WebSphere® software

IBM

Patterns: Direct Connections for Intra- and Inter-enterprise

- Select an intra- or inter-enterprise integration approach
- Explore J2EE, Web services and EDI solutions
- Learn by example with sample scenarios

Mark Endrei
Filippo Diotalevi
Wolfgang Dostal
Karsten Elich
Uday Manginapudi
Wendy Neave
William Patrey

Redbooks

ibm.com/redbooks

International Technical Support Organization

Patterns: Direct Connections for Intra- and Inter-enterprise

February 2004

SG24-6933-00

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

Second Edition (February 2004)

This edition applies to IBM WebSphere Application Server V5.0.2, and IBM WebSphere Studio Application Developer V5.1, for use with Microsoft Windows 2000.

Contents

Notices

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

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

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

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

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

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

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

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

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

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

Trademarks

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

eserver™
Redbooks (logo) ™
AIX®
alphaWorks®
CICS®
CrossWorlds®
DB2 Connect™
DB2®
developerWorks®
Domino®
e-business on demand™
IBM®
IMS™
Lotus®
MVS™
Notes®
OS/390®
OS/400®
pSeries®
RACF®
Redbooks™
SupportPac™
Tivoli®
VisualAge®
VSE/ESA™
WebSphere®
z/OS®
zSeries®
ibm.com®
MQSeries®

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

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

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

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

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

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

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

Preface

The Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying Web applications. This IBM® Redbook focuses on point-to-point application integration using the Process-focused Application Integration::Direct Connection application pattern for intra-enterprise, and the Extended Enterprise::Exposed Direct Connection application pattern for inter-enterprise.

Part 1 guides you through the process of selecting an Application and Runtime pattern. Next, the platform-specific Product mappings are identified based upon the selected Runtime pattern.

Part 2 presents guidelines on applying the Patterns approach to a sample business scenario and on selecting application integration technologies.

Part 3 provides detailed design, development, and runtime guidelines for intra-enterprise integration solutions. It teaches you by example using IBM WebSphere Application Server V5.0 with Web services, J2EE Connectors, and JMS.

Part 4 provides detailed design, development, and runtime guidelines for inter-enterprise integration solutions. It teaches you by example using IBM WebSphere Application Server V5.0 with Web services.

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.

The IBM redbook team (Left to right: Uday Manginapudi, Karsten Elich, Wendy Neave, Filippo Diotalevi, William Patrey, Wolfgang Dostal, Mark Endrei)

Mark Endrei is an IT Architect at the International Technical Support Organization, Raleigh Center. He writes about WebSphere® and Patterns for e-business. Before joining the ITSO early in 2001, Mark worked in IBM Global Services Australia as an IT Architect. He holds a bachelor's degree in Computer Systems Engineering from the Royal Melbourne Institute of Technology, and an MBA in Technology Management from Deakin University/APESMA.

Filippo Diotalevi is an IT Professional in IBM Global Services, Application Management Services, in Milan, Italy. He has two years of experience in the e-business field. He holds a degree in Computer Engineering from University of Padova, Italy. His areas of expertise include Enterprise Java programming, VisualAge® and WebSphere.

Wolfgang Dostal is an IT architect in IBM Application Management Services in Frankfurt, Germany. He holds a Ph.D. degree in Physics from the University of Mainz, Germany. His areas of expertise include object technology in distributed environments, Enterprise Application Integration, XML, and J2EE. He has about 10 years of experience in object-oriented technologies. He has also been teaching since 1990, and at present teaches software engineering, object technology and Web-based applications at the University for Applied Science Mainz. He has made extensive contributions to professional journals and seminars on Web services and related topics.

Karsten Elich is an IT Architect in IBM Germany. He has 7 years of experience in the application design and architecture field. He holds an M.S. degree in Chemistry and a Ph.D. in Physical Chemistry from Johannes Gutenberg-University in Mainz. His areas of expertise include designing and implementing e-business solutions.

Uday Manginapudi is a IT Architect with IBM Business Consulting Services India. He has more than 13 years of experience in the IT industry. He holds a Bachelor's Degree in Engineering. His areas of expertise include e-Business Integration, particularly EAI.

Wendy Neave is a Senior IT Specialist in IBM Global Services in Australia. She has 14 years experience in application design and development. Wendy holds a Bachelor of Education (Environmental Science) and an Associate Diploma in Computing. She is also a Sun Certified Programmer for Java 2. Her areas of expertise include object-oriented analysis, design and development, Java, and WebSphere.

William Patrey is an Advisory IT Specialist in the United States. He has 9 years of experience in the Information Technology field. He holds a Master's degree in History from Virginia Tech. His areas of expertise include application development, performance testing, and networking. He has written extensively on software testing and performance test tools.

The business interaction patterns framework presented in this redbook, and its application to the Patterns for e-business, is based on the work of:

Paul Verschueren, IBM UK
Jonathan Adams, IBM UK

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

Jeff Barrett, IBM Austin
Russell Butek, IBM Austin
Alison Chandler, IBM ITSO Poughkeepsie
Eric Erpenbach, IBM Rochester
Michele Galic, IBM ITSO Raleigh
Robert Haimowitz, IBM ITSO Raleigh
Martin Keen, IBM ITSO Raleigh
Chris Rayns, IBM ITSO Poughkeepsie
Carla Sadtler, IBM ITSO Raleigh
Geert Van De Putte, IBM ITSO Raleigh
Phil Wakelin, IBM UK
Adrian Warman, IBM UK

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

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

Part 1

Patterns for e-business

Part 1 provides an overview of IBM Patterns for e-business. It introduces the interaction patterns that both Application Integration and Extended Enterprise patterns are based on. It guides you through the process of selecting Application and Runtime patterns for intra- and inter-enterprise integration. The platform-specific product mappings are identified based upon the selected Runtime pattern.

Included in Part 1 are the following chapters:

- Chapter 1, “Patterns for e-business” on page 3
- Chapter 2, “Fundamental concepts in Process Integration” on page 17
- Chapter 3, “Application Integration” on page 33
- Chapter 4, “Extended Enterprise” on page 69
- Chapter 5, “Node types and Product descriptions” on page 97

1

Patterns for e-business

This redbook is part of the Patterns for e-business series. In this introductory chapter we provide an overview of how IT architects can work effectively with the Patterns for e-business.

The role of the IT architect is to evaluate business problems and build solutions to solve them. To do this, the architect begins by gathering input on the problem, an outline of the desired solution, and any special considerations or requirements that need to be factored into that solution. The architect then takes this input and designs the solution. This solution 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 the knowledge gained from each engagement and using it to build a repository of assets. IT architects can then build future solutions based on these proven assets. This reuse saves time, money, and effort; and in the process, it helps ensure delivery of a solid, properly architected solution.

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

The IBM Patterns for e-business are further augmented with guidelines and related links for their better use.

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

1.1 The Patterns for e-business layered asset model

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

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

These assets and their relationships to each other are shown in Figure 1-1 on page 5.

