WebSphere MQ

There are certain other technologies that, although primarily intended for other uses, provide certain features that are appropriate to Enterprise Service Bus capability. Though they would not be considered prime candidates for ESB implementation, they do provide features that promote ESB-like access to their function:

- WebSphere Business Integration Server Foundation
- WebSphere Business Integration Interchange Server

In this book, we cannot describe all of the possible implementations of each pattern in each technology, but the more common product-based solutions are covered as follows:

- Implementation of the ESB pattern using the Web Services Gateway is described in Chapter 8, “Enterprise Service Bus: Router variation” on page 175.
- Implementation of the ESB pattern using WebSphere Business Integration Message Broker is described in Chapter 9, “Enterprise Service Bus: Broker variation” on page 219.

- Implementation of the Exposed ESB Gateway composite pattern using the Web Services Gateway is described in Chapter 11, “Exposed ESB Gateway composite pattern” on page 299.

An assessment of product features against the ESB capability model is made in 5.3, “Product capabilities for the Enterprise Service Bus” on page 144.

Standard, proprietary, and customized approaches

The choice among standard, proprietary, and customized approaches to various aspects of ESB implementation is discussed in 4.5.8, “Architecture decision questions” on page 125, particularly question 3 on page 126, “Technologies for Interoperability.” There are two senses in which this question applies when identifying implementation technologies for the ESB:

- The decision whether to base ESB implementation primarily on a product, such as WebSphere Business Integration Message Broker, or on a custom implementation perhaps using features of other products such as WebSphere Application Server or WebSphere MQ.
- The decision to implement each aspect of service interaction, such as security, transactionality, and service interface definition, using open standards, proprietary features, or customized formats.

The first case is the more fundamental, but similar issues apply in both:

- Balancing the cost of development and maintenance of specific solutions against the availability of interoperable or open-standard product implementations that support appropriate service levels.
- Balancing requirements to support emerging Web services specifications against requirements for high service levels.
- Balancing the desire for a single simple solution against the desire to apply the best technical solution to each requirement across the enterprise.

Other aspects of customized implementation are discussed in 5.3.5, “Options for customized implementation” on page 149.

5.3 Product capabilities for the Enterprise Service Bus

As the role of the ESB capabilities is so crucial to identifying appropriate implementation technologies, this section presents an analysis of product features against the ESB capability model.

5.3.1 Assessment of ESB capabilities by product

Table 5-1 rates each product against the ESB capabilities that are defined in the previous chapter. WebSphere Application Server and WebSphere MQ are included to indicate the base features that they provide that may be leveraged in a more customized solution.

Table 5-1 Matching ESB capabilities to products

Enterprise Service Bus Capability	WebSphere Application Server V5.1	WebSphere MQ V5.3	WebSphere Application Server Network Deployment V5.1	WebSphere Business Integration Message Broker V5.0	WebSphere Business Integration Connect V4.2.1
Communication	Medium	Limited	Limited	Strong	Fairly Strong
Integration	Medium	Strong	Medium	Strong	Limited
Security	Strong	Limited	Fairly Strong	Limited	Strong
Message processing	Medium	Limited	Limited	Strong	Medium
Modeling	Limited	Limited	Limited	Fairly Strong	Fairly Strong
Service interaction	Strong	Limited	Strong	Strong	Strong
Quality of service	Strong	Strong	Medium	Strong	Medium
Service level	Strong	Strong	Medium	Strong	Strong
Management and autonomic	Medium	Medium	Medium	Medium	Fairly Strong
Infrastructure intelligence	Limited	Limited	Limited	Limited	Limited

5.3.2 WebSphere Business Integration Message Broker

Here we list the capabilities of WebSphere Business Integration Message Broker as they relate to an ESB:

- Communication
  - Dynamic routing using database or service lookup.
  - Support for request-response, fire and forget, and publish and subscribe.
  - Asynchronous and synchronous delivery support.

  - Event-driven processing (using WebSphere Business Integration Adapters).
  - Transaction management support and assured once-only delivery of persistent WebSphere MQ messages.
- Service Interaction
  - The implementation of a service as a message flow can be changed without affecting the service requester.
  - A service provider's implementation can change without affecting its access from a message flow.
  - A message flow can handle a service request with no, partial, or complete SOAP (or message) validation and processing.
- Integration
  - Integration with relational databases (which can be under transactional control). This access can be used for data enrichment to provide additional information that is required for service provider processing.
  - Connectivity to applications, such as COTS packages, using WebSphere Business Integration Adapters.
  - Aggregation: The processing of a single service request from a client by fanning out several requests to service providers and aggregating the results into a single response.
  - Protocol transformation support: HTTP to JMS and vice versa.
- Service Level
  - High performance and throughput characteristics (documented on the IBM SupportPacs Web site).
  - Partial parsing is automatically supported. For example, the SOAP header could be parsed and the body transported.
  - High levels of availability can be achieved using multiple brokers and execution groups underpinned with WebSphere MQ clustering.
- Quality of service
  - Message flows can be transactions, using WebSphere MQ as a transaction manager, for example.
  - Use of WebSphere MQ as the JMS transport assures the once-only delivery of persistent messages.
- Security
  - Authentication and authorization for access over JMS can be provided by WebSphere MQ infrastructure.

  - Authentication and authorization for access over HTTP must be provided by an external HTTP server. (Custom security can be implemented within WebSphere Business Integration Message Broker and this could make a call out to an LDAP directory using a plug-in node that is provided as a SupportPac™.)
- Message processing
  - Decoding and encoding of SOAP messages (including faults).
  - Ability to define logic, such as dynamic routing, based on content.
  - SOAP header validation. (XML and custom message formats can also be validated.)
  - Comprehensive message transformation facilities. Stylesheet transformation can be used.
  - Able to act as an intermediary between a service requester and service provider, with independence between the two.
- Management and autonomic
  - Messages can be logged in whole or in part. Such information can be used to input into metering and monitoring.
  - Services that are created from message flows could be published in an external directory such as UDDI. A client could then use UDDI to discover the services.
- Modeling
  - Broad support for data formats, including SOAP. Business entity support through XSDs.
  - Stateless.
- Infrastructure intelligence
  - Message flow processing can be adapted according to business rules.

5.3.3 WebSphere Application Server Network Deployment

Here we list the capabilities of the Web Services Gateway component of WebSphere Application Server Network Deployment as it relates to an ESB.

- Communication
  - Internet routing supported with proxy.
  - Supports SOAP, UDDI, WSDL, and Web services for J2EE.
  - Uses SOAP/JMS channels that support point-to-point and publish/subscribe styles of messaging.
  - Asynchronous messaging supplied through the JMS interface.

  - Synchronous messaging can be managed through Apache SOAP or SOAP/HTTP transports.
- Service interaction
  - WSDL is used by the Gateway as the service interface definition and the service implementation definition.
  - Can publish services to a UDDI directory (either a private IBM, or external, directory).
- Integration
  - Java provides the language interface.
  - Protocol transformation support.
- Service level
  - Selective SOAP parsing is supported. This option can be enabled or disabled at the service level and does not affect the ability to apply filters to the SOAP message.
  - Web service performance can be monitored through the Performance Monitoring Infrastructure (PMI), including the number of asynchronous and synchronous requests and responses.
- Security
  - Tokens, keys, signatures, and encryption according to the WS-Security specification may be applied to every deployed Web service.
  - Authentication and authorization is available through the WebSphere Application Server.
  - HTTPS is supported.
  - Proxy authentication can enabled.
  - Message-level security, as part of the WS-Security specification, is implemented using JAX-RPC.
- Message processing
  - JAX-RPC is used as message handlers to provide the ability to extend the basic service implementation. For example, JAX-RPC handlers can be used to manage message-level security (encryption/decryption), logging, and auditing.

5.3.4 WebSphere Business Integration Connect

The capabilities of WebSphere Business Integration Connect as they relate to an ESB are:

- Communication
  - Support for EDI, Web Services, SMTP, and HTTP communication with external partners.
  - The Advanced and Enterprise editions provide SOAP/HTTP support.
- Service interaction
  - Supports SOAP and WSDL.
  - Provides a directory of services and interactions.
- Integration
  - Provided through Web services or JMS, so the integration capabilities of the WebSphere Business Integration family can be used.
- Service level
  - Takes advantage of WebSphere Application Server's proven resilience, performance, and scalability.
- Security
  - Message encryption and decryption, digital signature verification, and provides features for non-repudiation.
- Message processing
  - Transform and validate XML.
- Management and autonomic
  - Logs entries about the processing of messages.
- Modeling
  - Provides features to model interactions with partners, and support for various industry standard communication mechanisms such as RosettaNet.

5.3.5 Options for customized implementation

As discussed in 5.2.3, "SOA product mappings" on page 142, customized effort may be required in an ESB implementation either to implement a customized ESB or to support service interaction characteristics that are not supported by standards or product features.

Such customized development should be weighed carefully against business benefit; unless it is restricted to relatively simple function, it can imply significant development and maintenance costs, not to mention the eventual cost of migration to an open-standard or product-supported solution.

Customized implementation using application server features

Most application servers, including WebSphere Application Server, provide several runtime features that support ESB capabilities, such as support for Web Services standards and for programming models that enable data and message manipulation. The development tooling for such application servers, such as WebSphere Studio Application Developer, includes tools and wizards to simplify the development of application, framework, or infrastructure code to leverage those runtime features. Service interactions using many protocols, such as SOAP/HTTP, SOAP/JMS, and RMI/IIOP can be enabled this way. As application servers support these standards in the same way as more dedicated Enterprise Service Bus technologies, such as through JAX-RPC–compliant implementations, they can provide similar features, such as the ability to implement intermediaries using JAX-RPC handlers.

Such application servers and tooling also provide support for a wide variety of integration methods, either directly (databases, J2EE connectors, and so forth), or through support for Enterprise Application Integration middleware (such as WebSphere MQ).

In its most basic form, tooling wizards could be used to create basic code to expose a variety of systems as service providers. A danger in doing this, however, is that the result is, effectively, a large set of unrelated point-to-point interactions that all happen to be hosted in the same application server runtime. This effect can be reduced by adding framework or utility functions, such as common approaches to logging, security, and data format transformation. However, there is a need to limit the extent of application development that is undertaken in such an approach, so as to avoid implementing an overly sophisticated solution that will be costly to maintain.

Customized implementation using messaging middleware

Customized development based on messaging middleware may be an option when Web services support is not critical and quality-of-service requirements demand the use of mature middleware. Of course, there is a spectrum of approaches to the use of messaging middleware to support SOA: Chapter 9, “Enterprise Service Bus: Broker variation” on page 219 describes the use of WebSphere Business Integration Message Broker to provide an ESB infrastructure that supports Web Services interactions. There is no reason why the same infrastructure should not support a variety of other message-based and event-based interactions that are part of an overall SOA but that, for various reasons, do not use the Web services standards.

Where the Web services standards are not used, either for specific interactions or for all interactions within an SOA, several decisions must be made:

- In order to fulfill the criteria for service interactions, some form of explicit interface definition is required. This definition is usually, but not always, machine-readable (for example, WSDL can be read by application development tools or ESB middleware). Machine-readable interface specifications increase the options that are available to loosely couple service interactions. In some cases, a proprietary interface definition might be provided by the messaging middleware, in other cases a customized model might be used.
- Some form of service messaging model is also required, such as to provide a message body using some format for application data and message headers and describing other aspects of the interaction such as security or transactional context. Again, these features may be provided by middleware, or a customized approach could be used. It is important to note that there are many choices of non-Web services messaging models that nevertheless provide interoperability and conform to open standards. XML or industry formats that are based on it, such as ebXML, are good examples.
- Applications that are service requesters will have to invoke and receive service requests that are defined by the interface definition and that use the service messaging model. When the interface definition or messaging model is proprietary, applications will either have to construct appropriate messages themselves, or a framework will have to be provided to assist them.

Given the rapid emergence and maturity of Web services standards, the amount of effort that should be put into customized implementations is questionable, unless the implementation is to provide support for an open standard that is not directly supported by the product. Preferably, as a starting position, service interactions should use open standards or supported features of product technologies in order to minimize development, maintenance, and migration cost.

In terms of fulfilling the minimum ESB capabilities, it is also important to consider how routing, addressing, and service directory features are provided. Some middleware technologies, such as WebSphere Business Integration Message Broker, provide sophisticated support for these capabilities, whereas others such as the base WebSphere MQ product do not, so they have to be implemented in some other manner.

Finally, approaches that are based on messaging middleware can be combined with approaches that are based on Web services in an overall infrastructure for SOA. In some cases, the same technology, such as WebSphere Business

Integration Message Broker, support both approaches. In other cases, it might be combined, perhaps with a gateway technology such as WebSphere Business Integration Connect, to provide additional features.

Customized implementation of emerging open standards

Whichever approach is taken to ESB implementation, the rapidly changing and emerging nature of the Web services standards means that it is always possible that the chosen technologies do not provide support for a particular Web services standard, as that standard is too recent. However, it may be that there is a desire to use such an emergent standard to implement some aspect of service interactions.

In this case, it is possible to use features of the ESB technology runtime to provide a customized implementation of an open standard, rather than using product features. Although a development and maintenance cost is involved in doing this, the use of an open standard reduces the eventual migration cost to a product-supported solution.

6

Endpoint enablement roadmap

An Enterprise Service Bus (ESB) can interoperate with a wide variety of endpoints. Perhaps the most obvious is an endpoint wrapped as a Web service. In this instance, there are several options for the protocol that is used to communicate with the Web service. This chapter discusses two: an HTTP service bus and a JMS service bus.

The evolution of service-oriented architecture can drive organizations to enable service-based access to their legacy transactions. Organizations should therefore plan to define and implement the most appropriate form of service enablement for each of their legacy systems; options include leveraging legacy XML or Web services support (such as is provided by CICS Transaction Server), the use of adapters or connectors that are fronted by application servers, and the use of EAI or Gateway technology to provide legacy connectivity.

This chapter discusses two such service enablement options: WebSphere Business Integration Adapters and the J2EE Connector Architecture.

6.1 Web services

Web services are a common implementation of an SOA. They offer a standard interface for many different types of endpoints (such as J2EE, messaging, and enterprise systems) and therefore are well suited to an ESB architecture.

Web services can communicate using SOAP messages over a variety of protocols. Each protocol effectively provides a service bus between multiple endpoints. The most common service bus implementations include:

- HTTP service bus
- JMS service bus

6.1.1 HTTP service bus

The HTTP service bus is the most familiar way to send requests and responses between service requesters and providers, due to wide adoption of the HTTP protocol through the creation of the Internet. Even non-transactional companies are connected to the Internet and are therefore able to use the HTTP service bus inter-enterprise.

Organizations are already well-equipped to handle HTTP security requirements and have put in measures to ensure that only valid HTTP requests are received through firewalls, proxy servers, demilitarized zones, HTTP servers, authorization, authentication procedures, and so forth. As a result, HTTP is usually one of the first transport layers an organization would use when thinking about inter-enterprise solutions.

Figure 6-1 on page 155 shows the implementation of services on an HTTP service bus. This is an implementation of the Application Integration::Direct Connection runtime pattern.

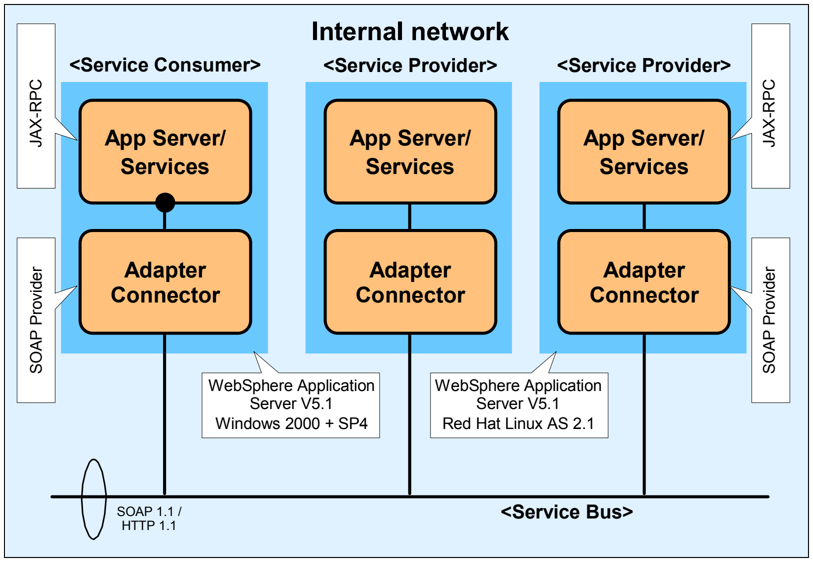


Figure 6-1 Runtime pattern and Product mapping for an HTTP bus

Use of an ESB extends the HTTP service bus concept. It enables the service requester to communicate using HTTP and permits the service provider to receive the request using a different transport mechanism.

Many ESB implementation providers have an HTTP service bus in addition to at least one other protocol. Any of these protocols can be used for ESB interactions and often are chosen based on service-level requirements.

Advantages of HTTP

There are several advantages to using HTTP as a transport for Web services interactions, including:

- HTTP is a widely adopted protocol. Any organization with a Web server has implemented HTTP, and any client that uses a Web browser uses HTTP. Therefore, the HTTP infrastructure is widely available.
- The HTTP protocol is open and deployed on many different system types, including non-traditional computing devices such as PDAs.
- Most enterprises allow HTTP to travel freely through protocol firewalls. Therefore, there are fewer barriers to extended enterprise use of HTTP as a transport for Web services.

Disadvantages of HTTP

HTTP is a lightweight and stateless protocol that was not originally designed to carry application data. Disadvantages of using it for Web services include:

- The protocol is stateless. If any state data is required to maintain an application session, the applications must create and manage the state data.
- HTTP is not a reliable protocol. If reliable delivery of application data is required, the application must either:
  - Develop a reliability framework, such as exchanging receipt messages.
  - Use a more reliable protocol.

Further information

For more information about the design, development, and runtime of an HTTP service bus, consult Chapter 5, “HTTP service bus” in the redbook *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303.

6.1.2 JMS service bus

While it does not quite provide the level of interoperability based on the wide adoption that the HTTP service bus can boast, the JMS service bus brings advantages in terms of quality of service.

JMS, part of the J2EE standard, provides a conventional way to create, send, and receive enterprise messages. A JMS service bus can provide asynchronous and reliable messaging to a Web service invocation. This means that the requestor can receive acknowledgement of assured delivery and communicate with enterprises that may not be available.

Figure 6-2 on page 157 shows the implementation of services on an MQ bus, using JMS. This is an implementation of the Application Integration::Direct Connection runtime pattern.

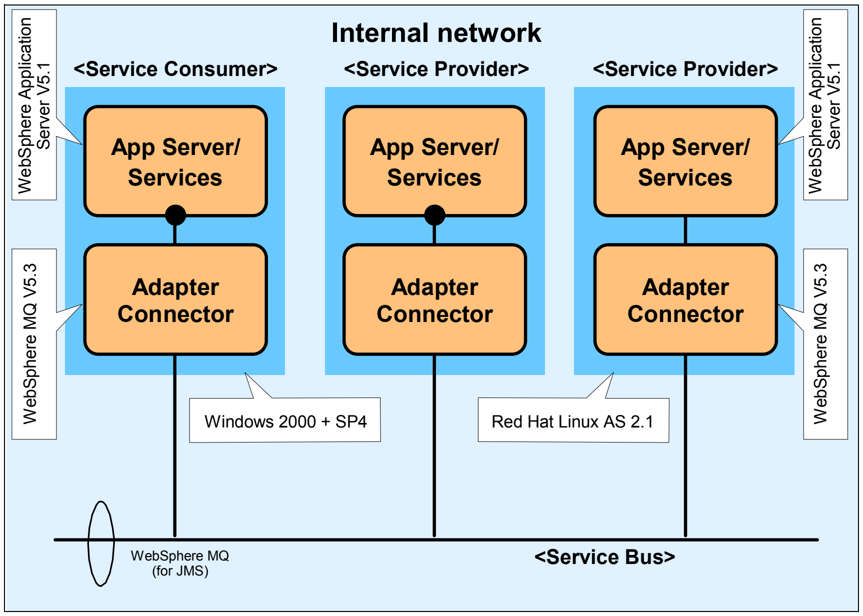


Figure 6-2 Runtime pattern and product mapping for an MQ bus using JMS

As with the HTTP service bus, use of an ESB extends the JMS service bus, enabling service requesters and providers to communicate using different protocols.

IBM JMS implementations

IBM provides two implementations of JMS:

1. A JMS provider included with WebSphere Application Server V5.1 (Embedded Messaging), which can be used for asynchronous communication between applications running on WebSphere Application Server V5.1 servers.
2. IBM WebSphere MQ V5.3 includes built-in JMS Provider support with enhanced performance features for integrating JMS applications with other applications. IBM WebSphere MQ takes care of network interfaces, assures once and once-only delivery of messages, deals with communications protocols, dynamically distributes workload across available resources, and handles recovery after system problems. IBM WebSphere MQ is available for most popular operating system platforms.

Advantages of JMS

There are several advantages to using JMS as a transport for Web services interactions, including:

- JMS provides a more reliable transport than alternatives such as HTTP.
- Asynchronous requests can be deployed readily.
- It leverages existing, enterprise-proven messaging systems.

Disadvantages of JMS

Although JMS is an open standard for Java-based systems, the actual transport system must be provided by a software product. Therefore, there are several considerations, including:

- The communicating Web services must have access to JMS providers that can communicate with each other. Generally, this implies that the same product must be installed. For example, both systems must have IBM WebSphere MQ installed.
- JMS is a Java-based standard and is not as readily accessible to systems that are not based on Java.

These disadvantages only occur for use with the Direct Connection pattern. That is, the service requestor and service provider have adopted use of the same product (IBM WebSphere MQ, for example). When using an Enterprise Service Bus, this is not necessarily a requirement.

Keep in mind that the capabilities of an ESB, which are listed in 4.3, “A capability model for the Enterprise Service Bus” on page 82, identify protocol transformation as ESB functionality. This means that by using an Enterprise Service Bus as a key component of the SOA, two organizations can send and receive messages regardless of the original message structure.

Using the ESB capabilities enables:

- A service provider to receive a JMS message that is sent from a service requester that originated from different JMS providers.
- A service provider to receive a JMS message that is sent from a service requester that originated the call using HTTP.

Further Information

For more information about the design, development, and runtime of a JMS service bus, consult Chapter 6, “JMS service bus” in the redbook *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303.

6.2 WebSphere Business Integration Adapters

This section describes how to communicate with existing business logic by using WebSphere Business Integration Adapters. It discusses application interface choices, available adapters, and using adapters with the Enterprise Service Bus.

6.2.1 Application interfaces

Organizations may have custom or off-the-shelf applications that provide important business information. Perhaps the company has purchased (or merged with) another organization and would like to use that application as a product endpoint. How the application exchanges information to the ESB depends on the application accessibility options.

Alternative ways an application can exchange information with the ESB include:

- Application-provided Web service interface

  Some applications and legacy application servers have adopted the open standards philosophy and have included a Web services interface. The WSDL defines the interface to communicate directly with the application business logic. Where possible, taking a direct approach is always preferred.

- Non-Web service interface

  The application does not expose business logic via Web services. An application-specific adapter can be supplied to provide a basic intermediary between the application API and the ESB. The adapter has two parts: the adapter framework and the application-specific component.

- Service wrapper as interface to adapter

  In some cases the adapter may not supply the correct protocol (JMS, for example) that the ESB expects. In this case, the adapter would be Web service enabled.

Figure 6-3 on page 160 illustrates these alternatives.

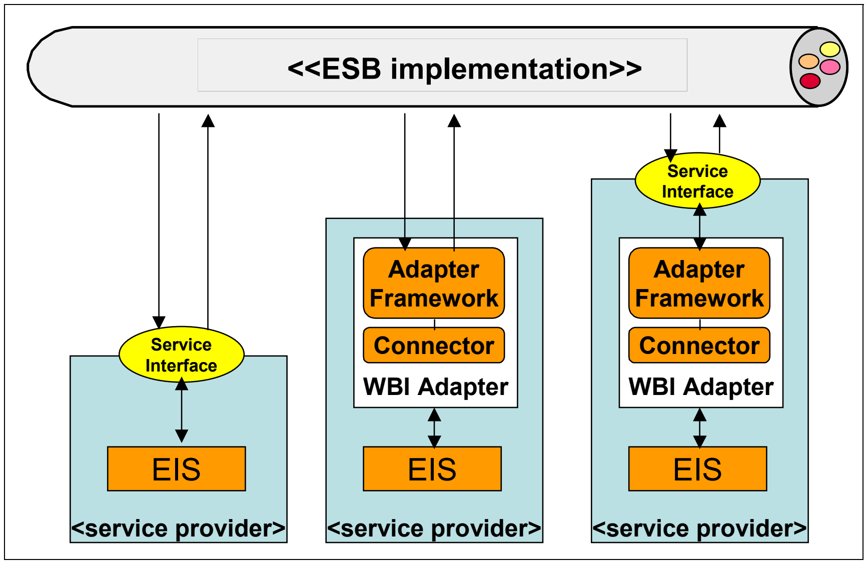


Figure 6-3 Options for accessing an Enterprise Information Store (EIS) from an ESB

6.2.2 Available adapters

Adapters are useful for extracting data and transactional information from packaged applications and belong to one of the following four areas:

- Application adapters

  Including Ariba Buyer, Clarify, eMatrix, JDEdwards, mySAP.com, Oracle Applications, PeopleSoft, Portal Infranet, QAD MFG/PRO, Retek, SAP Exchange Infrastructure, Siebel, WebSphere Commerce, and others.

- Technology adapters

  Including ACORD XML, COM, CORBA, e-mail, EJB, Exchange, FIX Protocol, iSeries™, iSoft Peer-to-Peer Agent, JDBC (SQL and stored procedure access), JMS, JText, Lotus Domino, SWIFT, WebSphere MQ, WebSphere Business Integration Message Broker, WebSphere MQ Workflow, Web Services, and XML.

  Some of these technology adapters can use Data Handlers. These include Data Handlers for EDI, SOAP, XML, and various text formats.

- Mainframe adapters

  Including ADABAS, CICS Transaction Server, DB2 (on zOS), IDMS Database, IMS Transaction Manager, IMS Database Manager, Natural, and VSAM.

- Adapter development tools

  The WebSphere Business Integration Adapter Development Tools are an integrated toolkit that provides a framework for developing custom adapters in Java or C++.

For a complete and current list of WebSphere Business Integration Adapters, see:

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

6.2.3 Capabilities of the adapters and the ESB

The WebSphere Business Integration Adapters are built using a common adapter framework and all typically have the following characteristics:

- Bi-directional
  - An event detection mechanism is provided to enable the adapter to capture application-generated events. These events trigger processing in an integration broker. This is known as event processing.
  - The adapter can make changes in applications at the request of an integration broker. This is known as request processing.
- Configurable through metadata

  The adapter represents application entities as business objects. These are defined in metadata and used by the adapter at runtime. This means that the adapter operates generically for all business objects and business object operations (create, retrieve, update, and delete, for example).

  The metadata holds information that defines how to invoke application APIs for each business object.

  Business object definitions can be created directly from application entities at development time for many adapters using an Object Discovery Agent.
- A reusable infrastructure component

  The adapter is a reusable infrastructure component across many integration brokers. Currently these include:
  - WebSphere InterChange Server
  - WebSphere Business Integration Message Broker
  - WebSphere Business Integration Server Foundation
- Multi-threaded business object processing

  Most adapters are built using Java. This means that request processing can be multi-threaded. For event processing, multiple adapter instances can be created to provide scalability and resilience options.

The WebSphere Business Integration Adapters communicate with integration brokers by sending and receiving business objects over JMS. (Note that there are additional options when using WebSphere InterChange Server.) When looking at how an ESB can interact with an adapter, there are two options:

- An ESB implementation can access the adapter by using the native business object over JMS mechanism.
- A Web services wrapper can be implemented to access the adapter. Care should be exercised when considering this option to make sure that real benefits would be realized, such as increasing reuse.

Figure 6-3 on page 160 shows an enterprise with existing applications providing connectivity to the ESB using off-the-shelf adapters to provide them with an SOA without changing their back-end systems. (Note that event processing may require application changes to record the events. IBM provides artifacts and samples that are specific to each adapter).

The service requestor calls a service provided by the Enterprise Service Bus, the bus provides the required capabilities (security and transformation, for example) and the request is sent to the adapter. The adapter takes in the request and translates it to the EIS language. The response is followed through the reverse of the same process.

6.2.4 Intelligent use of the adapters with an ESB

One of the inherent issues of adapter technology revolves around the complexity of interacting with many applications, and commercial off-the-shelf (COTS) packages in particular.

WebSphere InterChange Server has a tight coupling with the WebSphere Business Integration Adapters. It is designed to manage the process of interacting with applications and is particularly well suited to COTS packages.

For example, a new order record may consist of customer details, various contact information for queries and delivery, and a list of the products that are being ordered. Some COTS packages place restrictions on order placement (for example, an order can be placed only for a valid customer and if a delivery contact is correctly specified). WebSphere InterChange Server is designed to manage these interactions with the application so that it can verify that the prerequisite information correctly exists or to create or update it if it does not. Then the order can be created in the application with low risk of failure. The characteristics that define this behavior are configurable in prebuilt collaborations (integration processes) that IBM supplies to run on WebSphere InterChange Server.

The WebSphere Business Integration Adapter for Web Services can be used to provide a Web service interface to the collaboration from the ESB. Figure 6-4 shows how this might look.

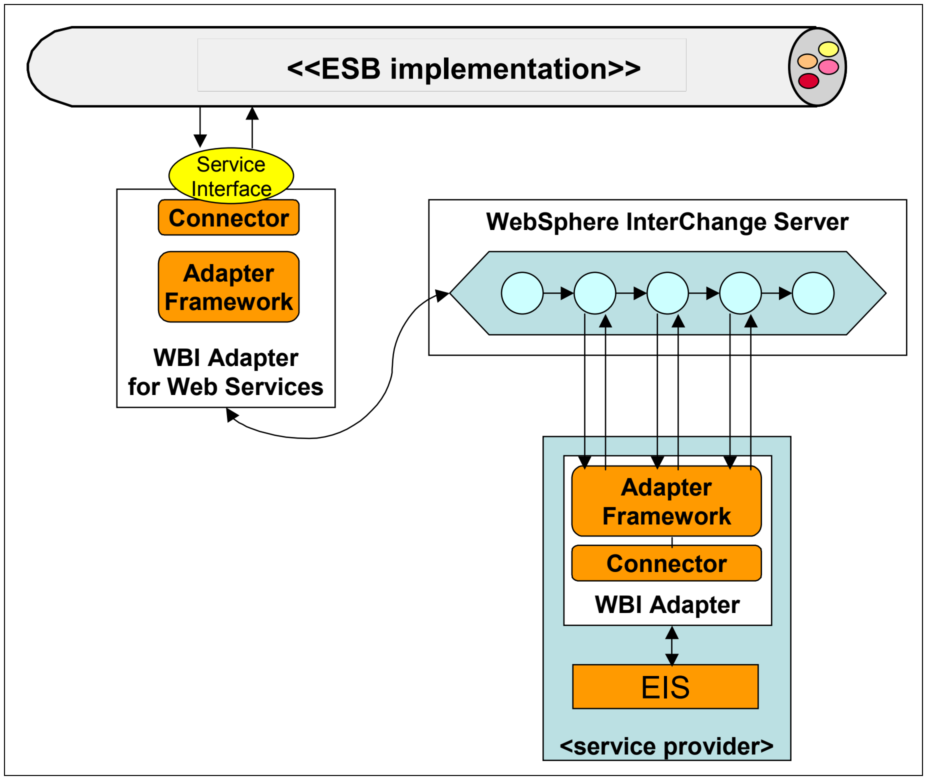


Figure 6-4 Using WebSphere InterChange Server to access applications

Note: The use of WebSphere InterChange Server as described here should not be confused with using WebSphere InterChange Server to provide:

- An implementation of the capabilities of the ESB
- Process management of services provided by the ESB

6.2.5 Further Information

Consult the redbook *Using Web Services for Business Integration*, SG24-6583, for further information.

6.3 J2EE Connector Architecture

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

J2EE Connector Architecture 1.0 support is a requirement of the J2EE 1.3 specification. It provides access to a range of systems through a common client interface (CCI) API. Application programmers code to the single API rather than having unique interfaces for each proprietary system. The link from the API to the enterprise system is called a resource adapter and is provided by a third-party vendor. This is somewhat analogous to the model for JDBC drivers. Resource adapters are packaged as resource adapter archive (RAR) files.

IBM supplies resource adapters for enterprise systems such as CICS Transaction Server and IMS Transaction Server.

IBM provides a tool to generate enterprise beans and Web services that encapsulate interactions with a resource adapter. This tool is part of WebSphere Studio Application Developer Integration Edition.

Figure 6-5 on page 165 shows how a J2EE Connector Architecture endpoint that is exposed as a Web service can be invoked using the Web Services Gateway as the ESB implementation. These protocols are used:

1. HTTP to communicate between the service requester and the ESB.
2. JMS to communicate between the ESB and the J2EE Connector Architecture Web service.
3. An enterprise-specific protocol to communicate between the J2EE Connector Architecture resource adapter and the legacy system application.

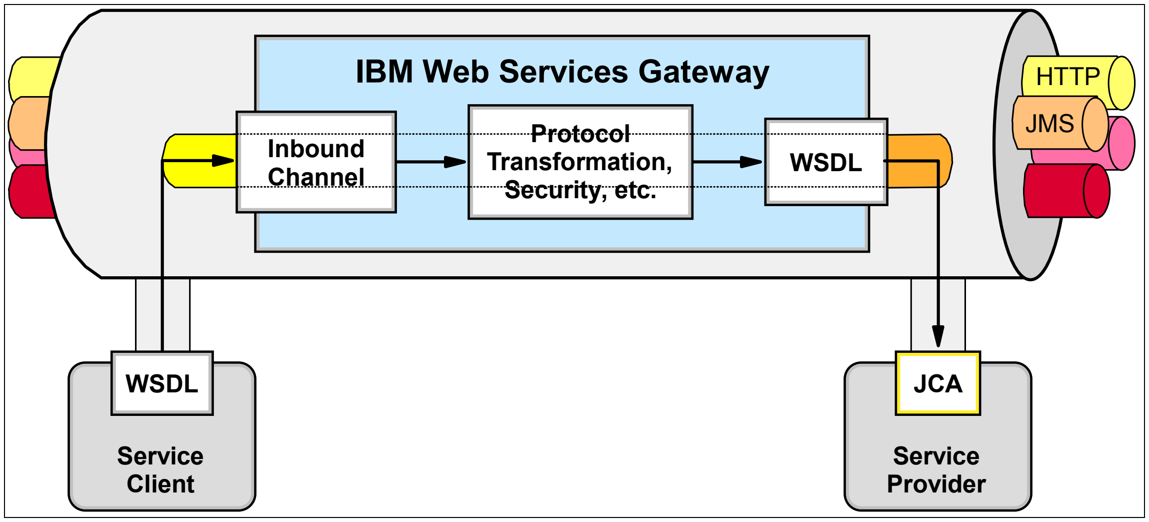


Figure 6-5 Using the J2EE Connector Architecture as an endpoint

Further information

For more information about building J2EE Connector Architecture Web services, consult *Exploring WebSphere Studio Application Developer Integration Edition*, SG24-6200. For information about the CICS resource adapters, see *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401.

6.4 Alternatives

While it is not the intent of this redbook to concentrate on endpoint enablement, it is critical to the ESB concept to ensure that enterprises are aware that there are a multitude of ways to wrapper existing code with a service interface.

Any application that is described using WSDL can be deployed as a Web service. The WSDL specification defines SOAP bindings for services; however, it is possible to add binding extensions.

Some of the available alternative endpoints are:

- Enterprise JavaBeans (EJB)

  Using RMI-IIOP as the access protocol, an EJB method can be invoked.

- Message-driven bean (MDB)

  Using an MDB enables an EJB to act as a JMS endpoint.

- Java class

  Services can be built from a single Java class using in-thread Java method invocations as the access protocol.

- SQL and stored procedures

  A database stored procedure can be exposed as a stateless session bean. That stateless session bean is then deployed into a SOAP router or as a SOAP service, changing the access mechanism from JDBC to RMI-IIOP and then to SOAP.

  It may be valid to access an endpoint database from the ESB using SQL. In this case it is unlikely to make sense to implement a Web service interface.

- Native MQ

  Many legacy systems can be accessed using WebSphere MQ messaging. In this case the queue is enabled as an endpoint for the ESB.

  When considering integrating such MQ-enabled applications with the ESB, the starting point will often be an XSD, a custom wire format or a tag-delimited string that is provided by the application. For example, these can be imported into WebSphere Business Integration Message Broker to create a message set. To provide access as a Web service, the message flow must implement the transformation of the message portions of the WSDL definition to this message set.

  When considering using JMS to access MQ-enabled applications, an assessment must be made to establish whether the applications can handle the JMS folder in the message header. For example, WebSphere Business Integration Message Broker can be used to strip this JMS folder.

- JAX-RPC

  Java API for XML-based RPC (JAX-RPC) is a highly interoperable way for service entities to interoperate over ESB. A service requestor who creates a JAX-RPC client can access a Web service that is running on a non-Java platform and vice versa.

Part 3

Scenario implementation

Now that the Enterprise Service Bus patterns have been defined and mapped to IBM product mappings, we implement some of these patterns in:

7

The business scenario used in this book

A common business scenario is used throughout Part 3 of this book: the WS-I Supply Chain Management sample application.

This chapter describes the sample scenario, the three stages of the business scenario, and the relevant chapter in which each stage is described.

7.1 WS-I sample application

The Web Services Interoperability Organization (WS-I) has developed a supply chain management business scenario to demonstrate features of the WS-I Basic Profile 1.0. The WS-I sample business scenario and the technical solution overview are described in the following documents:

- WS-I Supply Chain Management Use Cases 1.0
- WS-I Usage Scenarios 1.0
- WS-I Supply Chain Management Technical Architecture 1.0

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

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

We use this business scenario to show how the Patterns for e-business, service-oriented architecture, and Enterprise Service Bus approach can be used to develop solutions to real-world business requirements that are based on interoperability principles as defined in the WS-I Basic Profile.

This business scenario is a simplified supply chain for a consumer electronics retailer. This chapter describes the evolution of scenarios as the supply chain management organization moves from a directly connected intra-enterprise environment to an expanded organization that has divested its business and operates in an inter-enterprise environment.

7.2 Scenarios

This section describes the stages of the business scenario that is used throughout the book. Each stage builds a layer of complexity onto the previous stage.

7.2.1 Stage I: internal supply chain management on demand

In a typical B2C model, customers may access the retailer’s Web site, review the catalog, and place orders for products such as televisions, DVD players, and video cameras. The retailer system requests fulfilment of a consumer’s order from the internal company warehouse, which responds as to whether line items from the order can be filled. If stock for any line item falls below a minimum threshold in the warehouse, a replenishment order is sent to an external manufacturer using the B2B model.

The manufacturer does not immediately fulfill replenishment orders, but completes the order at some later time (possibly after completing a manufacturing run).

The business scenario is shown in Figure 7-1.

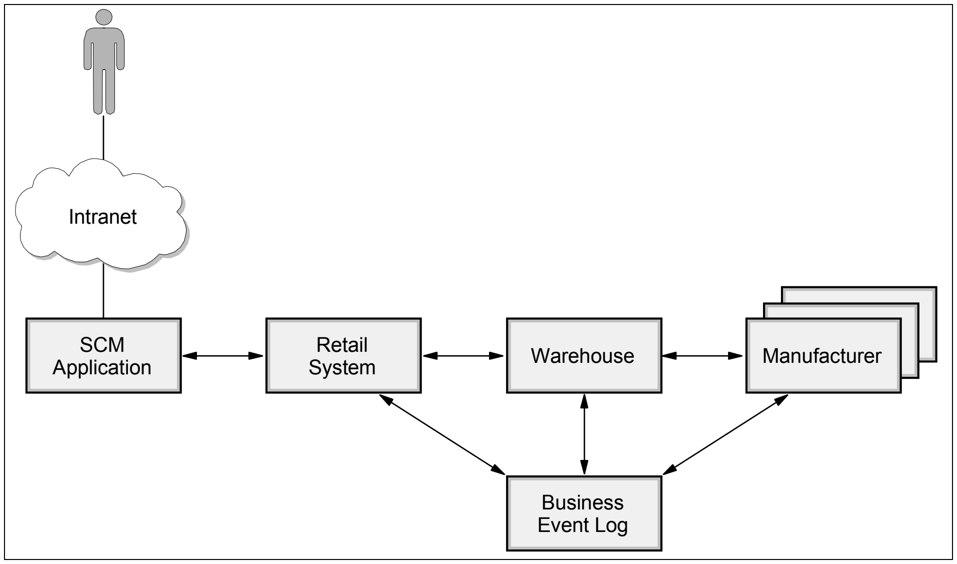


Figure 7-1 Stage 1: internal SCM

Initially, this business scenario uses the Direct Connection pattern to communicate between each service. We discuss how to take this application and model it with an Enterprise Service Bus: Router variation in Chapter 8, “Enterprise Service Bus: Router variation” on page 175.

7.2.2 Stage II: addition of warehouses

The company has a requirement to stock the parts that it offers to customers in more than one warehouse. However, the customer must see the order as a single transaction with the company. This is shown in Figure 7-2.

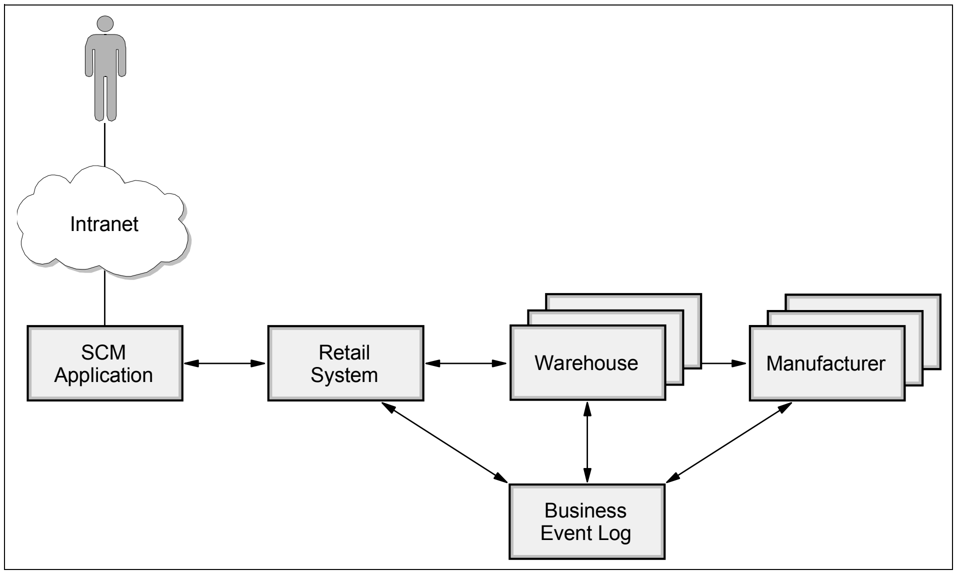


Figure 7-2 Stage II: additional warehouses

The Enterprise Service Bus: Router variation is no longer sufficient to model these interactions because aggregation is required from the retail system to the warehouse. In this instance, the Enterprise Service Bus: Broker variation can be used. This is discussed in Chapter 9, "Enterprise Service Bus: Broker variation" on page 219.

One of the three manufacturers is modeled as an external business process, mapping to the Serial Workflow pattern. It is accessed by the Enterprise Service Bus as discussed in Chapter 10, "Business Service Choreography" on page 271.

7.2.3 Stage III: divested inter-enterprise manufacturers

The company has decided to divest itself of the three manufacturers. Each will be sold off to another company or be established as a new company in its own right. Various interactions must now take place securely over the Internet. This is shown in Figure 7-3.

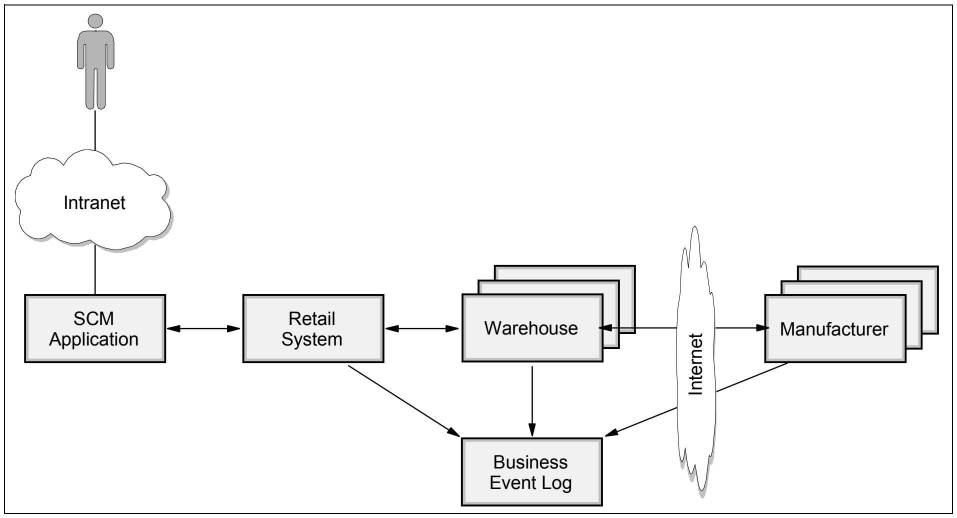


Figure 7-3 Stage III: divested manufacturers

Each manufacturer runs within its own Enterprise Service Bus. Communication between two Enterprise Services Bus implementations is represented by the Exposed ESB Gateway pattern. This is discussed in Chapter 11, “Exposed ESB Gateway composite pattern” on page 299.

8

Enterprise Service Bus: Router variation

In this chapter, the Enterprise Service Bus (ESB) is moved from concept to practical implementation by applying the service-oriented architecture (SOA) Enterprise Service Bus:Router variation. Using a simple scenario (described in Chapter 7, “The business scenario used in this book,” beginning on page 169) we demonstrate how an ESB can be designed and implemented using this pattern.

In this chapter, the following points are discussed:

- The sample business scenario that our solution is meant to address
- Design guidelines that describe the design approaches for using the Router variation to mediate service interactions in an ESB
- Development guidelines that show how to gain access and modify the flows that mediate service interactions in an ESB
- Runtime guidelines that discuss the considerations for configuring the Router

This chapter primarily focuses on the Web Services Gateway, which is supplied with WebSphere Application Server Network Deployment V5.1.1, as the product mapping of the Enterprise Service Bus: Router variation.

8.1 Business scenario

Our supplied business scenario represents a simplified SCM solution that is based on the WS-I SCM sample scenario as defined in Chapter 7, “The business scenario used in this book,” beginning on page 169. The sample scenario is used to illustrate the benefits of applying an SOA using an Enterprise Service Bus within an single enterprise.

In Figure 8-1, the Supply Chain Management application makes requests to the Retail System to help customers buy electronics goods online. The Retailer fulfills stock from the Warehouse and the Warehouse replenishes stock from the Manufacturers.