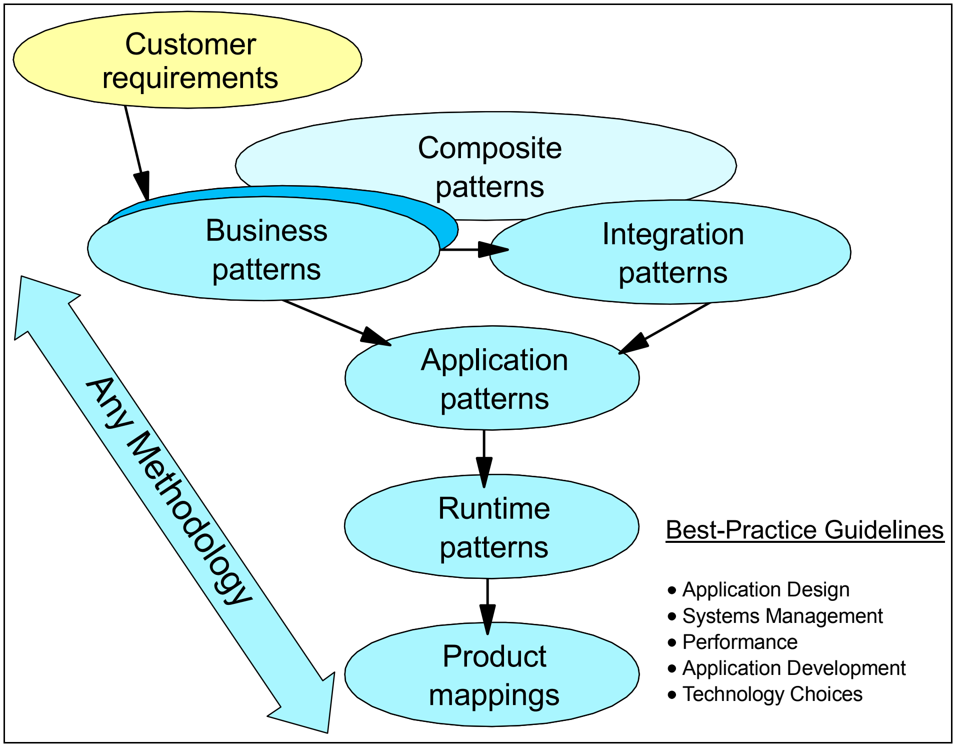


Figure 1-1 The Patterns for e-business layered asset model

Patterns for e-business Web site

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

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

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

1.2 How to use the Patterns for e-business

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

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

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

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

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

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

Business patterns

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

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

Table 1-1 The four primary Business patterns

Business Patterns	Description	Examples
Self-Service (User-to-Business)	Applications where users interact with a business via the Internet or intranet	Simple Web site applications
Information Aggregation (User-to-Data)	Applications where users can extract useful information from large volumes of data, text, images, etc.	Business intelligence, knowledge management, Web crawlers
Collaboration (User-to-User)	Applications where 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.

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

Integration patterns

Integration patterns allow us to tie together multiple Business patterns to solve a business problem. The Integration patterns are outlined in Table 1-2 on page 8.

Table 1-2 Integration patterns

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

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

Custom design

We can illustrate the use of a Custom design to address a business problem through an iconic representation as shown in Figure 1-2.

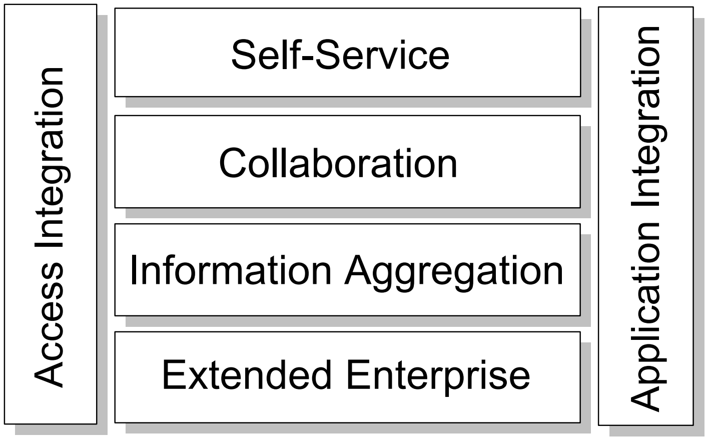


Figure 1-2 Patterns representing a Custom design

If any of the Business or Integration patterns are not used in a Custom design, we can show the unused patterns as lighter blocks than those that are used. For example, Figure 1-3 on page 9 shows a Custom design that does not have a Collaboration business pattern or an Extended Enterprise business pattern for a business problem.

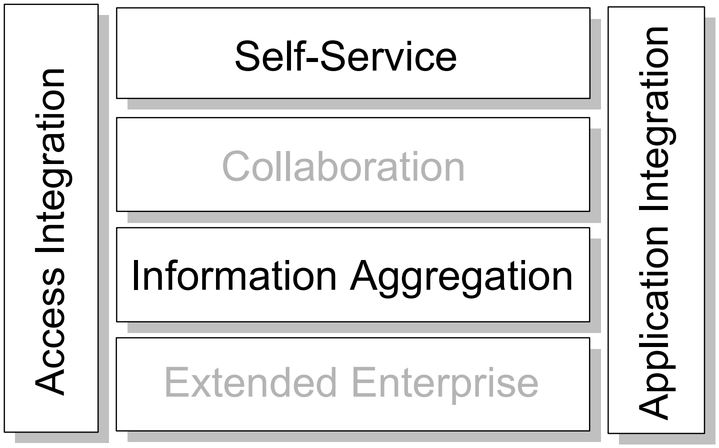


Figure 1-3 Custom design with Self-Service, Information Aggregation, Access Integration and Application Integration

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 1-3 can also describe a Sell-Side Hub composite pattern.

Composite patterns

Several common uses of Business and Integration patterns have been identified and formalized into Composite patterns. The identified Composite patterns are shown in Table 1-3 on page 10.

Table 1-3 Composite patterns

Composite Patterns	Description	Examples
Electronic Commerce	User-to-Online-Buying	• www.macys.com • www.amazon.com
Portal	Typically designed to aggregate multiple information sources and applications to provide uniform, seamless, and personalized access for its users.	• Enterprise Intranet portal providing self-service functions such as payroll, benefits, and travel expenses. • Collaboration providers who provide services such as e-mail or instant messaging.
Account Access	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	Allows 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)

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

1.2.2 Selecting Application patterns

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

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

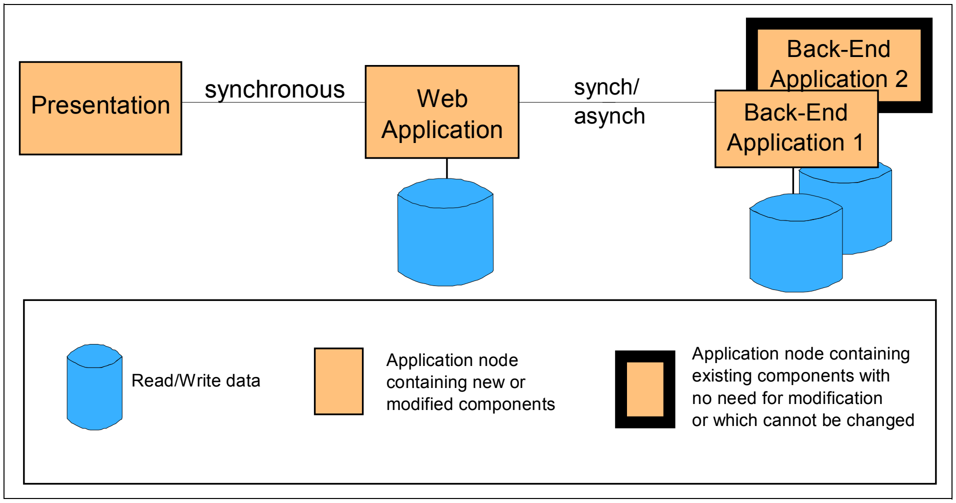

Presentation
synchronous
Web Application
synch/ asynch
Back-End Application 2
Back-End Application 1
Read/Write data
Application node containing new or modified components
Application node containing existing components with no need for modification or which cannot be changed

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

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

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

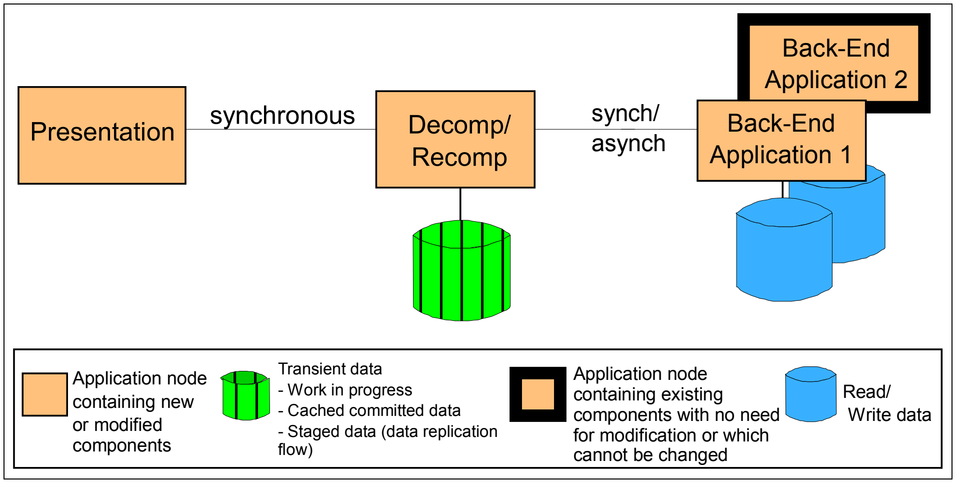

Presentation
synchronous
Decomp/ Recomp
synch/ asynch
Back-End Application 2
Back-End Application 1
Application node containing new or modified components
Transient data
- Work in progress
- Cached committed data
- Staged data (data replication flow)
Application node containing existing components with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

Figure 1-5 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 further refined with more explicit functions to be performed. Each function is associated with a runtime node. In reality these functions, or nodes, can exist on separate physical machines or can co-exist on the same machine. In the Runtime pattern this is not relevant. The focus is on the logical nodes required and their placement in the overall network structure.

As an example, let's assume that our customer has determined that his 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 his situation.

He knows that he will have users on the Internet accessing his business data and he 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 his company network.

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

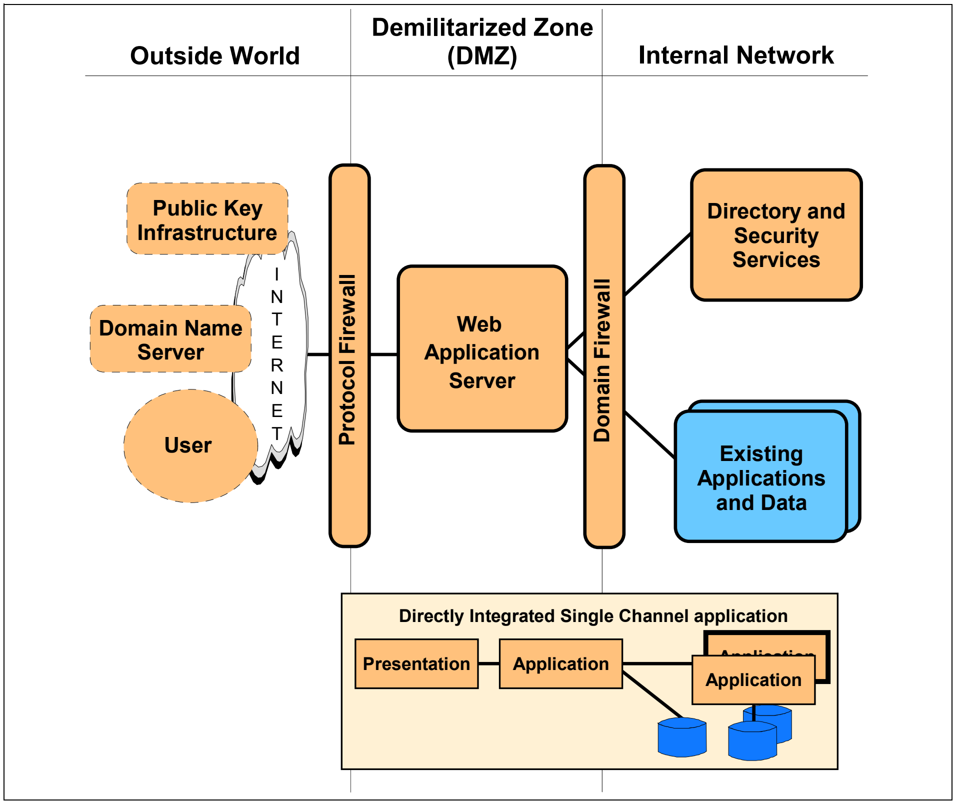


Figure 1-6 Directly Integrated Single Channel application pattern::Runtime pattern

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

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

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

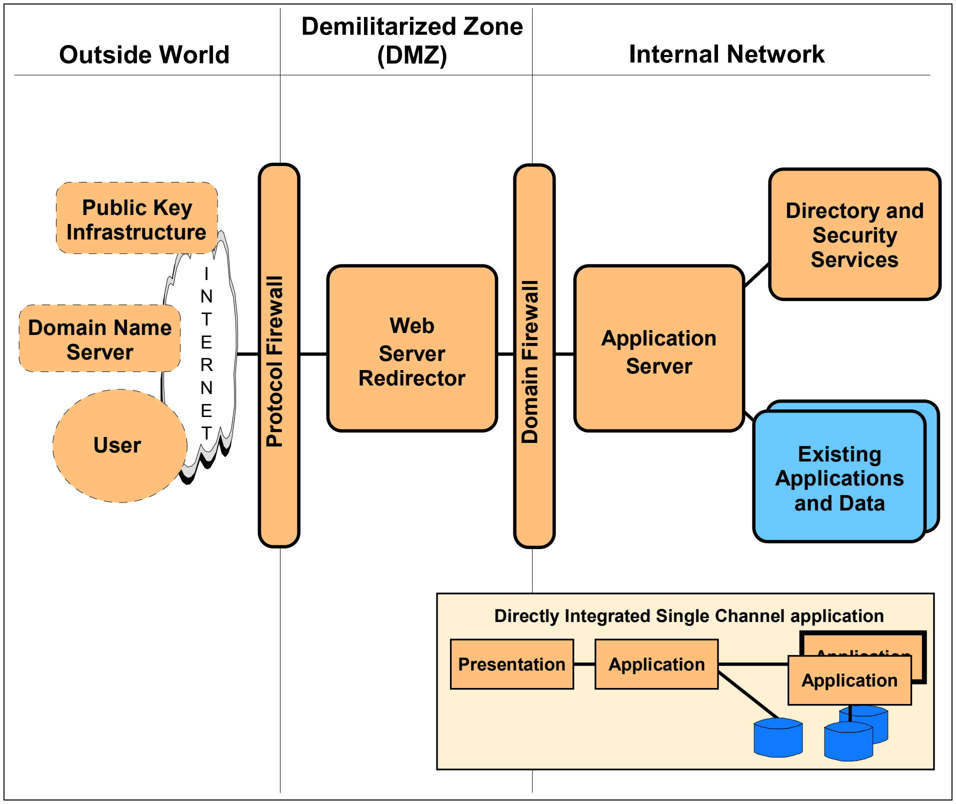


Figure 1-7 Directly Integrated Single Channel application pattern::Runtime pattern: Variation 1