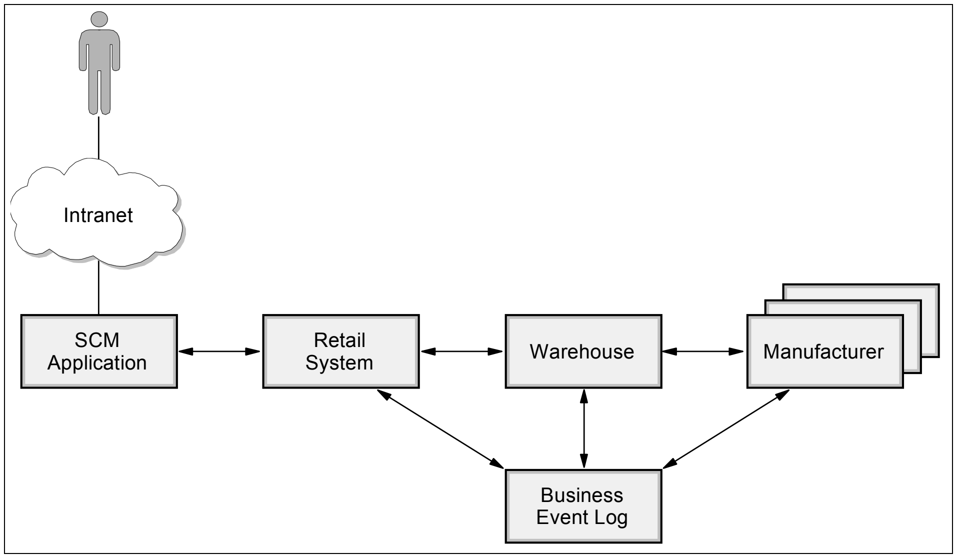


Figure 8-1 High-level business context showing the existing infrastructure

The organization has a few concerns with their current ability to meet market demands. Competition is increasing, customers have more options and opportunities, and the enterprise is under management pressure to do more for less. Consequently, the proposed solution must focus on solving the following business issues:

- Quicker response to change:
  - Improve the ability to respond to changes including both business requirements and changes in technology.
  - Mergers and acquisitions means that their service providers often change. These changes must be implemented with speed in a central location.

  - Rapidly grow the business without constant infrastructure change.
- Reduce costs:
  - Suppliers throughout the supply chain have different infrastructures. Managing all of these differences can be costly due to increases in trained resources, time, and training.
  - Reuse of existing code by combining independent business requirements in an innovative way.
  - Utilize existing resources more efficiently.
  - Make use of industry open standards in all possible locations of the solution.

8.2 Design guidelines

This section discusses the design guidelines relating to the Enterprise Service Bus: Router variation. The business scenario that is used in this chapter is based on Stage I of the business scenario described in 7.2, “Scenarios” on page 170.

8.2.1 Design overview

Figure 8-2 shows an overview of the steps that might be taken to design a solution to address business requirements. We follow these steps in this section.

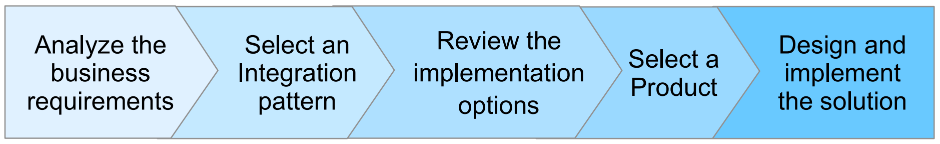

Analyze the business requirements
Select an Integration pattern
Review the implementation options
Select a Product
Design and implement the solution

Figure 8-2 Design overview

Selecting the pattern

This business scenario has the following interaction requirements:

- Only one warehouse provider is used.
- A warehouse calls each manufacturer in isolation. Interactions with more than one manufacturer requires multiple calls.

Choosing the relevant SOA pattern

The simplest form of mediation is the Enterprise Service Bus: Router variation. This uses the Process Integration Router pattern. This pattern meets the needs of the business scenario in this chapter, because no aggregation is required. The Router application pattern is shown in Figure 8-3 on page 178.

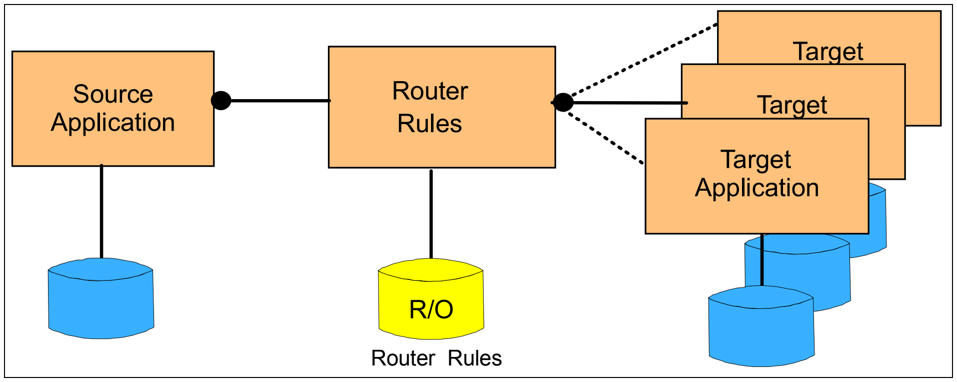


Figure 8-3 Router application pattern

The Router application pattern is a variation of the Broker application pattern that dictates that a single request received by the Router is routed to only one target.

For further information on the Router pattern, consult *Patterns: Broker Interactions for Intra- and Inter-enterprise*, SG24-6075.

Applying the SOA pattern to the scenario

The Enterprise Service Bus: Router variation is implemented for our business scenario as shown in Figure 8-4.

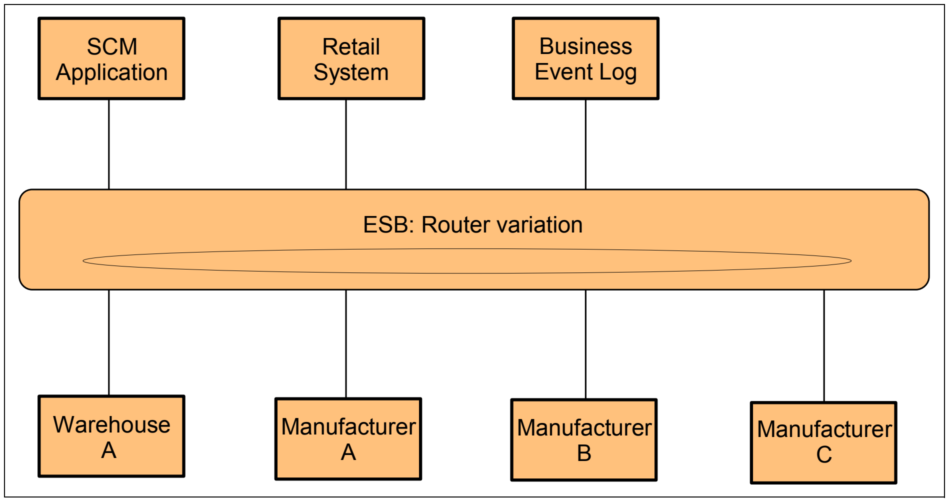


Figure 8-4 The Router pattern in the ESB

Implementing the Router variation in the ESB

The use of an Enterprise Service Bus in the context of a business scenario such as this offers the following benefits:

- The ESB is a bus with a single configuration and distributed deployment. Managing communications through the bus provides many advantages, including decoupling of service requesters and providers, and centralized control of a service namespace.
- Protocol conversion occurs inside the ESB (for example, SOAP/HTTP to SOAP/JMS). Requesters using one protocol can invoke services that are exposed using a different protocol.
- The ESB can provide logging and transformation of service requests and service responses.
- The ESB can provide centralized security for Web services invocations. It can, for example, authenticate all service requesters centrally.
- The ESB provides a common access point for service requesters that need access to services providers. The ESB intercepts and routes requests to the relevant service provider. A change in the location of the service provider only affects the ESB routing; the service provider location remains transparent to the service requester.

Specifically, we use the Enterprise Service Bus: Router variation to provide:

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

Product implementation options

We now consider ESB capabilities in the context of selecting a product to implement the Enterprise Service Bus: Router variation. Our product selection for the scenario was based on:

- The products that are currently available
- The ability of the products' capabilities to map to the requirements
- The existing infrastructure of our organization (for example, whether the company already uses one of the products)

The following products could be used to implement an Enterprise Service Bus: Router variation:

- WebSphere Application Server Network Deployment V5.1.1 Web Services Gateway

- WebSphere Business Integration Message Broker V5.0
- WebSphere InterChange Server V4.2

To help with selecting the appropriate product, refer to:

- The description of each product, in 5.1, “Runtime product descriptions” on page 134
- The ESB capabilities of each product, described in 5.3, “Product capabilities for the Enterprise Service Bus” on page 144

Product selection for scenario implementation

To address the requirements of the business scenario using the Router pattern, we selected the Web Services Gateway component of WebSphere Application Server Network Deployment V5.1.1. Figure 8-5 shows the product mapping.

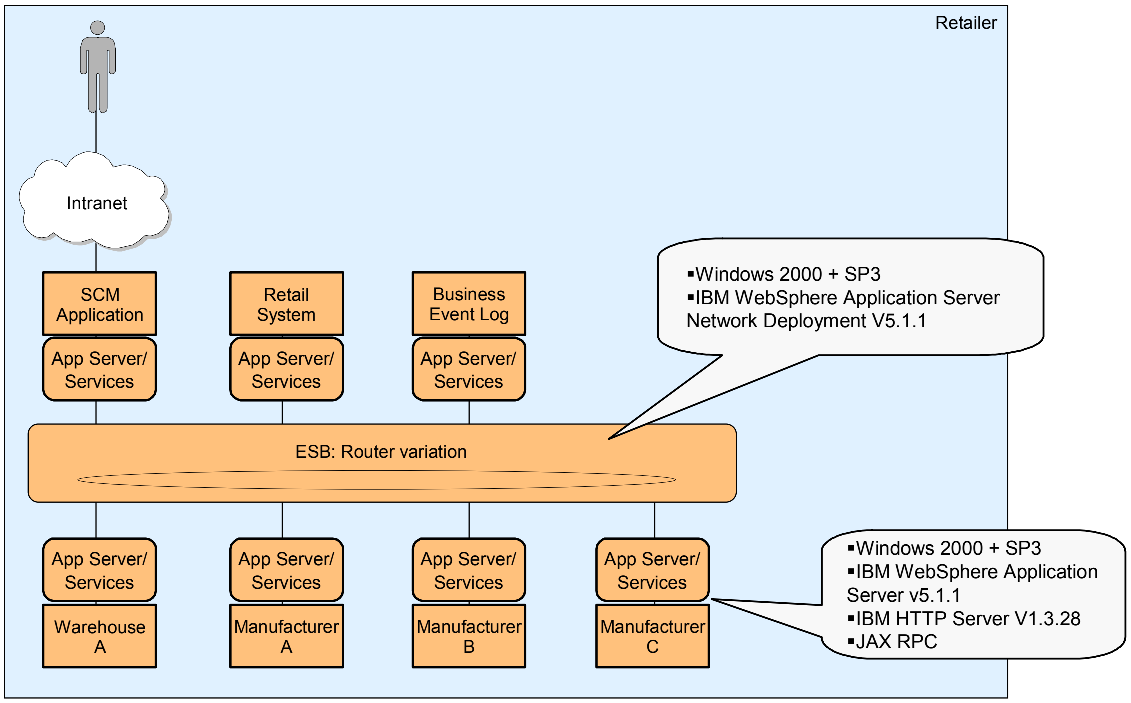


Figure 8-5 Product mapping for Application Integration::Router pattern on the ESB

In our lab environment, only Windows 2000 machines were used. However, it is likely that many enterprises will choose to implement their Enterprise Service Bus on other platforms such as IBM pSeries® running AIX.

The Router is leveraged to manage communication between all service requesters and service providers. Figure 8-6 illustrates a different view of our solution to the scenario requirements in terms of service requesters, service providers, and the protocols that are used for communication with the ESB.

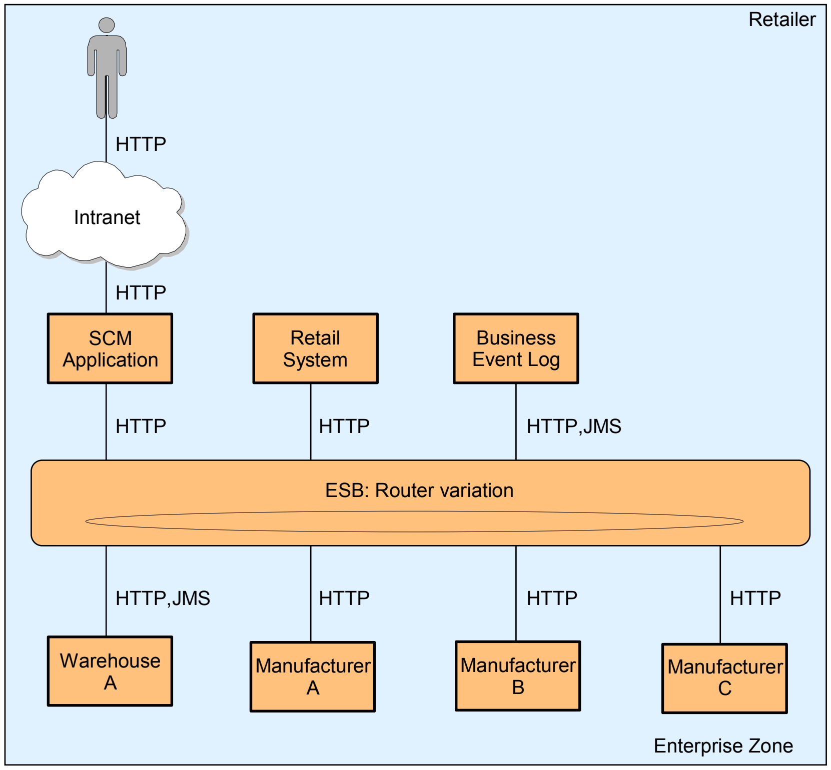


Figure 8-6 Solution to the scenario requirements

8.2.2 Router variation

This section describes design considerations for an Enterprise Service Bus: Router variation. First, the Web Services Gateway architecture is described, then design considerations for the Router variation are discussed.

Web Services Gateway architecture

The entry point to the Web Services Gateway is defined by a channel. A channel defines the protocol that you can use to access the Web Services Gateway. The incoming message is assessed on arrival through the channel to determine which service is required. Each service that is exposed by the Web Services Gateway has to be bound to one or more channels. One or more filters can be bound to a service for manipulating both request and response messages. Inbound JAX-RPC handlers can be bound to a channel, and outbound JAX-RPC handlers can be bound to the invocation of a target service deployed in the Web Services Gateway. The WSDL service definition specifies the provider service interface and implementation that is used to access the target service (Figure 8-7).

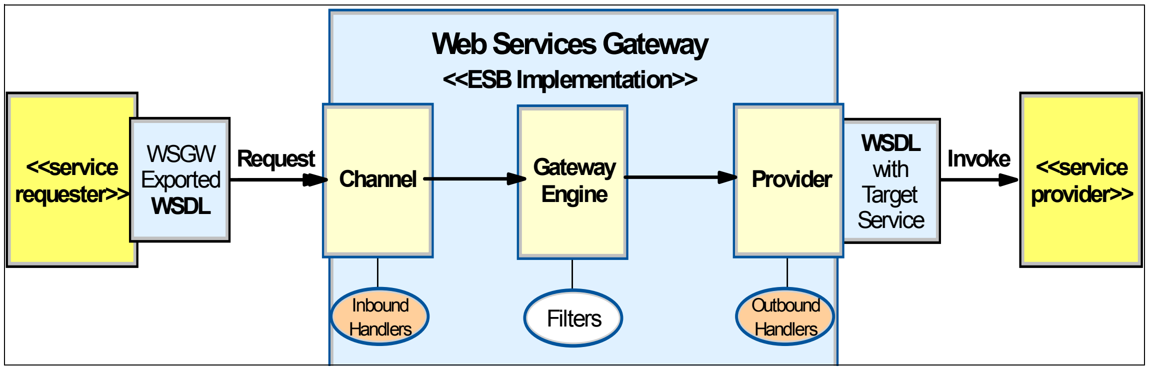


Figure 8-7 Web Services Gateway architecture

The response from the target service flows along the exact same path back to the service requester. There is no extra channel for an immediate response. However, the implementation of the handleResponse method of a JAX-RPC handler might be different to the implementation of the handleRequest method. Similarly one or more response filters can be deployed independently of the request filters.

The process of deploying a target service into a Web Services Gateway channel generates two different external WSDL files; an implementation definition and an interface definition. These new WSDL files can be exported for use by client applications, and are the externalization of the service capabilities that are offered by the internal target service. The implementation WSDL definition is used to simplify the connection process for a client, particularly when dynamic invocation is used. Having obtained the implementation definition, the client can then access the WSDL interface definition produced by Web Services Gateway, which provides full information about the target service (as presented externally by the Web Services Gateway).

For further information about Web Services Gateway V5.1.1 features, consult:

```
http://www.ibm.com/developerworks/websphere/techjournal/0403_flurry/0403_fl
urry.html
```

Design approach

This section discusses considerations for designing a Router ESB solution.

Design alternative: location of service definitions

The interface, binding, and service endpoint of a Web service are defined in WSDL files. The Router must have access to these service definitions and can get it through the following options:

- Copy the WSDL files to a local directory that the Router has access to.
- Publish the WSDL files on an HTTP server, where they can be retrieved by the Router.
- Publish the WSDL files to a private UDDI registry.

To achieve the first option with the Web Services Gateway, copy the WSDL files into a subfolder of the Web Services Gatewayinstallation directory within WebSphere Application Server. While this option is the easiest to configure, it should be considered for prototypes only.

Either the HTTP server or UDDI registry should be used for production-ready implementations. The HTTP server approach is the simplest to implement, but the UDDI approach offers several advantages:

- UDDI registries allow a fine degree of classification for Web services in your organization, which enables service requesters to quickly find the Web services to fit their needs.
- Providers of Web services within a enterprise could change frequently because of organizational changes or business reasons. From an administrative point of view it might be easier that each service provider keeps its published data up to date using UDDI, versus maintaining the WSDL files located on a HTTP server.

Reasons for not using the UDDI approach include:

- In an ESB scenario, the only client that uses the UDDI registry to obtain WSDL interface definition and implementation definitions of the Web services in your organization is the Router. The service requesters will obtain the interface definition and implementation definition from the Router. This limits the need for most UDDI features.
- If the Router supports the WS-Inspection specification (as the Web Services Gateway does) the service requesters can easily locate the WSDL documents of the Web services that are exposed by the Router.

The reasons for using UDDI in this scenario are more organizational than technical, so we do not use a UDDI registry in our sample scenario. Instead we place the WSDL files on an HTTP server.

Design alternative: client design considerations

When you introduce a Router ESB into an SOA, point your service requesters to this ESB rather than directly to the service provider. There are two alternatives for changing the service requester:

1. Use the WSDL files that are generated by the Router in the service requester, and regenerate the Web service client stubs.
2. Change the end-point address in your existing WSDL files on the client to point to the Router.

Generally, the first option is the safer and cleaner approach. You use a separate set of WSDL files in your service requesters to invoke a service that is exposed by the Router. These WSDL files are maintained by the Router, and they contain information such as the namespace of the Router, so they are decoupled from the service provider. The warning on this approach is that you have to make changes to your client code to reflect the changes.

The second option is easier to implement as it requires no changes to your client code. You simply change the end-point address to point to the Router and then redeploy your client application. As a drawback, the service requester uses the same interface description to invoke a service exposed by the Router as the Router uses to invoke the target service. This creates a tighter coupling with the service provider and could lead to problems where the Router and service provider publish to the same UDDI registry.

Mediation

Here we discuss the types of mediation that can be performed by a Router ESB.

Design alternative: selective SOAP parsing

If mediation of information in a SOAP body is not required, it is not necessary for a Router to parse this SOAP body. This brings several benefits:

- Performance is improved.
- Incompatibilities between the parser in the Router and the supplied SOAP body are not an issue.

By default, the Web Services Gateway parses the SOAP body of any message it receives. Use the selective SOAP parsing feature to turn parsing off. With selective SOAP parsing turned on, the SOAP body will pass through the Web Services Gateway unchecked.

Selective SOAP parsing is available only for the SOAP/HTTP channel. See "Design alternative: WS-Security" on page 187 for an explanation of how to use selective SOAP parsing and WS-Security together to establish end-to-end security.

Design alternative: custom versus standards-based mediation

A Router can intercept SOAP requests and examine them in several ways. For example, the Web Services Gateway offers two alternatives: filters or JAX-RPC handlers.

JAX-RPC is the standards-based way for heterogeneous systems to communicate. It is part of the J2EE Web services specification and is supported in WebSphere Application Server V5.1.1. As the standards-based approach, using JAX-RPC handlers with the Web Services Gateway is recommended. JAX-RPC handlers are available for SOAP/HTTP channels and SOAP/JMS channels.

JAX-RPC handlers can act between service requesters and the Web Services Gateway (inbound handlers) or between the Web Services Gateway and service providers (outbound handlers). You can have multiple inbound or outbound handlers, which are called a handler chain. JAX-RPC handlers generally work best when acting on SOAP headers. You can configure which headers a handler is intended to process.

Filters are a custom Web Services Gateway solution that do not benefit from the advantages of using a J2EE open standard such as JAX-RPC handlers. Filters are still supported and are the only option when using Apache SOAP channels.

Design alternative: static versus dynamic service provider routing

A Router can determine the service provider to route a request to in the following ways:

- Statically, by looking up the endpoint for a given Web service request from within the Router application or from an external routing table
- Dynamically, by examining the SOAP message and determining the endpoint, based on the content of this message.

The static approach has the following advantages:

- Easiest to configure, with no programming required
- Best performance
- Ensures that all requests for a particular Web service operation are forwarded to the same service provider
- Can be changed in the Router application or external routing table without requiring a change to the service requester

In most cases, the static approach is sufficient. However, the dynamic approach offers the chance to use mediation to examine the SOAP message to determine which service provider endpoint to use.

The Web Services Gateway implements this dynamic approach when it is configured in proxy operation mode.

In proxy operation mode you configure only one Web Services Gateway service, (the proxy service) without any target services. You can make any SOAP request to this service. As a consequence of not having to define target services, some other mechanism is needed to tell the Web Services Gateway where to route a request to. This will be achieved by writing and configuring a JAX-RPC handler that selects the appropriate target service and sets the endpoint URL to this target service into a SOAP message property named `transport.url`.

To select the appropriate target service's endpoint URL, an agreement regarding a routing parameter is needed between the service requesters and the routing handler:

- The service requester could place a HTTP request parameter into the request. The handler can receive the full request URL using the SOAP message property `inbound.url`.
- The service requester could place a routing parameter in a SOAP header.

The content of the agreed routing parameter can be either:

- The target service's endpoint URL

  or
- Some identification of the target service (such as the WSDL port type). The routing handler uses this information to look up the target service endpoint address from, for example, a UDDI registry.

As this book is being written, the proxy operation mode is supported only by the SOAP/HTTP channel. To use the proxy operation mode, you have to configure the Web Services Gateway to use selective SOAP parsing. (See "Design alternative: selective SOAP parsing" on page 184.)

The Web Services Gateway has to know whether a received message is a request-response message or a one-way message. Since the SOAP body will not be parsed, this information cannot be gathered from the message automatically. Therefore the clients have to place the operationMode HTTP parameter into the request. The value of this parameter can either by oneway or requestResponse. The default value is requestResponse.

Regarding the Enterprise Service Bus: Router variation, the proxy operation mode of the Web Services Gateway provides the following advantages:

- For an ESB it is very likely to apply a basic set of mediations such as logging and security to all service invocations. In proxy mode, these mediations must be configured only once by configuring a handler chain that is applied to the proxy service.

- The service providers in an organization are subject to change over time. Provided that each service provider keeps its routing directory entry (for example, in a UDDI registry) updated, the proxy operation mode allows for a zero ESB administration after it is set up.

Security

The aim of applying security in an SOA is to provide end-to-end security between a service requester and a service provider. By introducing a third party such as the Router between the communication, end-to-end security can no longer be achieved using transport-level security mechanisms such as SSL.

Design alternative: WS-Security

WS-Security describes how to secure SOAP messages that provide message integrity, message confidentiality, identification, and authentication. WS-Security provides security on the message level; you can apply any of the aforementioned security mechanisms to parts of the message.

With WS-Security, you can use the Router as a security endpoint to enforce different security constraints on inter-enterprise messages that are sent to the Router (versus intra-enterprise messages).

To address confidentiality, you can encrypt the SOAP body or parts of it. This enables a Router such as the Web Services Gateway to process the message without affecting end-to-end security by decrypting and encrypting data. In the case of the Web Services Gateway, selective SOAP parsing must be turned on so the encrypted SOAP body is not parsed.

There are some cases in which WS-Security protected messages cannot flow through the Router, such as a message with a digitally signed body. In this instance, the message contains SOAP headers with signing information, which would be altered in their transit through the Router, thereby invalidating the message.

In contrast to confidentiality where you want to establish an end-to-end security context, it might be worth thinking of intercepting authentication at the Router stage. Each service requester could be authenticated at the Router. After successful authentication at the Router, a new authentication context is established between the Router and the service provider.

This eases the necessary configuration effort for authentication because each service requester must be authenticated only at the Router, as opposed to authenticating each service requester at each potential service provider side.

Version 1 of the WS-Security specification was recently (April 2004) ratified by OASIS. The Web Services Gateway that is part of WebSphere Application Server Network Deployment Version 5.1 currently implements the WS-Security draft recommendation (April 2002).

Design alternative: basic authentication, role-based authorization

If you need a role-based authorization model to protect operations of services exposed by the Web Services Gateway, you can make use of the basic authentication and authorization mechanism based on the broader security features of WebSphere Application Server.

These mechanisms are:

- Web Services Gateway-level authentication
- Web service operation-level authorization

Using Gateway-level authentication, you can set up the Web Services Gateway channel applications to provide access only to service requesters that supply the correct user ID and password and, thus, restrict the access to the ESB.

Additionally, the Web Services Gateway enables you to assign the J2EE role-based authorization model to operations of services that are exposed by the Web Services Gateway. Consult the WebSphere Application Server Network Deployment InfoCenter for detailed information.

8.3 Development guidelines

To implement the Router ESB design, very little work is required in the development environment. We now develop a very basic JAX-RPC handler that writes the message contents unformatted to the systems log.

8.3.1 Using JAX-RPC handlers in an ESB

JAX-RPC handlers, which are compliant with J2EE Web services (as defined in JSR 101 and JSR 109), are a new feature of the Web Services Gateway that is part of WebSphere Application Server, Network Deployment Edition Version 5.1.

Handlers can be generic (such as a handler used for logging) or application-specific by processing a specific SOAP header only or by fulfilling a SOAP actor function.

JAX-RPC handlers add means of mediation to our ESB implementation:

- JAX-RPC handlers provide a standards-based approach for managing message-level security as defined by the WS-Security specification. Within an ESB implementation, it might be of particular interest to:
  - Add or act on identification and authentication information.
  - Validate message integrity.
- Logging

  Web service requests and responses that pass through the Web Services Gateway can be logged.
- Message transformation

  JAX-RPC handlers can add, delete, or modify any SOAP headers. JAX-RPC handlers can also modify the SOAP body of a message within the limitations defined in JSR 109. Some of the elements that cannot be changed in a SOAP body are:
  - The WSDL operation
  - The types of the parts in a message
  - The number of parts in a message
- Configure timeouts

  You can configure a timeout for outbound requests by setting a SOAP message property called `timeout`.
- Terminating a request

  You can terminate an incoming request in a handler (for example, after unsuccessful authentication of the service requester).
- Dynamic service routing
  - In scenarios in which you have configured multiple target services or target ports for a Web Services Gateway service, you can use a JAX-RPC handler to determine the correct service provider. The handler can choose the intended target service or target port from the list of available target services or target ports.
  - When operating the Web Services Gateway in proxy mode, use a JAX-RPC handler to identify the target service, and route this request to the target service by setting the endpoint URL in a SOAP message property called transport.url.
- Establish a shared context

  You can use JAX-RPC handlers to establish a shared context between the service requesters, service providers, or any JAX-RPC handler–enabled intermediary such as the Web Services Gateway. Establish a shared context by placing some piece of information into the message header in a handler on one node and act on that information in a handler on another node.

JAX-RPC handlers can be applied to SOAP/HTTP channels and SOAP/JMS channels.

In our sample scenario, we do not have a mediation requirement that could be satisfied using a JAX-RPC handler. Therefore, we provide a rather simple JAX-RPC handler that logs the request and response messages.

Writing a JAX-RPC handler

We only cover what is needed to write a generic handler. For more about SOAP header processing and SOAP actors, consult the SOAP 1.1 specification.

To develop a handler, you have to implement the Handler interface, which is defined as shown in Example 8-1.

Example 8-1 Interface definition of a JAX-RPC handler

```
abstract public interface javax.xml.rpc.handler.Handler extends
java.lang.Object {
   abstract public boolean handleRequest(MessageContext arg);
   abstract public boolean handleResponse(MessageContext arg);
   abstract public boolean handleFault(MessageContext arg);
   abstract public void init(HandlerInfo arg);
   abstract public void destroy();
   abstract public QName[] getHeaders();
}
```

The methods `init` and `destroy` are for lifecycle control. Similar to servlets, handler instances might be shared between invocations. The J2EE container calls init when a handler is instantiated and before any request will be dispatched to the handleRequest, handleResponse, or handleFault methods. The J2EE container calls the destroy method to inform a handler instance that it will be removed from the container's working set. Thus, anything that is needed in the handler in order to process requests can be set up in the init method and can be tidied up in the destroy method.

The HandlerInfo object that is passed to the init method provides context information from the runtime system. The HandlerInfo.getHandlerConfig() method returns name/value pairs that are configured in the handlers deployment descriptor. The method HandlerInfo.getHeaders() returns the set of SOAP headers that are defined in the deployment descriptor for this handler.

The JSR 109 specification states that a Handler.init() method must retain the information that is defined by HandlerInfo.getHeaders(). Additionally, the JSR

109 specification states that a Handler implementation must implement the getHeaders() method to return the results of the HandlerInfo.getHeaders() method.

The handleRequest method is invoked when a request message arrives. Similarly, the handleResponse message is invoked when a response message arrives. The handleFault method performs SOAP fault processing.

Instead of implementing the Handler interface directly, you can inherit the javax.xml.rpc.handler.GenericHandler class, which provides default implementations for all of the above-mentioned Handler methods except getHeaders().

Note: IBM provides another default handler implementation with the class com.ibm.wsspi.webservices.rpc.handler.GenericHandler, which implements the interface com.ibm.wsspi.webservices.rpc.handler.Handler. While it is possible to start your handler development using these classes, your handlers will not be portable to any other JAX-RPC implementation.

Putting all of this information together, our first handler looks like Example 8-2.

Example 8-2 JAX-RPC handler implementation - part one

```
package com.ibm.ral.itso;

import java.io.IOException;
import javax.xml.namespace.QName;
import javax.xml.rpc.handler.*;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.*;

public class LogHandler extends GenericHandler {

   private QName[] headers = null;

   public void init(HandlerInfo handlerInfo) {
      headers = handlerInfo.getHeaders();
   }

   public QName[] getHeaders() {
      return headers;
    }
}
```

Note that we also have to implement the init method in order to implement the getHeaders method as required by JSR 109. In the next step, we implement the handleRequest and handleResponse methods to log the message contents, as shown in Example 8-3.

Example 8-3 JAX-RPC handler implementation - part two

```
public boolean handleRequest(MessageContext msgContext) {
   System.out.println("a request message is passing the gateway...");
   logMessage(msgContext);
   return true;
}
public boolean handleResponse(MessageContext msgContext) {
   System.out.println("a response message is passing the gateway...");
   logMessage(msgContext);
   return true;
}
protected void logMessage(MessageContext msgContext) {
   SOAPMessageContext smc = (SOAPMessageContext) msgContext;
   SOAPMessage sm = smc.getMessage();
   try {
      sm.writeTo(System.out);
      System.out.println();
   } catch (SOAPException e) {
      e.printStackTrace();
   } catch (IOException e) {
      e.printStackTrace();
   }
}
```

Both methods, `handleRequest` and `handleResponse`, are passed a reference to a `MessageContext`. Note that casting this reference to a `SOAPMessageContext` without testing (as shown) is not recommended by JSR 109, because in the future there might be support for message types other than SOAP by container vendors.

A MessageContext serves two purposes:

- It can be used to pass parameters between handlers in a handler chain or just different handler methods in the same handler.
- It provides access to the SOAP message. In our simple sample, we only use the SOAPMessageContext to get the SOAP message, which we want to log to the system log entirely.

8.3.2 Developing a handler in WebSphere Studio

Developing a handler in WebSphere Studio Application Developer is easy:

1. Create a Java project. For the project name, type: `Handler`
2. Create a package in this Java project. For the package name enter: `com.ibm.ral.itso`
3. Adjust the Java Build Path for your Java Project:
   – Add WAS_50_PLUGINDIR/lib/qname.jar
   – Add WAS_50_PLUGINDIR/lib/webservices.jar
4. Create a handler class:
   – Right-click the created package and select **New** → **Class**.
   – For the class name enter: `LogHandler`
   – For the superclass enter: `javax.xml.rpc.handler.GenericHandler`
   – Click **Finish**.
5. Complete the implementation of your handler, as shown in Example 8-2 on page 191 and Example 8-3 on page 192.

8.3.3 Preparing a handler for deployment to Web Services Gateway

To deploy a JAX-RPC handler to the Web Services Gateway, first you must make it available to WebSphere Application Server. You can do one of the following:

- Export the required class files from WebSphere Studio Application Developer and copy them into the directory structure under <install_root>/classes (providing the package name in this directory structure as well). Using this method, our sample handler LogHandler.class file is copied to the <install_root>/classes/com/ibm/ral/itso directory, where <install_root> is the directory in which you have installed WebSphere Application Server.
- Export the class files that are required for your handler to a JAR file and copy the JAR file to the install_root/lib/app directory.

The deployment of the handler into the Web Services Gateway configuration is described in the Runtime guidelines section in 8.4.7, “JAX-RPC handler runtime guidelines” on page 208.

8.4 Runtime guidelines

In this section we start with a brief look at installing and configuring the Web Services Gateway. After that, we replace the point-to-point interactions in our business scenario by implementing the Enterprise Service Bus: Router variation using the Web Services Gateway. The following three-step process must be performed for every Web service interaction:

1. Deploy a Web Services Gateway service.
2. Retrieve the endpoint address of this service.
3. Configure the service requesters to point to the endpoint address of the Web Services Gateway service.

Note: In order to run this scenario, we used WebSphere Application Server Network Deployment V5.1.1 (Fix Pack 1). This release contains several essential fixes to the Web Services Gateway.

8.4.1 Installing the Web Services Gateway

The Web Services Gateway is a J2EE application that is supplied with WebSphere Application Server Network Deployment edition. We installed the Web Services Gateway on a standalone IBM WebSphere Application Server base V5.1.1 server. For the installation of the Web Services Gateway, we followed the installation guidelines in the WebSphere InfoCenter, which is located at:

`http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp`

We only installed the SOAP/HTTP channel for this sample scenario, as we only have HTTP traffic incoming to the Web Services Gateway.

Note: After installing, be sure to update your Web server plug-in by selecting **Environment** → **Update Web Server Plugin** in the WebSphere Administrative Console.

8.4.2 Configuring the Web Services Gateway

When configuring the Web Services Gateway before first use, you have to configure a namespace URI. This is an important step because you have to redeploy all of your Web Services Gateway services if you make changes to the namespace URI.

The Web Services Gateway namespace URI is used as the targetNamespace in the WSDL implementation definition files that are exported from the Web Services Gateway. Thus, there will be a different set of WSDL files for the services that are exposed by the Web Services Gateway than for those that point directly to the service provider. The namespace URI differentiates these files.

You should therefore specify a namespace URI that reflects your organization and the services that will be exposed by this ESB.

To configure the Web Services Gateway and other settings:

1. Open the IBM Web Services Gateway Admin Tool. For the default installation on server1, the URL for accessing the console will be:

   ```
   http://<hostname>/wsgw/admin
   ```

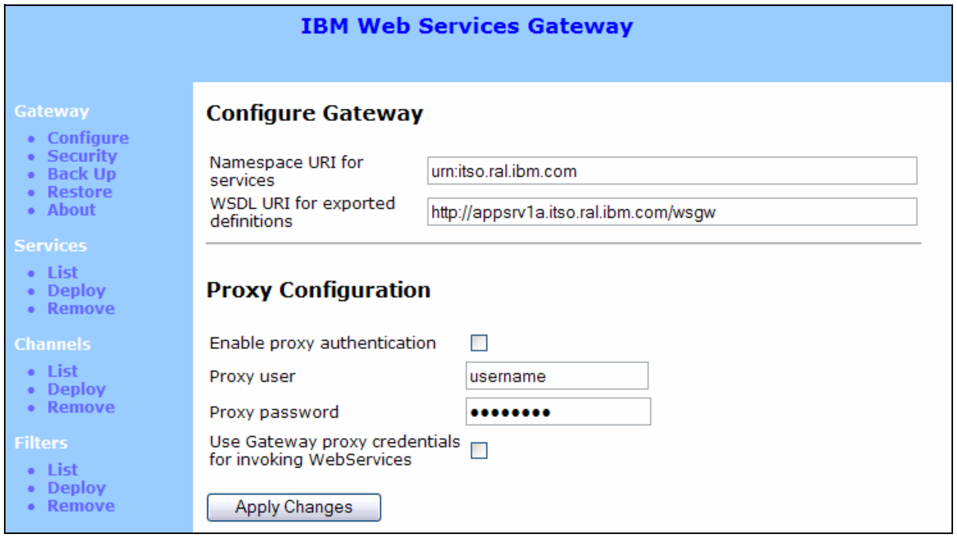
2. Click **Gateway** → **Configure** in the navigation panel.
3. In the Configure Gateway window, set the following properties (Figure 8-8):

   Namespace URI `urn:itso.ral.ibm.com`

   Note: The namespace URI can be set to a URN or URL. We found that URLs with path names (such as http://itso.ral.ibm.com/ESB) were not supported as valid namespace URIs at runtime.

   WSDL URI `http://appsrv1a.itso.ral.ibm.com/wsgw`

   This is the URI that Web service clients will use to access the WSDL file and the exposed Web service.



Configure Gateway

Namespace URI for services	urn:itso.ral.ibm.com
WSDL URI for exported definitions	http://appsrv1a.itso.ral.ibm.com/wsgw

Proxy Configuration

Enable proxy authentication	☐
Proxy user	username
Proxy password	••••••••
Use Gateway proxy credentials for invoking WebServices	☐

Figure 8-8 Web Services Gateway configuration

4. In this scenario we did not configure Proxy settings because we are performing intra-enterprise operations. These settings may be necessary when using the Web Services Gateway in an inter-enterprise environment.

5. Click **Apply Changes** to configure the Web Services Gateway.

8.4.3 Deploying Web Service Gateway channels

Currently, these channels types are available in the Web Services Gateway:

- Apache SOAP channel
- SOAP/HTTP channel
- SOAP/JMS channel

The Apache SOAP channel and the SOAP/HTTP channel both support SOAP applications that are SOAP 1.1 compatible (such as Apache SOAP 2.3 and Axis SOAP 1.0). So if you have an application that uses a production-supported Axis 1.0 SOAP stack, generating SOAP 1.1 and using HTTP as the transport protocol, then it can use either channel. However, the use of the newer SOAP/HTTP channel is recommended in this case.

If you are using the Apache SOAP Channel, then the SOAP message format must be RPC style. To handle document-style SOAP messages, you must use the SOAP/HTTP channel (which supports both RPC and document-style SOAP messages).

If you deploy Web services that pass attachments in a MIME message, then these Web services can only be accessed using the SOAP/HTTP channel.

When this redbook was written, SOAP/JMS channels provided no support for SOAP with attachments.

For each channel type, two versions are supplied with the Web Services Gateway. The WebSphere InfoCenter refers to these two versions as inbound and outbound channels. For example, you can use these two versions to grant different access rights to different groups of service requesters using Web Services Gateway-level authentication as described in “Design alternative: basic authentication, role-based authorization” on page 188.

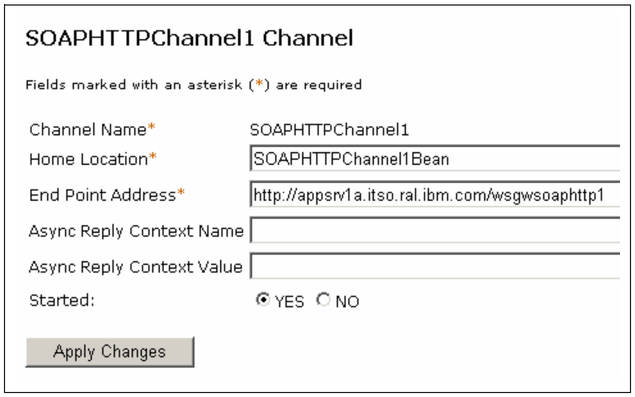
Our scenario uses RPC-style messages as well as document style messages. HTTP is the only protocol that is used by service requesters to request services from service providers, so we only need to deploy the SOAP/HTTP channel.

To deploy the SOAP/HTTP Channel, perform the following steps:

1. Click **Channels** → **Deploy** in the navigation panel on the left.
2. In the Deploy Channel window (Figure 8-9 on page 197), the following fields are required:

Channel Name	`SOAPHTTPChannel1`
Home Location	`SOAPHTTPChannel1Bean`

End Point Address	`http://<hostname>:port/wsgwsoaphttp1`
Async Reply Context Name	leave blank (not supported)
Async Reply Context Value	leave blank (not supported)


SOAPHTTPChannel1 Channel

Fields marked with an asterisk (*) are required

Channel Name*	SOAPHTTPChannel1
Home Location*	SOAPHTTPChannel1Bean
End Point Address*	http://appsrv1a.itso.ral.ibm.com/wsgwsoaphttp1
Async Reply Context Name	
Async Reply Context Value	
Started:	[✓ YES] NO

Figure 8-9 SOAP/HTTP channel settings

8.4.4 Deploying Web Service Gateway services

After installing and configuring the Web Services Gateway, the next step is to deploy Web services. This section describes how to deploy services, starting with a simple deployment, and then describes some design alternatives.

Deploying a basic Gateway service

Services that are deployed to the Web Services Gateway are called Gateway services. This section describes how to deploy a basic Gateway service.

Note: To deploy a Gateway service, the Web Services Gateway must be able to access the WSDL file that defines your Web service, either using a URL or business key of a UDDI registry. Therefore, you should place your WSDL files either on an HTTP server or publish them to a UDDI registry. The following step-by-step instructions assume that the WSDL files can be found by using the URL http://appsrv1a.itso.ral.ibm.com/wsdl/*xy.wsdl* where *xy.wsdl* is the name of the WSDL file.

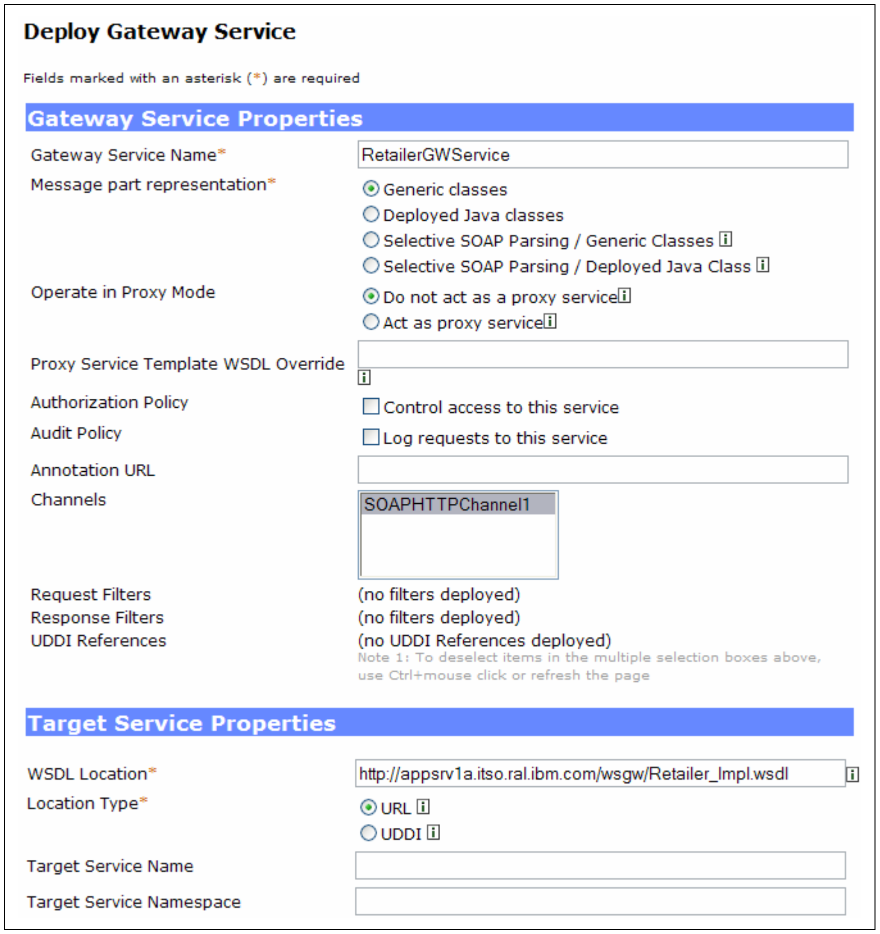
We deploy the Retailer Web service. To deploy a service:

1. Click **Services** → **Deploy** in the left frame of the Web Services Gateway Admin Tool.

2. In most cases, only three fields have to be populated (see Figure 8-10 on page 199):

Gateway Service Name	Sets a name for the Gateway service. We set this to `RetailerGWService`.
Channels	Specifies the channel on which the Web service call will be received. We chose the **SOAPHTTPChannel1** channel.
WSDL Location	If you are retrieving the Web services WSDL files from an HTTP server, set this to point to the address of the WSDL file that defines the location of the Web service. We set this to `http://appsrv1a.itso.ral.ibm.com/wsgw/Retailer_Impl.wsdl`.

Note: By leaving the Target Service Name and Target Service Namespace sections blank, these attributes will be populated by the Web Services Gateway. This assumes that your WSDL file defines only one WSDL service.



Deploy Gateway Service

Fields marked with an asterisk (*) are required

Gateway Service Properties

Gateway Service Name*	RetailerGWService
Message part representation*	[✓ Generic classes] Deployed Java classes / Selective SOAP Parsing / Generic Classes / Selective SOAP Parsing / Deployed Java Class
Operate in Proxy Mode	[✓ Do not act as a proxy service] Act as proxy service
Proxy Service Template WSDL Override	
Authorization Policy	☐ Control access to this service
Audit Policy	☐ Log requests to this service
Annotation URL	
Channels	SOAPHTTPChannel1
Request Filters	(no filters deployed)
Response Filters	(no filters deployed)
UDDI References	(no UDDI References deployed)

Note 1: To deselect items in the multiple selection boxes above, use Ctrl+mouse click or refresh the page

Target Service Properties

WSDL Location*	http://appsrv1a.itso.ral.ibm.com/wsgw/Retailer_Impl.wsdl
Location Type*	[✓ URL] UDDI
Target Service Name	
Target Service Namespace	

Figure 8-10 Deploy Gateway Service

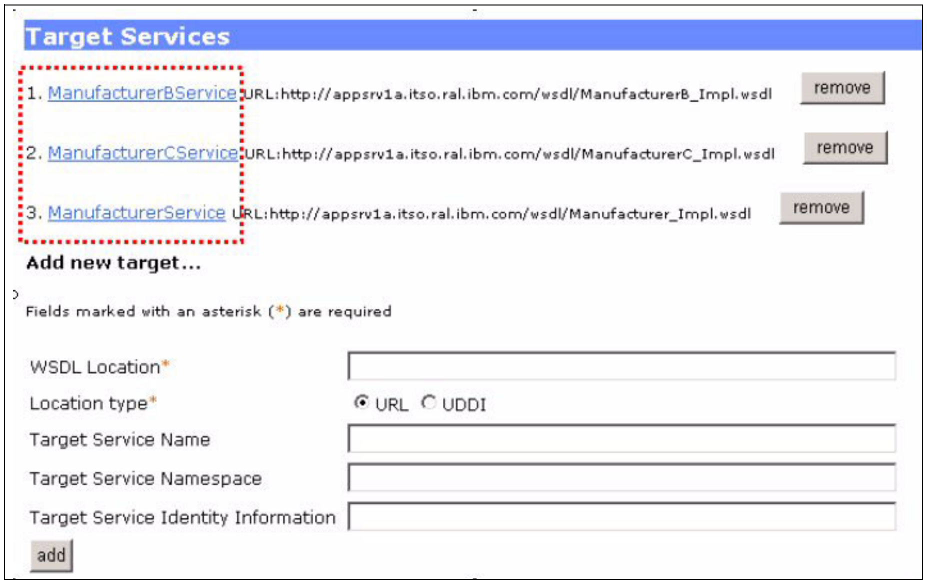
We created three additional Gateway services, as shown in Table 8-1. Each service retrieved the WSDL file from http://appsrv1a.itso.ral.ibm.com/wsdl.

Table 8-1 Other Web Services Gateway services

Gateway service name	Channels	WSDL location
ManufacturerGWService	SOAPHTTPChannel1	Manufacturer_Impl.wsdl
ManufacturerBGWService	SOAPHTTPChannel1	ManufacturerB_Impl.wsdl
ManufacturerCGWService	SOAPHTTPChannel1	ManufacturerC_Impl.wsdl
WarehouseCallBackGWService	SOAPHTTPChannel1	WarehouseCallBack_Impl.wsdl

Design alternative: multiple target services

For our sample scenario, we created a Gateway service for each Manufacturer. All Manufacturer Web services share the same WSDL port type definition so it would be possible to define only one Gateway service for this WSDL port type and configure multiple target services for this Gateway service (see Figure 8-11). In this case you would need a JAX-RPC handler to select the correct target service for the incoming request.



Target Services

1. ManufacturerBService URL:http://appsrv1a.itso.ral.ibm.com/wsdl/ManufacturerB_Impl.wsdl remove
2. ManufacturerCService URL:http://appsrv1a.itso.ral.ibm.com/wsdl/ManufacturerC_Impl.wsdl remove
3. ManufacturerService URL:http://appsrv1a.itso.ral.ibm.com/wsdl/Manufacturer_Impl.wsdl remove

Add new target...

Fields marked with an asterisk (*) are required

WSDL Location*	
Location type*	[✓ URL] UDDI
Target Service Name	
Target Service Namespace	
Target Service Identity Information	

add

Figure 8-11 Multiple target services

Design alternative: protocol transformation

The Web Services Gateway can perform protocol transformation of Web service requests. For example, a request that is received from a service requester over the SOAP/HTTP channel can be forwarded to a service provider as a SOAP/JMS request.

No configuration is necessary in the Web Services Gateway to perform this. The only requirement is that the WSDL service implementation file contains the relevant binding and endpoint for the protocol you wish to use.

In our scenario, the Retailer service places a call to the Warehouse Web service with a SOAP/HTTP request. The Web Services Gateway transforms this request to SOAP/JMS, because the WSDL implementation file specifies that this protocol is to be used (Example 8-4 on page 201).

Example 8-4 Warehouse_Impl.wsdl

```
<wsdl:service name="WarehouseService">
   <wsdl:port name="Warehouse" binding="intf:WarehouseSoapJMSBinding">
      <wsdlsoap:address location="jms:/queue?destination=jms/WarehouseQ&amp;
                        connectionFactory=jms/WarehouseQCF&amp;
                        targetService=Warehouse"/>
   </wsdl:port>
</wsdl:service>
```

Note: If the Web Services Gateway and service provider are located on different machines, you have to configure a JMS provider on the Web Services Gateway machine to point to the queue manager of the service provider.

We defined the Warehouse service in the Web Services Gateway as shown in Table 8-2. This service retrieved the WSDL file from http://appsrv1a.itso.ral.ibm.com/wsdl.

Table 8-2 Warehouse Gateway service

Gateway service name	Channels	WSDL location
WarehouseGWJmsService	SOAPHTTPChannel1	Warehouse_Impl.wsdl

Design alternative: multiple services in a single WSDL file

Some Web services may define multiple services in a single WSDL service implementation file. For example the LoggingFacility Web service defines two services, one for a SOAP/HTTP service and another for a SOAP/JMS service as shown in Example 8-5.

Example 8-5 Multiple services in a single WSDL file

```
<wsdl:service name="LoggingFacilityService">
   <wsdl:port name="LoggingFacility" binding="intf:LoggingFacilitySoapBinding">
      <wsdlsoap:address
            location="http://appsrv1a:9080/LoggingFacility/services/LoggingFacility"/>
   </wsdl:port>
</wsdl:service>

<wsdl:service name="LoggingFacilityJMSService">
   <wsdl:port name="LoggingFacility" binding="intf:LoggingFacilitySoapJMSBinding">
      <wsdlsoap:address location="jms:/queue?destination=jms/LoggingFacilityQ&amp;
            connectionFactory=jms/LoggingFacilityQCF&amp;targetService=LoggingFacility"/>
   </wsdl:port>
</wsdl:service>
```

When deploying such services to the Web Services Gateway, it is not sufficient simply to supply the WSDL location, as the Web Services Gateway is unable to determine which service is intended to be deployed. Therefore, you must also specify the target service name and the target service namespace.

For example, to define the LoggingFacility service, which uses SOAP/HTTP, we specified:

- Gateway Service Name set to **LoggingFacilityGWHttpService**
- Channels set to **SOAPHTTPChannel1**
- WSDL location set to **http://appsrv1a.itso.ral.ibm.com/wsdl/LoggingFacility_Impl.wsdl**
- Target Service Name set to **LoggingFacilityService**
- Target Service Namespace set to **http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/LoggingFacility.wsdl**

Similarly, we defined a second LoggingFacility service (which uses SOAP/JMS) called **LoggingFacilityGWJmsService** with the same settings, except Target Service Name, which was set to **LoggingFacilityJMSService**.

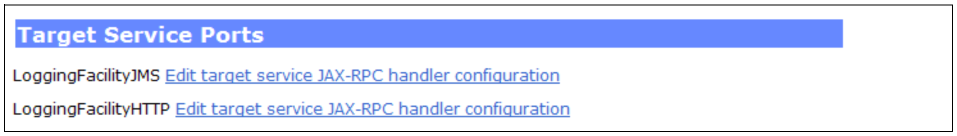
Design alternative: multiple target ports

Instead of defining two service definitions in a single WSDL file, you can define multiple WSDL ports with a single WSDL service (Example 8-6).

Example 8-6 Multiple ports in a single WSDL service

```
<wsdl:service name="LoggingFacilityService">
   <wsdl:port name="LoggingFacilityHTTP" binding="intf:LoggingFacilitySoapBinding">
      <wsdlsoap:address
         location="http://appsrv1a:9080/LoggingFacility/services/LoggingFacility"/>
   </wsdl:port>
   <wsdl:port name="LoggingFacilityJMS" binding="intf:LoggingFacilitySoapJMSBinding">
      <wsdlsoap:address
         location="jms:/queue?destination=jms/LoggingFacilityQ&amp;
         connectionFactory=jms/LoggingFacilityQCF&amp;targetService=LoggingFacility"/>
   </wsdl:port>
</wsdl:service>
```

In this instance, you would deploy a single Gateway service for both ports. The Web Services Gateway detects that multiple ports are defined in the service and creates target service ports for each of them (Figure 8-12 on page 203).


Target Service Ports

LoggingFacilityJMS Edit target service JAX-RPC handler configuration

LoggingFacilityHTTP Edit target service JAX-RPC handler configuration

Figure 8-12 Multiple target service ports

Define a JAX-RPC handler to select the appropriate target port for the target service. Failure to do so will mean the that Web Services Gateway will always select the first target service port in the list.

8.4.5 Extracting the endpoint address

The Gateway creates endpoint addresses, which the service requester must use to access the Gateway services. Allowing for the name of a Gateway service and the channel you have configured for a Gateway service, the endpoint address will be similar to:

```
http://appsrv1a.itso.ral.ibm.com/wsgwsoaphttp1/soaphttpengine/urn%3Aitso.ra
l.ibm.com%23RetailerGWService
```

To extract the endpoint address:

1. Open the Web Services Gateway Admin Tool and click **Services** → **List** in the left frame.
2. Click **RetailerGWService** in the list of deployed services.
3. In the Service: RetailerGWService window, scroll down to the Exported WSDL definitions section and click **View external WSDL implementation definition**.
4. Copy the value from the location attribute of the soap:address element to a temporary text file or to your service requester's WSDL implementation file. (See "Configuring the service requesters to point to the Gateway" on page 205.)

Repeat these steps for the remaining Gateway services.

Design alternative: export the Gateway-generated WSDL files

Instead of copying the endpoint address, you can export the WSDL implementation definition file that the Web Services Gateway generates and regenerate the Web service client. In the generated WSDL implementation definition file, the namespace URI that you have configured in 8.4.2, "Configuring the Web Services Gateway" on page 194 is specified as the targetNamespace.

The WSDL interface definition file will be imported from a location within the Web Services Gateway (see Example 8-7). Thus you have two sets of WSDL files:

- WSDL files that define how to reach a service provider and, in our case, are located on an HTTP server
- WSDL files that define how to reach the Web Services Gateway target service, and are located within the Web Services Gateway

Example 8-7 WSDL implementation definition file generated by the Gateway

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:itso.ral.ibm.com"
xmlns:interface="http://www.ws-i.org/SampleApplications/SupplyChainManagement/2
002-08/Retailer.wsdl"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:import
      namespace="http://www.ws-i.org/SampleApplications/SupplyChainManagement
         /2002-08/Retailer.wsdl"
      location="http://appsrv1a.itso.ral.ibm.com
         /wsgw/ServiceInterface?name=RetailerGWService"/>
  <wsdl:service name="RetailerGWService">
    <wsdl:port name="RetailerPortTypeSOAPHTTPBindingPort"
            binding="interface:RetailerPortTypeSOAPHTTPBinding">
      <soap:address
            location="http://appsrv1a.itso.ral.ibm.com/wsgwsoaphttp1
            /soaphttpengine/urn%3Aitso.ral.ibm.com%23RetailerGWService"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

If you regenerate the Web service client with the Web Services Gateway supplied WSDL, some code adjustments are necessary in your service requester's implementation code (see 8.4.6, "Configuring the service requesters to point to the Gateway" on page 205).

To export the WSDL implementation definition file that the Web Services Gateway has generated:

1. Select the a Gateway service from the list of deployed services.
2. In the Exported WSDL Definitions section, right-click **External WSDL implementation definition (WSDL only)** and select **Save Target As** and save the file using an appropriate filename.

Note: You do not have to export the External WSDL interface definition from the Web Services Gateway because the External WSDL implementation definition imports it using a URL that points to a location within the Web Services Gateway.

Note: Instead of saving the file, you can copy the link to this file (select **Copy Shortcut**) and use this URL to specify the WSDL file location in the Web service client wizard in WebSphere Studio Application Developer.

Design alternative: Use WS-Inspection to extract the endpoint

To help your service users locate the WSDL documents for services that are deployed to the Web Services Gateway, the Web Services Gateway also supports the WS-Inspection specification. To open a WS-Inspection document that contains references to the WSDL documents for all of the Web Services Gateway-deployed services, issue an HTTP GET against:

```
http://host:port/wsgw/wsinspection.wsil
```

Here, *host* and *port* are the host name and port number that your HTTP server is listening on. You can specify this URL in the Web service client wizard in WebSphere Studio Application Developer.

8.4.6 Configuring the service requesters to point to the Gateway

To regenerate your Web service clients in WebSphere Studio Application Developer you could either:

- Change the endpoint URL in your existing WSDL implementation definition file on the client side to the value that you have captured in the step “Extracting the endpoint address” on page 203.
- Use the Web Service client wizard to regenerate the Web service client by specifying the location of the WSDL implementation definition file that has been generated by the Web Services Gateway.

Changing the Web service endpoint URL

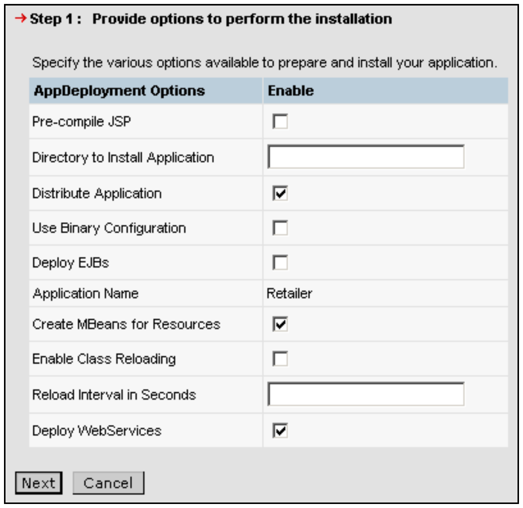
To change the endpoint URL for the Retailer Web service in the SCMSampleUI application to point to the service that is exposed by the Web Services Gateway (RetailerGWService in our scenario), open the Retailer_Impl.wsdl file and change soap:address:

```
<wsdlsoap:address
location="http://appsrv1a.itso.ral.ibm.com:9080/Retailer/services/Retailer"/>
```

To the value that you have captured in 8.4.5, “Extracting the endpoint address” on page 203:

```
<wsdlsoap:address
location="http://appsrv1g.itso.ral.ibm.com/wsgwsoaphttp1/soaphttpengine/urn
%3Aitso.ral.ibm.com%23RetailerGWService"/>
```

To pick up the changed endpoint URL in your client code, you can either regenerate the Web services client or use the Deploy WebServices option in WebSphere Application Server (Figure 8-13). WebSphere Application Server will regenerate the deployment code based on the Web services client deployment descriptors, updating it with the current endpoint address from the WSDL file. Similar to EJB deployment, this only has to be performed when the deployment details have changed.


→ **Step 1 : Provide options to perform the installation**

Specify the various options available to prepare and install your application.

AppDeployment Options	Enable
Pre-compile JSP	☐
Directory to Install Application	
Distribute Application	☑
Use Binary Configuration	☐
Deploy EJBs	☐
Application Name	Retailer
Create MBeans for Resources	☑
Enable Class Reloading	☐
Reload Interval in Seconds	
Deploy WebServices	☑

Figure 8-13 Deploying Web services using the WebSphere Administrative Console

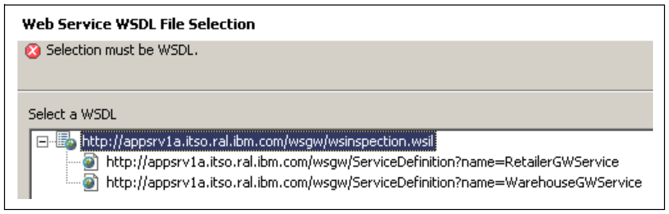
To regenerate the Web services client in WebSphere Studio Application Developer using the changed endpoint URL, right-click **Retailer_Impl.wsdl** and select **Web Services** → **Generate Client**. Follow the instructions in the wizard.

Note: No code changes are required in the service requester implementation, because only the endpoint URL has changed.

Regenerating the client using the generated WSDL file

To regenerate the Web service client based on the WSDL implementation definition file that is generated by Web Services Gateway:

1. Start the Web service client wizard: Select **File** → **New** → **Other** → **Web Services** → **Web Service Client**.
2. In the Web services client wizard, select **Java Proxy** on the Web Services page and click **Next**.
3. On the Client Environment Configuration page, select the Client Web Project (**SCMSampleUIWeb** in our case) and click **Next**.
4. On the Web Service Selection Page, specify either:
   - The location to the exported external WSDL implementation definition file.
   - The URL to the external WSDL implementation definition file.
   - The URL to the WSIL document of the Web Services Gateway. In this case you have to select the URL that points to the WSDL file that you want to use, in the next window (see Figure 8-14).



Selection must be WSDL.

http://appsrv1a.itso.ral.ibm.com/wsgw/wsinspection.wsil
http://appsrv1a.itso.ral.ibm.com/wsgw/ServiceDefinition?name=RetailerGWService
http://appsrv1a.itso.ral.ibm.com/wsgw/ServiceDefinition?name=WarehouseGWService

Figure 8-14 Selecting the WSDL file location from a WSIL document

5. Select **Define custom mapping for namespace to package** and click **Next**. On the next screen, click **Import** and select **Build** → **wsdl** → **Retailer-NStoPkg.properties**. Click **OK**.
6. Click **Finish** to create the Web service client.

After regenerating the Web service client, the following changes are made:

- A new Java package (in our case com.ibm.ral.itso) is created. This matches the namespace URI that we configured in 8.4.2, “Configuring the Web Services Gateway” on page 194.
- This package contains the JAX-RPC service interface (RetailerGWService) and the JAX-RPC service object (RetailerGWServiceLocator).
- A new service (service/RetailerGWService) is added to webservicesclient.xml.

To use this service, change your service requester to look up the reference to this service. For SCMSampleUIWeb, this code is located in the class shopper.Shopper. Change the code that retrieves a reference to the service object from:

```
RetailerService retailer = (RetailerService)
ctx.lookup("java:comp/env/service/RetailerService");
```

to:

```
RetailerGWService retailer = (RetailerGWService)
ctx.lookup("java:comp/env/service/RetailerGWService");
```

8.4.7 JAX-RPC handler runtime guidelines

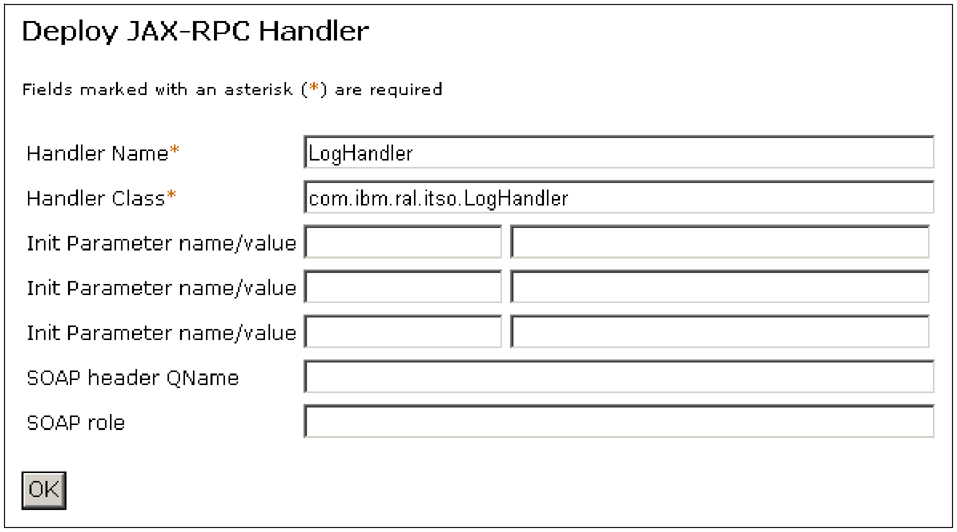
Two tasks are necessary to use JAX-RPC handlers developed in WebSphere Studio Application Developer in the Web Services Gateway:

- Deploying the handler to the Web Services Gateway
- Configuring inbound and outbound handler chains

Deploying JAX-RPC handlers to the Web Services Gateway

To deploy a JAX-RPC handler to the Web Services Gateway configuration:

1. Open the IBM Web Services Gateway Admin Tool.
2. Select **Handlers** → **Deploy.** (See Figure 8-15.)



Deploy JAX-RPC Handler

Fields marked with an asterisk (*) are required

Handler Name*	LogHandler	
Handler Class*	com.ibm.ral.itso.LogHandler	
Init Parameter name/value		
Init Parameter name/value		
Init Parameter name/value		
SOAP header QName		
SOAP role		

Figure 8-15 Deploy JAX-RPC Handler configuration

3. For Handler Name, enter a unique name by which the handler will be known in the Web Services Gateway. For our sample handler, enter `LogHandler`.
4. For Handler Class, enter the fully qualified name of the handler class. For our sample handler enter `com.ibm.ral.itso.LogHandler`.
5. Define optional properties:
   - Init parameter name / value

     You can define any name / value pair for configuration purposes. The handler can access these parameters using HandlerInfo.getHandlerConfig()
   - SOAP header QName
     - Here you can specify any SOAP header's qualified namespace to indicate to the runtime that this handler is either to consume or to produce a specific SOAP header.
     - Syntax: *{namespace_URI}local_name*
   - SOAP role
     - The SOAP role that a handler is supposed to fulfill.

Note: When you deploy a handler you can only configure three init parameters, one SOAP header QName, and one SOAP role. After deploying the handler you could add more of these configuration parameters.

Configuring inbound and outbound handler chains

You can configure one or more channels for a Gateway service to listen for incoming requests. For each of these Gateway service/channel combinations, you can configure a handler. This handler is called an inbound handler.

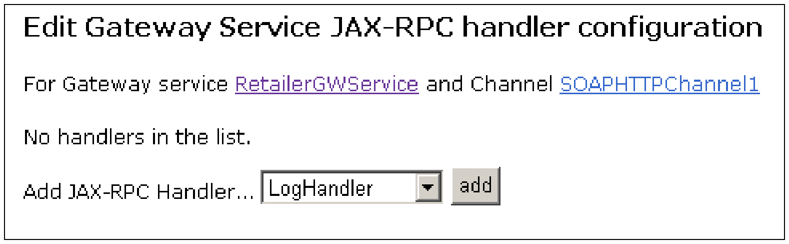
Additionally, you can configure multiple target services for a Gateway service, and each target service can have multiple target ports. For each of these Gateway service / target service / target port combinations, you can configure a handler. This handler is called an outbound handler.

For each the combinations, you can configure more than one handler. If you configure more than one handler, you also have to provide an execution order. This is referred to as a handler chain.

To configure an inbound handler for a the RetailerGWService:

1. Open the IBM Web Services Gateway Admin Tool.
2. Select **Services**-> **List**.
3. Click **RetailerGWService**.

4. In the Channels section, click **Edit JAX-RPC handler configuration**.
5. On the Edit Gateway Service JAX-RPC handler configuration page, you should see the previously deployed LogHandler in the dropdown list (Figure 8-16). Select **LogHandler** and click **Add**.


Edit Gateway Service JAX-RPC handler configuration

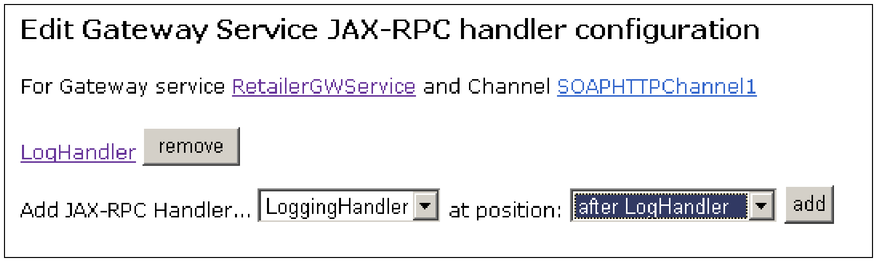
For Gateway service RetailerGWService and Channel SOAPHTTPChannel1

No handlers in the list.

Add JAX-RPC Handler... LogHandler add

Figure 8-16 Gateway service JAX-RPC handler configuration

6. To create a handler chain, you have to specify an order. Select another handler and specify whether it should be executed before or after LogHandler (Figure 8-17). We did not have a second handler to add.


Edit Gateway Service JAX-RPC handler configuration

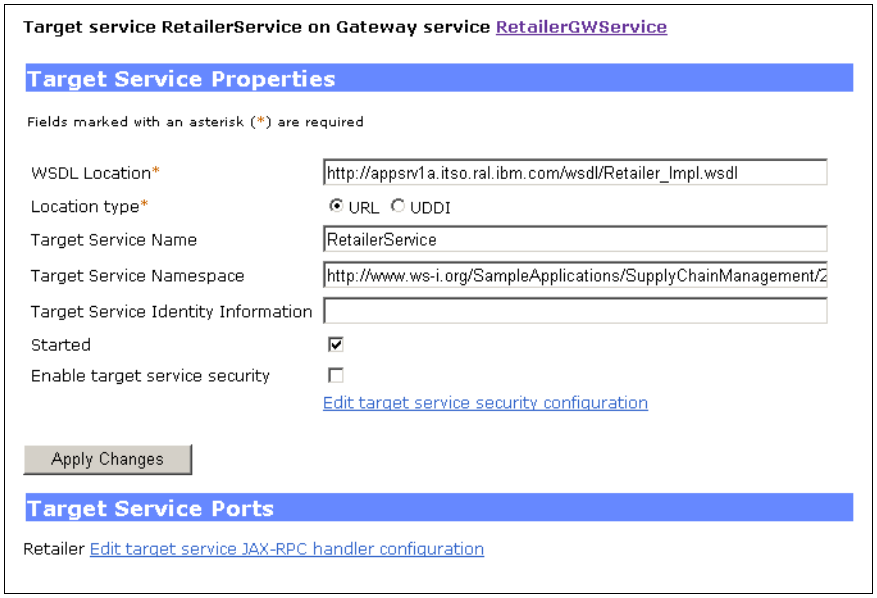
For Gateway service RetailerGWService and Channel SOAPHTTPChannel1

LogHandler remove

Add JAX-RPC Handler... LoggingHandler at position: after LogHandler add

Figure 8-17 Defining a handler chain

Outbound handlers can be configured in the Target Service Port section of a Gateway service by clicking **Edit target service JAX-RPC handler configuration** (Figure 8-18 on page 211).


Target service RetailerService on Gateway service RetailerGWService

Target Service Properties

Fields marked with an asterisk (*) are required

WSDL Location*	http://appsrv1a.itso.ral.ibm.com/wsdl/Retailer_Impl.wsdl
Location type*	[✓ URL] UDDI
Target Service Name	RetailerService
Target Service Namespace	http://www.ws-i.org/SampleApplications/SupplyChainManagement/2
Target Service Identity Information	
Started	✓
Enable target service security	☐
	Edit target service security configuration

Target Service Ports

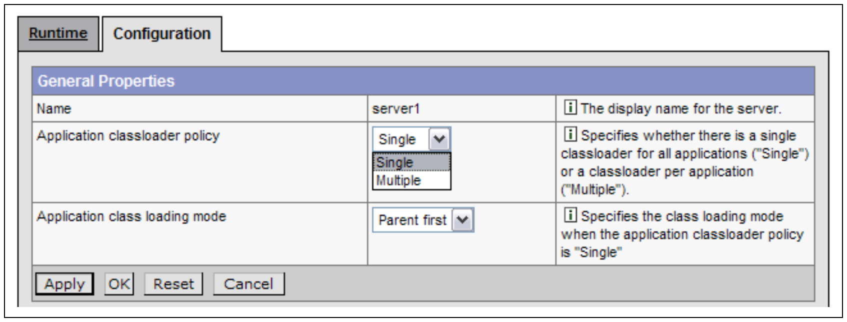
Retailer Edit target service JAX-RPC handler configuration

Figure 8-18 JAX-RPC handlers can be configured for target service ports

8.4.8 Runtime guidelines for selective SOAP parsing

The selective SOAP parsing feature can be switched on for a Gateway service, ensuring that the SOAP body will not be parsed by the Web Services Gateway. To use this feature, two configuration steps are required:

1. For the WebSphere Application Server instance where the Web Services Gateway is running, configure it to run in single class loader mode. Set this in the WebSphere Administrative Console by clicking **Servers** → **Application Servers** → **server1** and setting Application classloader policy to **Single** (Figure 8-19 on page 212). Restart WebSphere Application Server for this change to take effect.

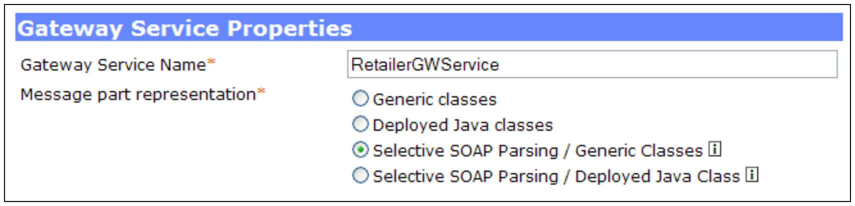


General Properties

Name	server1	The display name for the server.
Application classloader policy	Single (Single, Multiple)	Specifies whether there is a single classloader for all applications ("Single") or a classloader per application ("Multiple").
Application class loading mode	Parent first	Specifies the class loading mode when the application classloader policy is "Single"

Figure 8-19 Specifying a single classloader

2. In the Web Services Gateway Admin Tool, define a new Gateway service, and set the Message part representation to **Selective SOAP Parsing / Generic Classes** (Figure 8-20).



Gateway Service Properties

Gateway Service Name* RetailerGWService

Message part representation*
- Generic classes
- Deployed Java classes
- [✓ Selective SOAP Parsing / Generic Classes]
- Selective SOAP Parsing / Deployed Java Class

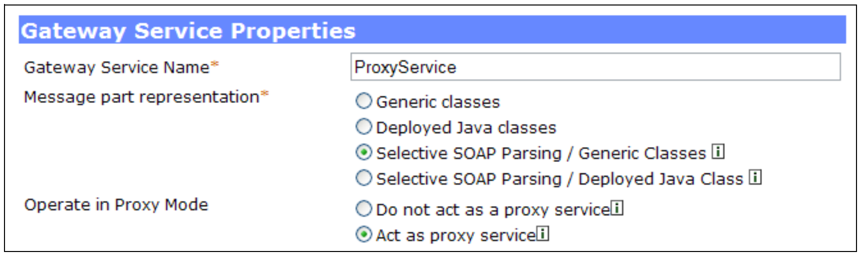
Figure 8-20 Turning on selective SOAP parsing

Note: It is not possible to turn on selective SOAP parsing for an existing Gateway service. You must create a new one.

8.4.9 Runtime guidelines for proxy operation mode

To configure the Web Services Gateway to use proxy mode, complete these steps:

1. Follow all of the steps in the section 8.4.8, “Runtime guidelines for selective SOAP parsing” on page 211.
2. Additionally, in the Gateway service definition, select the **Act as proxy service** option (Figure 8-21 on page 213).
3. Do not specify a target service in the Gateway service definition.
4. Create a JAX-RPC handler to route requests to the appropriate service provider.


Gateway Service Properties

Gateway Service Name*	ProxyService
Message part representation*	○ Generic classes ○ Deployed Java classes [✓ Selective SOAP Parsing / Generic Classes] ○ Selective SOAP Parsing / Deployed Java Class
Operate in Proxy Mode	○ Do not act as a proxy service [✓ Act as proxy service]

Figure 8-21 Act as proxy service

8.4.10 Other runtime issues

This section discusses some other runtime issues relating to the Web Services Gateway, such as how to handle exceptions and monitor SOAP messages.

Handling exceptions for the Web Services Gateway

During normal processing of a Web service invocation, a fault message might be generated by the target service and is passed back to the channel to be sent to the originator. As far as the Web Services Gateway is concerned, there is no difference between processing a normal output message and processing a fault message.

But when an exception occurs during processing of a request, the channel needs some way to decide what to do with the exception. What is needed is a service that provides a pluggable handler that can look at the message, exception, and other information to decide whether the exception should be thrown back to the originator, or whether a fault message should be constructed.

This service is not provided with the Web Services Gateway, but the Web Services Gateway does contain an interface to encapsulate such a service. The ExceptionHandler interface allows channels to call an exception handling service, and allows the exceptions to be reported to a third party for analysis.

The Home object for this service must implement the com.ibm.wsgw.beans.ExceptionHandlerHome interface and be located in JNDI at websphere/WSGW/ExceptionHandlerService.

Capturing Web service invocation information

The Web Services Gateway has not implemented a service that stores operational messages, but it does contain an interface (the MessageWarehouse interface) to encapsulate such a service. This interface is driven by channels on receipt of requests and before sending responses.

If you have your own system for handling (classifying, storing, and retrieving) operational messages, potentially you can use it to log the Web Services Gateway’s operational messages through the MessageWarehouse interface.

The Home object for this service must implement the com.ibm.wsgw.beans.MessageWarehouseHome interface and be located in JNDI at websphere/WSGW/MessageWarehouse.

Monitoring SOAP messages

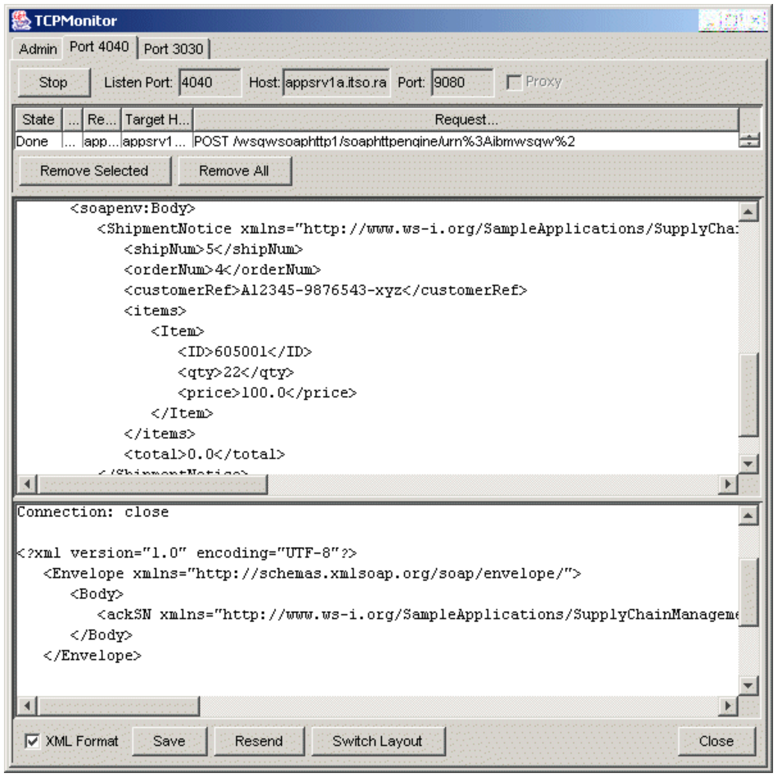
You can trace the XML messages exchanged between a Web service client and the server. In this section we look at two tools:

- The TCPMon tool provided with WebSphere Application Server V5.1.1
- The TCP/IP Monitor Server provided with WebSphere Studio Application Developer

WebSphere TCPMon tool

The TCPMon tool enables tracing of SOAP messages by redirecting messages from one port to another, displaying the contents as they go. WebSphere Application Server normally listens on port 9080. To trace messages that are sent to the application server, TCPMon can be configured, for example, to listen on port 9088 and redirect messages to 9080. The client is modified to use port 9088 to access the Web service.

This tool is provided with WebSphere Application Server V5.1.1. It enables you to view the contents of the SOAP messages that are exchanged between the source and target applications, as shown in Figure 8-22 on page 215.



```
State | ... | Re... | Target H... | Request...
Done  | ... | app...| appsrv1...  | POST /wsgwsoaphttp1/soaphttpengine/urn%3Aibmwsgw%2
```

```
      <soapenv:Body>
         <ShipmentNotice xmlns="http://www.ws-i.org/SampleApplications/SupplyCha
            <shipNum>5</shipNum>
            <orderNum>4</orderNum>
            <customerRef>A12345-9876543-xyz</customerRef>
            <items>
               <Item>
                  <ID>605001</ID>
                  <qty>22</qty>
                  <price>100.0</price>
               </Item>
            </items>
            <total>0.0</total>
```

```
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
         <ackSN xmlns="http://www.ws-i.org/SampleApplications/SupplyChainManageme
      </Body>
   </Envelope>
```

Figure 8-22 Tracing SOAP messages using TCPMon

You can start TCPMon from a command window as follows:

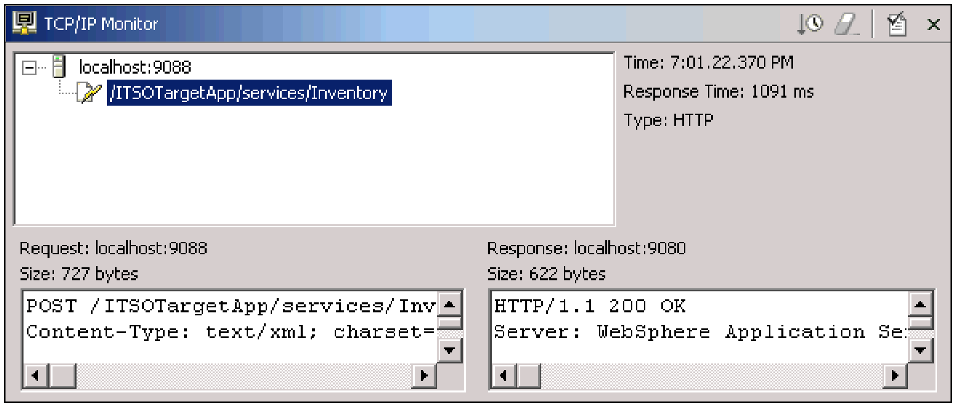
```
set CLASSPATH=%CLASSPATH%;<WAS_HOME>\lib\webservices.jar
<WAS_HOME>\java\bin\java com.ibm.ws.webservices.engine.utils.tcpmon
```

For more about TCPMon, see the InfoCenter article "Tracing Web services messages" at:

http://www.ibm.com/software/webservers/appserv/infocenter.html

TCP/IP Monitor Server

The TCP/IP Monitor Server that is provided with WebSphere Studio Application Developer (shown in Figure 8-23 on page 216) also allows tracing of SOAP messages. It works in a similar way to the WebSphere TCPMon tool. To use the TCP/IP Monitor Server, create a new Server and Configuration and select **Other** → **TCP/IP Monitor Server** for the server type.



Time: 7:01.22.370 PM
Response Time: 1091 ms
Type: HTTP

Request: localhost:9088
Size: 727 bytes

```
POST /ITSOTargetApp/services/Inv
Content-Type: text/xml; charset=
```

Response: localhost:9080
Size: 622 bytes

```
HTTP/1.1 200 OK
Server: WebSphere Application Se
```

Figure 8-23 Tracing SOAP messages using WebSphere Studio TCP/IP Monitor Server

Application server clustering

The Web Services Gateway is a J2EE enterprise application that runs on WebSphere Application Server. This means that in a Network Deployment environment, clustering of application servers can be used to improve performance and availability.

For information about WebSphere Application server clustering, see *IBM WebSphere V5.0 Performance, Scalability, and High Availability,* SG24-6198.

8.5 Further information

- Olaf Zimmermann, Mark Tomlinson, Stefan Peuser, *Perspectives on Web Services,* Springer, 2003, ISBN 3-540-00914-0
- Simple Object Access Protocol (SOAP) 1.1

  `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`
- Java API for XML-based RPC

  `http://java.sun.com/xml/downloads/jaxrpc.html`
- Web Services Security (WS-Security) Version 1.0

  `http://www.ibm.com/developerworks/webservices/library/ws-secure/`
- Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)

  `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-securi ty-1.0.pdf`
- WebSphere Application Server Information Center Library

  `http://www.ibm.com/software/webservers/appserv/infocenter.html`

- IBM developerWorks SOA and Web services zone

  `http://www.ibm.com/developerworks/webservices`
- *IBM WebSphere V5.0 Performance, Scalability, and High Availability,* SG24-6198

9

Enterprise Service Bus: Broker variation

In this chapter we expand on the concept of the Enterprise Service Bus (ESB) using the service-oriented architecture (SOA) Enterprise Service Bus: Broker variation. We explore the architectural implications of using the Enterprise Service Bus: Broker variation in an SOA by building on the scenarios that are supplied in Chapter 7, “The business scenario used in this book” on page 169.

In Chapter 8, “Enterprise Service Bus: Router variation” on page 175, we saw the implementation of the Router pattern to replace Direct Connections. In this chapter, we describe a fuller ESB implementation that reduces the number of service invocations by using the Enterprise Service Bus: Broker variation.

In this chapter, the following topics are discussed:

- The sample business scenario that our solution should address.
- Design guidelines that describe the design approaches for using the Enterprise Service Bus: Broker variation to mediate service interactions in an ESB.
- Development guidelines that show how development tools may be used to define the flows that mediate these service interactions in an ESB.
- Runtime guidelines that discuss the considerations for deploying the flows.

9.1 Business scenario

The sample application that was introduced in Chapter 7, “The business scenario used in this book” on page 169, is a simplified supply chain for a consumer electronics retailer. The supply chain components have been deployed as Web services, and the application can be used as shown in Figure 9-1.

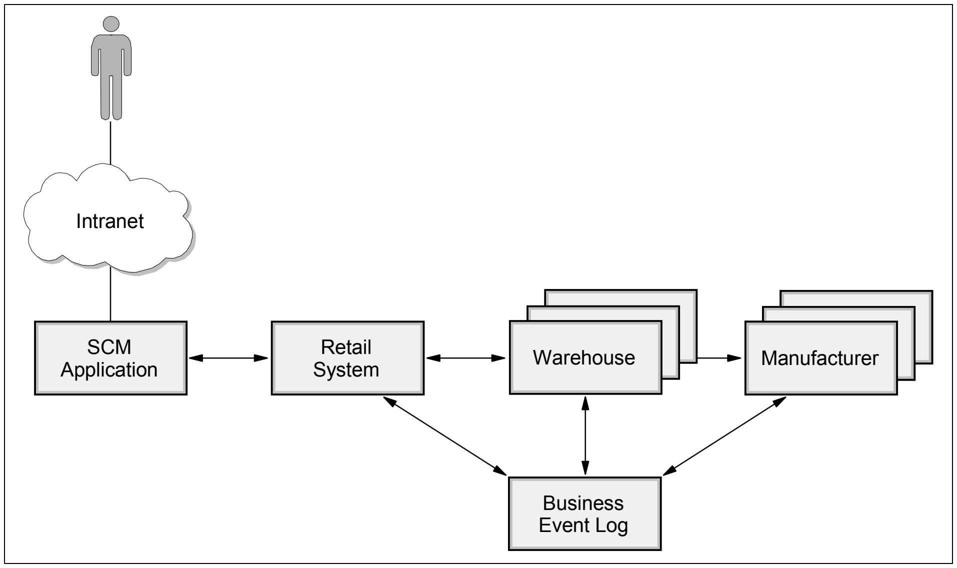


Figure 9-1 High-level business context showing the existing infrastructure

The company is facing several issues with the existing infrastructure, which it is looking to overcome. In addition to those discussed in the business scenario, there are two more challenges:

- The company has a requirement to stock the parts it offers to customers in more than one warehouse. Each part will only be held in one warehouse. However, the customer must see the order as a single transaction with the company. Therefore, an order must be seamlessly divided within the company so that requests for shipment by warehouses are only made for the parts it stocks. The responses from the warehouses must be aggregated to provide a single response to the customer.
- The company is looking to grow its product line through acquisition. This means being able to bring additional warehouses online and also broaden its supply chain by allowing more manufacturing facilities to make the company’s goods.

We will be implementing the Enterprise Service Bus: Broker variation to illustrate how a solution to these requirements can be provided.

9.2 Design guidelines

Many of the design guidelines that have been discussed in Chapter 8, “Enterprise Service Bus: Router variation” on page 175, apply when designing a solution using the Enterprise Service Bus: Broker variation. However, we discuss variations and additional considerations for meeting the requirements that are described in Stage II of the business scenario.

9.2.1 Design overview

An overview of the steps that might be taken to design a solution to address business requirements is shown in Figure 9-2. We now follow these steps.

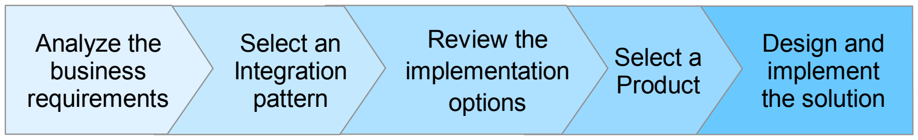

Analyze the business requirements
Select an Integration pattern
Review the implementation options
Select a Product
Design and implement the solution

Figure 9-2 Design overview

Selecting the pattern

We start by considering the requirements that are described in the business scenario in the previous section in conjunction with the Process Integration patterns found in 4.4, “SOA profile of the Application Integration patterns” on page 90. The need to ship orders from multiple warehouses can be summarized as follows:

- For a single client request, the ESB must distribute requests to multiple service providers and return a single response to the client.

Furthermore, there is a requirement to easily integrate acquired companies into the organization’s infrastructure:

- Additional warehouses must be added seamlessly behind the single client request from the retailer.
- Additional manufacturers could be accessed to provide goods to the warehouses.

Choosing the relevant SOA pattern

To address these requirements, we implement the Enterprise Service Bus: Broker variation for our scenario in this chapter. This variation uses the Process Integration Broker pattern, as shown in Figure 9-3 on page 222.

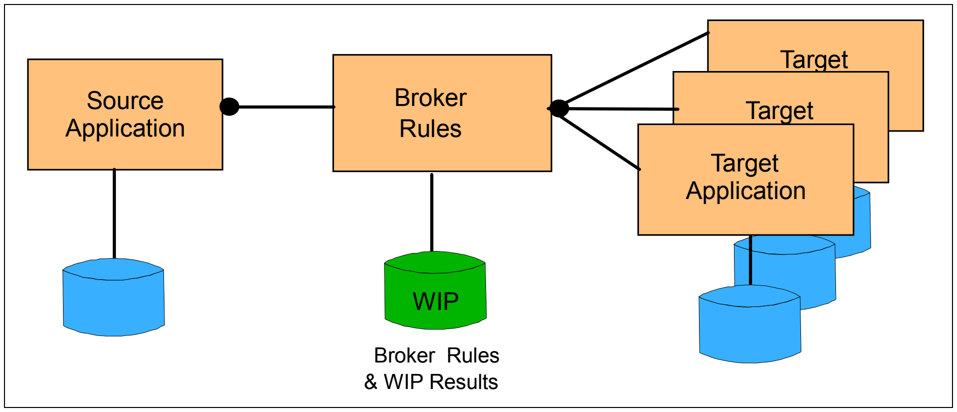


Figure 9-3 Broker application pattern

The Broker pattern dictates that a single request received from a source application can be sent to multiple target applications. The responses from the target applications can be aggregated into a single response, which is returned to the source application. For more information about the Broker pattern, consult *Patterns: Broker Interactions for Intra- and Inter-enterprise*, SG24-6075.

Applying the SOA pattern to the scenario

The Enterprise Service Bus: Broker variation is implemented for our business scenario, as shown in Figure 9-4.

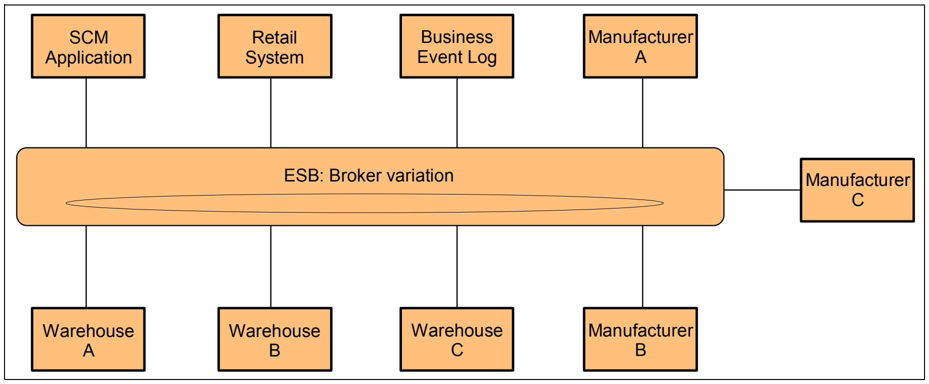


Figure 9-4 The Broker pattern in the ESB

Implementing the Enterprise Service Bus: Broker variation

Multiple similar interactions often benefit from a strategy of centralizing control to enable standardization and reuse across a distributed infrastructure. Implementing the Enterprise Service Bus: Broker variation provides message distribution that is based on predefined distribution rules that can invoke multiple targets concurrently. Key components of this solution design include:

- Message transformation
- Content-based routing
- Message decomposition/recomposition

The Broker pattern is described further in 3.4.4, “Broker application pattern,” in *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303.

Scenario requirement for the Broker variation

The implementation of the Enterprise Service Bus: Router variation in Chapter 8, “Enterprise Service Bus: Router variation” on page 175, eliminated the Direct Connection pattern for service invocations. However, it does not meet the additional requirements of our business scenario. In particular, the number of service invocations that are made by service clients will increase in line with the number of service calls. This may lead to higher latency in the execution of several application operations, such as the ship order request made by the Retailer. Perhaps, more important, it is not optimizing ongoing maintenance effort and overall flexibility of the ESB.