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

1.2.4 Review Product mappings

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

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

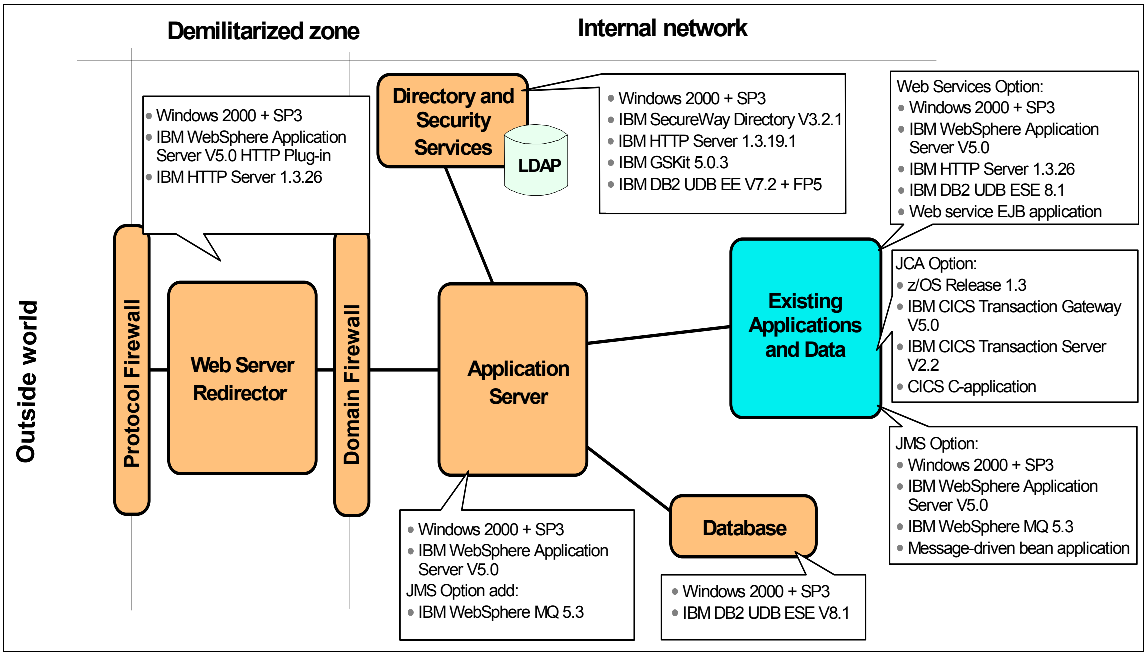


Figure 1-8 Directly Integrated Single Channel application pattern: Windows 2000 Product mapping

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:

- 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 on how to improve 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

Fundamental concepts in Process Integration

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

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

- Application Integration pattern
- Extended Enterprise business pattern

2.1 The need for a unifying technique

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

Table 2-1 provides an example of terminology overload for "synchronous verses asynchronous.

Table 2-1 Synchronous verses asynchronous terminology overload

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

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

2.1.1 Similarities between intra- and inter-enterprise integration

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

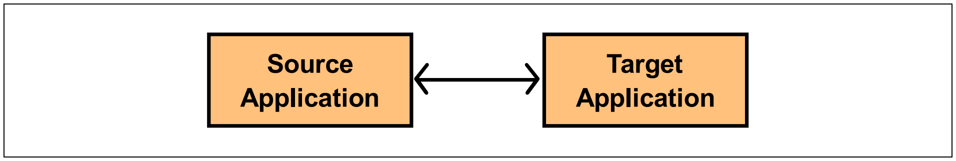


Figure 2-1 Intra-enterprise integration

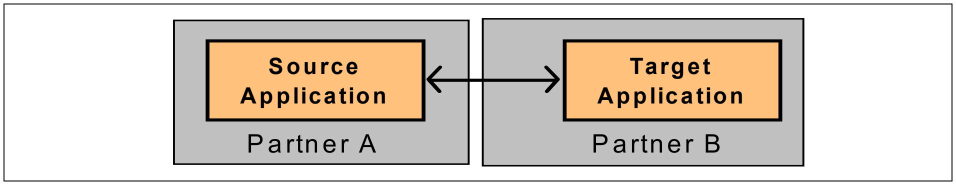


Figure 2-2 Inter-enterprise integration

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

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

2.1.2 Summary

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

2.2 Process Integration concepts and notations

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

2.2.1 Collaboration and Interaction

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

Collaboration

In the most general sense, a *collaboration* denotes N-to-N activities between sub-systems within a distributed system. As shown in Figure 2-3, complex collaborations between sub-systems can be broken down into more basic *interactions*. An interaction focuses on 1-to-1 or 1-to-N activities originating from a single sub-system.

In this way, complex collaborations involving many sub-systems can be decomposed into simpler interactions that are easier to analyze. Data analysts use a similar approach when analyzing complex data with many-to-many relationships. Normalization is used to reduce many-to-many relationships between data to 1-to-many relationships.

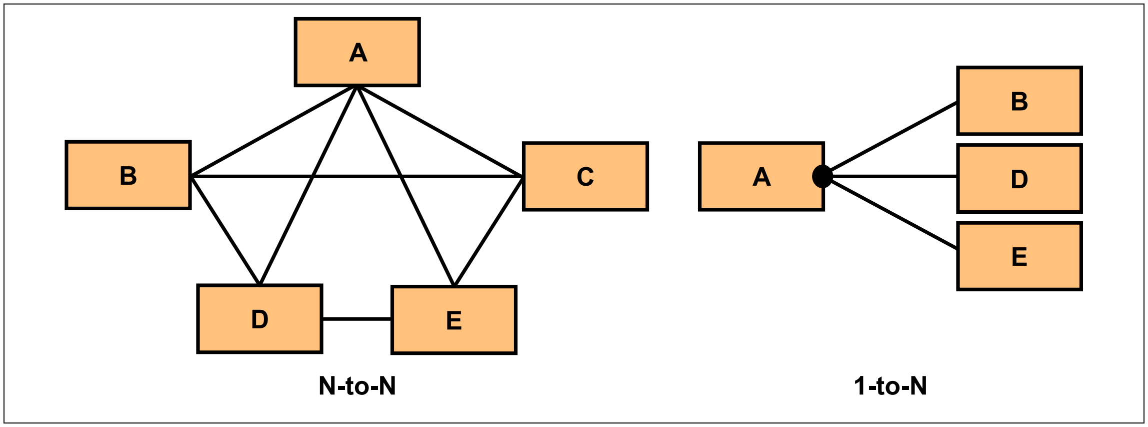


Figure 2-3 Collaboration topologies

Note that we do not show a link from A-to-C on the right of Figure 2-3. This is because, in breaking the interaction down, we found that A only initiates interactions with B, D, and E. The C-to-A interaction will be modeled in another 1-to-N interaction.