By introducing the Enterprise Service Bus: Broker variation, service invocations can be encapsulated behind a single request to the ESB from a service client. In our scenario, we are using this pattern to simplify the access by the Retailer to fulfill the order across several of the Warehouses: instead of implementing a service call to each warehouse in the Retailer, a single service request can be made by the Retailer to the ESB. The ESB will make a service request to each of the Warehouses, aggregate the responses, and return the results to the Retailer.

The general requirements for service routing and protocol conversion in this chapter are the same as for Chapter 8, “Enterprise Service Bus: Router variation” on page 175.

Product implementation options

We now consider the capabilities of the ESB in the context of selecting a product to implement the Enterprise Service Bus: Broker variation. Our product selection for the scenario was based on:

- Available products.
- The ability of the products’ capabilities to map to the requirements.
- The existing infrastructure of our organization. (For example, does the company already use one of the products?)

The following products could be used to implement an Enterprise Service Bus: Broker variation:

- WebSphere Business Integration Message Broker V5.0
- WebSphere InterChange Server V4.2
- WebSphere Business Integration Server Foundation V5.1

To help with selecting the appropriate product, refer to:

- The description of each product in 5.1, “Runtime product descriptions” on page 134.
- The ESB capabilities of each product, described in 5.3, “Product capabilities for the Enterprise Service Bus” on page 144.

Product selection for scenario implementation

To address the requirements that are described in 9.1, “Business scenario” on page 220 using the Enterprise Service Bus: Broker variation, we selected WebSphere Business Integration Message Broker. The product mapping is shown in Figure 9-5.

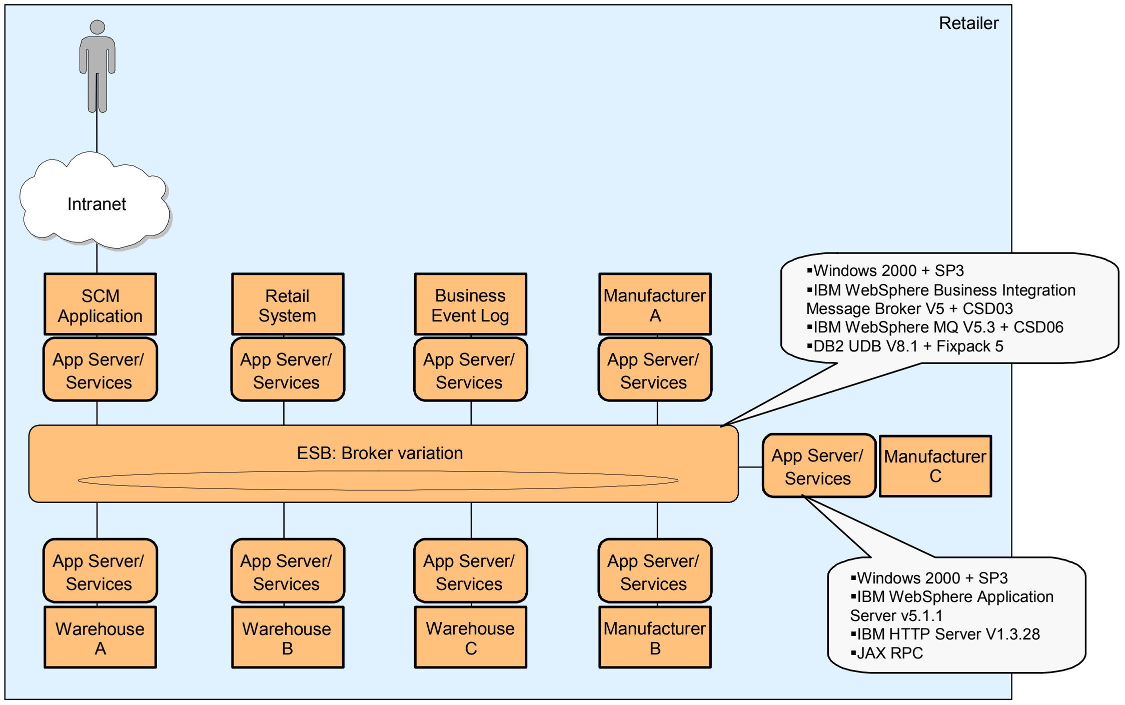


Figure 9-5 Product mapping for Application Integration::Broker pattern on the ESB

In our lab environment, only Windows 2000 machines were used. However, it is likely that many enterprises will choose to implement their Enterprise Service Bus on other platforms such as IBM pSeries running AIX.

Figure 9-6 illustrates a different view of our solution to the scenario requirements in terms of service requesters, service providers, and the protocols used for communication with the ESB.

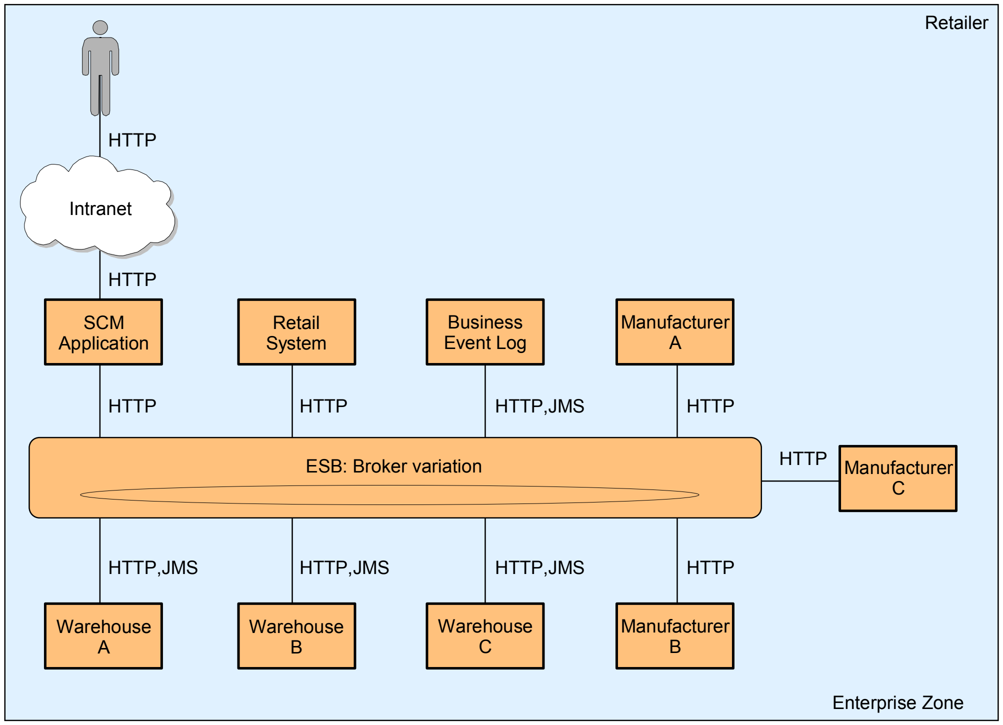


Figure 9-6 Solution to the scenario requirements

9.2.2 Broker design

Many of the design guidelines that are provided in 8.2.2, “Router variation” on page 181 apply to the Enterprise Service Bus: Broker variation. Additional considerations that should be given to the Enterprise Service Bus: Broker variation over the Enterprise Service Bus: Router variation now follow in three categories:

- ► Design approach
- ► Broker communication
- ► Inside the Broker

Design approach

Here we discuss some of the approaches to take when designing the Broker variation implementation.

Design alternative: top-down or bottom-up

A top-down approach develops the WSDL definition from scratch whereas a bottom-up approach uses an existing interface to derive the WSDL definition.

The introduction of the ESB adds further considerations. For a top-down approach these are as follows:

- Two WDSL definitions must be developed for each service: one provided by the ESB to the service client and one for the server-side service provider.
- The WSDL definitions for each service can be identical across the ESB except for the port element (which specifies the address for binding) and the binding element when different protocol support is required. The other major elements of the WDSL definition can be common across the ESB.
- Interoperability is facilitated by largely common definitions.

For a bottom-up approach the considerations are:

- The WSDL definition is derived for each server-side interface. Using the Enterprise Service Bus: Broker variation, multiple interfaces may have to be aggregated by the ESB to provide a service for clients. It is still possible to use a common WSDL definition for a service across the ESB as in the top-down approach.
- Such aggregation can involve complex service design and complex transformation logic to enable results from service providers to be merged for the client.
- Existing system implementations can be considered in the design of the service provided by the ESB.

It is also worth noting that the access of server-side functionality and data by the ESB does not have to be provided as a Web service. Many access mechanisms can be used by technologies that are capable of implementing the Enterprise Service Bus: Broker variation to reach these existing systems more easily. Examples include applications that are already MQ-enabled, those that are accessible through adapters, and SQL. In such cases, the Enterprise Service Bus: Broker variation would encapsulate such access behind a service interface that it provides to consumers.

Formally, this type of access is still part of an ESB because it is still able to implement a single point of control over the namespace for the services that are contained within. However, it is likely that use of such an access mechanism represents a Direct Connection pattern implementation from the ESB to an application.

Chapter 6, “Endpoint enablement roadmap” on page 153, discusses endpoint access in more detail. This includes accessing endpoints that do not provide a Web services interface.

For our scenario implementation, the WS-I has provided the WDSL for the services. We have used the same WSDL for service clients and service providers, changing the port in the service requester WSDL definition to access the service on the ESB.

As we write this book, WebSphere Business Integration Message Broker is more suited to a bottom-up approach because it is highly capable of reflecting server-side interfaces.

Design alternative: location of business logic

We consider business logic in terms of two types of functionality:

1. Aggregation

   One of the additional requirements of the business scenario is to support the ship order request across multiple warehouses. This requires the use of aggregation to manage multiple requests to the Warehouses in response to a single request from the Retailer. With this type of requirement, careful consideration must be given regarding the location of business logic.

   It is worth noting here that in some ways, aggregation is the result of business logic because decisions must be made about whether to return one, some, or all of the responses to the original requester.

   In practice, requirements are more complicated than for our scenario. For example, more than one warehouse might be allowed to stock each part, and fulfilment for a part on an order might be by more than one warehouse. In this case, the best implementation of the aggregation may include some business logic.

2. Process management

   In the context of business logic, we must also explore process management. An ESB does not provide full process choreography: business processes execute in an external engine and invoke services that are provided by the ESB. Similarly, the ESB can invoke business processes that are provided by a separate process engine.

   However, there may be situations where the combination of low-level services into a single exposed service could take place inside the ESB. Internal serial process logic can be used to handle sequences of calls such as those involved in screen-scraping dialogs with legacy systems.

Such business logic can be implemented in applications. In this case, these points should be noted:

- Applications are owned by business divisions (such as Sales and Manufacturing) in most organizations. This means that they have control over the logic.
- Flexibility of business logic within a business division is facilitated. Services must be designed to allow changes to business logic by a business division without the need for other service providers to change their WSDL definitions.

The alternative is the ESB. However, the drawbacks of locating business logic in the ESB are:

- The ESB is rarely owned by a business division, and so development of and changes to logic in the ESB may require consensus across divisions in many organizations.
- The impact of changes to logic in the ESB on business divisions may be high.
- The logic is centralized in the ESB, facilitating reuse, allowing a single point of maintenance, and enabling it to be provided as a service.

The important design point here is that within an architecture for each ESB implementation it should be clear where such logic should be assigned.

Note: Although an ESB will probably not be owned by a business division, an ESB implementation is unlikely to be successful unless there are governance processes that enable a business division to decide what simple and complex services it wishes to see deployed and to request these. Projects determine the functions that are required of the back-end systems, and they should be involved in defining the WSDL exposed by the ESB.

For the scenario implementation, we chose to ensure that the business logic was contained in one of the endpoints and not in the ESB.

Design alternative: service granularity

Service granularity is an important consideration when implementing the Enterprise Service Bus: Broker variation. This is because the amount of transformation, aggregation and routing implemented in the ESB can significantly affect the granularity. This in turn affects efficiency, the amount of reuse, and ongoing maintenance effort.

The core principles are as follows:

- Services that implement business interactions are likely to be more stable. Overall success is more likely.

- Services should be designed as complete units of work. A Web service probably maps best to the implementation of a use case.
- Implement a façade over fine-grained object methods. This minimizes network latency and the number of cross-network calls. There may have to be some serial logic when services are being built from calls to existing back-end systems and there is a desire not to expose the fine details.
- Coarser granularity improves performance and hides implementation details from consumers (including the developers of client applications).
- Services can be used by multiple service clients if they are general enough.
- Services on the ESB should not implement Business Service Choreography business logic (except for non-functional requirements or as described in "Design alternative: aggregation versus serial processing" on page 235). They can encapsulate external serial processing.

Implementing services for demanding non-functional requirements for fine-grained access is not recommended. For example, high-performance access to an endpoint should be direct. Alternatively, see the database lookup pattern described in "Design alternative: externalizing service lookup" on page 232.

In our scenario implementation, the WSDL was provided by the WS-I, so this was not a factor that we had to consider. However the primary use of WebSphere Business Integration Message Broker is typically mediation and it is likely that the granularity of integration endpoints will be significant in determining the amount of transformation, routing, and aggregation.

Broker communication

Here we discuss how the ESB is accessed and how it accesses service providers when it implements the Enterprise Service Bus: Broker variation.

Design alternative: JMS versus HTTP

The two primary options for transporting SOAP messages are HTTP and JMS. These technology options are discussed in Chapter 6, "Endpoint enablement roadmap" on page 153.

It is important to recognize that there is no single point of control over the namespace with HTTP because of the characteristics of the widely used domain controller infrastructure. This differs from the scope of an ESB that is defined by a group of services that are accessible through one or more protocols whose addressing and naming is controlled at a single point.

A Broker variation solution can aggregate service calls across multiple providers using multiple transport protocols. However, the reliability of the service that is provided by the ESB should be considered. Implementing JMS as the communication from the ESB to providers may allow a transactional controller where appropriate technologies allow and are selected.

WSDL definition changes are required to provide protocol conversion from HTTP to JMS in the Broker variation solution. In summary, changes must be made to the transport attribute of the soap:binding and the location attribute of the soap:address.

It is expected that many enterprises will use JMS as a reliable transport for their protocol to the service providers where the services are internal and communication does not pass through a firewall. WebSphere MQ is a JMS provider that assures once-only delivery of persistent messages.

Design alternative: externalizing the service endpoint

In any particular situation there may be multiple namespaces, but the only one that counts in the context of an ESB is that which contains the service names. By using some kind of external data store to manage this namespace we have a single point of control irrespective of the transport mechanism.

This control over service mapping via a service namespace:

- Enables a single point of control over routing from the indirect to the direct address.
- Achieves location transparency by decoupling the client and service invocation.

In order to implement the service mapping or routing rules that are required in order to achieve this, some form of service routing table is implied. Although collections of message flows that implement individual service mappings could be used to implement this, a better practice is to externalize the routing information to a database or directory.

Approaches to external service lookup are discussed further in "Design alternative: externalizing service lookup" on page 232.

Inside the Broker

We now consider design alternatives to be made within a Broker variation solution.

Design alternative: single versus individual flows

There are two extremes of approach to implementing logic inside the Broker variation solution:

- Each service call that is made by a client can be routed through its own flow inside the ESB.

- All client service calls can be targeted at a single generic flow inside the ESB where the originator is identified to route the request.

An individual flow for each client service operation may be quicker to build than flows with generic handling, but they are not easy to manage and maintain. There are two alternatives:

- Implement flows with some degree of generalism, perhaps by service.
  - These can be implemented as a distinct flow for each service provided by the ESB, for example.
  - Each flow provides routing and mediation support for the operations that are provided by the service.
  - This approach lends itself well to services that are invoked over JMS.
  - Management of the ESB reflects the breakdown of the implemented flows. This would provide a level of simplification.
- Implement a single generic flow for all services.
  - The approach can be implemented by placing a generic entry point façade over the set of flows that were implemented in the alternative approach.
  - The single generic flow approach can easily support services that are invoked over HTTP.
  - Management involves a single flow that may not provide the granularity for most simplicity, although it does provide a manageable entry point.
  - When the routing directory is external, the routing is maintained by updating the routing directory and not by amending and redeploying the logic in the flow.
  - Where there are other mediation requirements, such as transformation, these must be implemented in a second flow, and so some of the advantages of the single generic flow approach will be lost.

For an ESB, all the approaches are valid because each reflects the single namespace requirement over the services in an ESB. From a best practices viewpoint, some degree of generalism should be employed.

Use of a generic flow requires a service location lookup. This is discussed in "Design alternative: externalizing service lookup" on page 232.

We now look at the design for using HTTP and JMS as the entry point protocol in turn:

- HTTP

  Where HTTP is used as the transport protocol by service clients to reach the ESB, a common URI can be used to access the generic flow. Either a URL selector or an entry in the SOAP header can be used to identify the required service provider.

- JMS

  A generic flow can only listen to a single queue, so different service operations are requested on the same queue. This means that the service information that can be used by the ESB to identify the target provider must be carried in the SOAP header (or RFH2 header in the case of WebSphere Business Integration Message Broker).

Design alternative: externalizing service lookup

There is an additional requirement for the ESB when implementing a single generic flow: The address of the service provider must be established based on the incoming service request.

One option is to write the mapping of originator service to provider service(s) into the flow. This hard-codes the service routing and is not recommended.

An alternative approach is to externalize the lookup in a directory. The advantages of this approach are:

- Loose-coupling of services.
- A standards-based mechanism can be used.
- It allows a single point of control over routing from an indirect to a direct address.
- The address information can be maintained in the directory without having to maintain the logic that is defined in the ESB. For example, the services might change physical location over time.
- Flexibility to support the promotion of code from one environment to another, such as test to production.
- Several identical services can be deployed in different locations to provide high availability (via failover) and some level of load balancing.

The technology choices for implementing this directory include a relational database and possibly UDDI. The purpose of UDDI is more oriented to being accessed by a service client. For the ESB we require a directory for use by the ESB. A UDDI implementation provides these additional capabilities over a database implementation:

- Dynamic discovery of service interfaces at runtime
- Allows the use of different providers of the same service

For our ESB, these additional capabilities are not required. In our scenario implementation of the Enterprise Service Bus: Broker variation using WebSphere

Business Integration Message Broker, we chose to build on the core dynamic routing capability of the product and implement a lookup using a table in a database. Figure 9-7 shows this best-practice pattern for providing a solution for our directory.

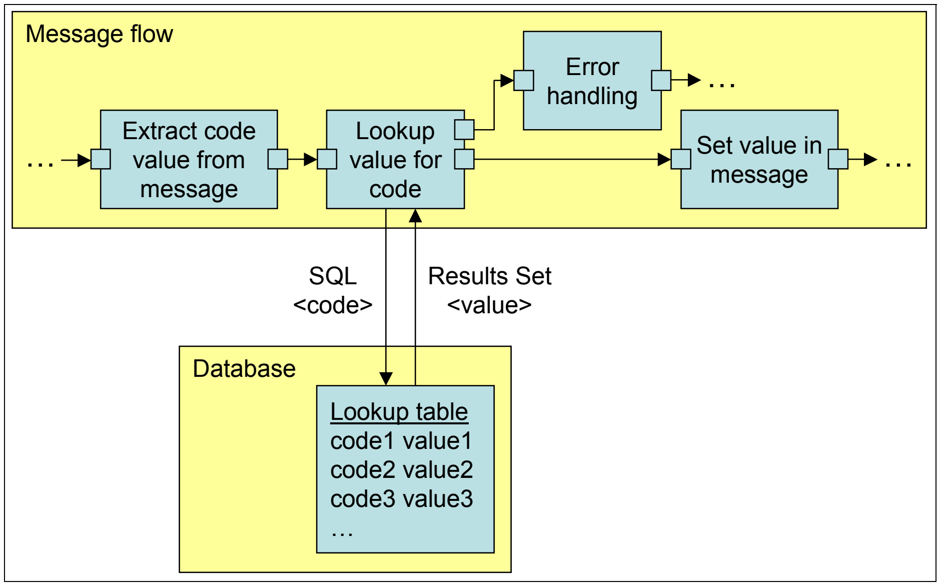


Figure 9-7 Database lookup pattern

Suppose that the service client makes a request to the URL:

```
http://localhost:7080/ServiceBroker/Manufacturer
```

The WebSphere Business Integration Message Broker HTTP Input node picks up the request because the HTTP server in WebSphere Business Integration Message Broker is listening on port 7080.

1. From a field on the incoming message, `Manufacturer` is stripped from the incoming request URL.
2. Then a call is made to the database to look up the address of the service provider using the name `Manufacturer`.
3. The flow sets the service provider's address in the message structure.

The flow makes the service request to the provider.

For access to a service provider over JMS, the design must be altered as follows:

1. From a field on the incoming message, `Manufacturer` is stripped from the incoming request URL.

2. A call is made to the database to look up the target queue name for the service provider using the name `Manufacturer`.
3. The flow sets the destination queue name in the message structure.
4. The appropriate JMS headers must be created in the message.

The request is made by putting the message on a queue.

Note: The use of WebSphere Business Integration Message Broker in our implementation requires one flow to put the MQ message and another to handle the response. A description of how to do this is included in 9.3, "Development guidelines" on page 236.

Note: To support both HTTP and JMS for service providers, a column must be added to the service lookup table in the database to hold the protocol. The flow must then route accordingly.

In these examples, note that access to the database using Web services for the address information would be an example of an anti-pattern and should be avoided: SQL access should be used directly.

Design alternative: partial versus full SOAP intermediary

A true SOAP intermediary will carry out the validation and processing of SOAP headers. The simple advantage of not validating and processing SOAP headers is performance, because parsing of the message is not required. However, SOAP processing will be of minor impact to performance compared with the overall solution design.

There are several advantages to performing the validation and processing of SOAP headers:

- SOAP faults are handled properly so that they correctly identify the service that is causing the fault:
  - Generation of SOAP faults
  - Encoding of SOAP faults
- Operations supported by the service can be validated.
- The SOAP header must be properly validated and processed when the mustUnderstand attribute is set to `1`.
- Decoding and encoding of the SOAP service requests and responses.
- Setting of the operation name and related namespace.
- Switching service style from document to RPC.

In our scenario implementation that uses WebSphere Business Integration Message Broker, we have accessed the elements of the SOAP message when required. However, time constraints prevented us from implementing a full SOAP intermediary.

If this functionality is required, download SupportPac IA81 (WebSphere Business Integration Message Broker and Web Services) from the IBM SupportPacs Web site or at:

`http://www.ibm.com/support/docview.wss?rs=203&uid=swg24006268`

It provides the following:

- A library of procedures handling the SOAP elements of Web service messages
- Examples of how Web services may be used with WebSphere Business Integration Message Broker
- Sample scenarios
- Supporting assets in the form of message sets, message flows, custom nodes, and test messages

Design alternative: aggregation versus serial processing

The alternative here relates to the specific implementation options in the product that is selected to implement the Enterprise Service Bus: Broker variation.

True aggregation in this context means that requests are made to more than one service provider by the ESB in parallel on behalf of a single service request from a client. It involves the management of the responses from each of the service providers and the merging and transformation of the results into a single response to the original requestor. This means considering the following:

- Overall time-out for the aggregation
- Handling of partial set of responses through the non-response or error from one or more providers
- Handling of invalid replies
- Construction of the SOAP body response by merging the results from the service providers appropriately
- Transactional requirements

Not all products are capable of aggregation and neither are all requirements where a single request requires access of multiple providers suited to it. In such cases it makes sense to consider serial calls from the ESB to the service provider.

We provide the following guidance for when to use aggregation:

- End-to-end requests to service provider are fast.
- There are many requests to be made to service providers for each client request.

The duration of a serial process implementation is the sum of the duration of the steps, whereas for aggregation it is the duration of the longest individual call.

The important distinction between this type of serial process and the type discussed in Chapter 10, “Business Service Choreography” on page 271 is that any one call to a provider does not depend on the previous calls. If such a dependency is a requirement, then the serial process should not be implemented on the ESB.

In our scenario implementation we have used aggregation for the ship order service operation. Figure 9-8 shows the high-level design for the message flow.

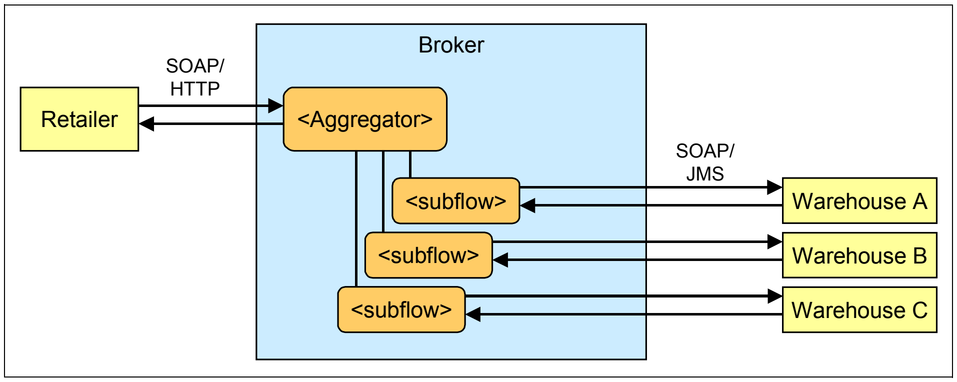


Figure 9-8 Aggregation design

9.3 Development guidelines

In this section we take the draft ESB design shown in Figure 9-5 on page 224 and present a detailed description of the steps that are taken to implement the ESB using WebSphere Business Integration Message Broker V5. Since this redbook is not an introductory text for the technologies that we use to build example ESB implementations, a certain level of understanding is assumed. Readers who are not familiar with WebSphere Business Integration Message Broker should refer to 9.5, “Further information” on page 268.

Within WebSphere Business Integration Message Broker, Brokered service invocations are managed as message flows. Our working scenario includes examples of service invocations with several differing characteristics. Collectively, these message flows facilitate the centralized brokering of the service invocations in our sample scenario and provide an example implementation of the Application Integration::Broker pattern. As shown in Table 9-1, the service invocations within our scenario can be conveniently partitioned into the following types:

- Serial HTTP invocation

  In this simple invocation type, a client makes a single service request over HTTP. WebSphere Business Integration Message Broker behaves as a simple SOAP intermediary, dynamically routing the request to an available service endpoint. The resulting HTTP data stream that contains the service response is returned to the client.

- Serial HTTP to JMS invocation

  In this invocation type the target service is exposed using JMS transport bindings; however, the service client makes an HTTP-based service invocation. WebSphere Business Integration Message Broker mediates between the disparate client and service by providing a protocol switch capability.

- Parallel HTTP to JMS invocation

  This service invocation type requires the mapping of a single source HTTP request to multiple concurrent target JMS services. The HTTP responses that are returned from the individual services are aggregated and returned to the caller. The decision to route to one or more of several preconfigured service endpoints is managed dynamically at execution time.

We chose to support these invocation types via the implementation of the three distinct message flows shown in Table 9-1.

Table 9-1 Implemented message flows

Invocation Type	**Message Flow Name**
Serial HTTP	HTTPtoHTTPMsgFlow
Serial HTTP to JMS	JMSLoggingRequestMsgFlow
Parallel HTTP to JMS	AggQtyMsgFlow

9.3.1 Prerequisite configuration

WebSphere Business Integration Message Broker provides explicit support for the handling of incoming HTTP requests. This capability is provided by the HTTPListener service. The port that is used by this service is defined during the creation of a Broker instance, and it defaults to port 7080. The examples and discussion in the following sections of this chapter use this default value. In the event that your environment has been configured to use a different port, you may need to adjust values in the following discussion accordingly.

WebSphere Business Integration Message Broker provides native ODBC support, enabling executing message flows to interact with external data repositories (typically relational databases). This capability is usually used from within WebSphere Business Integration Message Broker to enrich message data en route to its target application, or to retrieve details of the target endpoint location. In order to enable this capability, it is necessary to define an ODBC data source via which the external data repository can be accessed. Because we make use of this native ODBC support to access DB2 UDB, the definition and population of tables, together with the configuration of an ODBC data source, are prerequisite activities to the development steps outlined below.

9.3.2 Broker variation implementation

In this section we describe the development of three message flows that collectively implement all of the service invocation types that are contained within our scenario.

Important: Some of the code fragments that are presented in the remainder of this section have a line length that exceeds our page width. For this reason we use the '||' characters to represent a line break that has been inserted for formatting purposes. For example, the following formatted line of code:

```
OutputRoot.XMLNS.SoapNS:Envelope.SoapNS:Body.ns1:ShipGoods. ||
ItemList = NULL;
```

represents:

```
OutputRoot.XMLNS.SoapNS:Envelope.SoapNS:Body.ns1:ShipGoods.ItemList = NULL;
```

Serial HTTP flow implementation

Our serial HTTP message flow is invoked by the arrival of an HTTP request at the HTTP listener port. The message flow parses the incoming message structure to deduce the target service name, which is then used as the key in the retrieval of the target service endpoint address from an external data repository. The target service is invoked by transmitting the incoming SOAP request envelope over HTTP to the endpoint address. The resultant SOAP response envelope is returned unmodified to the requesting client. Figure 9-9 on page 239 depicts a high-level representation of the Application Integration::Router pattern implementation in our environment.

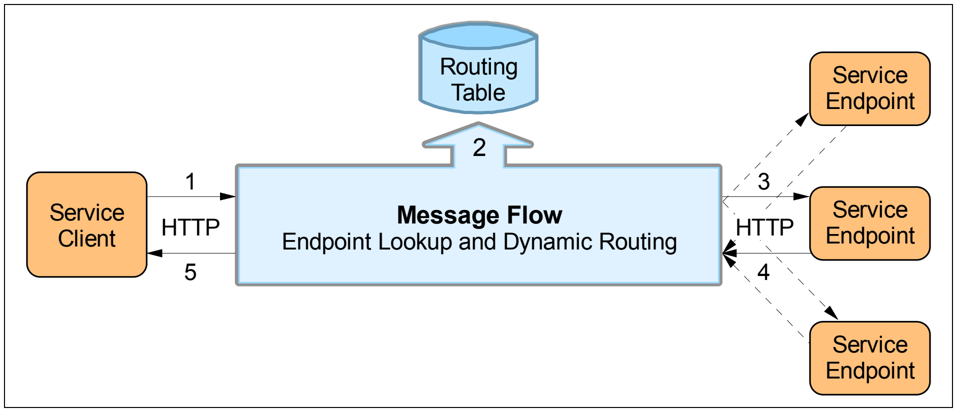


Figure 9-9 High-level representation of the serial HTTP message flow

The various activities that make up the overall message flow functionality are represented by the nodes of a directed graph. The nodes shown in Table 9-2 were used to build the serial HTTP message flow.

Table 9-2 Simple Router pattern message flow nodes

Node Name	Node Type
HTTP Input	HTTP Input
IdentifyService	Compute
HTTP Request	HTTP Request
PostRequest	Compute
HTTP Reply	HTTP Response

Figure 9-10 shows the directed graph representation of our simple Router pattern message flow.

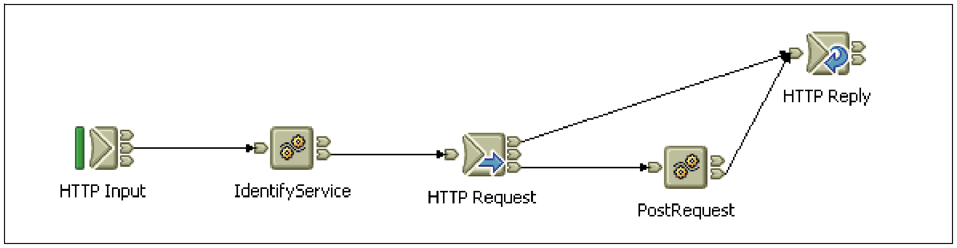


Figure 9-10 HTTPtoHTTPMsgFlow message flow

HTTP Input : HTTP Input

This initial node facilitates the invocation of our message flow as a result of the arrival of an incoming HTTP message to the WebSphere Business Integration Message Broker HTTP Listener. The URL Selector property value is used to register a message flow as the recipient of HTTP request messages that are addressed to matching URI values. Table 9-3 summarizes the relevant property values of the HTTP Input node.

Table 9-3 HTTP Input node configuration property values

Category	Name	Value
Basic	URL Selector	/ServiceBroker/*
Basic	Maximum client wait time	33
Default	Message Domain	XMLNS

Compute : IdentifyService

The Compute node is provided to enable explicit manipulation of message content via the provision of ESQL code. In the IdentifyService node, we parse the URL field of the incoming HTTP Request to determine the service name.

Example 9-1 Parsing the service name

```
DECLARE SVC CHAR;
DECLARE SVCURL CHAR;
DECLARE STEMLEN INTEGER;
DECLARE SVCLEN INTEGER;

CALL CopyMessageHeaders();
...
-- Extract the required service name from the client's URL
SET SVCURL = InputRoot.HTTPInputHeader."X-Original-HTTP-Command";
SET STEMLEN = POSITION('/' IN SVCURL FROM 1 REPEAT 4);
SET SVCLEN = LENGTH(SVCURL)-9-STEMLEN;
SET SVC = SUBSTRING(SVCURL FROM STEMLEN+1 FOR SVCLEN);
```

This value is used to retrieve the service endpoint URL from a predefined database table. We dynamically specify the service endpoint URL by overriding the default setting of the HTTP Request node Web service URL with the retrieved value.

Example 9-2 Overriding default Web service URL with retrieved value

```
-- Use the service name to look up the service provider's URL
SET OutputRoot.HTTPRequestHeader."X-Original-HTTP-URL"
```

```
    = THE (SELECT ITEM S.LOCATION FROM Database.SERVICE_ROUTER AS S
          WHERE S.NAME = SVC);
```

Finally, we must ensure that the HTTP message body is forwarded to the service endpoint.

Example 9-3 Copy HTTP request body to Output Root

```
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
...
SET OutputRoot.*[J+1] = InputRoot.*[J];
```

Table 9-4 presents the relevant property values of the IdentifyService node.

Table 9-4 IdentifyService node configuration property values

Category	Name	Value
Basic	Data Source	BROKERDB
Basic	Compute Mode	Message

HTTP Request : HTTP Request

The HTTP Request node enables external HTTP applications to be consumed from within a Message Flow. Although we specify a default value of the service endpoint address, in our scenario this value is always overridden in the IdentifyService node as described above. Table 9-5 presents the relevant property values of the HTTP Request node.

Table 9-5 HTTP Request node configuration property values

Category	Name	Value
Basic	Web Service URL	http://localhost
Basic	Request Timeout	20
Advanced	Use whole input message as request	Checked
Advanced	Replace input message with web-service response	Checked
Advanced	Generate default HTTP headers from input	Checked
Default	Message Domain	XMLNS

Compute : PostRequest

We have provided a Compute node following the service request as a container for logic that should be executed prior to returning the service output to the requesting client. Because the HTTP Reply node that is described is configured to generate default HTTP headers, we simply suppress the HTTP header returned by the service invocation.

Example 9-4 Suppressing HTTP response header from service

```
CALL CopyEntireMessage();

-- Remove HTTP header from the service’s response
Set OutputRoot.HTTPResponseHeader = null;
```

Note that all Configuration properties for the PostRequest node are set to their default values.

HTTP Response : HTTP Response

The HTTP Response node is responsible for returning the output from the service endpoint back to the client. Table 9-6 shows configuration property values that are used in this example.

Table 9-6 HTTP Response node properties

Category	Name	Value
Basic	Ignore Transport Failures	Checked
Basic	Generate default HTTP headers from input or response	Checked

Note: This HTTPtoHTTPMsgFlow is able to handle both Document and RPC-style Web service requests. However, it can invoke Web services only over HTTP using request-response and not one-way calls.

Serial flow with protocol switch implementation

Our example scenario provides an example of an HTTP-aware service client that requires access to a service that is exposed only via a JMS transport binding. This transport capability mismatch necessitates the introduction of a protocol switch capability to mediate among the disparate applications. This requirement is well-suited to the capability provided by WebSphere Business Integration Message Broker. In this section we describe the development of a message flow that implements the required protocol switch to facilitate interoperability between the client and service.

Table 9-11 on page 243 presents a high-level representation of the functionality that is provided by this message flow.

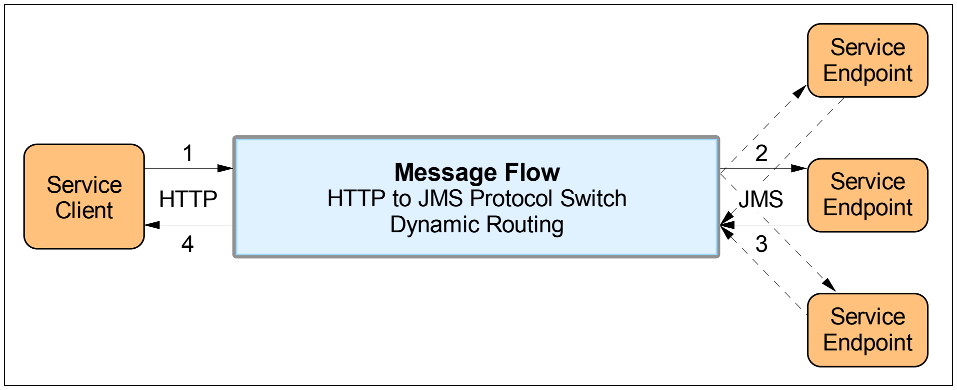


Figure 9-11 High-level representation of serial flow with protocol switch implementation

The nodes in Table 9-7 were used to build the serial protocol switch message flow.

Table 9-7 Serial flow with protocol switch message flow nodes

Node name	Node type
HTTP Input	HTTP Input
PrepareJMS	Compute
MQOutput	MQ Output
PrepareHTTPResponse	Compute
HTTP Reply	HTTP Reply

Figure 9-12 shows the directed graph representation of our message flow.

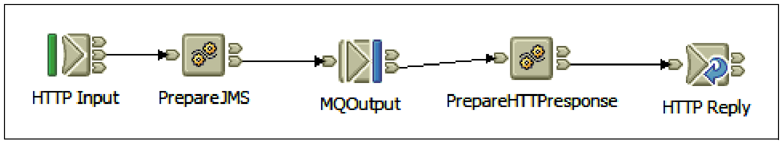


Figure 9-12 JMSLoggingRequestMsgFlow message flow

HTTP Input : HTTP Input

In a similar manner to that described for our earlier message flow in “Serial HTTP flow implementation” on page 238, the incoming HTTP Request is accepted by the HTTP Input node. Note that in contrast to the previous flow, we have specified the URL Selector property value in the form of a fully qualified URL in this message flow by way of example (Table 9-8 on page 244).

Table 9-8 HTTP Input node configuration property values

Category	Name	Value
Basic	URL Selector	http://localhost:7080/LoggingFacility
Basic	Maximum client wait time	30
Default	Message Domain	XMLNS

Compute : PrepareJMS

The subsequent PrepareJMS Compute node is responsible for preparing the in-flight message for onward transmission over JMS. We first set the MQMD header fields required by WebSphere MQ, and then specify JMS header fields that will be used by the target JMS Web service implementation. Finally we suppress the HTTP header information accompanying the incoming request. Figure 9-5 provides the salient code associated with this node.

Example 9-5 ESQL for Compute node PrepareJMS

```
CALL CopyEntireMessage();

-- Add MQMD headers for MQ Application
CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN 'MQMD';
SET OutputRoot.MQMD.Version  = MQMD_CURRENT_VERSION;
SET OutputRoot.MQMD.StrucId  = MQMD_STRUC_ID;

-- Add the MQRFH2 header to hold the jms folder
-- Set the response queue, target service and message content type
CREATE NEXTSIBLING OF OutputRoot.MQMD DOMAIN 'MQRFH2';
SET OutputRoot.MQRFH2.mcd.Msd = 'jms_bytes';
SET OutputRoot.MQRFH2.jms.Dst = 'queue:///LoggingFacilityQ';
SET OutputRoot.MQRFH2.usr.targetService = 'LoggingFacility';
SET OutputRoot.MQRFH2.usr.contentType = 'text/xml; charset=utf-8';

-- Clear the http header
SET OutputRoot.HTTPInputHeader = NULL;
```

Note: The values of destination and targetService in the MQRFH2 header could be set dynamically in ESQL so that the flow becomes more generic. This would provide capability similar to the previous HTTP routing message flow, as described in “Serial HTTP flow implementation” on page 238. The routing table that is accessed in DB2 UDB would have additional columns, including service name, destination queue, and the name of the target service. A reply to queue could also be held on the table if a response was required.

MQOutput : MQOutput

Here we complete our protocol switch by writing the in-flight message to a preconfigured WebSphere MQ queue. It should be noted that in this example the forwarding of the message on to the target service is an asynchronous operation. The MQOutput node does not block until the corresponding service response message arrives; logical control advances immediately to the next node in the message flow. Table 9-9 summarizes configuration property values for the MQOutput node.

Table 9-9 Configuration property values for MQOutput node MQOutput

Category	Name	Value
Basic	Queue Name	LoggingFacilityQ
Advanced	Destination Mode	Queue Name
Advanced	Transaction Mode	Automatic
Advanced	Persistence Mode	Automatic
Advanced	New Message ID	Checked
Advanced	New Correlation ID	Checked
Advanced	Message Context	Default

Note that all other configuration property values for this node are left unset and unchecked.

Compute : PrepareHTTPResponse

Following interaction with WebSphere MQ, we must now ensure that the in-flight message is compatible for returning to the service client over HTTP. This requires the suppression of the WebSphere MQ-specific MQMD message header, which is achieved with a single ESQL statement executed within this node, as shown in Table 9-6. Note that all node properties are set to their default values.

Example 9-6 ESQL code for Compute node PrepareHTTPResponse

```
CALL CopyEntireMessage();

-- Remove the MQMD header
SET OutputRoot.MQMD = NULL;
```

HTTP Reply : HTTP Reply

Finally, a response is returned to the originating client over HTTP. Note that the specified properties of the HTTP Reply node ensure that the header information required by the HTTP transport protocol are generated (Table 9-10).

Table 9-10 Configuration property values for HTTP Reply node

Category	Name	Value
Basic	Ignore Transport Failures	Checked
Basic	Generate default headers from input or response	Checked

Note: This message flow shows protocol conversion from HTTP to JMS for Web services that use fire and forget for the JMS message. It will be a frequent requirement to support services in this situation that are request-response.

In this case, two message flows must be developed because the MQInput node in WebSphere Business Integration Message Broker has no input terminal. Thus one flow begins with an HTTPInput node and the other ends with an HTTPReply node. In this case, the HTTP context must be passed from one flow to the other so that the response can be returned to the requestor.

Unfortunately, WebSphere Business Integration Message Broker does not provide any built-in mechanism for persisting data across message flows. However, there are several ways in which this information can be passed. One is shown in Example 4 of IBM SupportPac IA81, and another, which uses aggregation, is shown in the next message flow.

Parallel HTTP-JMS protocol switch flow implementation

In our example Supply Chain Management scenario, a given retailer's stock is sourced from multiple different warehouses. In order to isolate the retailer from the potentially changing number and identity of available warehouses, we implement the logic that is responsible for routing warehouse requests in the Enterprise Service Bus. Because our front-end application permits the customer to submit an order that consists of several items, the order may have to be satisfied by shipments from multiple warehouses. To optimize the process from a performance perspective, we need to transmit warehouse requests that correspond to a single customer order concurrently. Since the varying number of warehouses that are required to fulfill a given request is transparent to the retailer, there is an implicit need to aggregate responses from individual warehouses before returning the response.

In this section, we describe the development of a message flow that provides the concurrent service invocation and response aggregation capability. Figure 9-13 on page 247 depicts a high-level representation of the retailer-to-warehouse interaction.

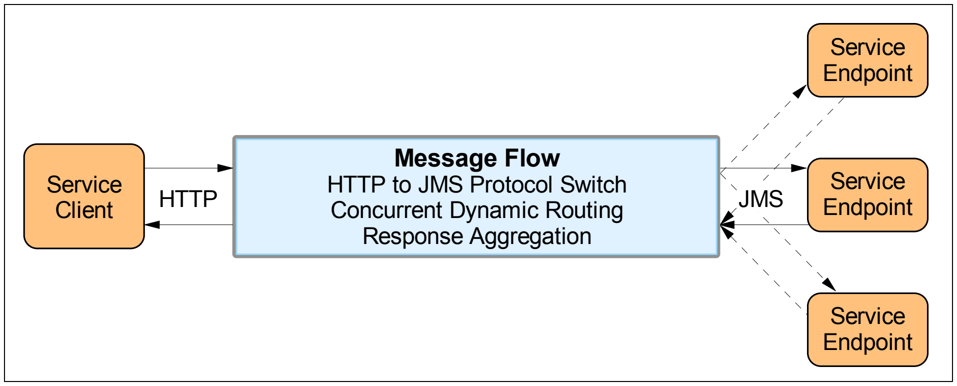


Figure 9-13 High-level representation of Broker pattern implementation

The nodes that are used to define the WebSphere Business Integration Message Broker message flow are tabulated in Table 9-11.

Table 9-11 AggQtyMsgFlow nodes

Node name	**Node type**
HTTP Input	HTTPInput
AddMQMD	Compute
AggCtrlQty	AggregateControl
Whse A Request	Compute
Whse B Request	Compute
Whse C Request	Compute
SetHTTPContext	Compute
AggReqWhseA	MQOutput
WriteHTTPContext	MQOutput
AggReqWhseB	MQOutput
AggReqWhseC	MQOutput
HTTPContext	AggregateRequest
WhseA	AggregateRequest
WhseB	AggregateRequest
WhseC	AggregateRequest

Node name	Node type
Replies	MQInput
AggregateReply	AggregateReply
ConsolidateResponse	Compute
HTTP Reply	HTTPReply

Figure 9-14 shows the directed graph representation depicting the relative position of each of the nodes in the overall message flow.

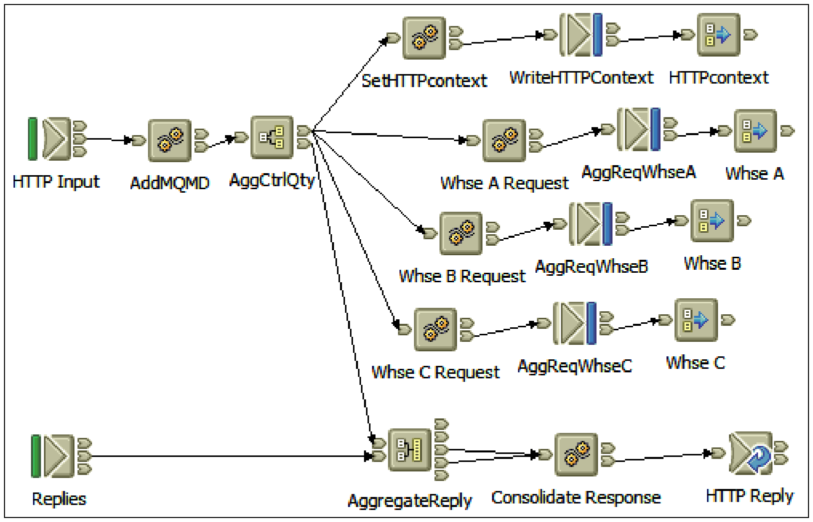


Figure 9-14 AggQtyMsgFlow message flow

In the following sections we consider the implementation of each message flow node in turn.

HTTPInput : HTTP Input

As described for the two previously documented message flows, the initial HTTP Input node is used to register this message flow for a given URI pattern. Figure 9-12 shows the configuration properties for the AggQtyMsgFlow HTTP Input node.

Table 9-12 Configuration property values for AggQtyMsgFlow HTTPInput node

Category	Name	Value
Basic	URL Selector	/ShipGoods
Basic	Maximum client wait time	30

Compute : AddMQMD

Because the request will be forwarded to multiple concurrent endpoints over JMS, we must ensure that the message format meets the requirements of WebSphere MQ and JMS. The ESQL code presented in Example 9-7 is responsible for adding the requisite MQMD and JMS headers. Note that we dynamically specify the reply-to queue to which the JMS Web service implementation should return its response in the JMS RFH2 header Rto field.

The last steps are to save the HTTP context of the service requestor and to suppress the existing HTTP header.

Example 9-7 ESQL code to prepare message for forwarding over JMS

```
CALL CopyEntireMessage();

-- Add the MQMD header for MQPUT in the correct position
CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN 'MQMD';
SET OutputRoot.MQMD.Version  = MQMD_CURRENT_VERSION;
SET OutputRoot.MQMD.StrucId  = MQMD_STRUC_ID;

-- Add the MQRFH2 header to hold the jms folder
-- Set the response queue, target service and message content type
CREATE NEXTSIBLING OF OutputRoot.MQMD DOMAIN 'MQRFH2';
SET OutputRoot.MQRFH2.mcd.Msd = 'jms_bytes';
SET OutputRoot.MQRFH2.jms.Dst = 'queue:///WarehouseQ';
SET OutputRoot.MQRFH2.jms.Rto = 'queue://SOA.QUEUE.MANAGER/WSHIPRESPONSE';
SET OutputRoot.MQRFH2.usr.targetService = 'Warehouse';
SET OutputRoot.MQRFH2.usr.contentType = 'text/xml; charset=utf-8';

-- Save the http message context for later retrieval
SET Environment.Variables.context =
CAST(InputLocalEnvironment.Destination.HTTP.RequestIdentifier AS CHARACTER);

-- Clear the http header
SET OutputRoot.HTTPInputHeader = NULL;
```

AggregateControl : AggCtrlQty

WebSphere Business Integration Message Broker provides out-of-the-box support for concurrently executing multiple endpoint invocations via the AggregateControl node. We use an instance of this node type in our scenario to fan out the concurrent invocation of multiple warehouses. A name is specified for the AggregateControl instance, which is then used subsequently on a corresponding AggregateReply node. Responses that are received after the time interval that is specified in the Timeout value will not be included in the aggregated response. However, additional capability could be incorporated into the message flow to cope with this eventuality.

WebSphere Business Integration Message Broker implicitly partitions the overall flow (presented in Figure 9-14 on page 248) into two separate flows: The first terminates after the Aggregate Request nodes, and the second flow commences upon receipt of the response messages at the Replies MQInput node.

To successfully return an HTTP response to the originating client requestor, the second flow needs HTTP context information to insert into the HTTP response message. There are several ways to ensure that this information is available to the second flow. In this example we chose to forward the data directly to the AggregateReply node using the built-in Aggregation capability.

In fact, taking this approach in our implementation enables easy separation of the two parts of the flow into two discrete message flows.

Table 9-13 shows configuration property values for the AggCtrlQty node.

Table 9-13 Configuration property values for AggCtrlQty node

Category	Name	Value
Basic	Aggregate Name	StockShip
Basic	Timeout	22

Compute : SetHTTPContext

In this Compute node, we create a new message body containing the HTTP context so that we can use it in the secondary message flow to return the response message to the original service requestor in the HTTP Reply node.

Example 9-8 SetHTTPContext node ESQL code

```
CALL CopyMessageHeaders();

-- Create the body of the message.
CREATE NEXTSIBLING OF OutputRoot.MQRFH2 DOMAIN 'XMLNS';

-- Create one element in the body which carries the HTTP context.
SET OutputRoot.XMLNS.HTTPcontext = Environment.Variables.context;
```

Compute : Whse A Request

In our implementation, we have configured our message flow to spawn concurrent execution activity through WebSphere MQ messages for each preconfigured warehouse endpoint service. A copy of the in-flight message structure that contains details of all items in the order is forwarded to each execution thread. Each execution thread must determine which of the items within the order are stocked by the warehouse to which this thread corresponds. Note that in our scenario a given item is only stocked by a single warehouse.

The incoming SOAP header makes use of XML schema namespaces, so we need to namespace-qualify any reference to contained attributes. As a convenience we can declare namespace instances in ESQL for subsequent use. Example 9-9 shows an example of ESQL namespace declarations.

Example 9-9 Required namespace declaration in Whse A Request

```
DECLARE SoapNS NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
```

We must then determine how many items have been ordered.

Example 9-10 Determining how many parts have been ordered

```
DECLARE CRD INTEGER CARDINALITY(InputRoot.XMLNS.SoapNS:Envelope. ||
   SoapNS:Body.*:ShipGoods.*:ItemList.*:Item[]);
```

Next, we must iterate through each item and retrieve the identity of the warehouse that stocks this part from an external database table. If the warehouse identity value is not equal to `‘A’` we delete the item entry from the message. Example 9-11 shows the ESQL code used to achieve this.

Example 9-11 Checking Warehouse A for items

```
DECLARE I INTEGER 1;
DECLARE J INTEGER 1;

CALL CopyEntireMessage();

LP : WHILE I <= CRD DO
   SET WHS = THE (SELECT ITEM P.WAREHOUSE FROM Database.PART_LOCATION AS P
                WHERE P.PART_NUMBER =
   InputRoot.XMLNS.SoapNS:Envelope.SoapNS:Body.*:ShipGoods.*:ItemList.*: ||
      Item[I].*:ProductNumber);
   IF WHS <> 'A' THEN
      SET OutputRoot.XMLNS.SoapNS:Envelope.SoapNS:Body.*:ShipGoods.* ||
         :ItemList.*:Item[J].*:ProductNumber = NULL;
      SET OutputRoot.XMLNS.SoapNS:Envelope.SoapNS:Body.*:ShipGoods.* ||
         :ItemList.*:Item[J].*:Quantity = NULL;
      SET OutputRoot.XMLNS.SoapNS:Envelope.SoapNS:Body.*:ShipGoods.*||
         :ItemList.*:Item[J]= NULL;
   ELSE
      SET J = J+1;
   END IF;
   SET I = I+1;
END WHILE LP;
```

Finally, we handle the case in which none of the items is stocked by Warehouse A, by suppressing the service call.

Example 9-12 Handle item not found in warehouse case

```
-- If none of the parts requested are in Warehouse A then
-- do not send MQ message
IF (J=1) THEN
   RETURN FALSE;
END IF;
RETURN TRUE;
```

Figure 9-14 provides non-default configuration property values for the Whse A Request Compute node.

Table 9-14 Configuration property values for Whse A Request node

Category	Name	Value
Basic	Data Source	BROKERDB

The behavior of Compute nodes Whse B Request and Whse C Request is similar to that described for Whse A Request above, but is related to the checking for items in Warehouse B and Warehouse C respectively. However we amend the targetService set previous in the MQRFH2 header as shown in Example 9-13.

Example 9-13 Resetting the targetService

```
-- Override destination and target service operation name
SET OutputRoot.MQRFH2.jms.Dst = 'queue:///WarehouseBQ';
SET OutputRoot.MQRFH2.usr.targetService = 'WarehouseB';
```

MQOutput : WriteHTTPContext

This node is used to write a message containing the HTTP context information that must be made available to the secondary message flow. The properties in Table 9-15 on page 252 show that the message will be put on to the SendHTTPContext queue. This message will be picked up by a small flow described after this flow, “BounceHTTPContextMsgFlow” on page 258. The reply-to information is used by this flow to put the message straight on to the AggregateReply input queue.

Table 9-15 Configuration property values for WriteHTTPContext node

Category	Name	Value
Basic	Queue Manager Name	WBRK_QM
Basic	Queue Name	SendHTTPContext

Category	Name	Value
Advanced	New Message ID	Checked
Advanced	Message Context	Default
Request	Request	Checked
Request	Reply-to queue manager	WBRK_QM
Request	Reply-to queue	WSHIPRESPONSE

MQOutput : AggReqWhseA, AggReqWhseB, AggReqWhseC

Each concurrent thread of execution then has its own MQOutput node as the means to forward the request JMS message to its outbound destination. The outbound Queue that corresponds to each warehouse is given by the warehouse-specific *Queue Name* value.

The configuration property values shown in Table 9-16 are similar for each of the three MQOutput nodes AggReqWhseA, AggReqWhseB, and AggReqWhseC except where indicated.

Table 9-16 MQOutput node property values for AggQtyMsgFlow

Category	Name	Value
Basic	Queue Manager Name	WBRK_QM
Basic (Warehouse A)	Queue Name	WASHIPREQUEST
Basic (Warehouse B)	Queue Name	WBSHIPREQUEST
Basic (Warehouse C)	Queue Name	WCSHIPREQUEST

AggregateRequest : Whse A, Whse B, Whse C, HTTPContext

Each outbound concurrent thread of execution is terminated by an instance of the AggregateRequest Node. Note that the Folder Name property specifies the folder into which each individual Warehouse response will be written within the body of the aggregated response message structure.

Table 9-17 Aggregate Request node configuration property values

Node	Category	Name	Value
HTTPcontext	Basic	Folder Name	HTTPcontext
Whse A	Basic	Folder Name	WhseA
Whse B	Basic	Folder Name	WhseB

Node	Category	Name	Value
Whse C	Basic	Folder Name	WhseC

MQInput : Replies

In the description of the AddMQMD Compute node, we saw how the Rto JMS header field is used to specify the JMS destination to which each JMS service should send its response. In this example, we specify the same queue destination for each of the three Warehouse services. The Replies MQInput node represents the Reply-to-queue and behaves as the input vehicle for each of the service response messages.

Table 9-18 summarizes the relevant non-default configuration property values for this node.

Table 9-18 Replies MQInput node

Category	Name	Value
Basic	Queue Name	WSHIPRESPONSE
Default	Message Domain	XMLNS

AggregateReply : AggregateReply

The AggregateReply node is responsible for merging multiple concurrent threads of execution and is a direct counterpart to the AggregateControl node discussed earlier in “AggregateControl : AggCtrlQty” on page 249. This node accomplishes aggregation of the service response messages.

Table 9-19 on page 254 summarizes the relevant non-default configuration property values for this node.

Note that if the aggregation timeout is reached before all of the responses have been received, then the message is propagated to the Timeout terminal of the this node. In this message flow, the Timeout terminal is connected to the Compute : ConsolidateResponse node for onward processing. This means that if a Warehouse fails to respond in the defined time, then the result that is returned to the application user shows the ordered parts that are held in that Warehouse as being out of stock. (The remainder of the order is processed as normal.)

Table 9-19 Aggregate Reply configuration property values

Category	Name	Value
Basic	Aggregate Name	StockShip

Compute : ConsolidateResponse

The ConsolidateResponse Compute node is responsible for transforming the aggregated response as forwarded by the AggregateReply node into a format that is acceptable to the client requestor. The ESQL code that performs this transformation is explained step-by-step.

1. We first declare the namespaces and references that we will use subsequently in the ESQL module (Example 9-14).

Example 9-14 XML namespace declarations

```
DECLARE SoapNS NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
DECLARE ns2 NAMESPACE 'http://www.ws-i.org/SampleApplications/ ||
   SupplyChainManagement/2002-08/Warehouse.xsd';
```

2. We then create the SOAP envelope structure in the Output message structure (Example 9-15).

Example 9-15 Output message SOAP envelope definition

```
CREATE FIELD OutputRoot.XMLNS.SoapNS:Envelope;
```

3. The next ESQL code fragment checks whether the aggregated response that is forwarded by the AggregateReply node contains information returned by Warehouse A. If it does, we copy the information into the output message structure parts (Example 9-16).

Example 9-16 Copy response from warehouse A into output message structure

```
DECLARE C INTEGER 0;
DECLARE HDR BOOLEAN FALSE;

IF (CARDINALITY(InputRoot.ComIbmAggregateReplyBody.WhseA[]) <> 0) THEN
-- If parts were ordered from warehouse A the copy its response into the
-- output message
   SET OutputRoot.XMLNS.*:Envelope = InputRoot.ComIbmAggregateReplyBody. ||
      WhseA.XMLNS.*:Envelope;
   SET C = CARDINALITY(OutputRoot.XMLNS.*:Envelope.*:Body.*: ||
      ShipGoodsResponse. *:Response.*:ItemStatus[]);
   SET HDR = TRUE;
END IF;
```

4. Now we must determine whether the aggregated response input message structure contains information returned by Warehouse B. If it does, we copy it across to the output message structure, making sure that the information is

merged with any SOAP envelope information from Warehouse A if necessary.

In Example 9-17 we check for a response from Warehouse B, then determine whether required header information has been written during prior processing of Warehouse A data, and if no header has yet been defined we add it here.

Example 9-17 Checking for Warehouse B return data

```
IF (CARDINALITY(InputRoot.ComIbmAggregateReplyBody.WhseB[]) <> 0) THEN
   IF (NOT HDR) THEN
      -- If parts were ordered from warehouse B but not from warehouse A
      -- then copy its response into the output message
      SET OutputRoot.XMLNS.*:Envelope = InputRoot.ComIbmAggregateReplyBody. ||
         WhseB.XMLNS.*:Envelope;
      SET C = CARDINALITY(OutputRoot.XMLNS.*:Envelope.*:Body. ||
         *:ShipGoodsResponse.*:Response.*:ItemStatus[]);
      SET HDR = TRUE;
ELSE
```

5. If Warehouse A did return a response so that the SOAP header information already exists in our output message, then we follow the ELSE path to incorporate the response from Warehouse B into the aggregated response message structure. This is shown in Example 9-18 and Example 9-19 on page 257.

   We first perform the response check and, if necessary, define a message substructure to hold the data.

Example 9-18 Add Warehouse B message structure

```
DECLARE CRDB INTEGER CARDINALITY(InputRoot.ComIbmAggregateReplyBody. ||
   WhseB.XMLNS.*:Envelope.*:Body.*:ShipGoodsResponse.*:Response. ||
   *:ItemStatus[]);
DECLARE I INTEGER 1;
DECLARE TMP INTEGER 0;

LPB : WHILE I <= CRDB DO
   SET C = C + 1;
   CREATE FIELD OutputRoot.XMLNS.*:Envelope.*:Body.*:ShipGoodsResponse.||
      *:Response.*:ItemStatus[C] TYPE Name NAMESPACE ns2;
   SET OutputRoot.XMLNS.*:Envelope.*:Body.*:ShipGoodsResponse.*:Response. ||
      *:ItemStatus[C].(XML.NamespaceDecl)xmlns = ns2;
```

6. Next, we create fields for the individual response elements and copy their values across into the response message structure.

Example 9-19 Copy response fields into output message structure

```
   CREATE LASTCHILD OF OutputRoot.XMLNS.*:Envelope.*:Body.*: ||
      ShipGoodsResponse.*:Response.*:ItemStatus[C] DOMAIN('XMLNS') ||
      NAMESPACE ns2 NAME 'ProductNumber';

   SET OutputRoot.XMLNS.*:Envelope.*:Body.*:ShipGoodsResponse. ||
      *:Response.*:ItemStatus[C].ns2:ProductNumber = InputRoot. ||
      ComIbmAggregateReplyBody.WhseB .XMLNS.*:Envelope.*:Body. ||
      *:ShipGoodsResponse.*:Response.*:ItemStatus[I].*:ProductNumber;

   CREATE LASTCHILD OF OutputRoot.XMLNS.*:Envelope.*:Body.*: ||
      ShipGoodsResponse.*:Response.*:ItemStatus[C] DOMAIN('XMLNS') ||
      NAMESPACE ns2 NAME 'Status' VALUE 2;

   SET OutputRoot.XMLNS.*:Envelope.*:Body.*:ShipGoodsResponse.*: ||
      Response.*:ItemStatus[C].ns2:Status = InputRoot. ||
      ComIbmAggregateReplyBody.WhseB.XMLNS.*:Envelope.*:Body. ||
      *:ShipGoodsResponse.*:Response.*:ItemStatus[I].*:Status;

END WHILE LPB;
```

The remaining code in this node is responsible for copying across any response data from Warehouse C into the output message structure. The coding constructs that are used are similar in principle to those that are described for the manipulation of response data from Warehouses A and B, so for the sake of brevity we shall not explicitly present the code listing here. Interested readers can download and view the code package provided as an accompaniment to this IBM Redbook.

7. We extract the HTTP Context information from the incoming aggregated message tree structure and copy it into the response message for return to the client requestor (Example 9-20).

Example 9-20 Consolidate Response node: restoring HTTP context

```
IF (CARDINALITY(InputRoot.ComIbmAggregateReplyBody.HTTPcontext[]) <> 0) THEN
   SET OutputLocalEnvironment.Destination.HTTP.RequestIdentifier =
      CAST(InputRoot.ComIbmAggregateReplyBody.HTTPcontext.XMLNS.HTTPcontext
            AS BLOB);
...
END IF;
```

In Example 9-20 on page 257 we set a value in the Local Environment. This means that we have to change property value for this node: in the pull-down list, the Compute Mode must be set to `LocalEnvironment And Message`.

HTTPReply : HTTP Reply

Finally, we return the aggregated message, potentially containing responses from differing warehouse services to the requesting client.

BounceHTTPContextMsgFlow

The aggregation message flow AggQtyMsgFlow uses a small additional flow to enable the message that carries the HTTP context to pass between the first part of the flow and the second part. It is a trivial flow, which picks up a message from a queue and replies by putting the same content onto the reply-to queue. This reply-to queue is the same queue to which the aggregation replies are sent from the warehouses, WSHIPRESPONSE.

There are two nodes in this flow:

- MQInput
- MQOutput

Figure 9-15 shows the directed graph representation that depicts the relative position of these two nodes in the message flow.

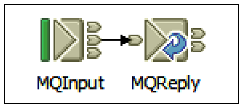


Figure 9-15 BounceHTTPContextMsgFlow message flow

We now look at the implementation of each of these nodes in turn.

MQInput

The MQInput waits for a message to be received on a specified WebSphere MQ queue. This queue must be defined as a local queue on the WebSphere Business Integration Message Broker's queue manager. When received, the node performs an MQ get operation to read the message from the queue. The configuration properties for this node are shown in Table 9-20.

Table 9-20 Configuration properties for BounceHTTPContextMsgFlow MQInput node

Category	Name	Value
Basic	Queue Name	SendHTTPContext
Default	Message Domain	XMLNS

MQReply

No configuration properties were changed from the default settings for this node. The reply-to queue is specified in the header of the message put on the queue

SendHTTPContext by the message flow AggQtyMsgFlow. See “MQOutput : WriteHTTPContext” on page 252.

Note: Information about how to use aggregation to consume Web services using SOAP over HTTP can be found in the redbook *Patterns: Broker Interactions for Intra- and Inter-enterprise*, SG24-6075.

9.4 Runtime guidelines

In this section, we present a discussion of the steps for restructuring our example Supply Chain implementation in order to incorporate a Broker component into the deployment infrastructure. At a high level, the starting point application infrastructure for this chapter essentially comprises a series of point-to-point interactions, as depicted in Figure 9-16. The discussion as presented here assumes that the configuration that is required to support the starting point scenario has already been defined. For example, the WebSphere Application Server JMS resource definitions that facilitate JMS Web service invocations are considered to be a prerequisite for the starting point configuration, and as such are not explicitly described in this chapter.

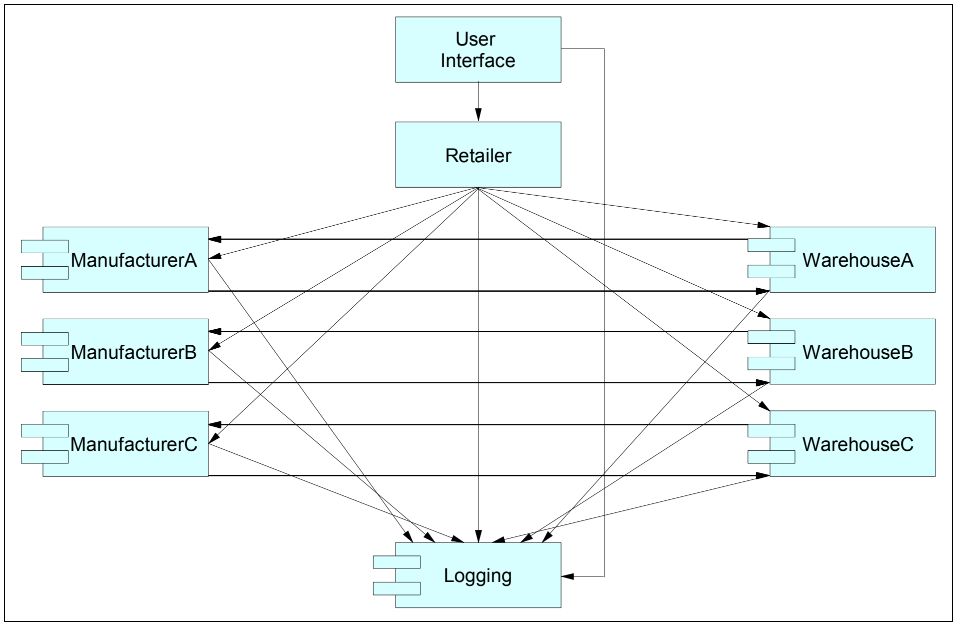


Figure 9-16 Scenario implemented using Direct Connection pattern

A major benefit of our restructuring to incorporate a Broker component is the rationalization of multiple point-to-point interactions into a logical hub-and-spoke architecture. This alternative architectural approach is shown in Figure 9-17.

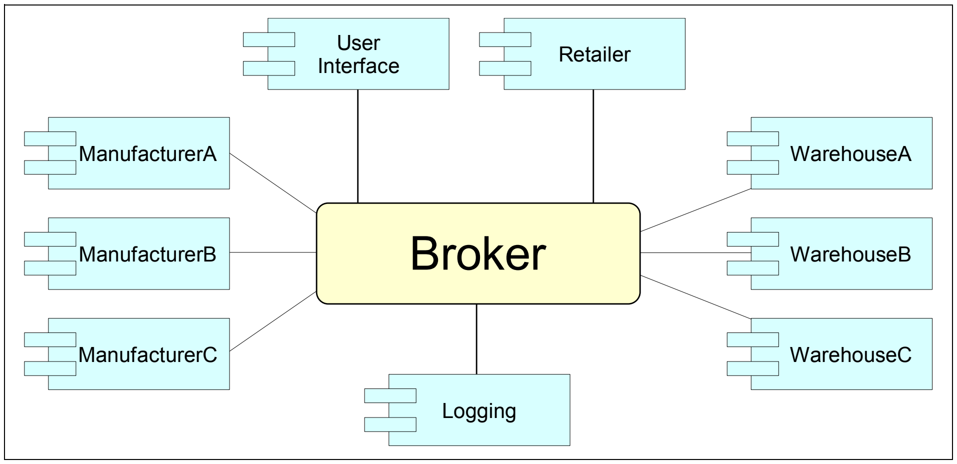


Figure 9-17 Scenario implemented using Broker pattern

In 9.3, “Development guidelines” on page 236, we saw how the interactions that comprise our overall scenario can be categorized into three types. We have chosen to handle each invocation type in a separate message flow, as summarized in Table 9-1 on page 237. We now examine the configuration steps that are required to replace the point-to-point interactions for each invocation type with the Broker pattern alternative.

9.4.1 Incorporation of Broker

To introduce the message flows that we have developed to handle each of our three service invocation types, we introduce a level of indirection into the service client to service provider interaction. Essentially this involves the modification of the WSDL that is used by each service client, replacing the service endpoint address in each case with that of WebSphere Business Integration Message Broker. After modifying each client WSDL file, it is necessary to reinstall the containing Enterprise Application into WebSphere Application Server, including the regeneration of Web services bindings.

Table 9-21 on page 261 shows the message flows that each Web service interaction uses.

Table 9-21 Message flow requirements for the scenario

Service requester	Operation	Service provider	Broker message flow requirement
SCM application	getCatalog	Retailer	HTTP routing
SCM application	submitOrder	Retailer	HTTP routing
Retailer	shipGoods	Warehouses	HTTP to JMS aggregation
Warehouse	submitPO	Manufacturer	HTTP routing
Manufacturer	submitSN	Warehouse	HTTP routing
SCM application	getEvents	LoggingFacility	HTTP routing
Retailer	logEvent	LoggingFacility	HTTP to JMS protocol conversion
Warehouse	logEvent	LoggingFacility	HTTP to JMS protocol conversion
Manufacturer	logEvent	LoggingFacility	HTTP to JMS protocol conversion

The steps that must be followed in order to achieve the required level of indirection are broadly similar, irrespective of the underlying transport protocol that is used for the service invocation. In the remainder of this section, we step through the process for the modification of the interaction between service client SCMSampleUI and service provider LoggingFacility. Note that these steps must be repeated for every serial HTTP service client.

Modifying WSDL definition files

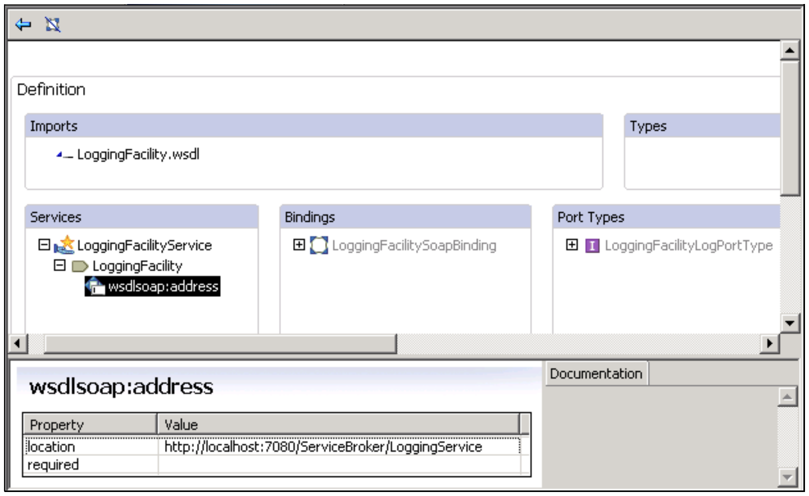
Open the service client WSDL file in WebSphere Studio Application Developer WSDL editor and navigate to the `soap:address` field. Recall that the URL pattern specified for the URL Selector property of the HTTP Input node of message flow HTTPtoHTTPMsgFlow was:

```
http://localhost:7080/ServiceBroker/*
```

Logic within the message flow will use the value concatenated with this URL stem as the key used on the retrieval of the service endpoint address from the externalized endpoint database. We chose to concatenate the service name to the URL stem in order to form the complete URL that will be used by the service client, giving the following value:

```
http://localhost:7080/ServiceBroker/LoggingFacility
```

This is shown in Figure 9-18 on page 262.



wsdlsoap:address

Property	Value
location	http://localhost:7080/ServiceBroker/LoggingService
required	

Figure 9-18 Changing service endpoint address used by serial HTTP client

Redeploying enterprise applications

After making the changes to the WSDL file used by the service client that were discussed in the previous section, we must redeploy the enterprise application:

1. Export the enterprise application as an EAR file from WebSphere Studio Application Developer.
2. From the WebSphere Application Server administration console, stop and uninstall any prior versions of the SCMSampleUI application that you might have installed in your environment.

You can now install the new enterprise application version from the admin console. We have changed the endpoint service address, so the Web services bindings must be regenerated by selecting the Deploy Web services check box on Step 1. (Alternatively, the client bindings can be generated in WebSphere Studio Application Developer.)

Additional configuration is required for the Warehouse service:

1. Export the Warehouse.ear file from WebSphere Studio Application Developer.
2. Regenerate the endpoints.

   This can be done directly from WebSphere Studio Application Developer or, as follows, on the command line from the directory where the file Warehouse.jar is located:

   ```
   <Studio_home>\v5.1.1\runtimes\base_v51\bin\endptenabler.bat
   ```

You are then prompted for the name of the EAR file, which is entered as:

```
Warehouse.ear
```

Take the default value for the HTTP router name. The HTTP context root should be supplied as follows:

```
/Warehouse
```

3. Deploy the Warehouse enterprise application.

   The following settings are required for the redeployment of the enterprise archive:

   a. Select the **Regenerate bindings** check box prior to Step 1.
   b. Select the **Deploy Web services** check box on Step 1.

9.4.2 Distributed WebSphere MQ configuration

Although the scenario that is described in this chapter was developed on a single machine, the individual services could be distributed across multiple machines in the future. For this reason, we have chosen to maintain some degree of isolation between the WebSphere MQ resources referenced by the runtime artefacts deployed within WebSphere Business Integration Message Broker and WebSphere Application Server. A WebSphere MQ queue manager is created during the installation of WebSphere Business Integration Message Broker, with a default name of WBRK_QM. Our definition of message flow AggQtyMsgFlow includes references to queues that are defined within its default queue manager WBRK_QM.

In a similar way, WebSphere MQ JMS resources that are defined within WebSphere Application Server reference queues that are defined within a dedicated queue manager, SOA.QUEUE.MANAGER. Because the queues within the respective queue managers are the foundation of the transport mechanism that is used to pass messages between our message flows and JMS services, we must define distributed WebSphere MQ resources to establish connectivity. This distributed WebSphere MQ queueing scheme is illustrated in Figure 9-19 on page 264. A more detailed description of the definition of the required WebSphere MQ objects is provided in Appendix B, “Configuring the scenario lab environment” on page 335.

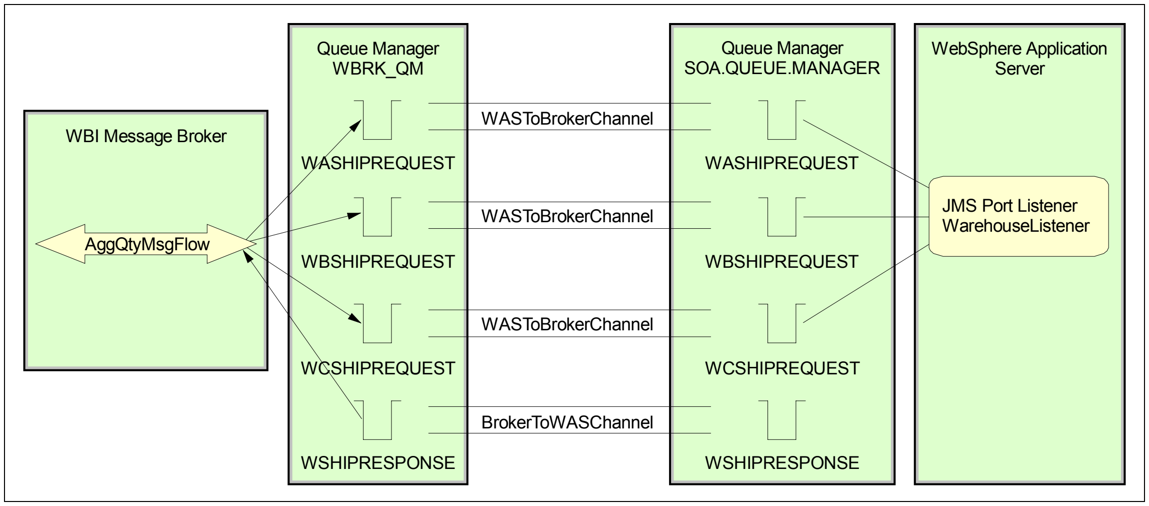


Figure 9-19 Distributed WebSphere MQ scheme

9.4.3 Externalized data definition

Our message flows consume externalized configuration data to determine service endpoint addresses and the mapping of specific items to the appropriate warehouse. We chose to use DB2 UDB as the data management technology, and we defined two tables to contain the values.

Table 9-22 shows a sample of the service endpoint routing data that is defined within the `SERVICE_ROUTER` table.

Table 9-22 Sample data from SERVICE_ROUTER table

Name	**Location**
Manufacturer	http://appsrv1l.itso.ral.ibm.com:9080/Manufacturer/services/Manufacturer
LoggingFacility	http://appsrv1l.itso.ral.ibm.com:9080/LoggingFacility/services/LoggingFacility
WarehouseCallBack	http://appsrv1l.itso.ral.ibm.com:9080/WarehouseCallBack/services/WarehouseCallBack
Retailer	http://appsrv1l.itso.ral.ibm.com:9080/Retailer/services/Retailer

The values in Table 9-23 on page 265 are used to determine the Warehouse corresponding to any given Item. These are held within the PART_LOCATION table in DB2 UDB. The primary key is made up of both columns because of the requirement that parts are only stocked in one warehouse.

Table 9-23 Data held in PART_LOCATION table

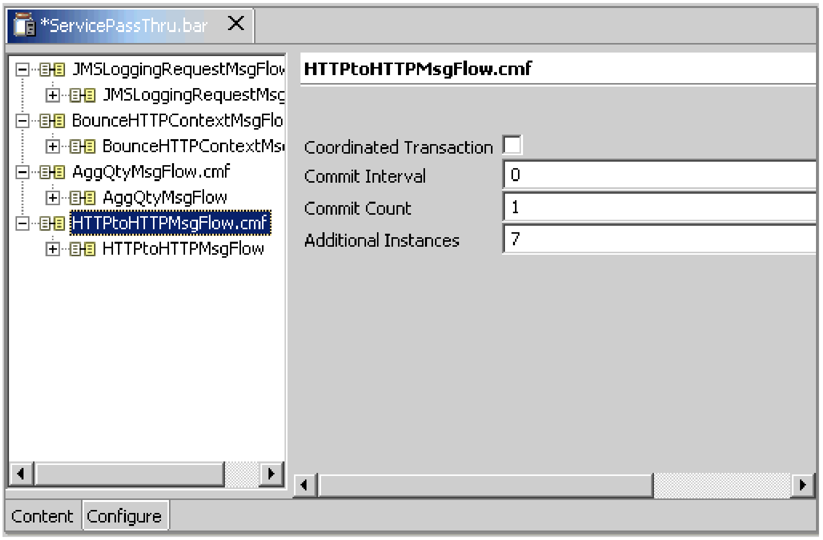
PART_NUMBER	Warehouse
605001	A
605002	B
605003	C
605004	A
605005	B
605006	C
605007	A
605008	B
605009	C

9.4.4 Message flow deployment

Administration of WebSphere Business Integration Message Broker is shipped with an Eclipse-based development and administration Toolkit. Message flows are packaged into Broker archives for deployment. Within a running Broker instance, Broker archives are deployed to an execution group. Broker archives can be deployed readily into an execution group via a simple drag-and-drop paradigm. Further details about how to package and deploy message flows can be found in “Deploy the message flow” on page 248 of *Patterns: Broker Interactions for Intra- and Inter-enterprise*, SG24-6075.

Broker archives (and, therefore, message flows) can be deployed to more than one execution group and more than one Broker to meet scalability and availability requirements. Our scenario has one restriction to this: a flow containing an AggregateReply node must not be deployed to multiple execution groups in a Broker; multiple instances of the flow are allowed. By splitting AggQtyMsgFlow into two message flows, the first part can be deployed to multiple execution groups. In this case, the AggregateControl message would be passed between the flows using WebSphere MQ.

To run the full scenario, multiple instances of the HTTPtoHTTPMsgFlow message flow are required concurrently. To enable more than one instance of a flow to be executed at once in an execution group, a configuration setting is required on the message flow in the Broker archive. The Additional Instances property should be set to 7 as shown in Figure 9-20 on page 266.



HTTPtoHTTPMsgFlow.cmf

Coordinated Transaction	☐
Commit Interval	0
Commit Count	1
Additional Instances	7

Figure 9-20 Setting the number of instances of a message flow

9.4.5 Troubleshooting message flows

Two main mechanisms are available for troubleshooting WebSphere Business Integration Message Broker message flows:

- The Flow Debug perspective in the Message Brokers Toolkit

  This is a visual debugging environment. You can set breakpoints in a flow and then step through the flow. While you are stepping, you can examine and change the message and variables in ESQL code or Java code. These capabilities enable you to debug a wide variety of error conditions in flows, such as:

  – Incorrectly wired activities (for example, outputs that are connected to the wrong inputs)

  – Incorrect conditional branching in transition conditions

  – Unintended infinite loops in flows

- Tracing an execution group in a Broker

  To trace the execution of message flows in an execution group:

  a. Enter the following command in a command window before putting messages onto the input queue or making an HTTP request:

```
mqsichangetrace <Broker> -u -e <execution group> -l debug -r
```

b. Put the message onto the input queue or make the HTTP request. After the message flow has completed execution, you can read the trace log and format it to show the result by using the following commands:

```
mqsireadlog <Broker> -u -e <execution group> -o trace.xml
mqsiformatlog -i trace.xml -o formattrace.log
mqsichangetrace <Broker> -u -e <execution group> -l none
```

The formatted trace is in the formattrace.log text file.

This sequence of commands can be used to create a .bat file, which is commonly known as usertrace.

A useful utility for placing messages on WebSphere MQ queues for message flow execution and reading the response messages from queues is provided as part of IBM SupportPac IH03. It includes the rfhutil utility and can be found at:

http://www.ibm.com/software/integration/support/supportpacs/

9.4.6 Quality of service capabilities

For an ESB implementation, the quality of service capabilities is greatly affected by the qualities of the environment and of the service providers. The following discussion focuses on the quality of service implications when using WebSphere Business Integration Message Broker.

Availability

Installing a WebSphere Business Integration Message Broker default configuration of one Broker with a single execution group leads to a single point of failure. This is easily remedied by deploying two Brokers on similar machines with a WebSphere MQ cluster configuration. Where connection is by IP address, an IP switching facility is required.

This deployment approach is in line with the attributes of an ESB implementation because it provides a physically distributed infrastructure, which has a central configuration.

Performance

Performance and capacity planning reports are available for the WebSphere Business Integration Message Broker in the form of IBM SupportPacs at:

http://www.ibm.com/software/integration/support/supportpacs/

Security

Security provides confidentiality and non-repudiation by authenticating the involved parties, encrypting messages, and providing access control. Security has additional importance if the activity occurs over the Internet. The service provider can have different approaches and levels of providing security, depending on the service requestor.

SSL is available for the Internet and WebSphere MQ protocols. However, SSL support is not provided by the HTTP nodes in WebSphere Business Integration Message Broker, so a Web server or the Web Services Gateway is needed. Where encryption is required directly to the Broker, JMS can be used with WebSphere MQ.

WebSphere Business Integration Message Broker does not provide WS-Security support. However, the overall infrastructure can provide this support by deploying Web Services Gateway.

Security requirements are broader when considering publish and subscribe. A finer granularity of security exists in WebSphere Business Integration Message Broker when using publish and subscribe.

Transactionality

WebSphere Business Integration Message Broker provides transactional support via WebSphere MQ (and JMS with WebSphere MQ) or supported databases.

For external activity, the message flow has to enlist in a global transaction such that all required resources are coordinated. If a message flow includes interaction with an external user database, the message flow can be configured such that all of its processing is coordinated within a transaction. This ensures that either all processing is successfully completed or no processing is completed; thus all affected resources (queues, databases, and so on) can maintain or return to a consistent state, and data integrity is preserved.

If an error is generated in a message flow node, the default behavior that will be taken by the Broker depends on how you have created and connected the message flow, and whether the message that is being processed is under transactional control.

9.5 Further information

- *Developing Solutions in WebSphere MQ Integrator*, SG24-6579
- *WebSphere MQ Integrator Deployment and Migration*, SG24-6509
- *Migration to WebSphere Business Integration Message Broker V5*, SG24-6995
- *Using Web Services for Business Integration*, SG24-6583
- *Patterns: Broker Interactions for Intra- and Inter-enterprise*, SG24-6075

- WebSphere Business Integration Message Broker SupportPacs

  `http://www.ibm.com/software/integration/support/supportpacs/`
- WebSphere Business Integration Message Broker information library

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

10

Business Service Choreography

This chapter describes the implementation of Business Service Choreography for our business scenario. In previous chapters, we have described the Enterprise Service Bus Router and Broker variations. Business Service Choreography is mainly concerned with business logic and therefore is likely to be implemented in a component outside the Enterprise Service Bus (ESB) rather than within the bus itself. This chapter primarily focuses on design and development of Business Service Choreography within the manufacturer service and its connection to the ESB.

In this chapter, the following topics are discussed:

- Sample business scenario with Business Service Choreography
- Design guidelines for designing a Business Service Choreography process
- Development guidelines describing the processes that are implemented in this chapter
- Runtime guidelines discussing the deployment and runtime options for a business process in relation to an Enterprise Service Bus

This chapter primarily focuses on developing BPEL4WS processes for WebSphere Business Integration Server Foundation.

10.1 Business scenario

As discussed in earlier chapters, the WS-I sample application is a simplified supply chain for a consumer electronics retailer. When the warehouse runs out of goods, it makes a request for additional stock from the manufacturers. The manufacturer receives the request and sends the shipment to the warehouse.

The basic infrastructure of multiple manufacturers remains the same as the previous scenario, as shown in Figure 10-1.

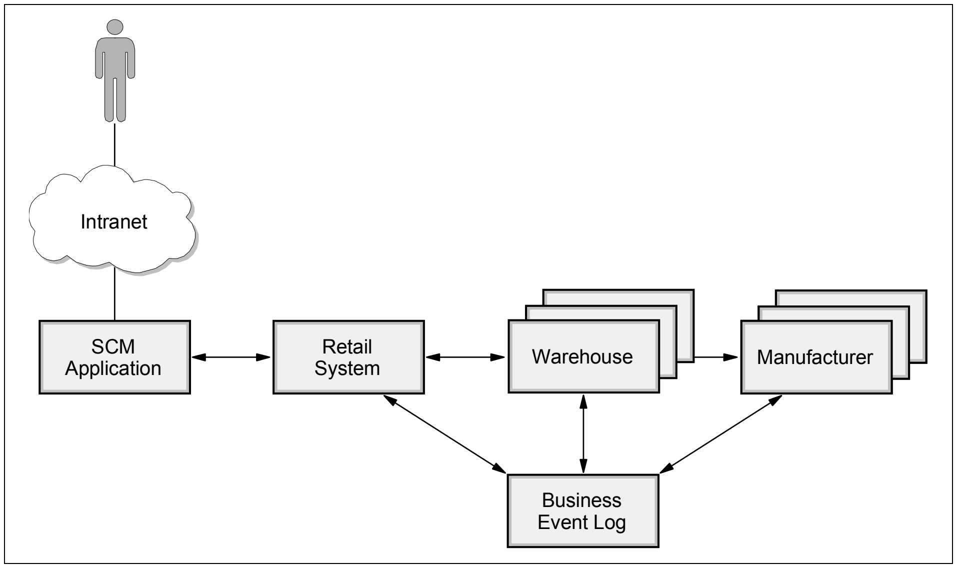


Figure 10-1 Existing infrastructure

One of the external manufacturers has decided to model the fulfillment operation as a business process. This gives the manufacturer two advantages:

- The sequence of activities within the business process can be changed easily.
- The manually executed approval activity can be incorporated with the automated activities in a single process.

In this chapter, we showcase the design and development of Business Service Choreography. We also discuss the connection of that process as a service to the existing Enterprise Service Bus within the organization.

10.2 Design guidelines

This section reviews the design considerations and critiques of the available product mappings for its implementation.

10.2.1 Design overview

This section outlines some things to think about when implementing Business Service Choreography as an off-the-ESB service.

Selecting the pattern

Typically, the business requirements drive the pattern selection; therefore we start to analyze our needs, leading to the selection of a pattern implementation for the manufacturer.

Currently, the manufacturer runs its business process through a mixture of manual and automated steps. The overall process is not explicitly modeled or documented in any great detail. As a result, it is difficult to measure business performance indicators, such as the response time for fulfilling orders, or to assess the impact of changes on those indicators. In order to improve both the measurability and flexibility of the business, the manufacturer has decided to explicitly model and automate its business processes, including a new workflow system to manage the execution of manual steps.

Specifically, the business requirements demand the following:

- Separation of business process flow logic from the individual application or function logic.
- Loose coupling between different services for quicker response and flexibility to adapt to the changing business needs, including replacing one service implementation with another.
- The inclusion of human intervention and manual steps within an automated process model.
  - A management/staff personnel intervention (or approval) to prompt the production of goods when the manufacturer is low on some requested goods.

Choosing the relevant SOA pattern

The Business Service Choreography component is suitable for this scenario. This component can use the Process Integration Serial Process application pattern to describe the interactions of a business process, as is shown in Figure 10-2 on page 274.

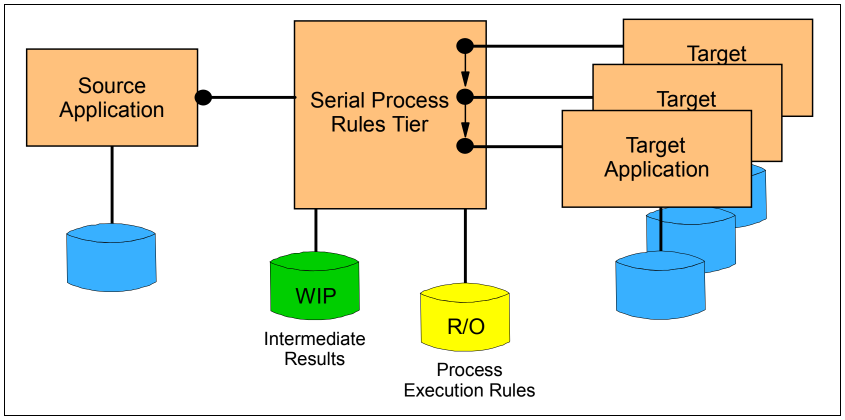


Figure 10-2 Serial Process application pattern

Where human interaction is involved (as is the case with this business scenario), the Serial Workflow variation applies. See Figure 10-3.

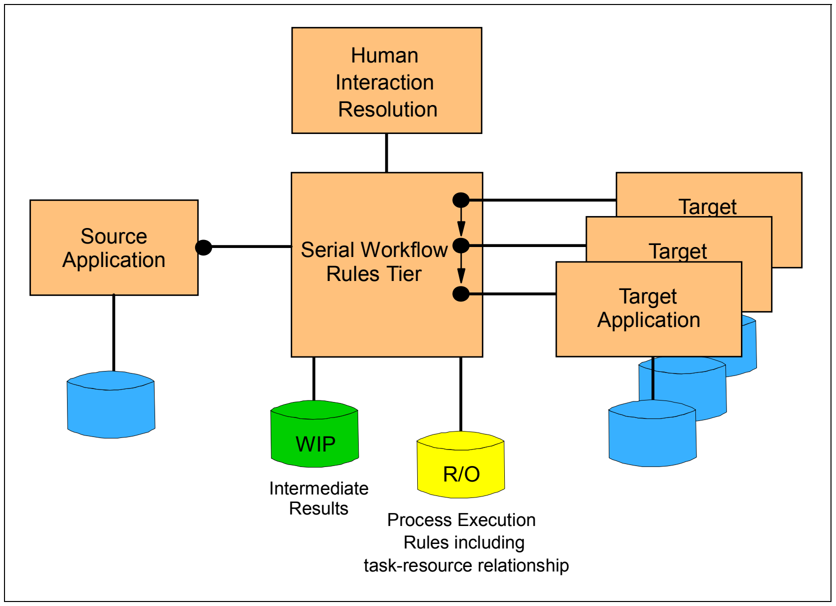

Human Interaction Resolution
Source Application
Serial Workflow Rules Tier
Target
Target
Target Application
WIP
Intermediate Results
R/O
Process Execution Rules including task-resource relationship

Figure 10-3 Serial Workflow application pattern

For more information on the Serial Process and Serial Workflow patterns, consult the redbook *Patterns: Serial and Parallel Processes for Process Choreography and Workflow*, SG24-6306.

Applying the SOA pattern to the scenario

Due to the requirement for human interaction, we have elected to use the Serial Workflow pattern to implement our Business Service Choreography process. This Business Service Choreography sits off the Enterprise Service Bus as an additional service provider (Figure 10-4).