Interaction

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

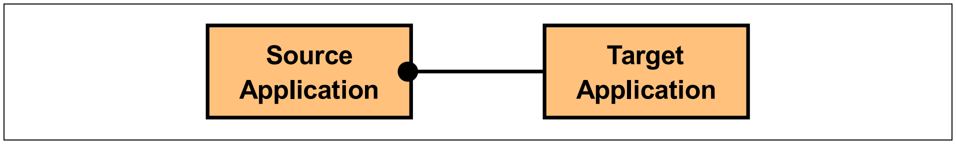


Figure 2-4 Definition of interaction

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

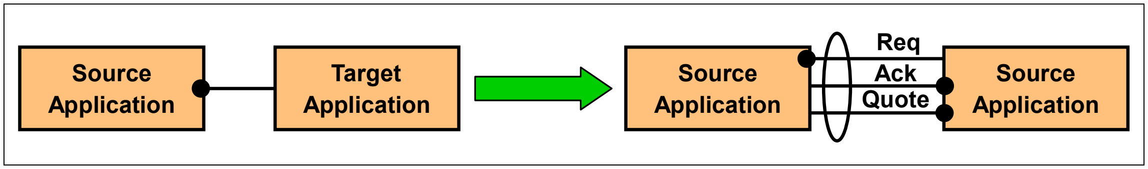


Figure 2-5 Decomposition of complex interactions

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

Complex interactions with multiple target applications also can be decomposed into multiple 1-to-1 interactions, as long as there is one initiating operation within a source application. The interaction patterns approach can then be applied to these 1-to-1 interactions.

2.2.2 Connectors and Adapters

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

Connectors

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

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

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

For lower-level modelling, a primitive connector can always be decomposed into a modelled connector and two adjacent primitive connectors, as shown in Figure 2-6. This way, connector models can be recursively decomposed until the correct level of detail is reached.

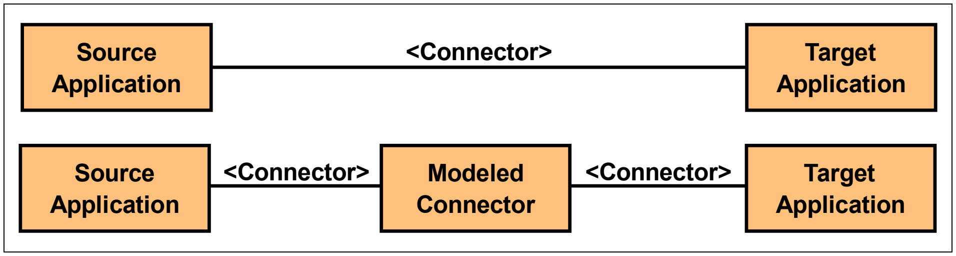


Figure 2-6 Decomposition of connectors

It is useful to distinguish two connector subtypes, as shown in Figure 2-7:

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

These connector subtypes are orthogonal, meaning a connector may be both an adapter connector and a path connector. The relationship between connectors and adapters is shown in Figure 2-7.

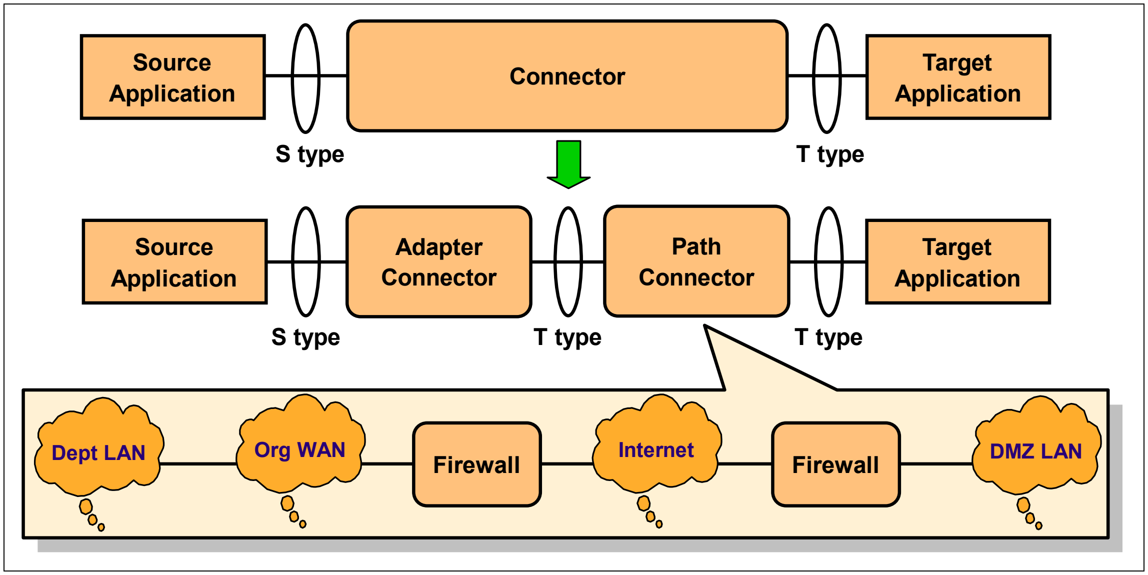


Figure 2-7 Relationship between connectors and adapters

Adapters

Adapters provide the logical connectivity to an application. Without adapters, each application would need to implement the specific interface of the target application.

It is useful to distinguish three types of adapters:

- Control adapters are not concerned with content. They are only concerned with the activities involved in flow operations:
  - Transforming the protocol used between the segments
  - Segmenting, batching, and sorting data blocks
  - Correctly interacting with the path connector to execute the transport operation (This includes respecting the protocol rules.)
- View adapters are concerned with transforming content but only in terms of its technical representation. Examples include:
  - Element demarcation schemes, such as delimited, fixed-length, and XML
  - Element sequencing schemes, such as keys and collation sequences
  - Element encoding schemes, such as character set, number format, and date format
- Model adapters transform the semantic content and normally require business input to define correct operational rules. Some examples are:
  - Splitting out subsets of data
  - Joining external data (augmentation/enrichment)
  - Summarization
  - Translation of identifiers (key management)

Coupling adapter connectors

Coupling adapter connectors can be used to implement a common integration protocol such as messaging, RMI/IIOP, SOAP/HTTP, and so on. As shown in Figure 2-8, the adapter functionality between the source application and the target application is decomposed into two halves. Each half adapts to and from a common intermediate protocol.

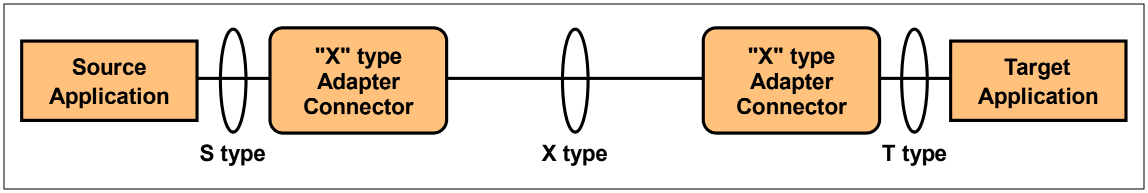


Figure 2-8 Coupling adapters

If there are multiple point-to-point connections between a group of sub-systems, this approach can significantly reduce the number of different adapters required. Each sub-system only needs one adapter (instead of needing a different adapter to connect to each sub-system).