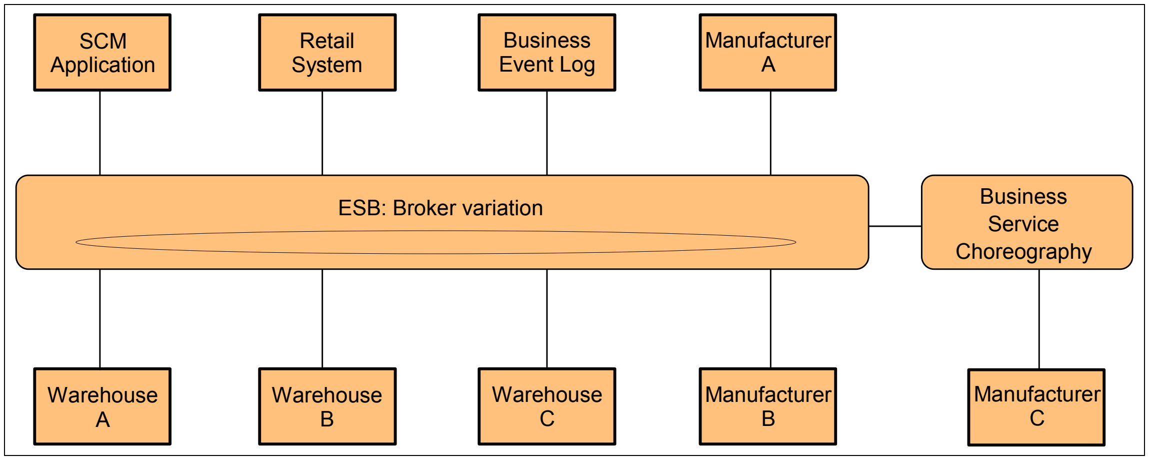


Figure 10-4 Business Service Choreography off the ESB

Business Service Choreography benefits

The key benefits of using Business Service Choreography are:

- Flexibility: achieved by externalization of process logic for the individual application.
- Human interaction: facilitating various staff functions such as approval, investigation, and claim.
- Reusability of the software components or services in the process, resulting in cost savings.
- The construction of processes from services with explicit interfaces enables the easy substitution of one service interface with another.

Product implementation options

Business Service Choreography can be implemented in a number of IBM products. We specifically considered the following:

- WebSphere Business Integration Server Foundation V5.1
- WebSphere MQ Workflow V3.5

To help with selecting the appropriate product, refer to the description of each product in 5.1, “Runtime product descriptions” on page 134.

Product selection for scenario implementation

To address the requirements described in the business scenario using Business Service Choreography, we selected the WebSphere Process Choreographer capability of WebSphere Business Integration Server Foundation. The product mapping is shown in Figure 10-5.

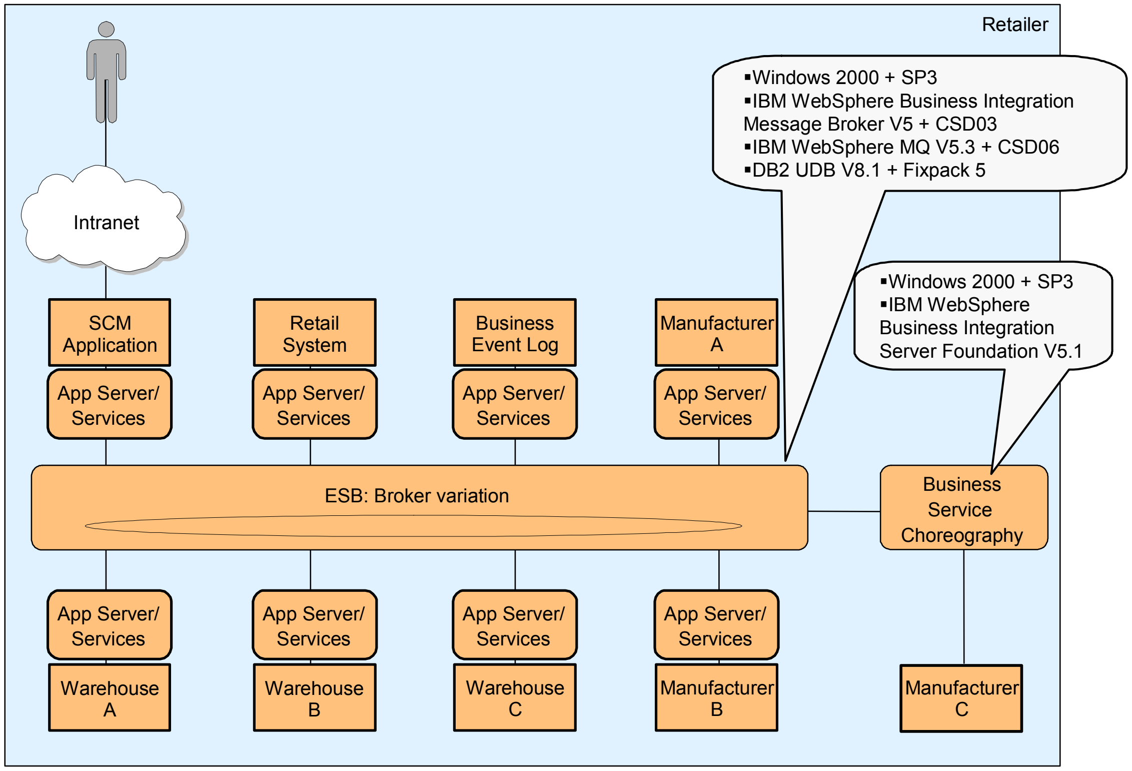


Figure 10-5 Product mapping for Business Service Choreography off the ESB

By selecting WebSphere Business Integration Server Foundation, we can take advantage of the support for the open-standard Business Process Execution for Web Services (abbreviated as BPEL4WS). The BPEL4WS V1.1 specification can be found at:

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

Figure 10-6 on page 277 illustrates a different view of our solution to the scenario requirements in terms of service requesters, service providers, and the protocols that are used for communication with the ESB.

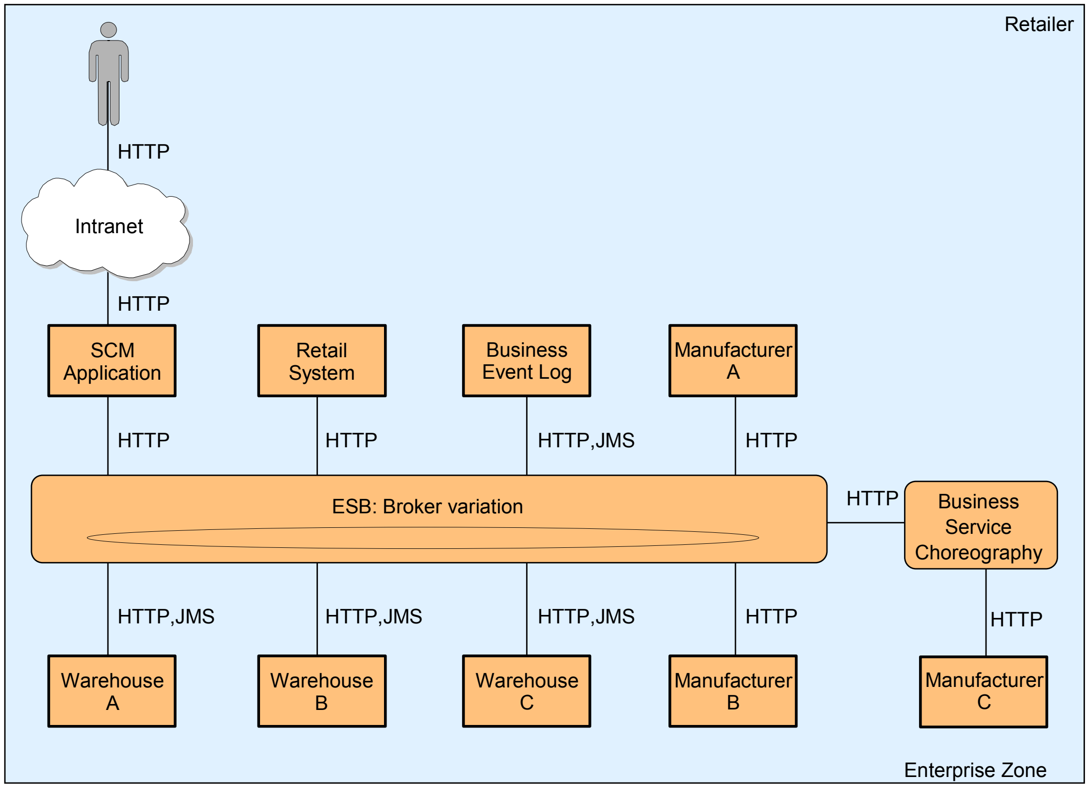


Figure 10-6 Business Service Choreography-based manufacturer off the ESB

Business Service Choreography design

We now look at the different design approaches that we can take to implement a solution based in Business Service Choreography. Business Service Choreography incorporates the business logic of the enterprise, thus is implemented off the bus.

This section includes:

- Solution overview
- Design alternatives

Solution overview

The Business Service Choreography process flow for the manufacturer consists of the following activities (Figure 10-7 on page 278):

- Manufacturer receives the request for shipment of goods.

- Manufacturer sends a response back to the warehouse indicating that the order was received.
  - This does not mean that the order was fulfilled and the goods will be shipped; it is just an acknowledgement of receipt of the order.
- The request is sent to the inventory division to determine whether the requested goods are available.
  - If the goods are in stock:
    - Make a callback to the warehouse and send the shipment.
  - If the goods are below the threshold or out-of-stock:
    - Ask the manager to authorize production for that specific product item.
    - Make a callback to the warehouse and send the shipment.

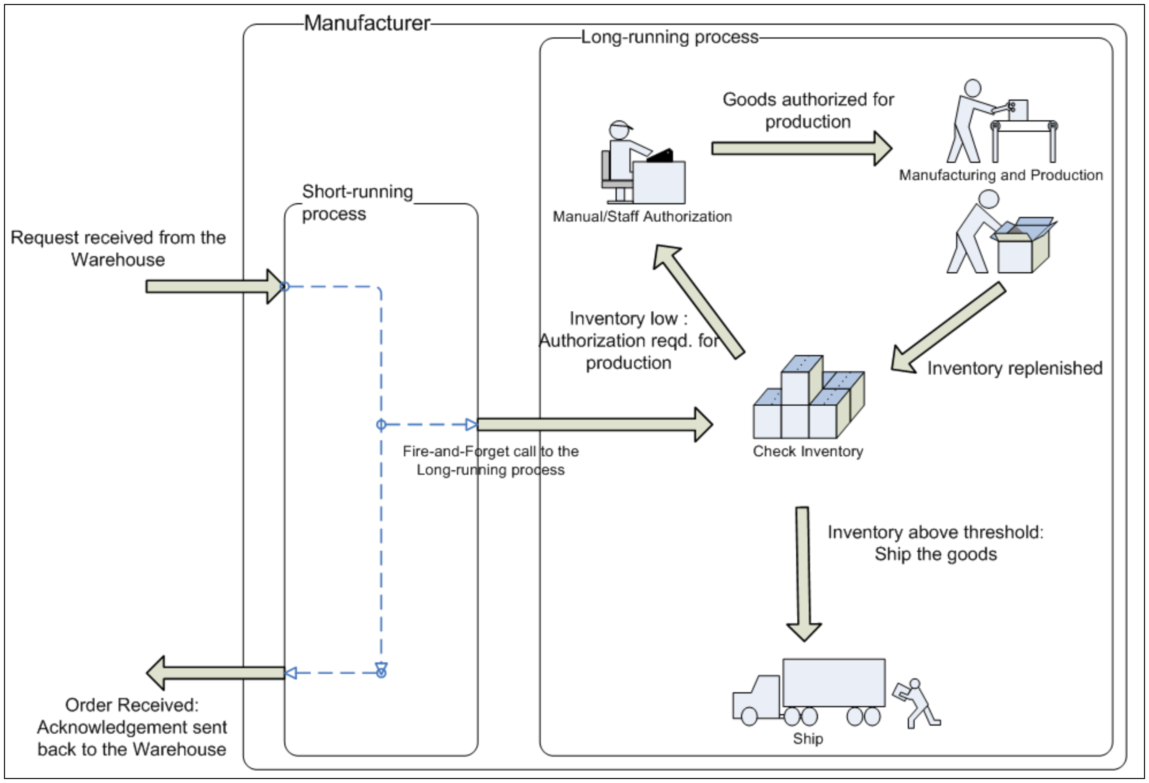

Manufacturer
Long-running process
Goods authorized for production
Manufacturing and Production
Manual/Staff Authorization
Short-running process
Request received from the Warehouse
Inventory low : Authorization reqd. for production
Inventory replenished
Fire-and-Forget call to the Long-running process
Check Inventory
Inventory above threshold: Ship the goods
Ship
Order Received: Acknowledgement sent back to the Warehouse

Figure 10-7 Manufacturer implemented as a Business Service Choreography process

Design alternatives

To design the Business Service Choreography as described in Figure 10-7, we consider the following design alternatives that are available to us and select the best that is appropriate for our scenario, based on their benefits and limitations.

Design alternative: top-down versus bottom-up modeling

In the top-down design approach, we already have the WSDL interface definitions defining the services and interface. The service implementation is designed and developed at a later stage. This kind of approach gives us the following benefits and impacts:

- Services defined according to the requirements of clients, the business definition of the process, or both.
- Parallel development between the service requester and service provider.
- Reuse of the existing WSDL and XSD to either migrate or enhance the current enterprise application.
- Services tend to be large-grained, which promotes flexibility and reusability.
- Service definitions may not precisely match functions that are offered by existing systems, so some transformation, aggregation, or modification may be required.

In the bottom-up approach, the service definition is based on existing implementations of function. Business services and processes must then be created by aggregation, transformation, or choreography of these existing functions. This kind of approach is usually taken when:

- The service provider already has an application and it would like to expose its implementation as a Web service.
- Existing service implementations are not amenable to change, perhaps because they are complex systems that support many processes.
- An existing service provider interface definition is found and used by a service requester, without the knowledge of the service provider. There is no agreement between the consumer and provider. The service provider may change and republish the service at any time.

Because we are using the WS-I sample supply chain management application scenario, we already have the existing WSDL and the XSD definitions, so we will use the top-down approach.

Design alternative: long-running versus short-running processes

A short-running process completes in a short interval of time. It does not persist any data and can be summarized as a single business transaction. A short-running process design is considered when:

- A synchronous interaction is desired.
- A process will run for a limited time.
- No human interaction is required.
- Runtime data persistence is not required.

A long-running process can potentially complete in a longer interval of time, such as an hour, several days, or several months. It persists the process instance data and may involve human interaction. A long-running process design is considered when:

- An asynchronous interaction is desired.
- A process will run for a potentially long, undetermined time.
- Human interaction is required.
- Runtime data persistence is required because either:
  - The process may take very long time to complete.
  - Human intervention is involved.
- The process may have to wait for external events, or its outcome may be affected by them.

Based on the requirements we have for the manufacturer, our need for human interaction (management authorization for stock production) dictates that we should implement a long-running process.

Design alternative: composite processes

Short-running and long-running processes can interoperate with each other. There are four invocation compositions:

- A long-running process invoking a short-running process

  Human-based or event-based activities require a process to be long-running, but such a process may also include simpler functions that are short-lived. Therefore, a short-running process containing these functions could be invoked. This is useful in situations where commonly occurring short-running business logic is reused as a subprocess in long-running processes.
- A short-running process invoking a long-running process

  In this instance a short-running process invokes a long-running process in a fire-and-forget manner. The short-running process then continues to the next activity in the flow. The long-running process executes in its own thread, enabling the short-running process to complete while the long-running process continues to execute.

This composition is useful when a synchronous request/response interaction is required by the client that invokes the process. The short-running process starts, invokes a long-running process, then returns an acknowledgement back to the client. The client does not have to wait for the long-running process to complete before it receives a response.

For example, when you purchase a book from an online bookstore, you receive an acknowledgement indicating that the order was placed and will be shipped by a certain date. Behind the scenes, this acknowledgement might have been generated by a short-running process that triggered a long-running process to check the inventory, order the book, and ship it to the respective customer.

- A long-running process invoking a long-running process and a short-running process invoking a short running process

  The last two compositions are usually preferred when you already have an existing process and would like to reuse this logic as part of a larger flow.

In our scenario, we used the short-running process invoking a long-running process composition for the manufacturer process:

1. When the warehouse calls the manufacturer to ship the order, an acknowledgement is sent back to the warehouse indicating successful placement of the order. This will be implemented in the short-running process.
2. The short-running process invokes a long-running process, which performs the inventory check and, if required, requests management approval. When the long-running process completes, a callback is sent to the warehouse, informing it that the order has shipped.

10.3 Development guidelines

In this section, we build the processes defined in Figure 10-7 on page 278 to create a business process implementation of the manufacturer service. Two processes are created:

- Short-running process

  Exposes a process interface that is invoked by a client (the warehouse service) to place an order for stock. The short-running process invokes a long-running process, then returns a reply to the client. This maps to the Serial Process pattern because it uses no human interaction.

- Long-running process

  Checks the inventory, replenishes stock, gains management approval if necessary, and ships the goods. The requirement for human interaction means that this maps to the Serial Workflow pattern.

Both processes are created in the BPEL4WS open standard, using WebSphere Studio Application Developer Integration Edition V5.1. This section assumes that you have some knowledge of using WebSphere Studio Application Developer Integration Edition to build business processes.

10.3.1 Long-running process

This section describes how to build the long-running process for our business scenario:

- Creating the skeleton process and process interface

  Describes how to create the long-running process skeleton and how to assign an existing WSDL interface to it.
- Building the process

  Describes each activity, partner link, and variable that is used in the process.

Creating the skeleton process and process interface

This section describes how to create a new BPEL4WS process and assign the relevant process interface to it. We go into detail in this section to get you started with a working process template. The remainder of the process is not described in step-by-step detail.

1. Create a new BPEL4WS process in WebSphere Studio in the Business Integration perspective:
   a. Create a new service project (**File** → **New** → **Service Project**) called `WSIManufacturer`.
   b. Create a new business process in this service project (**File** → **New** → **Business Process**) in package com.ibm.itso.process called `longRunning.bpel`.
   c. In the New Business Process wizard, you will be asked which process type to use. Select **Sequence-based BPEL Process**.
2. When a new process is created, you will see two files in the Services view. The .bpel file contains the BPEL process, and by default is opened in the BPEL Editor. The .wsdl file represents the generated process instance for the

process. We want to use our own predefined process interface. Complete the following steps to use the predefined interface:

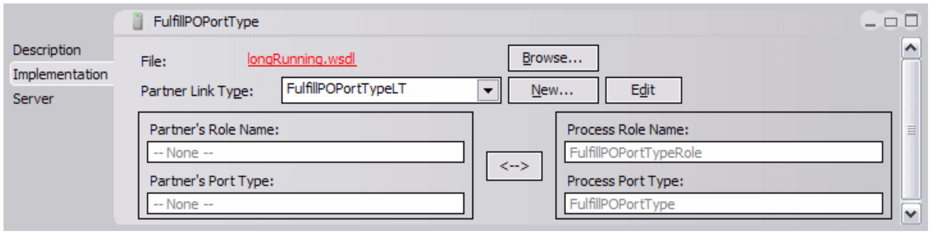
a. In the process editor, delete the variable called **InputVariable** and the partner link called **PartnerLink**.

b. Import the process interface WSDL file and associated XSD files into the com.ibm.itso.process package. These files are supplied with the source code that accompanies this redbook:

   - fulfillPOProcessInterface.wsdl
   - ManufacturerPO.xsd
   - ManufacturerSN.xsd
   - Configuration.wsdl
   - Configuration.xsd

c. Drag **fulfillPOProcessInterface.wsdl** into the process editor. When asked to select the Port Type, select **FulfillPOPortType**. This should create a new partner link.

d. Select **FulfillPOPortType** and display the **Implementation** tab properties. Click **<-->** to assign the partner role name and port type as the process role name and port type (Figure 10-8).



File: longRunning.wsdl

Partner Link Type: FulfillPOPortTypeLT

Partner's Role Name: -- None --

Partner's Port Type: -- None --

Process Role Name: FulfillPOPortTypeRole

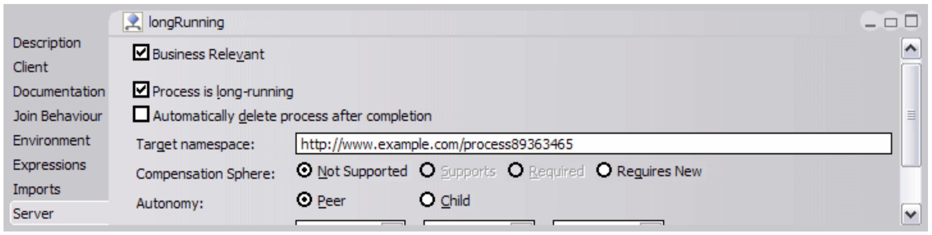
Process Port Type: FulfillPOPortType

Figure 10-8 Setting the process role name and port type

e. The Receive activity must point to an operation in this partner link and be assigned a variable. Click the **Receive** activity. In the **Implementation** tab, set the Partner Link to **FulfillPOPortType**. By default this should set the Operation to fulfillPO. To create a new variable click **New** and name the variable **fulfillPORequest**.

f. The fulfillPO operation defines a one-way interaction, so the Reply activity is not required. Delete it.

3. Finally, this process is long-running. The default is to create a short-running process, so we need to modify this. In the BPEL Editor, click **longRunning** at the top of the process and select the **Server** tab. Check the option **Process is long-running** (Figure 10-9).


longRunning

Description
Client
Documentation
Join Behaviour
Environment
Expressions
Imports
Server

[✓] Business Relevant

[✓] Process is long-running
[ ] Automatically delete process after completion

Target namespace: http://www.example.com/process89363465

Compensation Sphere: [✓ Not Supported] Supports Required Requires New

Autonomy: [✓ Peer] Child

Figure 10-9 Setting the process to be long-running

Building the long-running process

This section describes the activities and other entities that are part of the long-running process. The complete process is shown in Figure 10-10 on page 284.

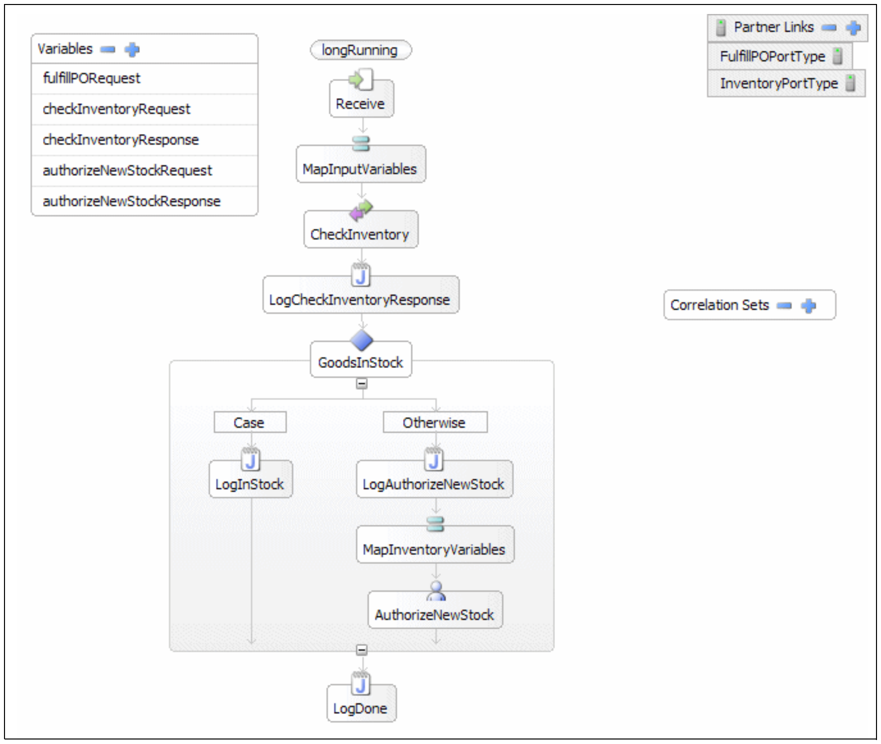


Figure 10-10 Long-running process in the BPEL Editor

Partner links

Partner links identify the parties that interact with your business process. Each partner link is defined by a partner type and a role name and can either set or modify the variables that are used by the activities within the process.

The following partner links are used in this process:

- FulfillPOPortType
  - Contains the interface for this process.
  - Is implemented by the fulfullPOProcessInteface.wsdl file.
- InventoryPortType
  - Contains the CheckInventory service.
  - Is implemented by the InventoryPortTypeJavaService.wsdl file.

Variables

Variables are Java representations of WSDL messages, and are used to set, get, and store message values. Each WSDL message that is used by an activity, Java snippet, or condition must be represented by a variable.

We created the following variables:

- fulfillPORequest

  Used to store the process input data. Represents the POFulfill message that is defined in fulfillPOProcessInterface.wsdl.
- checkInventoryRequest and checkInventoryResponse

  Used to store the input and output messages of the CheckInventory operation. Represents the CheckInventoryRequest and CheckInventoryResponse messages in Inventory.wsdl.
- authorizeNewStockRequest and authorizeNewStockResponse

  Used by the staff activity to store data from the management approval human interaction. Represents the AuthorizeNewStockRequest and AuthorizeNewStockResponse messages in AuthorizeNewStock.wsdl.

Receive activity

The receive activity represents the start of a process. It waits for external input, based on the partner link, port type, and operation that it expects to receive. This receive activity waits for the fulfillPO operation to be invoked.

Notice that this process does not contain a reply activity. This indicates that the process contains a one-way interface.

Assign activity : MapInputVariables

The assign activity is used to map one value to another. This assign activity maps the PurchaseOrder part that is defined in the process interface with the PurchaseOrder part that is used in the CheckInventory service. This ensures that data that is entered as input to the process is sent to the CheckInventory service. The assign is shown in Figure 10-11.

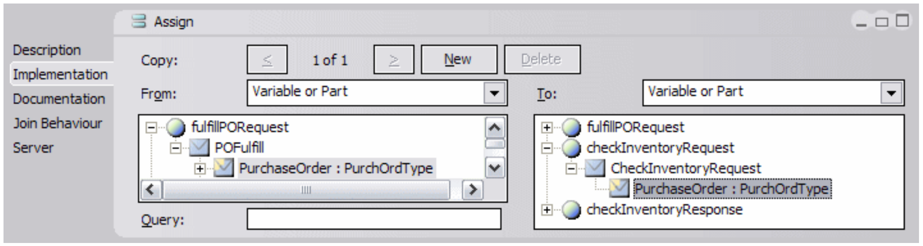


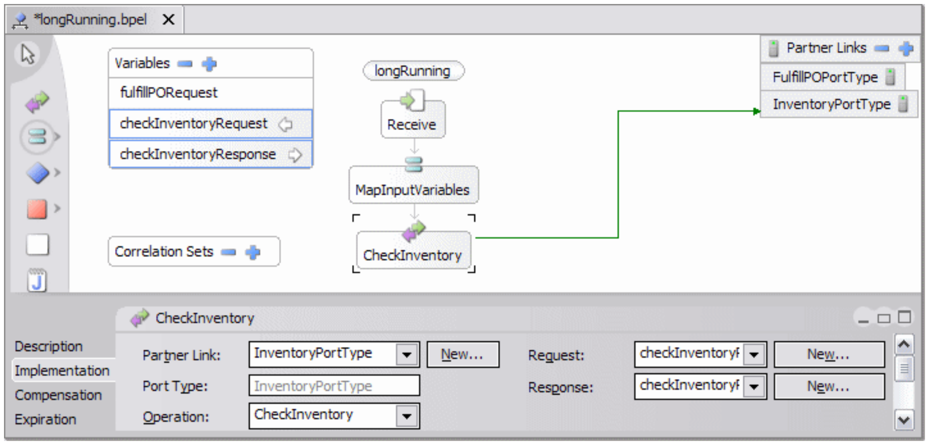
Figure 10-11 MapInputVariables assign activity

Invoke activity : CheckInventory

Invoke activities are used for invoking services. A service can be a Web service or any other type of service that can be described in WSDL and is supported by the Web Services Invocation Framework.

This activity invokes the checkInventory operation, which is a service with a Java class endpoint. To use this service, you must import the endpoint and service description into the workspace. The checkInventory operation is provided in the Java project InventoryProject and is supplied with this book.

To add the checkInventory operation to the process, drag the service file (InventoryPortTypeJavaService.wsdl) into the process editor to create a partner link, then add an invoke activity that uses the partner link (Figure 10-12). You will also create variables for the activity.



Partner Link: InventoryPortType
Port Type: InventoryPortType
Operation: CheckInventory
Request: checkInventoryF
Response: checkInventoryF

Figure 10-12 CheckInventory invoke activity, and InventoryPortType partner link

CheckInventory is service that provides the implementation for checking whether the goods are available in the stock. It accepts the parameters PurchaseOrder, ConfigurationHeader, StartHeader. If the goods are available in stock, it returns an empty object of type PurchaseOrder; otherwise it fills in the empty object with the number of goods required to be manufactured.

Java snippet : LogCheckInventoryResponse

A Java snippet is a BPEL4WS extension that enables you to add code to a process. This code is typically used for mapping variables, adding logging and tracing, and to perform other Java functions.

In this instance we use a Java snippet to output the response of the CheckInventory operation, stored in the checkInventoryResponse variable, to the system console (Example 10-1 on page 287).

Example 10-1 LogCheckInventoryResponse Java snippet

```
System.out.println("checkInventoryResponse: " +
        getCheckInventoryResponse().toString());
```

Switch structure : GoodsInStock

A switch structure is similar to the switch Java construct. It allows a single path of execution to be taken based on a specific case being met. In our example, we check to see whether the ordered goods are in stock. If they are, we log this fact and continue. If not, we execute a separate path that includes a staff activity to request permission to manufacture new stock.

A case statement checks whether an empty PurchaseOrder (indicating that all items are in stock) has been returned by the CheckInventory service. Example 10-2 shows the Java expression that is used.

Example 10-2 Case statement for GoodsInStock

```
CheckInventoryResponseMessage unfilledStock = getCheckInventoryResponse();
return (unfilledStock.getUnfilledPurchaseOrder().getItems().getItem() == null);
```

An `otherwise` statement indicates that if all case statements evaluate to false, the flow continues down the otherwise branch.

Figure 10-13 shows the GoodsInStock switch.

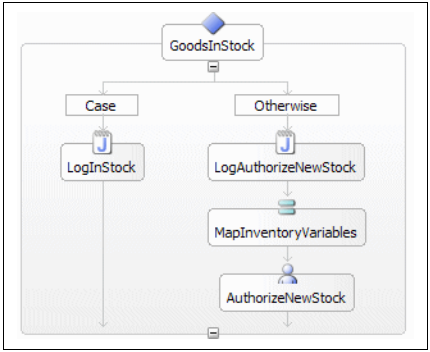


Figure 10-13 GoodsInStock switch

Java snippets : LogInStock and LogAuthorizeNewStock

These Java snippets report the decision of the switch to the system console, indicating either all items are in stock, or that management approval is required to replenish stock. These Java snippets are optional but are useful for logging and debugging purposes.

Assign activity : MapInventoryVariables

This assign activity maps the UnfilledPurchaseOrder part from the CheckInventoryResponse variable to the UnfilledStock part of the authorizeNewStockRequest variable.

The content of the authorizeNewStockRequest variable is used as the input message of the work item for the staff activity. The manager who claims the work item uses the data in the input message to make the decision as to whether to approve or reject the request.

Staff activity : AuthorizeNewStock

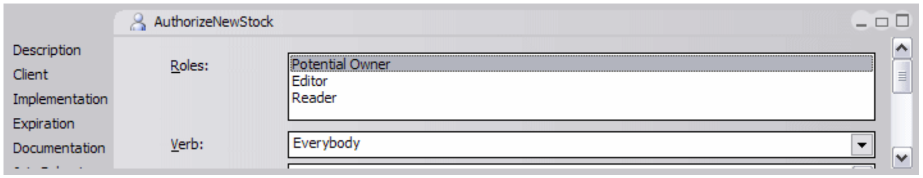
Staff activities are another BPEL4WS extension, and they enable human interaction in a process. In some ways, a staff activity is like a regular invoke activity in that it contains an input and output message that is defined in WSDL. The owner of the staff activity receives the input message and, based on this, populates the output message.

At runtime, when a staff activity is reached, a work item is generated. This work item must be claimed and completed before the process can continue. Permissions are granted as to who can view, edit, and claim a work item by assigning roles to a staff activity: potential owner, editor, and reader. Each role can be assigned to an individual user or a group of users, or be dynamically determined at runtime using a late binding.

Staff plug-in providers define the user registry to use for security credential resolution. Such user registries include the local operating system (such as the Windows user registry) and an LDAP directory.

We created a staff activity and assigned it the authorizeNewStock operation. The response message to this operation (mapped by the authorizeNewStockResponse variable) contains a single Boolean part where the manager indicates whether the request to manufacture new stock is authorized.

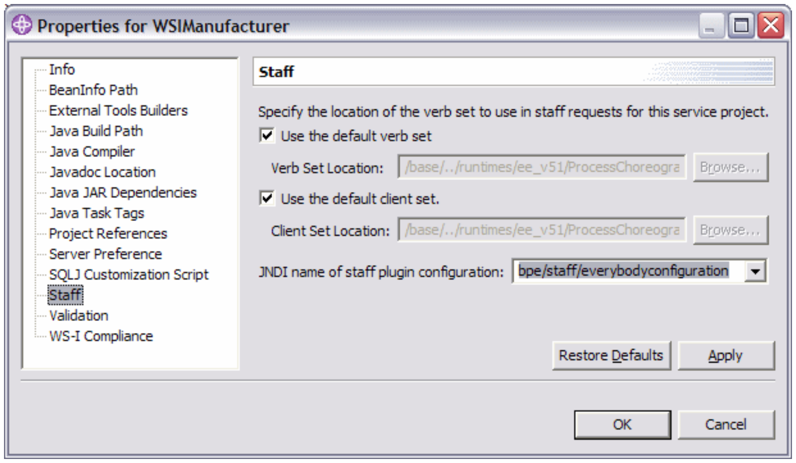
For simplicity of testing, we did not want to specify strict security constraints regarding who could be the potential owner of this staff activity. Therefore we set the potential owner to use the verb Everybody (Figure 10-14 on page 289) and the staff plug-in provider to the everybody configuration (Figure 10-15 on page 290). This means that any user, whether authenticated or not, has the ability to claim and complete work items that are generated by this staff activity.



Roles: Potential Owner
Editor
Reader

Verb: Everybody

Figure 10-14 AuthorizeNewStock staff activity potential owner



Specify the location of the verb set to use in staff requests for this service project.

☑ Use the default verb set

Verb Set Location: /base/../runtimes/ee_v51/ProcessChoreogra Browse...

☑ Use the default client set.

Client Set Location: /base/../runtimes/ee_v51/ProcessChoreogra Browse...

JNDI name of staff plugin configuration: bpe/staff/everybodyconfiguration

Figure 10-15 Staff plug-in provider

Also for simplicity, this process does not examine the response of the staff activity to determine whether the manager authorized the request. Normally you would examine the output message of a staff activity and divert the flow accordingly.

Java snippet : LogDone

The final activity in the process is a Java snippet that sends a message to the system console indicating that the process has completed and the stock is dispatched. It represents the convergence of both execution paths in the switch.

In the interest of time, the callback to the warehouse service to indicate that the stock is dispatched has not been implemented. This would take the form of an invoke activity that invokes the WarehouseCallBack Web service, and would most likely be routed back through the Enterprise Service Bus.

10.3.2 Short-running process

The short-running process achieves two goals:

- It facilitates a short request/response operation with a client, returning an acknowledgement that an order has been placed.
- It invokes the long-running process to dispatch the stock ordered in a fire-and-forget manner.

Although most of the actual functionality is contained within the long-running process, the short-running process is a useful facade. It provides the client with a short response time that would not have been guaranteed if the client invoked the long-running process directly.

This section has two parts:

- Creating the skeleton process and process interface

  Describes the existing WSDL interface that is assigned to the process.
- Building the process

  Describes each activity, partner link, and variable that is used in the process.

Creating the skeleton process and process interface

Because the short-running process contains a request/response interface, it has a Receive and Reply activity. We named the process shortRunning.bpel.

As with the long-running process, we used a predefined WSDL file to specify the process interface. Follow the steps described in "Creating the skeleton process and process interface" on page 282 to switch the process interface, but use the submitPO operation of the port type ManufacturerPortType defined in submitPOProcessInterface.wsdl.

Note: The short-running process will be deployed with a SOAP/HTTP process binding. Use of this binding requires that we make the following changes to our process interface file submitPOProcessInterface.wsdl:

- Replace all WSDL and XSD schema imports with inline code.
- Modify all simple types to not use the restriction element.
- Remove all references to the mustUnderstand attribute.

Building the short-running process

This section describes the activities and other entities that are part of the long-running process. Figure 10-16 on page 292 shows the complete process.

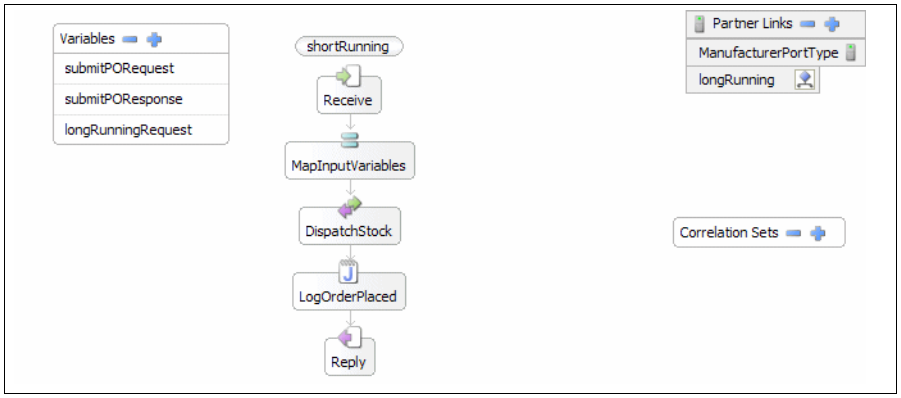


Figure 10-16 Short-running process in the BPEL4WS editor

Partner links

The following partner links are used in this process:

- ManufacturerPortType
  - Contains the interface for this process.
  - Is implemented by the submitPOProcessInteface.wsdl file.
- longRunning
  - References the long-running BPEL4WS process.
  - Is implemented by the longRunning.bpel file.

Variables

We created the following variables:

- submitPORequest

  Used to store the process input data. Represents the POSubmit message that is defined in submitPOProcessInterface.wsdl.
- submitPOResponse

  Used to store the process output data. Represents the ackPO message that is defined in submitPOProcessInterface.wsdl.
- longRunningRequest

  Used to store the data to send to the long-running process. Represents the POFulfill message that is defined in fulfillPOProcessInterface.wsdl.

Receive activity

The receive activity represents the start of a process. It waits for external input, based on the ManufacturerPortType partner link and port type, and the submitPO operation.

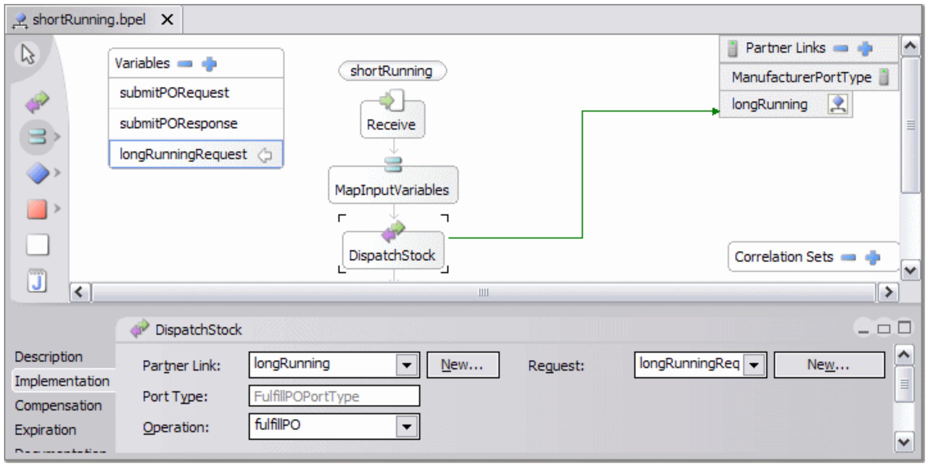
Assign activity : MapInputVariables

Maps the PurchaseOrder part of the submitPORequest variable to the PurchaseOrder part of the longRunningRequest variable. This ensures that the stock information that is sent to the receive activity is propagated to the long-running process.

Invoke activity : DispatchStock

We have discussed that an invoke activity can invoke a Web service or a service with a Web Services Invocation Framework endpoint. Additionally, an invoke activity can invoke a BPEL4WS process. The DispatchStock activity invokes the long-running BPEL4WS process we have built.

To invoke a BPEL4WS process, simply drag the .bpel file into the BPEL Editor to create a partner link, then assign the invoke activity to that partner link (Figure 10-17).



Partner Link: longRunning
Port Type: FulfillPOPortType
Operation: fulfillPO
Request: longRunningReq

Figure 10-17 Invoking a BPEL4WS process

The long-running process exposes the operation fulfillPO, which is a one-way operation, so the long-running process will be invoked in a fire-and-forget manner. After the long-running process instance has started, the short-running process will continue execution. When the short-running process completes, the long-running process may still be running.

Java snippet : LogOrderPlaced

This Java snippet (Example 10-3) serves two purposes:

- Sends a message to the system log to indicate that an order has been placed with the manufacturer.
- Sets the process output variable submitPORequest to the Boolean value of `true` to indicate successful completion of the short-running process.

Example 10-3 LogOrderPlaced Java snippet

```
System.out.println("Order has been placed with the manufacturer");
AckPOMessage submitPOResponse = getSubmitPOResponse(true);
submitPOResponse.setResponse(true);
```

Reply activity

The activity that returns a response to the process client. This is the response portion of the submitPO request/response operation.

10.4 Runtime guidelines

This section describes guidelines for running a Business Service Choreography process. It describes the deployment and testing options, and highlights the best options to use when interfacing with an Enterprise Service Bus.

10.4.1 Deploying a process

Before you can run a process, you must deploy it as an enterprise application. This deployment process serves two purposes:

- Packages the process and service project into an enterprise application that can be installed as an EAR file in WebSphere Business Integration Server Foundation.
- Creates a process binding that can be used to start the process instance.

When you deploy a process, you are required to enter the process binding type, even if ultimately you do not intend to start the process using the process binding. Multiple process bindings can be created for a given process.

Design alternative: process bindings

When deploying a process, you are given a choice of four process bindings:

- EJB binding

  Generates a stateless session Enterprise bean that acts as a facade to the process. A client can invoke a method on this session bean to initiate the process.

- JMS binding

  Generates a message-driven Enterprise bean that acts as a facade to the process. The message-driven bean contains a message selector that listens on a specific queue for a message containing a specific header property. A client can initiate the process by sending a properly formatted message to the appropriate JMS queue.

- SOAP/HTTP binding

  Exposes the process as a Web service provider with a SOAP/HTTP interface. There are two types of SOAP/HTTP binding:

  – IBM Web Service

    Uses the SOAP provider supplied with WebSphere Business Integration Server Foundation V5.1. This creates a JAX-RPC compliant Web service.

  – Apache

    Uses the Apache SOAP provider. Provided for compatibility with earlier Web services runtimes.

- SOAP/JMS binding

  Exposes the process as a Web service with a SOAP/JMS interface. This binding uses the IBM Web Service SOAP provider.

We created the following process bindings:

- Short-running process: SOAP/HTTP IBM Web Service binding

  To expose a Business Service Choreography process to an Enterprise Service Bus, the SOAP/HTTP or SOAP/JMS bindings should be used. This enables the Enterprise Service Bus to invoke the process as a regular Web service.

- Long-running process: JMS binding

  The long-running process is invoked by the short-running process. This invocation does not use a process binding. Therefore the choice of process binding for the long-running process is irrelevant, unless you intend to invoke the long-running process directly. The default process binding for a long-running process is the JMS binding.

Preparing a business process container

After a process binding has been created for a process, and the process is packaged into an enterprise application (EAR file), it can be run in a business process container. Two business process containers are provided:

- WebSphere Studio Application Developer Integration Edition test server

  You can use WebSphere Studio to create a WebSphere Business Integration Server Foundation test server. This server contains an automatically configured business process container, which uses a local Cloudscape™ database for persistence, and an internal JMS provider. This server is useful for testing business processes.

- WebSphere Business Integration Server Foundation

  After a process is tested and working, it should be deployed to WebSphere Business Integration Server Foundation. A business process container can be created automatically when WebSphere Business Integration Server Foundation is installed, or you can create one manually. This business process container can be configured to use a variety of databases and JMS providers.

Starting a process instance

After a process has been deployed and installed in a real or test WebSphere Business Integration Server Foundation server, you can create process instances from it in the following ways:

- Process Web Client

  The Process Web Client is an application that is supplied with WebSphere Business Integration Server Foundation that enables you to start, view, monitor, and terminate process instances. Additionally, it can be used to manipulate work items that are generated by staff activities. The Process Web Client can also be customized using JSPs.

  Note: The Process Web Client is not compatible with many process instances that contain complex types. The complex types that are used in the short-running process described in this chapter fall into this category. In this case, you must use another option to initiate the process instance, such as create customized JSPs for the Process Web Client that are compatible with the complex types, or build your own process client.

- Process binding

  A process binding is always created during deployment, regardless of whether you intend to use it. For testing purposes you can invoke process bindings as follows:

  – An EJB binding can be invoked by using the Universal Test Client of WebSphere Studio to invoke the relevant EJB method.

  – A SOAP/HTTP or SOAP/JMS binding can be invoked by using the Web Services Explorer of WebSphere Studio to invoke the relevant WSDL operation.

- WebSphere Process Choreographer API

  An API is provided to work directly with process instances. The API contains two interfaces: a session bean and a message-driven bean. Using either of these interfaces, you can work with process instances, work items, and process variables.

- Custom client

  You can create a custom client application that either uses the WebSphere Process Choreographer API or a process binding to invoke a process instance. We created a JSP application that uses the SOAP/HTTP process binding to invoke the short-running process.

To start a process instance from an Enterprise Service Bus, it is expected that you will use the SOAP/HTTP or SOAP/JMS process binding.

10.5 Further information

- *Patterns: Serial and Parallel Processes for Process Choreography and Workflow*, SG24-6306
- Business Process Execution Language for Web Services Version 1.1

  `http://www.ibm.com/developerworks/library/ws-bpel/`
- WebSphere Business Integration Server Foundation InfoCenter

  `http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp`

11

Exposed ESB Gateway composite pattern

This chapter takes the Enterprise Service Bus (ESB) implementation that was discussed in previous chapters and describes how access to the Internet can be provided to allow interactions between different enterprises.

We use the Exposed ESB Gateway composite pattern to expose access from our ESB to an external enterprise. This chapter explores the architectural implications of using it to add Internet access to a service-oriented architecture (SOA). The Exposed ESB Gateway was introduced in 4.4.5, “Extended Enterprise SOA patterns” on page 108.

We extending the scenario that was developed in previous chapters to illustrate many of the issues that are discussed and to describe how this was achieved.

In this chapter, the following topics are discussed:

- The sample business scenario that our solution must address
- Design guidelines that describe the design approaches for using the Exposed ESB Gateway composite pattern to allow Internet access to and from an ESB
- Runtime guidelines that illustrate the configuration of service invocation routing

11.1 Business scenario

The sample application that was introduced in Chapter 7, “The business scenario used in this book” on page 169, is a simplified supply chain for a consumer electronics retailer. The company has implemented the Enterprise Service Bus to meet the business requirements.

The company has decided to divest itself of the three manufacturers. Each will be sold off to other companies or established as new companies in their own right. Various interactions must now take place securely over the Internet. These are:

- The functionality enabling a warehouse to replenish stock from a manufacturer.
- The notification of shipment by a warehouse of replenishment stock by a manufacturer.
- The logging of business tracking information by the manufacturer.

Figure 11-1 shows the business infrastructure.