Connectors and synchronicity

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

- Synchronous interaction

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

- Asynchronous interaction

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

2.2.3 Classification of interaction between sub-systems

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

Parallel interaction

An interaction is denoted as parallel if it includes a set of concurrent 1-to-1 interactions between a source application and multiple target applications, as shown in Figure 2-9.

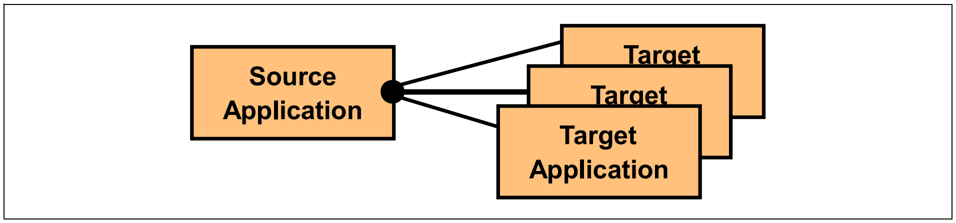


Figure 2-9 Parallel interaction

Serial interaction

An interaction is denoted as serial if it includes a series of 1-to-1 interactions between a source application and multiple target applications that are subject to time sequenced dependencies, as shown in Figure 2-10.

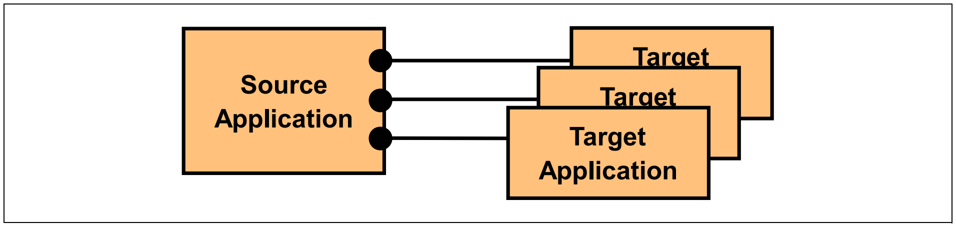
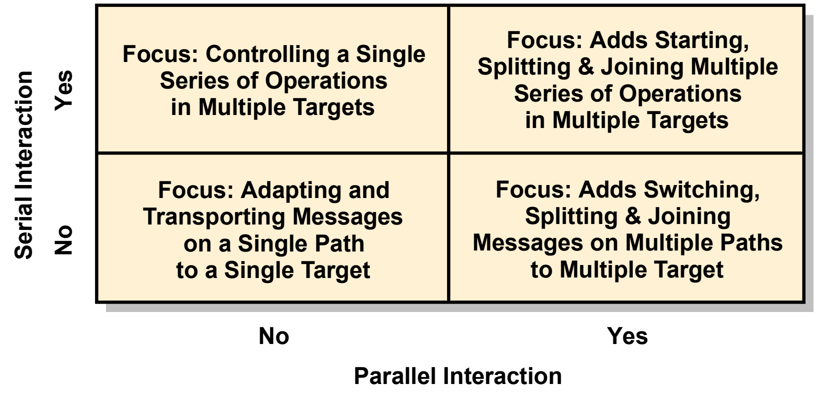


Figure 2-10 Serial interaction

Classification of Interactions

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



		No	Yes
Serial Interaction	Yes	Focus: Controlling a Single Series of Operations in Multiple Targets	Focus: Adds Starting, Splitting & Joining Multiple Series of Operations in Multiple Targets
	No	Focus: Adapting and Transporting Messages on a Single Path to a Single Target	Focus: Adds Switching, Splitting & Joining Messages on Multiple Paths to Multiple Target

Figure 2-11 Classification of interactions

This classification framework is used later in this chapter to classify Application patterns for both Application Integration patterns (intra-enterprise) and Extended Enterprise business patterns (inter-enterprise).

2.3 QoS capabilities framework

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

The following QoS concerns are defined in this section:

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

2.3.1 Operability

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

2.3.2 Availability

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

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

2.3.3 Federation

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

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

2.3.4 Performance

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

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

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

2.3.5 Security

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

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

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

2.3.6 Standards compliance

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

2.3.7 Transactionality

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

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

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

2.4 Application patterns for Application Integration

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

- **Direct Connection Application pattern**
  and its Message/Call Connection variations

  Allows a single interaction from the source application to be adapted and transported to one target application

- **Broker Application pattern**
  and its Router variation

  Allows a single interaction from the source application to be switched, split, and joined to multiple target applications concurrently

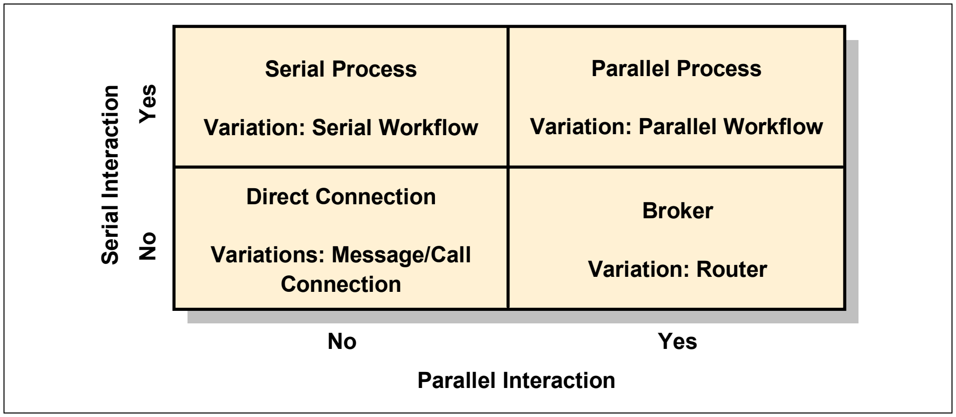
- **Serial Process Application pattern**
  and its Serial Workflow variation

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

- **Parallel Process Application pattern and its Parallel Workflow variation**

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

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



Serial Interaction	Parallel Interaction: No	Parallel Interaction: Yes
Yes	**Serial Process** **Variation: Serial Workflow**	**Parallel Process** **Variation: Parallel Workflow**
No	**Direct Connection** **Variations: Message/Call Connection**	**Broker** **Variation: Router**

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

2.5 Application patterns for Extended Enterprise

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

- **Exposed Direct Connection application pattern and its Exposed Message/Call Connection variations**

  Allows a single interaction from the source application to be adapted and transported to one partner target application

- **Exposed Broker application pattern and its Exposed Router variation**

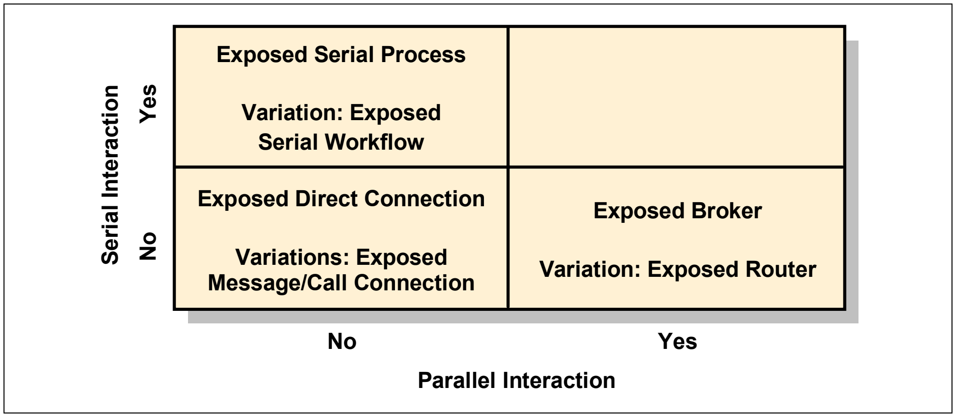
  Allows a single interaction from the source application to be switched, split, and joined to multiple partner target applications concurrently.

- **Exposed Serial Process application pattern and its Exposed Serial Workflow variation**

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

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

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



Serial Interaction	Parallel Interaction: No	Parallel Interaction: Yes
Yes	Exposed Serial Process Variation: Exposed Serial Workflow	
No	Exposed Direct Connection Variations: Exposed Message/Call Connection	Exposed Broker Variation: Exposed Router

Figure 2-13 Classification of Extended Enterprise patterns

2.6 Summary

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

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

- The Process-focused Application Integration::Direct Connection application pattern, which is introduced in Chapter 3, “Application Integration” on page 33.
- The Extended Enterprise::Exposed Direct Connection application pattern, which is introduced in Chapter 4, “Extended Enterprise” on page 69.

3

Application Integration

The Application Integration pattern (also known as Enterprise Application Integration or EAI) serves to integrate multiple Business patterns or to integrate applications and data within an individual Business pattern. It is applicable when integrating applications and data within the bounds of an organization.

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

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

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

Critical to selecting the right Application Integration pattern is an understanding of the integration requirements of the business problem being automated. Some examples of key questions to ask in determining an appropriate EAI design are listed in 3.1.5, “Application Integration solution requirements” on page 36.

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

What's next

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

For a better understanding of the issues and considerations surrounding an Application Integration solution, review the guidelines in the next section, which provides additional information on choosing this Integration pattern. Business and IT drivers, the e-business context appropriate for this solution type, and additional solution details are discussed here.

If you have established a sound understanding of the issues relating to your EAI deployment, the next step is to select an Application pattern. The Application Integration pattern can be implemented using any one of the four Process-focused application patterns and the Data-focused application patterns (see 3.2, “Application patterns” on page 39). These various designs provide solution flexibility to address the specific needs of the business process being automated.

3.1 General guidelines

To help you determine if the Application Integration pattern is appropriate for the design of your intra-enterprise application integration scenario, this section details the business and IT scenario into which a solution using the Application Integration pattern will fit.

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

3.1.1 Business and IT drivers

Businesses developing a solution needing the following characteristics should consider using the Application Integration pattern:

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

3.1.2 Context

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

In our sample business scenario, described in Chapter 6, “Business scenarios used in this book” on page 111, ITSO Electronics wants to integrate their retail and wholesale departments. Currently, both departments have proven IT infrastructures but have no interconnectivity. The Process-focused Application Integration patterns address this problem. These patterns can be applied in a case where the business process needs to be integrated between existing business systems within the organization. The Process-focused Application Integration patterns can be used to integrate the retail ordering and wholesale inventory systems in ITSO Electronics, eliminating ordering lag and providing an up-to-date inventory.

3.1.3 Solution

The Application Integration pattern typically consists of the following:

- Business applications and data that need to communicate, interact, and integrate with other business applications and data within the organization
- A network which:
  - Is based on TCP/IP and other Internet technologies, or on proprietary protocols
  - Can be a dedicated LAN or WAN connection
- Other business applications and data which can be:
  - Custom developed systems (old and new)

  - Enterprise Resource Planning systems and other packaged applications, such as SAP, BAAN and PeopleSoft
  - Databases

It is typically based on the patterns described in Chapter 2, "Fundamental concepts in Process Integration" on page 17.

3.1.4 Putting the pattern to use

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

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

3.1.5 Application Integration solution requirements

Choosing the right Application Integration pattern can only be done in the context of specific enterprise requirements. These requirements encompass not only the specific application integration to be deployed, but also the enterprise's IT infrastructure and technology preferences. This section details considerations to be made and questions to ask in determining which Application Integration pattern best fulfills your enterprise needs.

Request for information versus request for processing

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

Foreground versus background integration

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

Scope of integration

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

Operation latency (applications or data queries)

How long will it take the operation to complete in the application? Operations that can not complete in less than a couple of seconds dictate the need for asynchronous methods of integration. A query on product inventory may be a quick operation, whereas the computation of the production plan for the manufacturing of that inventory could take minutes to hours to complete.

Geographic proximity

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

Process re-engineering

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

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

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

Application portfolio

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

Your survey of applications will detail several key points about the enterprise environment:

- Can the application interfaces be extended as part of the integration activity? Homegrown applications may have standardized interfaces or be extensible to implement standards. Interfaces into pre-packaged applications typically can only be standardized through implementation of sophisticated adapters.
- Is there a central cornerstone application in the enterprise environment or a portfolio of peer applications? Is the business processing focused around one key application (perhaps an ERP system) with all other applications being subservient to that application?
- How many applications are being integrated? For instance, a typical Self-Service application may be integrating the Web application server with one back-end system. At the other end of the spectrum would be a project creating a centralized customer information system that may require feeds from 100 or more different applications. Integration of two applications has different pattern requirements than integration of 100 applications.

Invasive versus non-invasive

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

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

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

Enterprise architecture

To what degree is the overall enterprise process and data model defined? The enterprise architecture is an instantiation of the application functions, application data model, application interfaces, and application flow of control. Beyond capturing an accurate description of the current enterprise environment, a good Enterprise Architecture (EA) takes into account new business processing requirements.

The completeness of the EA often will dictate the level of invasiveness in the EAI integration. A well conceived EA enables a more extensible enterprise application integration design.

The different types of Enterprise Architectures dictate different Application Integration patterns. For instance, Self-Service is divided into *Web up* and *enterprise out* scenarios. Enterprise out scenarios have heavier legacy content than Web up scenarios. Key characteristics of the EA that affect the EAI approach include the:

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

3.1.6 What's next

If you have determined that the Application Integration pattern is appropriate for use in your solution, the next step is to select an Application pattern.

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

3.2 Application patterns

The various designs in the Application patterns that allow for solution flexibility in Application Integration are categorized as either Process-focused or Data-focused. These two categories enable different types of integration functionality.

The focus of this redbook is the Process-focused, Direct Connection application pattern. A brief overview of the other Application Integration patterns is provided.