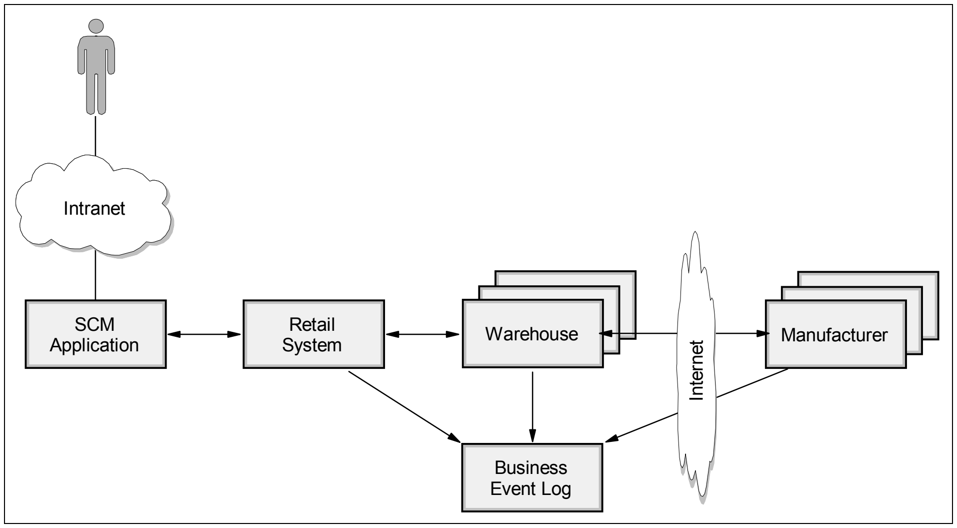


Figure 11-1 High-level business context with external manufacturers

11.2 Design guidelines

We now analyze the business requirements to select the appropriate design patterns and product components for implementation.

11.2.1 Design overview

Figure 11-2 shows an overview of the steps that might be taken to design a solution to address business requirements. We now follow these steps.

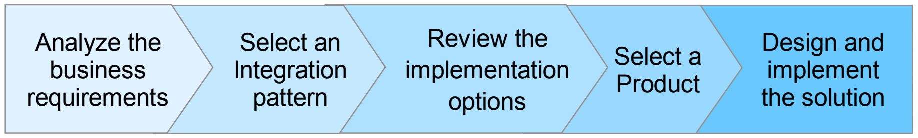

Analyze the business requirements
Select an Integration pattern
Review the implementation options
Select a Product
Design and implement the solution

Figure 11-2 Design overview

Selecting the pattern

We start by considering the requirements that were described in the business scenario in the previous section in conjunction with the Process Integration patterns that were defined in 4.4, “SOA profile of the Application Integration patterns” on page 90. To be able to access manufacturers in external organizations, the following areas must be considered:

- Addressing of remote services
- Security over the Internet
- Implementation of changes with minimal impact on the existing infrastructure

We previously implemented an ESB, and Figure 11-3 illustrates these areas.

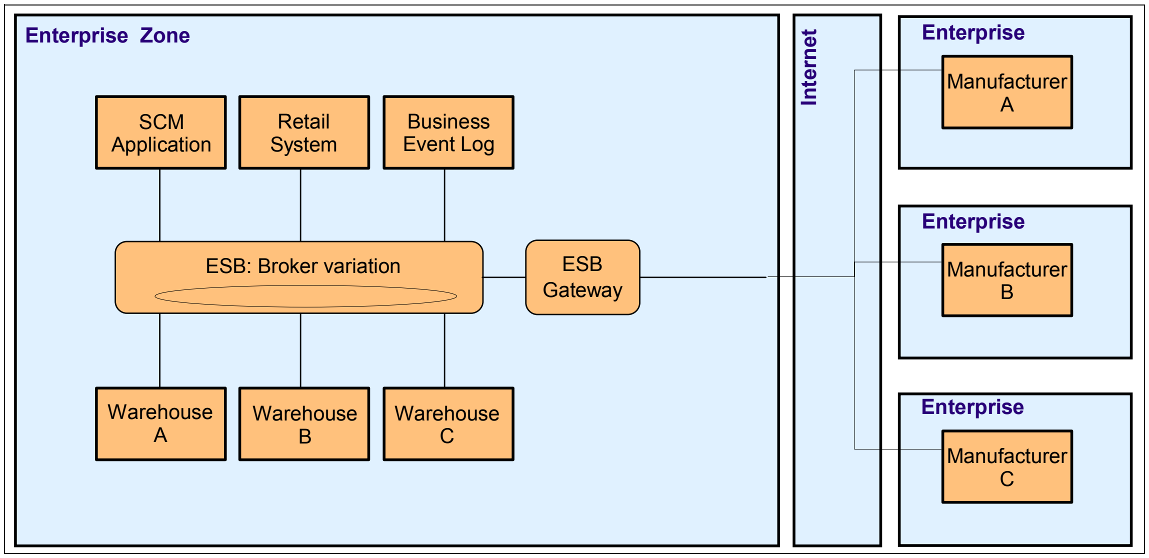


Figure 11-3 The Exposed ESB Gateway composite pattern applied to this scenario

When considering the patterns, first and foremost we must be able to access the Internet. The way in which this will be done is by using the addressing capability of the ESB and routing requests that are provided externally over the Internet. In this way we can eliminate the effect on our endpoint service requesters and service providers in the enterprise.

Furthermore, in our scenario each enterprise (the original retailer and the three new manufacturers) knows the specific enterprises it must interact with.

Choosing the relevant SOA pattern

To be able to manage the demands of linking the infrastructures over the Internet, we must use an Extended Enterprise SOA pattern. The Exposed ESB Gateway composite pattern (hereby referred to as the Exposed ESB Gateway pattern) enables us to grant access to an external enterprise from our own ESB, as shown in Figure 11-4.

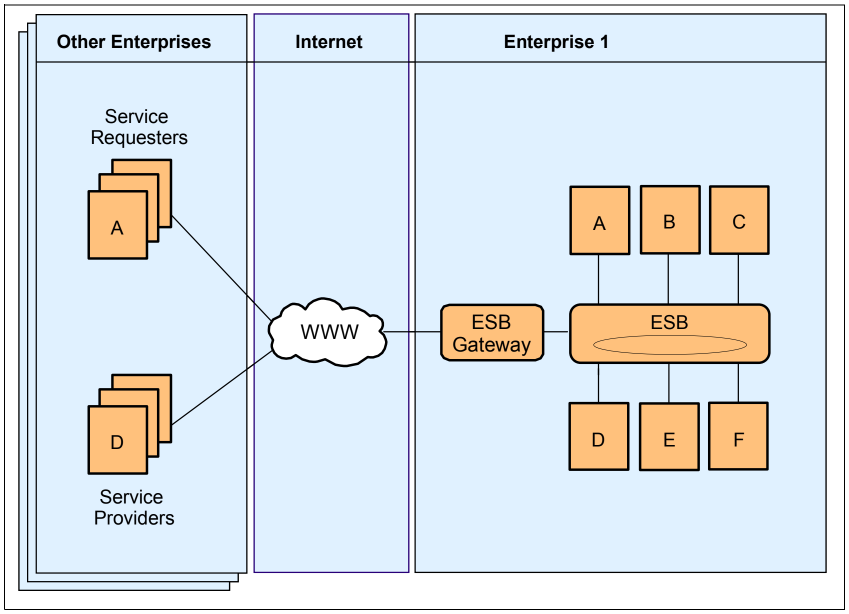


Figure 11-4 Exposed ESB Gateway composite pattern

We use an Extended Enterprise::Exposed Router pattern to model the interaction in the ESB Gateway node. Figure 11-5 on page 303 shows the Exposed Router application pattern and that the Exposed Router can route requests through to a single target application. Aggregation is not supported.

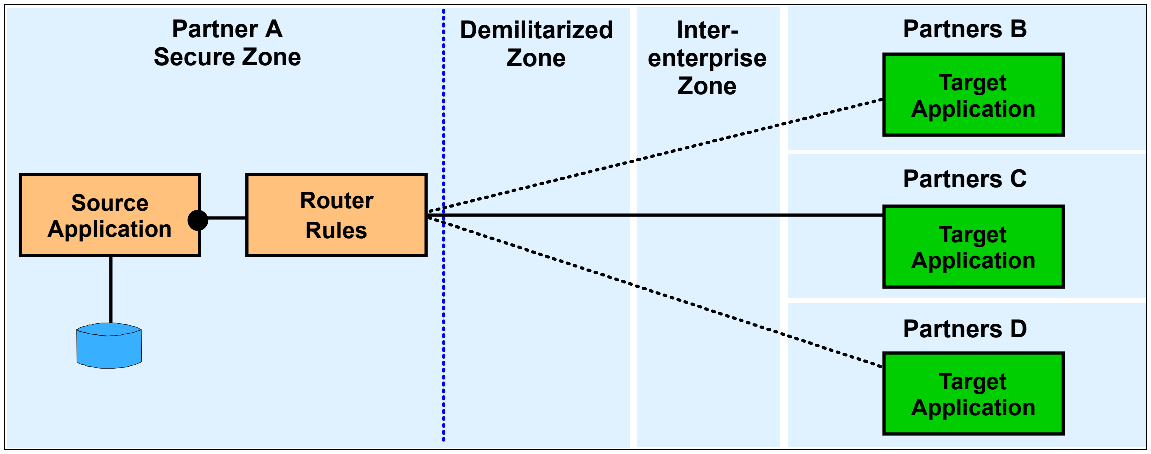


Figure 11-5 Extended Enterprise::Exposed Router application pattern

If you apply the Exposed Router application pattern (Figure 11-5) to the ESB Gateway in Enterprise 1 (Figure 11-4 on page 302) then:

- The Source Application is the Enterprise Service Bus in Enterprise 1.
- The Target Application is the service provider in Enterprise 2.

For further information about the Exposed Router pattern, consult *Patterns: Broker Interactions for Intra- and Inter-enterprise*, SG24-6075.

Implementing the ESB Gateway

The use of the ESB Gateway enables intelligent routing of a service request to an external service provider. The separation of the service interface in the ESB from its implementation in another enterprise helps address the likely differences in technology and information models.

By attaching an ESB Gateway to an ESB we achieve further important benefits:

- The ESB retains control over its service namespace.
  - It takes responsibility for routing service requests that are provided externally to the Exposed ESB Gateway.
  - It takes responsibility for routing service requests from outside the enterprise that have been routed to the ESB by the ESB Gateway.
- The separation of service requestors and service providers in the enterprise by using an ESB enables the addition of the ESB Gateway without affecting these endpoints.
- The ESB Gateway can receive service requests from other enterprises and route them to the ESB without affecting the endpoint service implementations.

Scenario requirement for the ESB Gateway

Three service invocations are related to the Manufacturer in our scenario:

- The ESB makes a call (on behalf of the Warehouse) to the Manufacturer to replenish stock.
- The Manufacturer sends business events to the ESB (which routes them to the LoggingFacility).
- The Manufacturer calls back to the ESB (which routes the call to the Warehouse) with a shipping note.

Prior to the divesting of the Manufacturer, these calls were direct between the ESB and Manufacturer. We now need to use the ESB Gateway to address these interactions between the enterprises.

From Chapter 9, “Enterprise Service Bus: Broker variation” on page 219 we have an ESB implemented in the Retail enterprise. In this chapter we introduce separate Manufacturer enterprises and will introduce an ESB Gateway for these enterprises. The enterprises will communicate with each other over the Internet, routing the requests according to the address resolution that is defined in each.

Product implementation options

We now need to consider the capabilities of products from which we can make a selection to implement the ESB Gateway alongside an ESB. Our product selection for the scenario was based on:

- The product that is currently available.
- The ability of the products’ capabilities to map to the requirements.
- The existing infrastructure of our organization (for example, whether the company already uses one of the products).

The following products can be used to implement the ESB Gateway in conjunction with an existing ESB.

- Web Services Gateway (as part of WebSphere Application Server V5.1.1 Network Deployment)
- WebSphere Business Integration Connect V4.2.1

To help with selecting the appropriate product, refer to the following:

- The description of each product, described in 5.1, “Runtime product descriptions” on page 134.

Product selection for the scenario implementation

The ESB Gateway does not make assumptions about the service providers that it invokes nor the service requesters from which it receives requests. In our scenario, the service provider is actually another ESB, enabling us to show the use of two ESB Gateways.

To address the requirements of the business scenario using the ESB Gateway, we selected the Web Services Gateway. The full runtime pattern is shown in Figure 11-6 and Figure 11-7 on page 306.

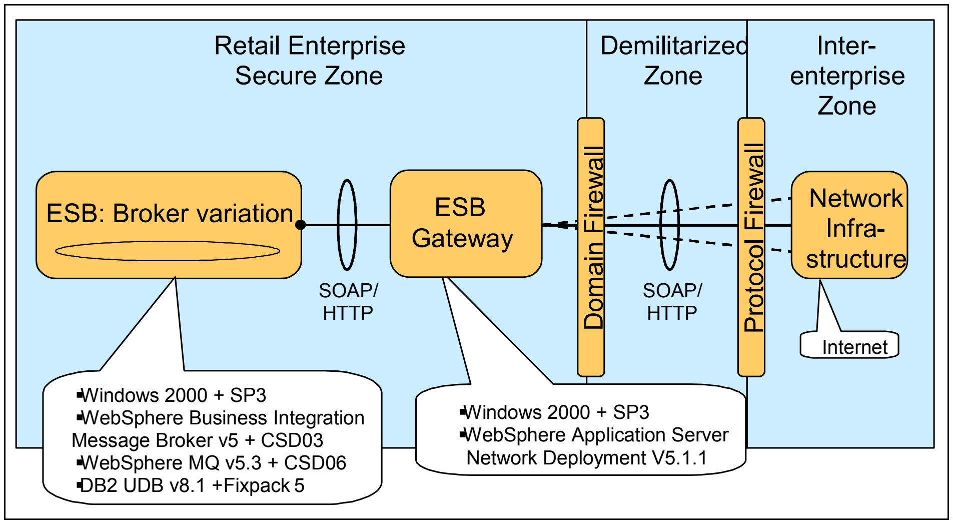


Figure 11-6 Exposed ESB Gateway composite pattern implementation: Retail

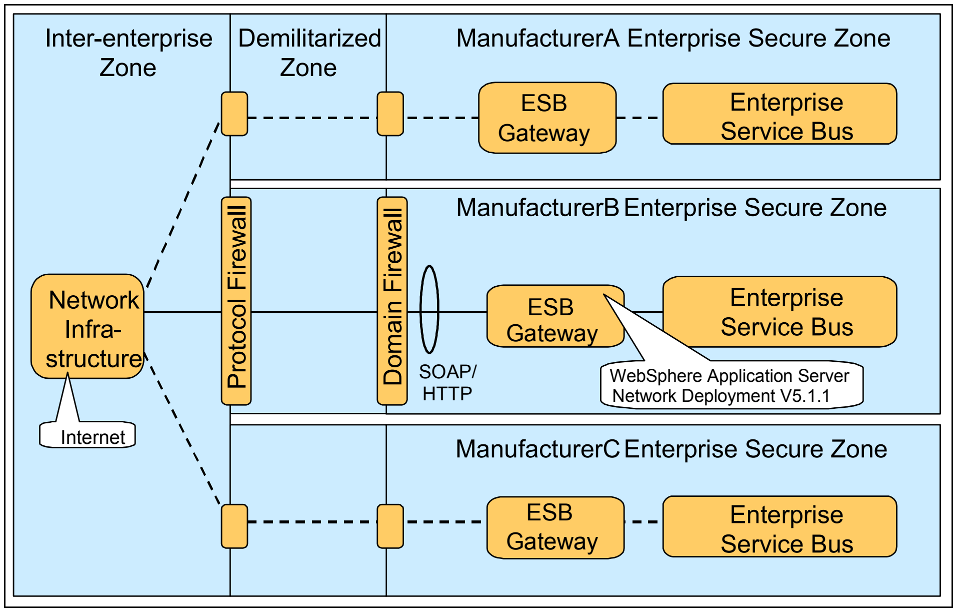


Figure 11-7 Exposed ESB Gateway composite pattern implementation : Manufacturer

Figure 11-8 on page 307 illustrates a different view of our solution to the scenario requirements in terms of the service clients, service provider, ESB, and protocols.

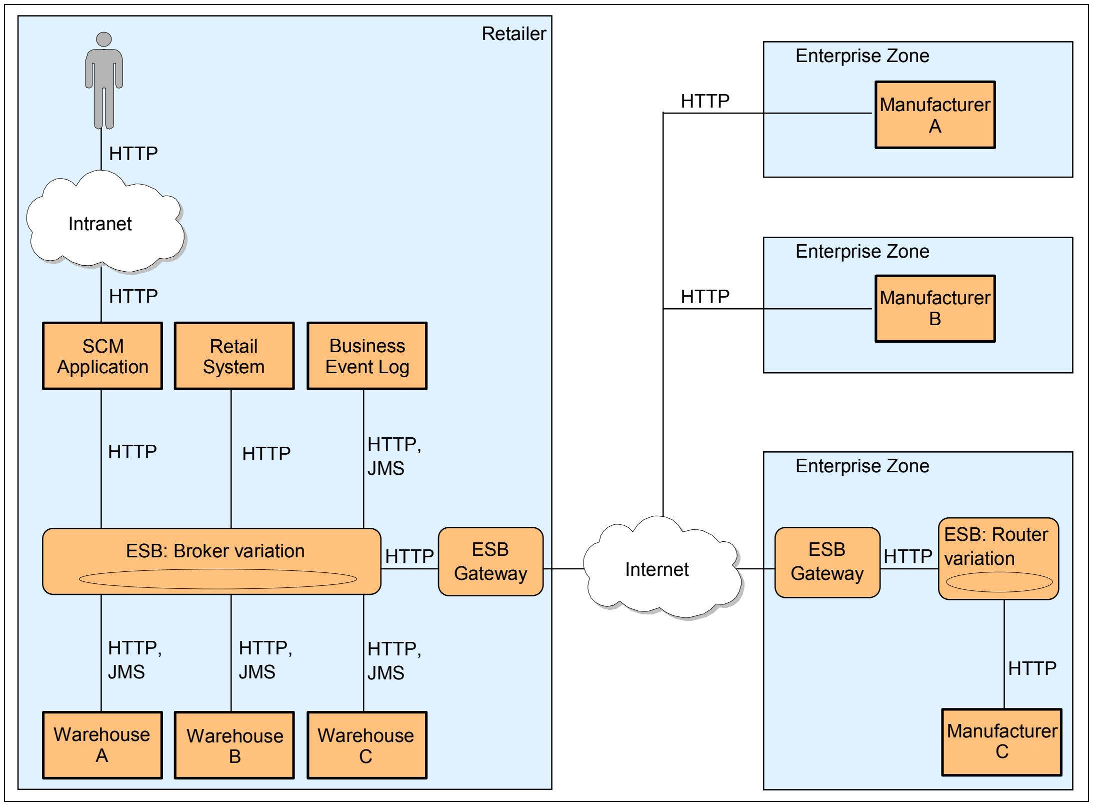


Figure 11-8 Solution to the scenario requirements

11.2.2 ESB Gateway design

Many of the design guidelines that are provided in 8.2.2, “Router variation” on page 181 apply to the ESB Gateway. Additional considerations are discussed here to reflect the following differences:

- The ESB Gateway is not part of the ESB.
- The ESB Gateway is a gateway to and from the Internet.

The starting point for these design discussions is an implemented ESB in the Retailer organization and an implemented ESB in one of the Manufacturers.

Design alternative: sharing of WSDL definitions

Invocation of inter-enterprise services implicitly requires the sharing of service definitions in the form of WSDL definition files. This can be achieved in several ways:

- In its most elementary form, organizations can simply provide WSDL definition files to trading partners. This means that WSDL definitions are common and static across the organizations. Changes to the definitions take time to roll out and must be made with the agreement of both the service provider and service requesters: Agreement must be reached between multiple organizations.
- More sophisticated techniques could include the population of shared UDDI directories, avoiding the need to share service definitions in advance of execution. This enables the service definition to be identified dynamically at runtime. It is up to the service provider to maintain the WSDL definition and for service requesters to discover the current version and bind to it dynamically at runtime.

Design alternative: service discovery timing

There are two alternatives when deciding when to discover service definitions.

- Service client development time

  Developers or configurers of the service client must import the WSDL definitions that are provided by the service provider prior to deployment and runtime execution. This leads to higher performance but lower flexibility over the alternative approach.
- Service client execution time

  Higher flexibility is possible at execution time because the service can be discovered dynamically. The service client then binds to the provider. However, this adds processing and reduces performance.

  An additional consideration is that this dynamic approach may improve certain qualities of service. For example:

  – If a service that has been discovered turns out to be unavailable then it enables another to be invoked.

  – If the algorithm for choosing a service provider may result in different services being selected over time.

Dynamic service discovery requires a directory.

Design alternative: managing the service namespace

The first point to recognize is that lookup of services in the ESB and lookup of services by the ESB Gateway are required. However, they are separate requirements because an ESB manages its own service namespace and not that of an ESB Gateway as well.

Options for directories include:

- URL to a WSDL definition

  In our scenario implementation of the Router variation in the ESB in Chapter 8, “Enterprise Service Bus: Router variation” on page 175, we look up the service in the ESB using a URL in the Web Services Gateway from a WSDL definition on the HTTP server.

  If client requests to the ESB Gateway require transformation to invoke the service provider, then development effort is required. This might be to implement a JAX-RPC handler in the case of the Web Services Gateway in our scenario.

  This option enables easy switching from SOAP/JMS inside the enterprise to or from SOAP/HTTP(S) outside.

- UDDI

  A UDDI registry can be used by the ESB Gateway to obtain the interface description and the implementation description of the Web services dynamically at execution time. This can be from public directories provided by organizations such as IBM, or by accessing privately managed directories (within or outside the enterprise).

  Although it may be easier in the short term to simply put the WSDL files of the Web services on a Web server than to implement a solution using a public or a private registry, consider the following trade-offs:

  - Web sites do not have a discovery protocol that enables consumers to search for and download WSDL files.
  - As the use of Web services becomes more popular, consumers are increasingly likely to use a UDDI registry to find Web services.
  - UDDI registries allow a fine degree of classification for Web services that enables consumers to quickly find Web services to fit their needs.
  - For dynamic service discovery at execution time, the trade-off is performance.

  For further information about UDDI, see Chapter 8, “Service Directory”, in *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303. For an example using UDDI with the Web Services Gateway, see the redbook *Patterns: Broker Interactions for Intra- and Inter-enterprise*, SG24-6075.

- Database lookup of service location

  In our scenario implementation of the Broker variation in the ESB in Chapter 9, “Enterprise Service Bus: Broker variation” on page 219, we look up the service in the ESB from a DB2 table using WebSphere Business Integration Message Broker.

This is suitable when the only the address location of the service definition is required to change other complex database schema structures and processing will be required. This would be the case when service clients of the ESB Gateway implementation use the same service interface definitions as the service providers (except for the address location): No transformation is required.

In our scenario implementation, we set the URL of the service provider in the WSDL definitions and imported them at configuration time into the Web Services Gateway. From the Web Services Gateway, the WSDL definitions are accessed via URLs.

Design alternative: security

This is usually an important topic for implementations that use the Internet for inter-enterprise communication. The main requirements are generally as follows:

- Authentication
- Authorization
- Confidentiality
- Integrity
- Non-repudiation
- Auditing

We now look at these requirements for securing Web services with the following security measures:

1. Securing the communications channel

   The choices are to use HTTP over Secure Sockets Layer (SSL) connections (HTTPS) or HTTP basic authentication.

   – HTTP basic authentication uses the HTTP header to carry user ID and password information. This information is sent over an SSL connection (HTTPS) but everything else is sent over HTTP. The mechanism tends to be limited because of the encryption key length that is used or the burden of using client-side certificates (which requires a PKI infrastructure).

   – HTTPS is widely used for secure communication over the Internet. However, it does not provide security behind the HTTPS endpoints. The entire message payload is encrypted and so this may have an adverse impact on performance. This also means that it cannot be used as a solution to meet requirements where part of the message must remain confidential between service endpoints.

     Both confidentiality and message integrity are provided with SSL.

    The use of SSL allows either or both the client and server to prove identity to the other. This can be achieved using certificates and provides an authentication mechanism.

   With a federated ESB implementation, it may make sense to only secure the communications between the intermediaries on either side of the Internet, between the ESB Gateways.

   Securing the communications channel in conjunction with J2EE security and JSR 109 can additionally address authorization requirements. The J2EE role-based authorization model can be assigned to operations of services exposed by the Web Services Gateway used in our scenario implementation.

2. XML document level security

   The choices here are to take the W3C recommendations to use XML Signature and XML Encryption to meet integrity and confidentiality requirements at the XML document level:

   - XML Encryption

     This provides the ability to encrypt certain portions of an XML document, such as part or all of the SOAP body. An XML syntax is used to represent the encrypted content and information that enables a service provider to decrypt it.

   - XML Signature

     Algorithms are available to sign XML documents, such as a SOAP envelope. XML Signature provides a mechanism for securely verifying the origin of such a message by using an XML-compliant syntax for representing the signature.

   For more information, see:

   `http://www.w3c.org`

   The use of the Structure Assertion Markup Language (SAML) standard allows the exchange of authentication and authorization information with XML between business partners. Further information can be found at:

   `http://www.oasis-open.org`

   The use of these capabilities has several drawbacks:

   - Performance may be affected adversely.
   - Current tooling may hide the XML layer from service developers.
   - A PKI infrastructure is required.

3. WS-Security

   Version 1.0 of WS-Security was ratified recently as an OASIS standard. It provides a mechanism for signing and securing Web services message exchanges. This includes adding and acting on identification and authentication information through security token propagation, providing message confidentiality and validating message integrity. These mechanisms can be used as part of a complete security solution.

The requirements that are addressed by the specification are to support:

- Multiple security tokens for authentication or authorization
- Multiple trust domains
- Multiple encryption technologies
- End-to-end message-level security (not just at the transport level)

For a federated ESB implementation, it makes sense to consider using WS-Security between the service endpoints rather than between specific pairs of intermediaries, such as ESBs and ESB Gateways. However, it may be appropriate to use WS-Security between ESB Gateways over the Internet because the increased openness allows encryption of part or all of the SOAP body only, for example.

Furthermore, whereas HTTPS with client authentication is implemented at a transport level with no fine level of control for services, WS-Security can be used to provide a high degree of flexibility for addressing security requirements for services, including the authentication of a service request from one ESB Gateway by another.

Further information on WS-Security can be found at:

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

Non-repudiation is not discussed in this book. Auditing can be a characteristic of products that are used to implement the ESB Gateway.

Design alternative: common WSDL definitions

The choice here is to use a common WSDL definition across the federated ESB for each service provider versus generating a definition at each intermediary, such as the ESB Gateway.

By changing the definition, transformation is required that demands resources at both implementation and execution time. However, control over the service interface may not be possible:

- Between enterprises because agreement between the enterprise providing the service and potential multiple enterprises consuming the service may not be possible or realistic.
- At the service provider endpoint that might result from taking a bottom-up approach to service definition with an existing system.

Design alternative: locating the HTTP server

When implementing an ESB Gateway, it is common for a standalone HTTP service to the deployed into the DMZ between two firewalls. In this case, HTTPS can be used to secure the communication link over the Internet for incoming service requests to the HTTP server. This then forwards the requests to the ESB Gateway sitting on the intranet using HTTPS. The ESB Gateway then converts the protocol to HTTP (or JMS) and makes a request to the ESB on behalf of the original requestor. This is shown in Figure 11-9.

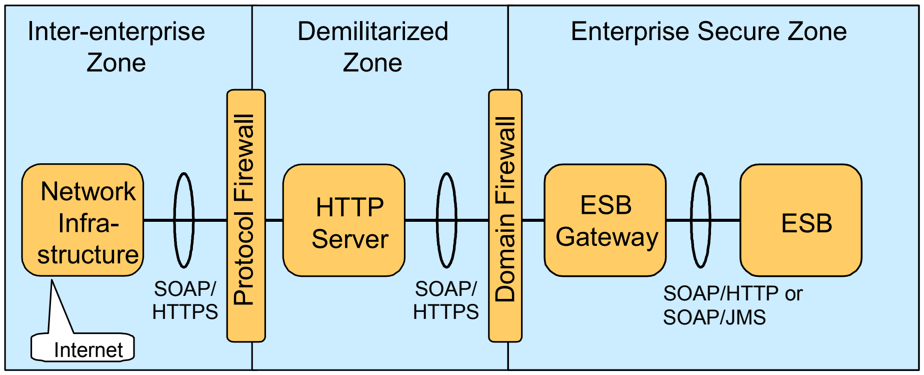


Figure 11-9 Infrastructure to receive service requests over the Internet

A full implementation of our scenario in this configuration would use the Web Services Gateway to implement the ESB Gateway.

Note that WebSphere Application Server provides an embedded HTTP server that avoids requests from HTTP clients that have to access a separate HTTP server. Further detail and alternative topology options are described in Chapter 3 of *IBM WebSphere Application Server V5.1 System Management and Configuration WebSphere Handbook Series,* SG24-6195.

Design alternative: change of namespace

In our scenario, the Manufacturers are divested. It is realistic to expect that the namespaces that are used to identify the target service will change to reflect the new organizations' identities. For the Retail enterprise, the changes would be reflected in its ESB Gateway implementation and not in the ESB because the ESB should not be aware that the provider of a service it requests is in another enterprise.

Design alternative: inter-enterprise process management

A B2B transaction may require business process management of several interactions between enterprises. Where one interaction depends upon at least one previous interaction then this should not be implemented as part of the ESB Gateway. Neither should it be implemented as part of the ESB. The business process manager is a separate component that makes service calls to the ESB.

11.3 Runtime guidelines

The starting point for the development of this scenario is the implementation of Chapter 9, “Enterprise Service Bus: Broker variation” on page 219. This was built on server appsrv1l.itso.ral.ibm.com using WebSphere Business Integration Message Broker to implement the ESB.

For the implementation in this chapter we show the divesting of one of the Manufacturers. This was moved to a new server, entsrv1w.itso.ral.ibm.com.

We see that some of the benefits of using an ESB over the point-to-point setup of the original starting point of the scenario were realized in the development of the scenario in this chapter. All the changes were relatively trivial configuration changes.

The movement of the Manufacturer service affects three service invocations that must now pass between the two servers. These are:

- The replenishment of stock ordered by the Warehouse with the Manufacturer.
- The callback by the Manufacturer to the Warehouse with the shipping note.
- Calls made by the Manufacturer to the LoggingFacility to record these events:
  - Replenishing the stock
  - Producing and shipping the additional stock

11.3.1 Transfer of the Manufacturer service implementation

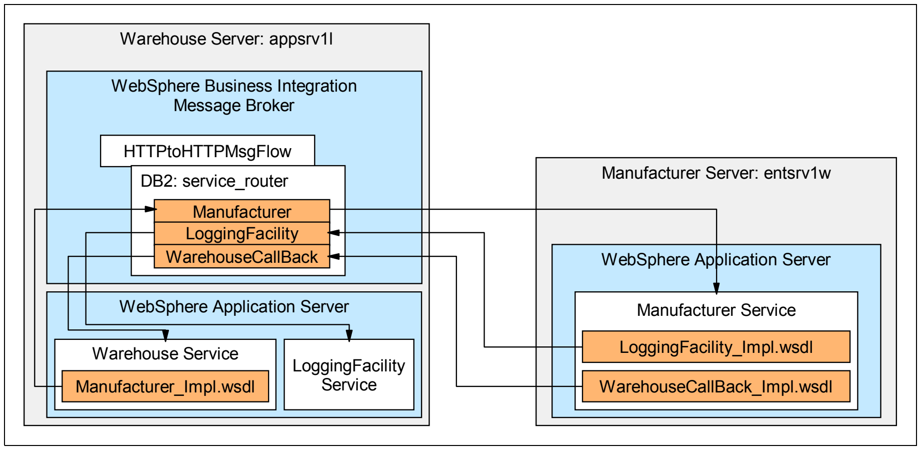
Only one change was needed to enable the scenario application to function as before: to update the service routing entry in the ESB for the Manufacturer service by issuing the command in Example 11-1 in DB2 UDB.

Example 11-1 Amending the service namespace entry for Manufacturer

```
UPDATE service_router
SET location = ‘http://entsrv1w.itso.ral.ibm.com:9080/Manufacturer/services/Manufacturer’
WHERE name = ‘Manufacturer’
```

This entry is used by the message flow HTTPtoHTTPMsgFlow in WebSphere Business Integration Message Broker to look up the target service location.

Figure 11-10 shows the resulting configuration.

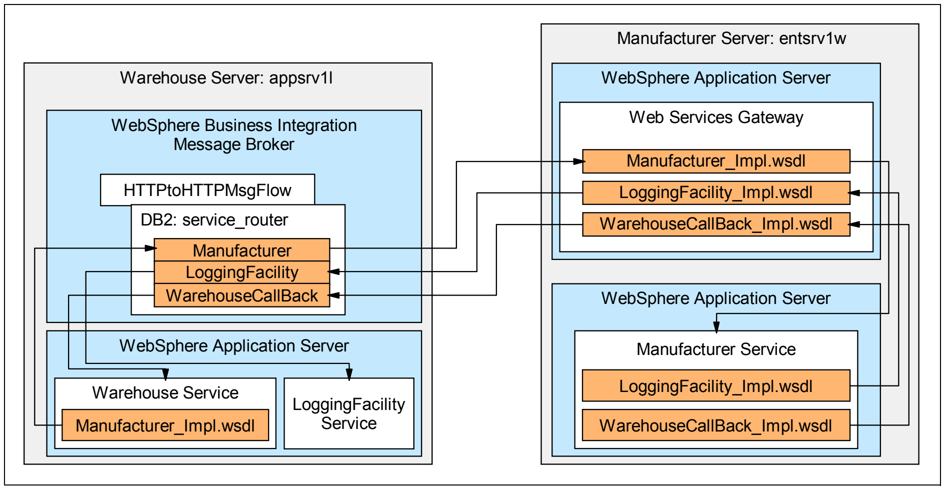


Warehouse Server: appsrv1l
WebSphere Business Integration Message Broker
HTTPtoHTTPMsgFlow
DB2: service_router
Manufacturer
LoggingFacility
WarehouseCallBack
WebSphere Application Server
Warehouse Service
Manufacturer_Impl.wsdl
LoggingFacility Service
Manufacturer Server: entsrv1w
WebSphere Application Server
Manufacturer Service
LoggingFacility_Impl.wsdl
WarehouseCallBack_Impl.wsdl

Figure 11-10 The infrastructure with the Manufacturer service installed remotely

11.3.2 Configuration of a single gateway

We next configured the Web Services Gateway on the Manufacturer's server, entsrv1. Figure 11-11 shows the target configuration.



Warehouse Server: appsrv1l
WebSphere Business Integration Message Broker
HTTPtoHTTPMsgFlow
DB2: service_router
Manufacturer
LoggingFacility
WarehouseCallBack
WebSphere Application Server
Warehouse Service
Manufacturer_Impl.wsdl
LoggingFacility Service
Manufacturer Server: entsrv1w
WebSphere Application Server
Web Services Gateway
Manufacturer_Impl.wsdl
LoggingFacility_Impl.wsdl
WarehouseCallBack_Impl.wsdl
WebSphere Application Server
Manufacturer Service
LoggingFacility_Impl.wsdl
WarehouseCallBack_Impl.wsdl

Figure 11-11 The infrastructure with one gateway configured

The detailed steps that are required for this task are covered in 11.3, "Runtime guidelines" on page 314, and this is a summary:

1. The Web Services Gateway was installed on entsrv1w.

2. Configure the gateway with the settings shown in Table 11-1.

Table 11-1 Settings for configuring the gateway

Attribute	Value
Namespace URI for services	urn:entsrv1w.itso.ral.ibm.com
WSDL URI for exported definitions	http://entsrv1w.itso.ral.ibm.com/wsgw

3. Deploy a SOAP/HTTP channel in the Web Services Gateway with the settings shown in Table 11-2.

Table 11-2 Settings for deploying a channel

Attribute	Value
Channel Name	SOAPHTTPChannel1
Home Location	SOAPHTTPChannel1Bean
End Point Address	http://entsrv1w.itso.ral.ibm.com/wsgwsoaphttp1

4. Set up the service locations for the three service invocations.

   If it doesn't already exist, create a new directory in the HTTP server on entsrv1w called `wsdl` in C:\<IBM HTTP Server>\htdocs\en_US.

   – Replenish stock order from the Warehouse to the Manufacturer.

     Update the service router table in DB2 UDB for WebSphere Business Integration Message Broker to point to the Web Services Gateway on entsrv1w with the statement in Example 11-2.

Example 11-2 Updating the routing entry in DB2 UDB

```
UPDATE service_router
SET location =
‘http://entsrv1w.itso.ral.ibm.com/wsgwsoaphttp1/soaphttpengine/urn%3Aentsrv1w.itso.ral.ibm.com%
23ManufacturerWsgwService’
WHERE name = ‘Manufacturer’
```

If not already present, copy the following files from the Warehouse to the new directory on entsrv1w:

- Manufacturer_Impl.wsdl
- Manufacturer.wsdl
- Configuration.wsdl
- ManufacturerPO.xsd
- ManufacturerSN.xsd
- Configuration.xsd

In the Manufacturer_Impl.wsdl file, set the import location attribute paths to:

```
http://entsrv1w.itso.ral.ibm.com/wsdl/
```

Then specify the SOAP address location to point to `Manufacturer` on `entsrv1w` as follows:

```
<wsdlsoap:address
location="http://entsrv1w.itso.ral.ibm.com:9080/Manufacturer/services
/Manufacturer"/>
```

Note that this is only necessary because we implemented an ESB and the original WSDL in the Warehouse pointed to WebSphere Business Integration Message Broker.

– Logging Facility calls from the Manufacturer

  Update the LoggingFacility_Impl.wsdl for the Manufacturer EJB in WebSphere Studio Application Developer to point to Web Services Gateway by setting the SOAP address location as follows:

```
<wsdlsoap:address
location="http://entsrv1w.itso.ral.ibm.com/wsgwsoaphttp1/soaphttpengi
ne/urn%3Aentsrv1w.itso.ral.ibm.com%23LoggingFacilityWsgwService"/>
```

  Do not export the Manufacturer.ear file and reinstalling the Manufacturer service in WebSphere Application Server for now.

  If they are not already present, copy the following files to the new directory on entsrv1w as before:

  - LoggingFacility_Impl.wsdl
  - LoggingFacility.wsdl
  - LoggingFacility.xsd

  Set the import location attributes as before.

  Set the SOAP address location attribute to point to the broker on appsrv1l as follows:

```
<wsdlsoap:address
location="http://appsrv1l.itso.ral.ibm.com:7080/ServiceBroker/Logging
Facility"/>
```

– Call back from the Manufacturer to the Warehouse.

  Update the WarehouseCallBack_Impl.wsdl for the Manufacturer EJB in WebSphere Studio Application Developer to point to Web Services Gateway by setting the SOAP address location as follows:

```
<wsdlsoap:address
location="http://entsrv1w.itso.ral.ibm.com/wsgwsoaphttp1/soaphttpengi
ne/urn%3Aentsrv1w.itso.ral.ibm.com%23WarehouseCallBackWsgwService"/>
```

Export the Manufacturer.ear file and reinstall the Manufacturer service in WebSphere Application Server on entsrv1w.

If not already present, copy the following files to the new directory on entsrv1w as before:

- WarehouseCallBack_Impl.wsdl
- Warehouse.wsdl
- Warehouse.xsd

Set the import location attributes as before.

Set the SOAP address location attribute to point to the broker on appsrv1l as follows:

```
<wsdlsoap:address
location="http://appsrv1l.itso.ral.ibm.com:7080/ServiceBroker/Warehou
seCallBack"/>
```

5. Three services were deployed in Web Services Gateway: one for each of the service invocations that are required to support the Manufacturer. The settings are shown in Table 11-3, Table 11-4, and Table 11-5 on page 318.

Table 11-3 Manufacturer gateway service settings

Attribute	Value
Gateway Service Name	ManufacturerWsgwService
Channels	SOAPHTTPChannel1
WSDL Location	http://entsrv1w.itso.ral.ibm.com/wsdl/Manufacturer_Impl.wsdl

Table 11-4 LoggingFacility gateway service settings

Attribute	Value
Gateway Service Name	LoggingFaciltyWsgwService
Channels	SOAPHTTPChannel1
WSDL Location	http://entsrv1w.itso.ral.ibm.com/wsdl/LoggingFacility_Impl.wsdl

Table 11-5 WarehouseCallBack gateway service settings

Attribute	Value
Gateway Service Name	WarehouseCallBackWsgwService
Channels	SOAPHTTPChannel1

Attribute	Value
WSDL Location	http://entsrv1w.itso.ral.ibm.com/wsdl/WarehouseCallBack_Impl.wsdl

11.3.3 Configuring a second gateway

The benefits that are derived by using an ESB (and the first Web Services Gateway) are now becoming considerable because we are now isolated from the endpoint services. In fact, the configuration to insert the second Web Services Gateway was quicker to complete than the installation of the product itself.

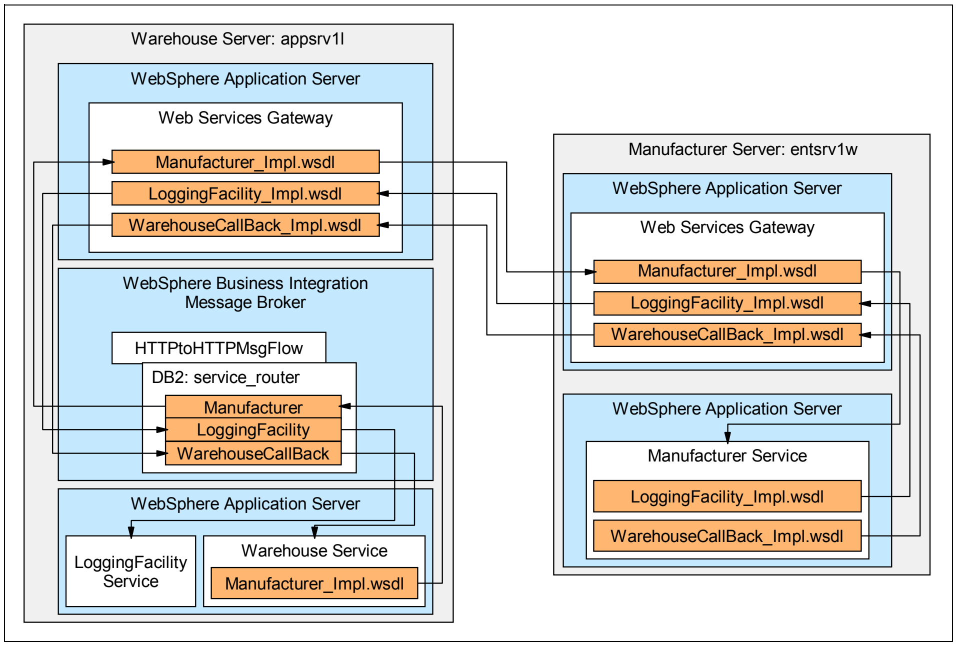

Warehouse Server: appsrv1l
WebSphere Application Server
Web Services Gateway
Manufacturer_Impl.wsdl
LoggingFacility_Impl.wsdl
WarehouseCallBack_Impl.wsdl
WebSphere Business Integration Message Broker
HTTPtoHTTPMsgFlow
DB2: service_router
Manufacturer
LoggingFacility
WarehouseCallBack
WebSphere Application Server
LoggingFacility Service
Warehouse Service
Manufacturer_Impl.wsdl
Manufacturer Server: entsrv1w
WebSphere Application Server
Web Services Gateway
Manufacturer_Impl.wsdl
LoggingFacility_Impl.wsdl
WarehouseCallBack_Impl.wsdl
WebSphere Application Server
Manufacturer Service
LoggingFacility_Impl.wsdl
WarehouseCallBack_Impl.wsdl

Figure 11-12 The infrastructure with two configured Web Services Gateways

The resulting configuration is shown in Figure 11-12 on page 319. For the configuration steps, we follow the same summary style as was used in the previous section.

1. The Web Services Gateway was installed on appsrv1l.
2. Configure the gateway with the settings shown in Table 11-6.

Table 11-6 Settings for configuring the gateway

Attribute	**Value**
Namespace URI for services	urn:appsrv1l.itso.ral.ibm.com
WSDL URI for exported definitions	http://appsrv1l.itso.ral.ibm.com/wsgw

3. Deploy a SOAP/HTTP channel in the gateway with the settings shown in Table 11-7.

Table 11-7 Setting for deploying a channel

Attribute	**Value**
Channel Name	SOAPHTTPChannel1
Home Location	SOAPHTTPChannel1Bean
End Point Address	http://appsrv1l.itso.ral.ibm.com/wsgwsoaphttp1

4. Set up the service locations for the three service invocations.

   If it doesn't already exist, create a new directory in the HTTP server on appsrv1l called `wsdl` in C:\<IBM HTTP Server>\htdocs\en_US.

   – Replenish stock order from the Warehouse to the Manufacturer.

     Update the service router table in DB2 UDB for WebSphere Business Integration Message Broker to point to the gateway on appsrv1l with the statement in Example 11-3.

Example 11-3 Updating the routing entry in DB2 UDB

```
UPDATE service_router
SET location =
'http://appsrv1l.itso.ral.ibm.com/wsgwsoaphttp1/soaphttpengine/urn%3Aappsrv1l.itso.ral.ibm.com%
23ManufacturerWsgwService'
WHERE name = 'Manufacturer'
```

If they are not already present, copy these files from the Warehouse to the new directory on appsrv1l:

- Manufacturer_Impl.wsdl
- Manufacturer.wsdl
- Configuration.wsdl
- ManufacturerPO.xsd
- ManufacturerSN.xsd
- Configuration.xsd

In the Manufacturer_Impl.wsdl file, set the import location attribute paths to:

```
http://appsrv1l.itso.ral.ibm.com/wsdl/
```

Then specify the SOAP address location to point to the Web Services Gateway on entsrv1w as follows:

```
<wsdlsoap:address
location="http://entsrv1w.itso.ral.ibm.com/wsgwsoaphttp1/soaphttpengi
ne/urn%3Aentsrv1w.itso.ral.ibm.com%23ManufacturerWsgwService"/>
```

– Logging facility calls from the Manufacturer

  In the LoggingFacility_Impl.wsdl file for the Web Services Gateway on entsrv1w, set the SOAP address location attribute to point to the Web Services Gateway on appsrv1l as follows:

```
<wsdlsoap:address
location="http://appsrv1l.itso.ral.ibm.com/wsgwsoaphttp1/soaphttpengi
ne/urn%3Aappsrv1l.itso.ral.ibm.com%23LoggingFacilityWsgwService"/>
```

  If they are not already present, copy these files to the new directory on appsrv1l as before:

  - LoggingFacility_Impl.wsdl
  - LoggingFacility.wsdl
  - LoggingFacility.xsd

  Set the import location attributes as before.

  Set the SOAP address location attribute in LoggingFacility_Impl.wsdl to point to the broker on appsrv1l as follows:

```
<wsdlsoap:address
location="http://appsrv1l.itso.ral.ibm.com:7080/ServiceBroker/Logging
Facility"/>
```

– Call back from the Manufacturer to the Warehouse.

  In the WarehouseCallBack_Impl.wsdl file for Web Services Gateway on entsrv1w, set the SOAP address location attribute to point to the Web Services Gateway on appsrv1l as follows:

```
<wsdlsoap:address
location="http://appsrv1l.itso.ral.ibm.com/wsgwsoaphttp1/soaphttpengi
ne/urn%3Aappsrv1l.itso.ral.ibm.com%23WarehouseCallBackWsgwService"/>
```

  If they are not already present, copy these files to the new directory on appsrv1l as before:

  - WarehouseCallBack_Impl.wsdl
  - Warehouse.wsdl

- Warehouse.xsd

  Set the import location attributes as before.

  Set the SOAP address location attribute in WarehouseCallBack_Impl.wsd to point to the broker on appsrv1l as follows:

```
<wsdlsoap:address
location="http://appsrv1l.itso.ral.ibm.com:7080/ServiceBroker/Warehou
seCallBack"/>
```

5. Three services were deployed in the Web Services Gateway: one for each of the service invocations that are required to support the Manufacturer. The settings are shown in Table 11-8, Table 11-9, and Table 11-10 on page 322.

Table 11-8 Manufacturer gateway service settings

Attribute	Value
Gateway Service Name	ManufacturerWsgwService
Channels	SOAPHTTPChannel1
WSDL Location	http://appsrv1l.itso.ral.ibm.com/wsdl/Manufacturer_Impl.wsdl

Table 11-9 LoggingFacility gateway service settings

Attribute	Value
Gateway Service Name	LoggingFaciltyWsgwService
Channels	SOAPHTTPChannel1
WSDL Location	http://appsrv1l.itso.ral.ibm.com/wsdl/LoggingFacility_Impl.wsdl

Table 11-10 WarehouseCallBack gateway service settings

Attribute	Value
Gateway Service Name	WarehouseCallBackWsgwService
Channels	SOAPHTTPChannel1
WSDL Location	http://appsrv1l.itso.ral.ibm.com/wsdl/WarehouseCallBack_Impl.wsdl

6. Redeploy the following services in the Web Services Gateway on entsrv1w after removing and adding a new WSDL location for each to pick up the updated files.

   – LoggingFacilityWsgwService

– WarehouseCallBackWsgwService

Note: When inserting and configuring the new Web Services Gateway, we only needed to make changes to routing information in the ESB and the Web Services Gateway on either side of the new Web Services Gateway. No other changes were required. In particular, our endpoints remained unchanged.

11.3.4 Securing the communications channel

In this section, we secure the communications link between the two Web Services Gateways for the three service operations using HTTPS (Figure 11-13 on page 323). Note that no changes were required in the service endpoints or the ESB.

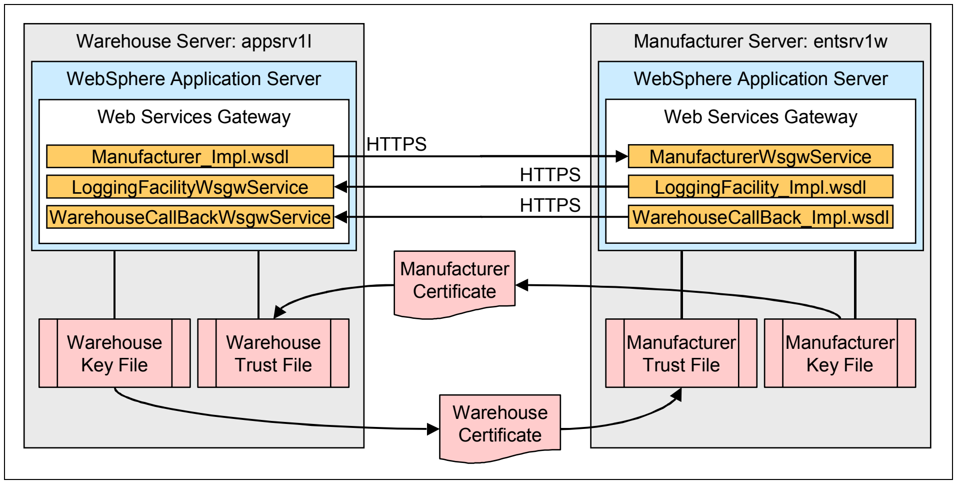


Figure 11-13 Securing the communications link

Generating self-signed certificates

Here we use the IBM key management tool, ikeyman, which is provided with WebSphere Application Server to generate the self-signed certificates and key databases. It can be found in C:\<WebSphere Application Server>\bin.

Three types of files will be created:

- Key database file

  This is a password-protected file that holds certificates of the organization that are associated with the organization’s public and private keys.

- Trust database file

  This is a password-protected file that holds the certificates of other trusted organizations that are associated with those organizations' public keys.
- Certificate file

  These are individual certificate files that are associated with a public key of an organization that are provided by that organization to others.

For these first two steps we first show the configuration for the Warehouse that is making a request to the Manufacturer to replenish stock. In this case, the Warehouse is the client that is running on server appsrv1l. The service provider, Manufacturer, which is running on server entsrv1w, is the server.

1. Generating a certificate for the Manufacturer on entsrv1w.
   a. Launch **`ikeyman.bat`** from a DOS command prompt on entsrv1w.
   b. Select **Key Database File** → **New** from the menu.
   c. Using JKS as the Key database type, type `ManufacturerKey.jks` for the File Name. Enter **C:\kdb** (this is where we store our files) for the Location. Click **OK**.
   d. Enter a password for the key database and click **OK**.
   e. A list of signer certificates appears. Select all of them and click **Delete** to delete the entire list.
   f. Now select **Create** → **New Self-Signed** Certificate from the menu.
   g. Enter `manufacturercert` for the label and use X509 V3 as the version and 1024 as the key size. Enter `Manufacturer` for the common name and `Manufacturer` for the organization. Take the defaults for country and validity period. Click **OK**.
   h. Select the generated certificate and click **Extract Certificate**.
   i. Use **Base64-encoded ASCII data** as the Data type.
   j. Enter `ManufacturerCert.arm` as the filename and `C:\kdb` as the location and click **OK**.
   k. Close the file by selecting **Key Database File** → **Close** from the menu.
2. Generating a trust file using the Manufacturer's certificate for the Warehouse on appsrv1l.
   a. Copy **ManufacturerCert.arm** from C:\kdb on entsrv1w to C:\kdb on appsrv1l.
   b. Launch **`ikeyman`** on appsrv1l.
   c. Select **Key Database File** → **New** from the menu.

   d. Using JKS as the Key database type, type `WarehouseTrust.jks` for the File Name and **C:**`\kdb` for the Location. Click **OK**.
   e. Enter a password for the trust database and click **OK**.
   f. A list of signer certificates appears. Select all of them and click **Delete** to delete the entire list.
   g. Ensure that Signer Certificates is selected from the pull-down list in the Key Database Content panel and click **Add**.
   h. Enter `ManufacturerCert.arm` for the filename and `C:\kdb` for the location. Click **OK**.
   i. Enter `manufacturercert` for the label.
   j. Close the file by selecting **Key Database File** → **Close** from the menu.

Step 3 on page 325 is required to implement client authentication using a certificate for the replenish stock service operation and to enable the service requests from the Manufacturer to the Warehouse (the callback) and LoggingFacility (logEvent) to be secured.

3. Follow steps 1 on page 324 and 2 on page 324 to create the client side certificate by swapping `warehouse` for manufacturer and `appsrv1l` for entsrv1w (and vice versa).

Setting up WebSphere Application Server

1. Configuring the server side on the Manufacturer server, entsrv1w.
   a. Launch the WebSphere Administration console on the Manufacturer server, entsrv1w.
   b. Select **Security** → **SSL**. Click **New**.
   c. Enter `ManufacturerSSLSettings` for the Alias.
   d. For the Key File Name, enter `C:\kdb\ManufacturerKey.jks`, and enter its password in Key File Password. The Key File Format is JKS.
   e. For the Trust File Name, enter `C:\kdb\ManufacturerTrust.jks`, and enter its password in Trust File Password. The Trust File Format is JKS.
   f. Do *not* check the Client Authentication check box. Take the remaining default values and click **OK**.
   g. In the left-side panel, click **Servers** → **Application Servers** and **server1**. Click **Web Container** in Additional Properties and then **HTTP transports** in Additional Properties.
   h. Click the * for port 9443.
   i. Ensure that the Enable SSL check box is checked and select **<hostname>/ManufacturerSSLSettings** from the SSL pull-down list.

j. Click **OK** and save the configuration changes.

2. Configuring the client side on the Warehouse server, appsrv1l.

   a. Check that there is a copy of the ibmjsse.jar file in the <WebSphere Application Server>/java/jre/lib/ext directory.

   b. Edit the security properties file <WebSphere Application Server>/java/jre/lib/security/java.security so that it includes entries for both the Sun security provider and the IBM security provider. You should add the entries for sun.security.provider.Sun and com.ibm.jsse.IBMJSSEProvider to what already exists if not already present, and the result might look like Example 11-4 on page 326.

Example 11-4 Entries in java.security

```
...
security.provider.1=com.ibm.crypto.provider.IBMJCE
security.provider.2=sun.security.provider.Sun
security.provider.3=com.ibm.jsse.IBMJSSEProvider
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
security.provider.6=com.ibm.crypto.pkcs11.provider.IBMPKCS11
...
```

   The order is significant. The Sun security provider must come before the IBM provider.

   c. Launch the WebSphere Administration Console on the Manufacturer server, appsrv1l.

   d. Set up the following system properties:

      - javax.net.ssl.trustStore

        Value: <Your trust store database file location>

      - javax.net.ssl.trustStorePassword

        Value: <Your trust store database password>

      - java.protocol.handler.pkgs

        Value: com.ibm.net.ssl.internal.www.protocol

        This is the IBM reference implementation.

      You do this by navigating as follows:

      i. Click **Server** → **Application Servers** and **server1**.

      ii. Under Additional Properties, click **Process Definition**.

      iii. Under Additional Properties, click **Java Virtual Machine**.

e. For the Generic JVM arguments field, enter the value in Example 11-5. (Each name value pair is prefixed with -D and is separated from the next with a space.)

Example 11-5 Value for Generic JVM arguments field

```
-Djava.protocol.handler.pkgs=com.ibm.net.ssl.internal.www.protocol
-Djavax.net.ssl.trustStore=C:\kdb\WarehouseTrust.jks
-Djavax.net.ssl.trustStorePassword=<password>
```

If you are enabling client authentication or security for each Manufacturer as a service requestor, then step 3 on page 327 is required.

3. Follow steps 1 on page 325 and 2 on page 326, swapping `warehouse` for manufacturer and `appsrv1l` for entsrv1w (and vice versa).

Setting up the Web Services Gateway

1. Perform the following steps on the Warehouse server, appsrv1l.
   a. In the HTTP server, locate the Manufacturer_Impl.wsdl file in C:\<IBM HTTP Server>\htdocs\en_US\wsdl.
   b. Update the SOAP address location by changing the initial HTTP to `HTTPS` and adding the port number `9443` as follows:
   ```
   <wsdlsoap:address
   location="https://entsrv1w.itso.ral.ibm.com:9443/wsgwsoaphttp1/soapht
   tpengine/urn%3Aentsrv1w.itso.ral.ibm.com%23ManufacturerWsgwService"/>
   ```
   c. Redeploy the service ManufacturerWsgwService in the Web Services Gateway on appsrv1l using this amended WSDL definition, as shown in Table 11-7 on page 320.
2. Perform the following steps on the Manufacturer server, entsrv1w:
   a. In the HTTP server, locate the LoggingFacility_Impl.wsdl file in C:\<IBM HTTP Server>\htdocs\en_US\wsdl.
   b. Update the SOAP address location by changing the initial HTTP to `HTTPS` and adding the port number `9443` as follows:
   ```
   <wsdlsoap:address
   location="https://appsrv1l.itso.ral.ibm.com:9443/wsgwsoaphttp1/soaphttpe
   ngine/urn%3Aappsrv1l.itso.ral.ibm.com%23LoggingFacilityWsgwService"/>
   ```
   c. Redeploy the service LoggingFacilityWsgwService in the Web Services Gateway on entsrv1w using this amended WSDL definition, as shown in Table 11-4 on page 318.
   d. In the HTTP server, locate the file WarehouseCallBack_Impl.wsdl in C:\<IBM HTTP Server>\htdocs\en_US\wsdl.

   e. Update the SOAP address location by changing the initial HTTP to `HTTPS` and adding the port number `9443` as follows:

```
<wsdlsoap:address
location="https://appsrv1l.itso.ral.ibm.com:9443/wsgwsoaphttp1/soaphttpe
ngine/urn%3Aappsrv1l.itso.ral.ibm.com%23WarehouseCallBackWsgwService"/>
```

   f. Redeploy the service WarehouseCallBackWsgwService in the Web Services Gateway on entsrv1w using this amended WSDL definition, as shown in Table 11-5 on page 318.

3. Stop and start the WebSphere Application Servers on both servers, appsrv1l and entsrv1w.

11.3.5 Further runtime alternatives and considerations

Time and available infrastructure when writing this book prevented the configuration of several important aspects related to implementing a federated ESB using the Web Services Gateway. Some of these are now listed:

1. Configuring HTTPS with a Web server.

   To extend our scenario implementation to include a Web server to receive incoming requests over HTTPS, see Chapter 10 in *IBM WebSphere Security V5.0,* SG24-6573.

2. Configuring client authentication with HTTPS.

3. Obtaining a certificate signed by a certificate authority (CA).

   In our scenario configuration, we used self-signed certificates. See *IBM WebSphere Security V5.0,* SG24-6573, to learn how to request a certificate signed by a certificate authority.

4. Configuring proxy security.

   The Web Services Gateway requires access to the Internet for invoking Web services and potentially for retrieval of WSDL files. Many enterprise installations use a proxy server in support of Internet routing, and many proxy servers require authentication before they grant access to the Internet.

   The Web Services Gateway can be configured to add username and password details for outbound service requests so that accesses to the Internet by HTTP clients can be validated by such proxy servers.

   For more details, see “Enabling proxy authentication for the gateway” in the WebSphere Application Server Information Center at:

   `http://www.ibm.com/software/webservers/appserv/infocenter.html`

5. Configuring WS-Security.

   Web Services Gateway can be configured for secure transmission of SOAP messages using tokens, keys, signatures, and encryption in accordance with the Web Services Security (WS-Security) draft recommendation of April 2002. WS-Security was ratified in April 2004.

See “Enabling Web Services Security (WS-Security) for the gateway” in the WebSphere Application Server Information Center for further details.

6. Changing the namespace of the Manufacturer in our scenario.

   This requires configuration in the Web Services Gateway.

11.4 Further information

- Olaf Zimmermann, Mark Tomlinson, Stefan Peuser, *Perspectives on Web Services,* Springer, 2003, ISBN 3-540-00914-0
- *IBM WebSphere Application Server V5.1 System Management and Configuration WebSphere Handbook Series,* SG24-6195
- *IBM WebSphere Security V5.0,* SG24-6573
- The WebSphere Application Server (including Web Services Gateway) Information Center can be found at:

  `http://www.ibm.com/software/webservers/appserv/infocenter.html`

Part 4

Appendixes

A

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

```
ftp://www.redbooks.ibm.com/redbooks/SG246346
```

Alternatively, you can go to the IBM Redbooks Web site at:

```
ibm.com/redbooks
```

Select **Additional materials** and open the directory that corresponds with the redbook form number, SG24-6346.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
sg246346.zip	Scenario implementation zip file

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	10 MB minimum
Operating System:	Windows 2000 or XP
Processor:	Intel® Pentium® 4, 1 GHz or faster
Memory:	512 MB minimum, 1 GB prefered

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

For information about how to use the scenario implementation files that are included in the additional material, refer to Appendix B, “Configuring the scenario lab environment” on page 335.

B

Configuring the scenario lab environment

This section describes the steps that are necessary for configuring a working environment to use with the scenario implementations as described in Part 3 of this redbook.

Each scenario requires a base environment to be configured. This base environment is also described in the chapter.

Source code is supplied with the redbook to help configure the environment. Consult Appendix A, “Additional material” on page 333 for information about how to obtain it.

Setting up the base environment

Important: This base environment must be configured before you can test any of the more advanced scenarios.

This describes how to configure the basic, Direct Connection pattern implementation of the scenario. It is used as the basis for adding an Enterprise Service Bus. Figure B-1 shows the protocols that are used to send SOAP messages between the Web services in this scenario.

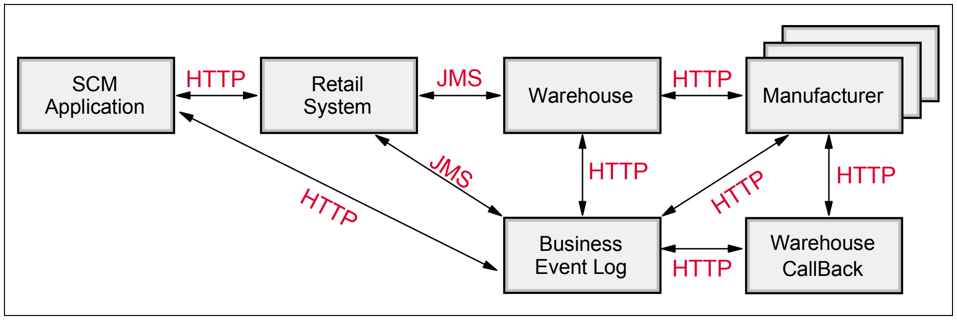


Figure B-1 Web services interactions

Creating an HTTP server to look up WSDL

Note: The scenario assumes that the Web services WSDL definitions are located on the machine appsrv1a.itso.ral.ibm.com.

Configure the following:

1. On the machine where you intend to run the scenario, open the Windows hosts file and set appsrv1a.itso.ral.ibm.com to localhost. If your WebSphere Studio workspace is on a remote machine, set the host file on this machine to point to the appsrv1a.itso.ral.ibm.com machine.
2. Install IBM HTTP Server. Create a `wsdl` directory in this server. Using the product defaults, this will create a directory in C:\Program Files\IBMHttpServer\htdocs\en_US\wsdl.
3. Copy the WSDL supplied with the redbook into this wsdl directory.
4. Test that your Web server is working by entering the following URL in a Web browser. It should show the WSDL interface for the Retailer service.

```
http://appsrv1a.itso.ral.ibm.com/wsdl/Retailer.wsdl
```

Creating a WebSphere Studio workspace

In WebSphere Studio Application Developer V5.1.1, open a new workspace, and import the following projects using **File** → **Import** → **Existing Project into Workspace**:

- Build
- LoggingFacility
- LoggingFacilityEJB
- LoggingFacilityJMS
- LoggingFacilityWeb
- Manufacturer
- ManufacturerEJB
- ManufacturerWeb
- ManufacturerB
- ManufacturerBEJB
- ManufacturerBWeb
- ManufacturerC
- ManufacturerCEJB
- ManufacturerCWeb
- Retailer
- RetailerWeb
- SCMSampleUI
- SCMSampleUIWeb
- UDDIUtility
- Warehouse
- WarehouseEJB
- WarehouseEJBRouter
- WarehouseJMS
- WarehouseWebTemp
- WarehouseB
- WarehouseBEJB
- WarehouseBEJBRouter
- WarehouseBJMS
- WarehouseC
- WarehouseCEJB
- WarehouseCEJBRouter
- WarehouseCJMS

Note that you may need to alter the Java Build Path properties of UDDIUtility to indicate the location of uddi4jv2.jar.

Exporting enterprise applications from WebSphere Studio

Export the enterprise applications from WebSphere Studio into EAR files so they can be installed into WebSphere Application Server.

1. In the J2EE perspective of WebSphere Studio, move to the J2EE Hierarchy view, and export the following enterprise applications:

   – LoggingFacility
   – Manufacturer
   – ManufacturerB
   – ManufacturerC
   – Retailer
   – SCMSampleUI
   – Warehouse

   The Warehouse enterprise application requires an additional step: The endpoint enabler must be run for it. To do this, in a command prompt navigate to the directory where you have exported Warehouse.ear, then type:

   ```
   <Studio_home>\v5.1.1\runtimes\base_v51\bin\endptenabler.bat
   ```

   You are then prompted for the name of the EAR file, which is entered as:

   ```
   Warehouse.ear
   ```

   Take the default value for the HTTP router name. The HTTP context root should be supplied as follows:

   ```
   /Warehouse
   ```

Configuring WebSphere MQ

Install WebSphere MQ V5.3 with CSD06 or above. Create a queue manager called `SOA.QUEUE.MANAGER`, with a listener set to listen on TCP/IP port `1418`. Define the following local queues from this queue manager:

- LoggingFacilityQ
- WSISampleAQ
- WSISampleBQ
- WSISampleCQ
- WarehouseQ
- WarehouseBQ
- WarehouseCQ

Configuring WebSphere Application Server

In the WebSphere Application Server Administrative Console:

1. Define the following resources for the WebSphere JMS MQ provider:
   a. Define WebSphere MQ Queue Connection Factories as shown in Table B-1.

Table B-1 WebSphere MQ Connection Factories

Name	JNDI name	Queue manager
LoggingFacilityQCF	jms/LoggingFacilityQCF	SOA.QUEUE.MANAGER
WSISampleQCF	jms/WSISampleQCF	SOA.QUEUE.MANAGER
WarehouseQCF	jms/WarehouseQCF	SOA.QUEUE.MANAGER

   b. Define WebSphere MQ Queue Destinations as shown in Table B-2.

Table B-2 WebSphere MQ Queue Destinations

Name	JNDI name	Base queue name
LoggingFacilityQ	jms/LoggingFacilityQ	LoggingFacilityQ
WSISampleAQ	jms/WSISampleAQ	WSISampleAQ
WSISampleBQ	jms/WSISampleBQ	WSISampleBQ
WSISampleCQ	jms/WSISampleCQ	WSISampleCQ
WarehouseQ	jms/WarehouseQ	WarehouseQ
WarehouseBQ	jms/WarehouseBQ	WarehouseBQ
WarehouseCQ	jms/WarehouseCQ	WarehouseCQ

   c. Define Listener Ports as shown in Table B-3.

Table B-3 Listener Ports

Name	Connection factory JNDI name	Destination JNDI name
LoggingFacilityListener	jms/LoggingFacilityQCF	jms/LoggingFacilityQ
WSISampleAListenerPort	jms/WSISampleQCF	jms/WSISampleAQ
WSISampleBListenerPort	jms/WSISampleQCF	jms/WSISampleBQ
WSISampleCListenerPort	jms/WSISampleQCF	jms/WSISampleCQ
WarehouseListenerPort	jms/WarehouseQCF	jms/WarehouseQ

Name	Connection factory JNDI name	Destination JNDI name
WarehouseBListenerPort	jms/WarehouseQCF	jms/WarehouseBQ
WarehouseCListenerPort	jms/WarehouseQCF	jms/WarehouseCQ

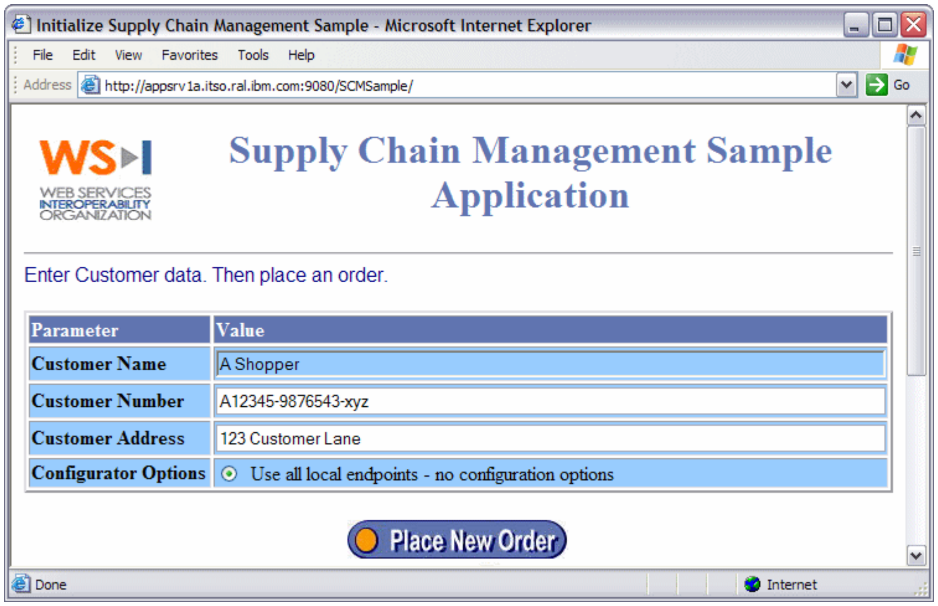
2. Restart the server to pick up the Listener Port definitions.
3. Install each of the EAR files you exported from WebSphere Studio into WebSphere Application Server.

Testing the business scenario

To test the business scenario, enter the following URL:

```
http://appsrv1a.itso.ral.ibm.com:9080/SCMSample
```

This should start the Supply Chain Management Sample Application, as shown in Figure B-2.


Initialize Supply Chain Management Sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://appsrv1a.itso.ral.ibm.com:9080/SCMSample/
Go
WS-I WEB SERVICES INTEROPERABILITY ORGANIZATION
Place New Order
Done
Internet

Supply Chain Management Sample Application

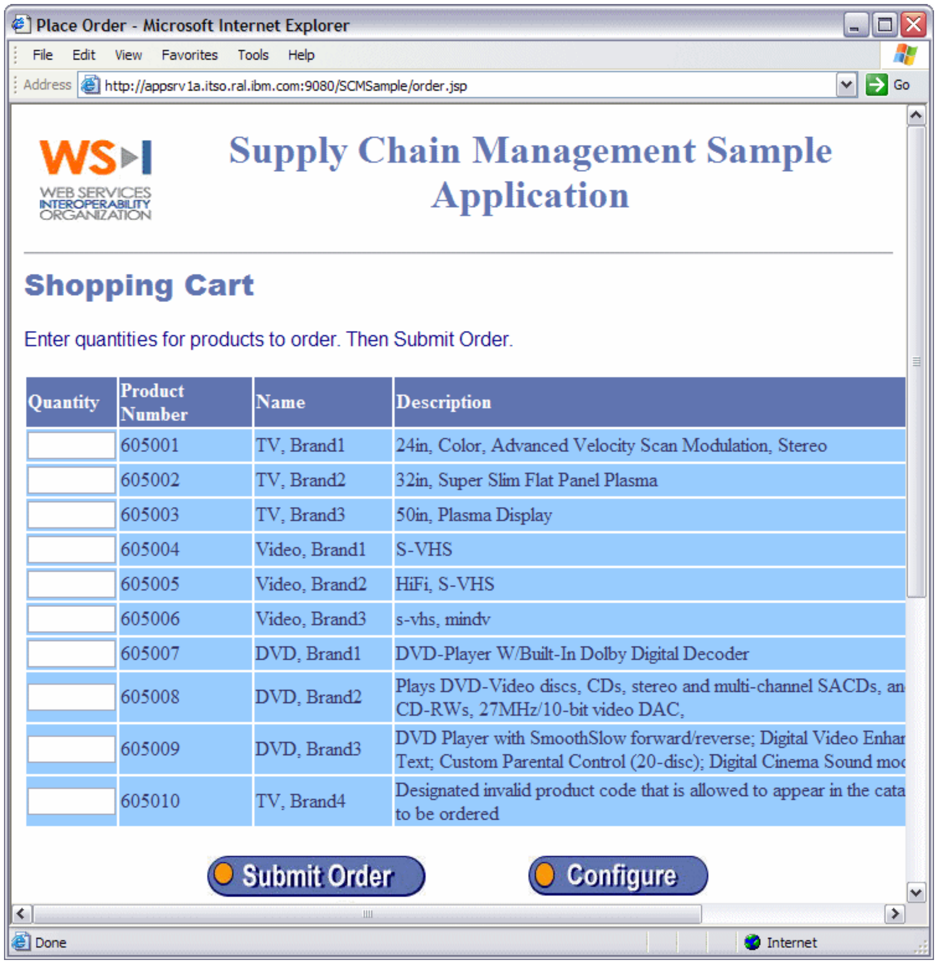
Enter Customer data. Then place an order.

Parameter	Value
Customer Name	A Shopper
Customer Number	A12345-9876543-xyz
Customer Address	123 Customer Lane
Configurator Options	● Use all local endpoints - no configuration options

Figure B-2 SCM sample application

The following steps describe how to use the SCM sample application:

1. To retrieve a list of products, click the **Place New Order** button. This displays a list of 10 products, as shown in Figure B-3 on page 341.


Place Order - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://appsrv1a.itso.ral.ibm.com:9080/SCMSample/order.jsp
Go
WS-I WEB SERVICES INTEROPERABILITY ORGANIZATION
Submit Order
Configure
Done
Internet

Supply Chain Management Sample Application

Shopping Cart

Enter quantities for products to order. Then Submit Order.

Quantity	Product Number	Name	Description
	605001	TV, Brand1	24in, Color, Advanced Velocity Scan Modulation, Stereo
	605002	TV, Brand2	32in, Super Slim Flat Panel Plasma
	605003	TV, Brand3	50in, Plasma Display
	605004	Video, Brand1	S-VHS
	605005	Video, Brand2	HiFi, S-VHS
	605006	Video, Brand3	s-vhs, mindv
	605007	DVD, Brand1	DVD-Player W/Built-In Dolby Digital Decoder
	605008	DVD, Brand2	Plays DVD-Video discs, CDs, stereo and multi-channel SACDs, an CD-RWs, 27MHz/10-bit video DAC,
	605009	DVD, Brand3	DVD Player with SmoothSlow forward/reverse; Digital Video Enhar Text; Custom Parental Control (20-disc); Digital Cinema Sound moc
	605010	TV, Brand4	Designated invalid product code that is allowed to appear in the cata to be ordered

Figure B-3 SCM sample application: shopping cart

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

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

   The warehouse stock level is stored in the org.ws_i.www.Impl.WarehouseImpl class in the WarehouseEJB project. For example, the stock level for the first three products is shown in Table B-4 on page 342.

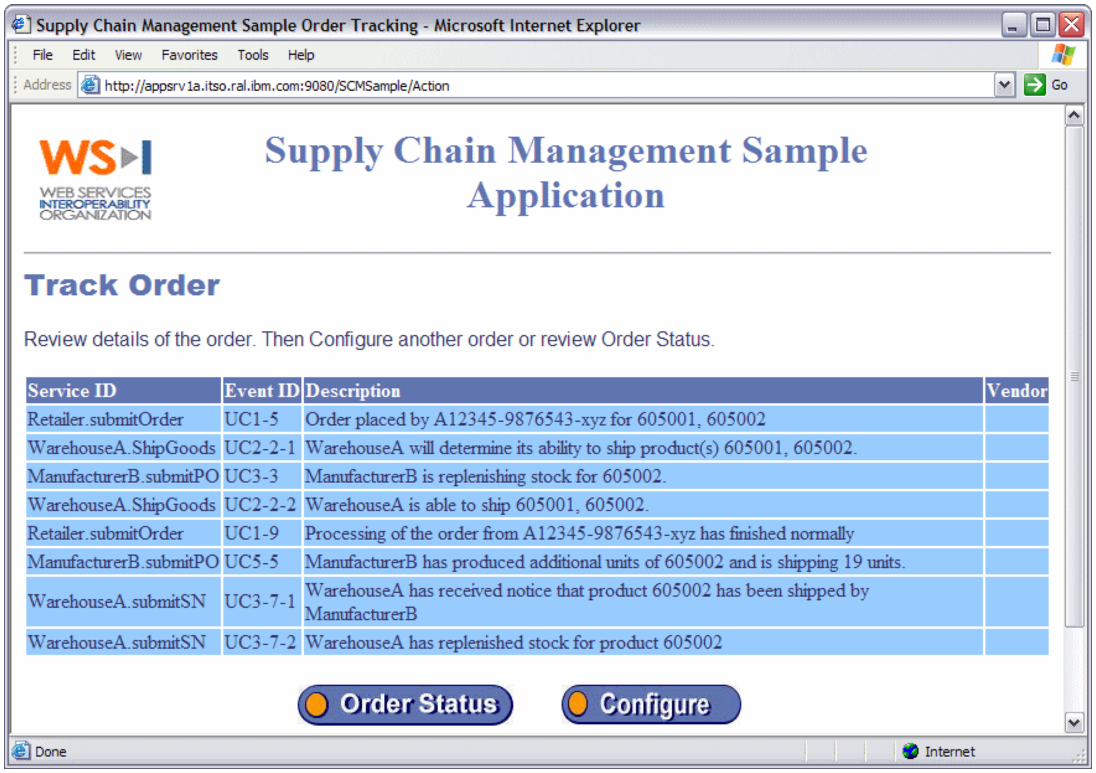
Table B-4 Warehouse stock levels

Product number	Current level	Minimum level	Maximum level
605001	10	5	25
605002	7	4	20
605003	15	10	50

If the current stock level falls below the minimum stock level, the stock is reordered so that, after the reorder has arrived, the stock will be at maximum stock level. For example, you order 6 items of 605001. This reduces the current stock level to 4 (10 - 6). A reorder will be made for 21 new items.

Each manufacturer only manufactures certain products. For example Manufacturer A manufacturers products 605001, 605004, and 605007.

3. Place orders for multiple products in the SCM sample application by entering quantities and pressing **Submit Order**. For example, order 3 items of product 605001 and 6 items of product 605002. This triggers a reorder of product 605002 with Manufacturer B.
4. The order status screen shows which orders were placed and which orders were not placed due to insufficient stock. Click **Track Order** to see the entries that were written to the LoggingFacility. As new entries are added to the Logging Facility, you must refresh this screen by clicking **Order Status** and then clicking **Track Order** again. Figure B-4 on page 343 shows the results of an order in which products 605001 and 605002 were shipped and a reorder for 19 units of product 605002 was placed with Manufacturer B.


Supply Chain Management Sample Order Tracking - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://appsrv1a.itso.ral.ibm.com:9080/SCMSample/Action
Go
WS-I WEB SERVICES INTEROPERABILITY ORGANIZATION
Done
Internet

Supply Chain Management Sample Application

Track Order

Review details of the order. Then Configure another order or review Order Status.

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

Figure B-4 SCM sample application: track order

5. To start a new order, click **Configure**. At this point, all state is lost, and the warehouse stock levels return to their default values.

Setting up the ESB Router variation scenario

After the base environment is installed you can configure the Enterprise Service Bus Router variation scenario as described in Chapter 8, “Enterprise Service Bus: Router variation” on page 175.

Setting up the ESB Broker variation scenario

Once the base environment is installed you must complete the steps in this section before implementing the Enterprise Service Bus Broker pattern scenario as described in Chapter 9, “Enterprise Service Bus: Broker variation” on page 219.

Installing WebSphere Business Integration Message Broker

Install WebSphere Business Integration Message Broker V5.0 with default settings. Run the **`mqsicreatebroker`** and **`mqsicreateconfigmgr`** commands to create a broker and configuration manger, using default values.

Setting up WebSphere MQ

In addition to the WebSphere MQ setup that was performed for the base environment, the following changes are necessary:

1. The queue manager WBRK_QM is created as part of the WebSphere Business Integration Message Broker installation. Create a listener set to listen on TCP/IP port `1415`.
2. Define the following local queues for the WBRK_QM queue manager:
   - WSHIPRESPONSE
   - SendHTTPContext
   - temp
3. Create a transmission queue for both the SOA.QUEUE.MANAGER and WBRK_QM queue managers called `XmitSenderScenario2`.
4. Create a sender channel for SOA.QUEUE.MANAGER with the settings shown in Table B-5.

Table B-5 Sender channel on SOA.QUEUE.MANAGER

Name	**Value**
Name	SOA_QM_to_WBRK_QM
Transmission protocol	TCP/IP
Connection Name	127.0.0.1(1415)
Transmission queue	XmitSenderScenario2

Note: You can set the Disconnect Interval to `0` on the Extended panel of the Sender channel properties to stop the channel timing out.

5. Similarly, create a Receiver Channel on SOA.QUEUE.MANAGER with settings as shown in Table B-6 on page 345.

Table B-6 Receiver channel on SOA.QUEUE.MANAGER

Name	**Value**
Name	WBRK_QM_to_SOA_QM
Transmission protocol	TCP/IP

6. On queue manager WBRK_QM, create a sender channel as defined in Table B-7.

Table B-7 Sender channel on WBRK_QM

Name	**Value**
Name	WBRK_QM_to_SOA_QM
Transmission protocol	TCP/IP
Connection Name	127.0.0.1(1418)
Transmission queue	XmitSenderScenario2

7. Create a Receiver Channel on WBRK_QM with settings as shown in Table B-8.

Table B-8 Receiver channel on WBRK_QM

Name	**Value**
Name	SOA_QM_to_WBRK_QM
Transmission protocol	TCP/IP

8. Start each sender channel.
9. Define a remote queue on SOA.QUEUE.MANAGER with the settings shown in Table B-9.

Table B-9 Remote queue on SOA.QUEUE.MANAGER

Name	**Value**
Name	WSHIPRESPONSE
Transmission Queue Name	XmitSenderScenario2
Remote Queue Manager Name	WBRK_QM
Remote Queue Name	WSHIPRESPONSE

10. In a similar fashion, define remote queues on WBRK_QM, setting the remote queue manager to SOA.QUEUE.MANAGER. Table B-10 shows the mapping between the queue name and the remote queue name on SOA.QUEUE.MANAGER to use.

Table B-10 Remote queues on WBRK_QM

Name	Remote queue name
LoggingFacilityQ	LoggingFacilityQ
WASHIPREQUEST	WarehouseQ
WBSHIPREQUEST	WarehouseBQ
WCSHIPREQUEST	WarehouseCQ

Setting up DB2

Install DB2 Universal Database™ V8.1 with Fixpack 5 to hold the tables that are used by the WebSphere Business Integration Message Broker message flows.

Define the following:

1. Create a new database called `BROKERDB`.
2. Create the table `SERVICE_ROUTER` in the BROKERDB database. Add the following columns:

NAME	Is a VARCHAR of length 20, is not nullable, and is the primary key.
LOCATION	Is a VARCHAR of length 120 and is not nullable.

3. Repeat the procedure to create the PART_LOCATION table. It contains the following columns:

PART_NUMBER	Is an INTEGER, not nullable, and the primary key.
WAREHOUSE	Is a VARCHAR of length 1 and is not nullable.

4. Create a third table called `HTTPCONTEXT`. This contains the field CLIENTID, which is a CHARACTER of length 51.
5. Populate the SERVICE_ROUTER and PART_LOCATION tables by running the insertdata.sql script: **`db2 -f insertdata.sql`**

Setting up ODBC data sources

To configure the ODBC data sources:

1. Open the DB2 Configuration Assistant, right-click **BROKERDB**, and select **Change Database**.

2. Click **4. Data Source** and select the **Register this database for ODBC** check box.
3. Click **OK** and close the DB2 Configuration Assistant.

Setting up the Message Brokers Toolkit

The message flows that are described in the Enterprise Service Bus Broker chapter have been prebuilt and supplied the with redbook source code. You can import each of the four message flows that are supplied with the redbook as projects in the Message Brokers Toolkit.

Setting up the Business Service Choreography scenario

To run the Business Service Choreography scenario in WebSphere Studio Application Developer Integration Edition, complete the following steps:

1. Import the InventoryProject and WSIManufacturer projects (from the source code supplied with this redbook) into a WebSphere Studio Application Developer Integration Edition workspace.
2. Import the TestClient.war Web project (also supplied with this redbook) into the workspace.
3. Follow the steps in 10.4, “Runtime guidelines” on page 294 to deploy the processes.
4. Before starting the server, be sure to add the Testclient.war Web application to the server. Also select the option on the server to **Create tables and data sources**.
5. After the server has started, use the customized Web client to invoke the long-running process. Run **TestClient.jsp** in the WebContent/sample/ManufacturerPortTypeProxy folder.
6. Select the **submitPO** method to run. This enables you to set the input message to send to the long-running process instance. Specify a prodID and a quantity, then click **Invoke**. The process will run.
7. If you specify a quantity that requires a reorder of stock, a staff work item will be generated. Use the Business Process Web Client to claim and complete this work item.
8. The process has completed when `Stock dispatched` is displayed in the console. You can also check the state of a process in the Created By Me view of the Business Process Web Client.

Abbreviations and acronyms

API	Application Programming Interface
B2B	Business-to-business
BPEL4WS	Business Process Execution Language for Web Services
CCI	Common Client Interface
CICS	Customer Information Control System
CORBA	Common Object Request Broker Architecture
COTS	Commerical-Off-The-Shelf
DMZ	Demilitarized zone
DNS	Domain Name System
DOS	Disk Operating System
DVD	Digitial Video Disc
EAI	Enterprise Application Integration
EAR	Enterprise Archive
ebXML	Electronic Business using XML
EDI	Electronic Data Interchange
EIS	Enterprise Information System
EJB	Enterprise JavaBean
EPI	External Presentation Interface
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IBM	International Business Machines Corporation
IDE	Integrated Development Environments
IIOP	Internet Inter-ORB Protocol
ITSO	International Technical Support Organization
J2C or JCA	J2EE Connector Architecture
J2EE	Java 2 Platform, Enterprise Edition
JAR	Java archive
JAX-RPC	Java API for XML-based Remote Procedure Calls
JDBC	Java database connectivity
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
JVM	Java Virtual Machine
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MDB	Message Driven Bean
OASIS	Organization for the Advancement of Structured Information Standards
OGSA	Open Grid Services Architecture
PKI	Public-Key Infrastructure
QoS	Quality of Service
RAR	Resource Adapter Archive
RMI	Remote Method Invocation
SCM	Supply Chain Management
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol

SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol / Internet Protocol‘
UDDI	Universal Description Discovery and Integration
UML	Unified Modeling Language
URL	Uniform Resource Locator
WAR	Web Archive
WSDL	Web Services Description Language
WS-I	Web Services Interoperability Organization
WSIF	Web Services Invocation Framework
WSIL	Web Services Inspection Language
XML	Extensible Markup Language
XSD	XML Schema Definition
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 355. Note that some of the documents referenced here may be available in softcopy only.

- *Developing Solutions in WebSphere MQ Integrator*, SG24-6579
- *IBM WebSphere Application Server V5.1 System Management and Configuration WebSphere Handbook Series,* SG24-6195
- *IBM WebSphere V5.0 Security WebSphere Handbook Series,* SG24-6573
- *IBM WebSphere V5.0 Performance, Scalability, and High Availability,* SG24-6198
- *Migration to WebSphere Business Integration Message Broker V5*, SG24-6995
- *Patterns: Broker Interactions for Intra- and Inter-enterprise*, SG24-6075
- *Patterns: Serial and Parallel Processes for Process Choreography and Workflow*, SG24-6306
- *Patterns: Service-Oriented Architecture and Web Services,* SG24-6303
- *Using Web Services for Business Integration*, SG24-6583
- *WebSphere MQ Integrator Deployment and Migration*, SG24-6509

Other publications

These publications are also relevant as further information sources:

- Olaf Zimmermann, Mark Tomlinson, Stefan Peuser, *Perspectives on Web Services,* Springer, 2003, ISBN 3-540-00914-0
- Jonathan Adams, Srinivas Koushik, Guru Vasudeva, George Galambos, *Patterns for e-business: A Strategy for Reuse*, IBM Press, 2001, ISBN 1-931182-02-7

- First look at the WS-I Basic Profile 1.0

  `http://www.ibm.com/developerworks/webservices/library/ws-basicprof.html`
- First look at the WS-I Usage Scenarios

  `http://www.ibm.com/developerworks/webservices/library/ws-iuse/`
- Preview of WS-I sample application

  `http://www.ibm.com/developerworks/webservices/library/ws-wsisamp/`
- “Security in a Web Services World: a Proposed Architecture and Roadmap,” a joint security whitepaper from IBM Corporation and Microsoft Corporation

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

  `http://www.ibm.com/developerworks/webservices/library/ws-secroad/`
- “Updated: Web Services Reliable Messaging,” a new protocol for reliable delivery between distributed applications

  `http://www.ibm.com/developerworks/webservices/library/ws-rm/`
- “Implementation Strategies for WS-ReliableMessaging,” how WS-ReliableMessaging can interact with other middleware communication systems

  `http://www.ibm.com/developerworks/webservices/library/ws-rmimp/`
- BPEL4WS specification

  `http://www.ibm.com/developerworks/library/ws-bpel/`
- Business Process with BPEL4WS, a series of introductory articles and references

  `http://www.ibm.com/developerworks/webservices/library/ws-bpelcol1/`
- BPEL4WS support in WebSphere Business Integration Server Foundation

  `http://www-306.ibm.com/software/integration/wbisf/features/`
- BPEL4WS support in WebSphere Studio Application Developer Integration Edition

  `http://www-306.ibm.com/software/integration/wsadie/features/`
- WS-AtomicTransaction specification

  `http://www.ibm.com/developerworks/library/ws-atomtran/`
- WS-BusinessActivity specification

  `http://www.ibm.com/developerworks/webservices/library/ws-busact/`
- “Transactions in the world of Web services,” part 1 and 2

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

- WS-Coordination specification

  `http://www.ibm.com/developerworks/library/ws-coor/`
- WS-Policy specification

  `http://www.ibm.com/developerworks/library/ws-polfram/`
- “Web Services Policy Framework,” new specifications improve WS-Security

  `http://www.ibm.com/developerworks/webservices/library/ws-polfram/summary.html`
- WS-Resource Framework overview

  `http://www.ibm.com/developerworks/webservices/library/ws-resource/ws-wsrfpaper.html`
- “Web Service Oriented Architecture - The Best Solution to Business Integration,” Annrai O'Toole, Cape Clear Software CEO

  `http://www.capeclear.com/clear_thinking1.shtml`
- “SOA - Save Our Assets,” Lawrence Wilkes, CBDI Forum (subscription required)

  `http://www.cbdiforum.com/report_summary.php3?topic_id=2&report=623&start_rec=0`
- The IBM series of articles “Migrating to a service-oriented architecture” by Kishore Channabasavaiah, Kerrie Holley, and Edward M. Tuggle Jr.

  `http://www.ibm.com/developerworks/library/ws-migratesoa/`
  `http://www.ibm.com/developerworks/webservices/library/ws-migratesoa2/`
- “Coarse-grained interfaces enable service composition in SOA,” Jeff Hanson, Builder.com

  `http://builder.com.com/5100-6386-5064520.html`
- Simple Object Access Protocol (SOAP) 1.1

  `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`
- Java API for XML-based RPC

  `http://java.sun.com/xml/downloads/jaxrpc.html`
- Web Services Security (WS-Security) Version 1.0

  `http://www.ibm.com/developerworks/webservices/library/ws-secure/`
- Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)

  `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf`
- Business Process Execution Language for Web Services Version 1.1

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

Online resources

These Web sites and URLs are also relevant as further information sources:

- IBM Patterns for e-business

  `http://www.ibm.com/developerWorks/patterns/`
- IBM Web Services

  `http://www.ibm.com/webservices`
- IBM on demand Operating Environment

  `http://www-3.ibm.com/software/info/openenvironment/`
- IBM developerWorks: SOA and Web services zone

  `http://www.ibm.com/developerworks/webservices`
- IBM alphaWorks

  `http://www.alphaworks.ibm.com/`
- IBM CICS

  `http://www.ibm.com/software/ts/cics`
- IBM developerWorks

  `http://www.ibm.com/developerworks`
- IBM Web services

  `http://www.ibm.com/software/solutions/webservices`
- IBM WebSphere Developer Domain

  `http://www7b.boulder.ibm.com/wsdd/`
- IBM WebSphere MQ

  `http://www.ibm.com/software/ts/mqseries`
- IBM WebSphere software platform

  `http://www.ibm.com/software/webservers/appserv`
- WebSphere Application Server Information Center

  `http://www.ibm.com/software/webservers/appserv/infocenter.html`
- IBM WebSphere MQ family SupportPacs

  `http://www.ibm.com/software/integration/support/supportpacs/`
- WebSphere Business Integration Message Broker information center

  `http://www.ibm.com/software/integration/wbimessagebroker/library/`
- WebSphere Business Integration Server V5.1.x information center

  `http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications, and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

Patterns: Implementing an SOA Using an Enterprise Service Bus

(0.5" spine)
0.475"<->0.875"
250 <-> 459 pages

Patterns: Implementing an SOA Using an Enterprise Service Bus

Design and implement an ESB using current WebSphere technologies

Service-oriented architecture and Web services

Learn by example with practical scenarios

Patterns for e-business is a group of proven, reusable assets that can be used to increase the speed of developing and deploying e-business applications. This IBM Redbook focuses on how the service-oriented architecture profile of the Process Integration patterns can be used to start implementing service-oriented architecture using an Enterprise Service Bus.

Part 1 presents a description of service-oriented architecture and how it applies to Web services and e-business on demand. Emerging service-oriented architecture trends are also discussed.

Part 2 provides a detailed description of the Enterprise Service Bus (ESB) concept, and how this fits with the Patterns for e-business. Common usage scenarios, a minimum capability ESB, and ESB patterns are described. IBM product mappings are then applied to the ESB patterns.

Part 3 guides you through the process of implementing an Enterprise Service Bus using current IBM technologies. Router and Broker interactions within an Enterprise Service Bus are covered, along with off-the-bus service choreography and the Exposed ESB Gateway to enable interaction in an inter-enterprise environment.

SG24-6346-00 ISBN 0738490008