For complete details on the other Application Integration patterns, see the IBM Patterns for e-business Web site:

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

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

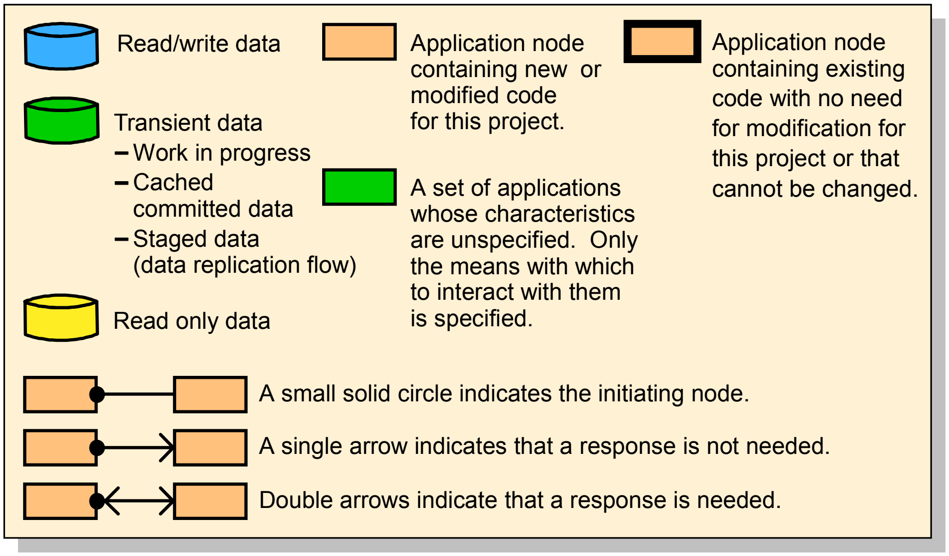

Read/write data
Transient data
– Work in progress
– Cached committed data
– Staged data (data replication flow)
Read only data
Application node containing new or modified code for this project.
A set of applications whose characteristics are unspecified. Only the means with which to interact with them is specified.
Application node containing existing code with no need for modification for this project or that cannot be changed.
A small solid circle indicates the initiating node.
A single arrow indicates that a response is not needed.
Double arrows indicate that a response is needed.

Figure 3-1 Application pattern diagram conventions

3.3 Process-focused Application Integration patterns

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

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

The Process-focused Application Integration patterns are presented here in order of increasing flexibility and sophistication. As the Application patterns build on each other, their capabilities and reliance on middleware increase, and they require less application development effort. From the following Application patterns, select the one that best fits your requirements:

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

Business and IT drivers

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

Table 3-1 Business drivers

Business drivers	Direct Connection Message variation	Direct Connection Call variation	Broker Router variation	Broker	Serial/Parallel Process
Improve the organizational efficiency	✓	✓	✓	✓	✓
Reduce the latency of business events	✓	✓	✓	✓	✓
Support a structured exchange within the organization	✓	✓	✓	✓	✓
Support real-time one-way “message” flows	✓		✓	✓	✓
Support real-time request/reply “message” flows		✓	✓	✓	✓
Support dynamic routing of “messages” to one of many target applications			✓	✓	✓
Support dynamic distribution of “messages” to multiple target applications				✓	✓
Support more flexible, time-sequenced business and human process flows					✓

Table 3-2 IT drivers

IT drivers	Direct Connection Message variation	Direct Connection Call variation	Broker Router variation	Broker	Serial/Parallel Process
Leverage existing skills	✓	✓	✓	✓	✓
Leverage the legacy investment	✓	✓	✓	✓	✓
Enable back-end application integration	✓	✓	✓	✓	✓
Minimize application complexity	✓	✓	✓	✓	✓
Minimize enterprise complexity			✓	✓	✓
Exploit parallelism				✓	✓

QoS concerns

This section highlights Quality of Service (QoS) capabilities that are of particular concern in the Process-focused Application Integration domain.

In 2.3, “QoS capabilities framework” on page 26, we described a QoS capabilities framework for Process Integration based on the following general concerns:

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

The following QoS concerns are of particular importance when working in the Process-focused Application Integration domain.

Important: This profile is intended as a very rough first guide to QoS concerns which differentiate this domain, suitable for high-level architectural design. It is not a substitute for thorough analysis at a later design stage.

Autonomic

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

Performance

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

Standards compliance

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

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

Transactionality

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

3.3.1 Direct Connection application pattern

The Direct Connection application pattern represents the simplest interaction type and is based on a 1-to-1 topology. It allows a pair of applications within the organization to directly communicate with each other. Interactions between a source and a target application can be arbitrarily complex. Generally, complexity can be addressed by breaking down interactions into more elementary interactions.

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

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

- Capacity and availability rules

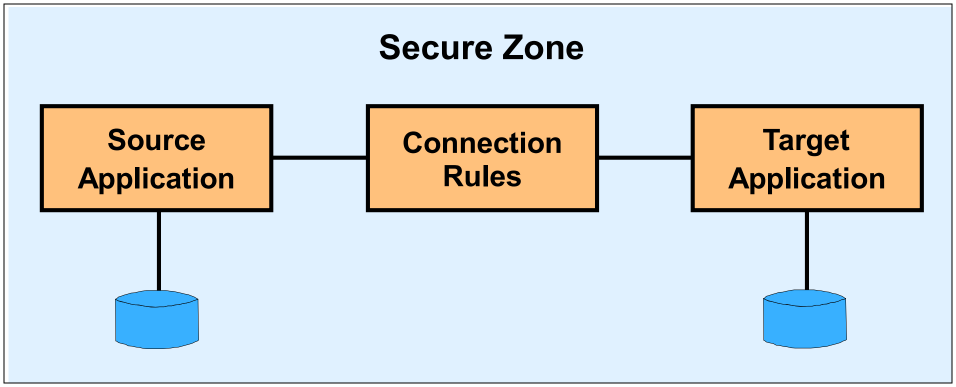


Figure 3-2 Direct Connection application pattern

Note: The Connection Rules component is not needed when there are no modeled rules associated with the connection.

The Direct Connection application pattern has two variations:

- Message Connection variation
- Call Connection variation

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

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

We examine these two variations in more detail later in this section.

Business and IT drivers

The business and IT drivers for choosing the Direct Connection application pattern are to:

- Improve the organizational efficiency
- Reduce the latency of business events
- Support a structured exchange within the organization

- Support real-time one-way message flows
- Support real-time request/reply message flows
- Leverage existing skills
- Leverage the legacy investment
- Enable back-end application integration
- Minimize application complexity

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

Solution

This Application pattern, as shown in Figure 3-2 on page 44, is divided into a number of logical components:

- The Source Application represents one or more applications that are interested in initiating an interaction with the target application.
- The Connection is the line between the source application and the target application representing a point-to-point connection between the two applications.
- The Connection Rules represent any business rules associated with the connection, such as data mapping rules and security rules.
- The Target Application represents a new application, a modified existing application, or an unmodified existing application. This application is responsible for implementing the necessary business services.

Guidelines for use

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

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

Benefits

The Direct Connection application pattern offers the following benefits:

- It works with applications that have simple integration requirements with only a few back-end applications.

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

Limitations

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

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

Putting the Application pattern to use

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

Message Connection variation

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

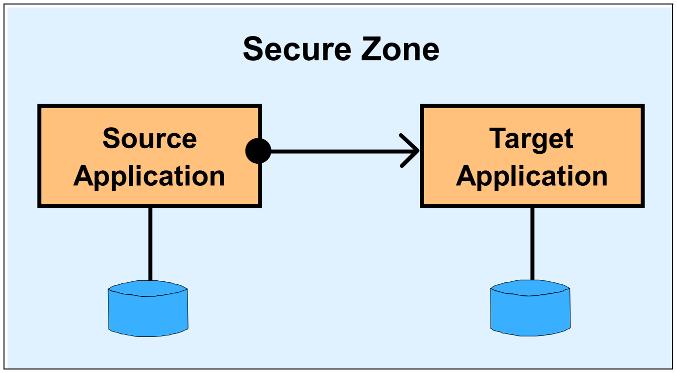


Figure 3-3 Message Connection variation

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

Business and IT drivers

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

- Support real-time one-way message flows

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

Putting the Application pattern to use

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

Call Connection variation

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

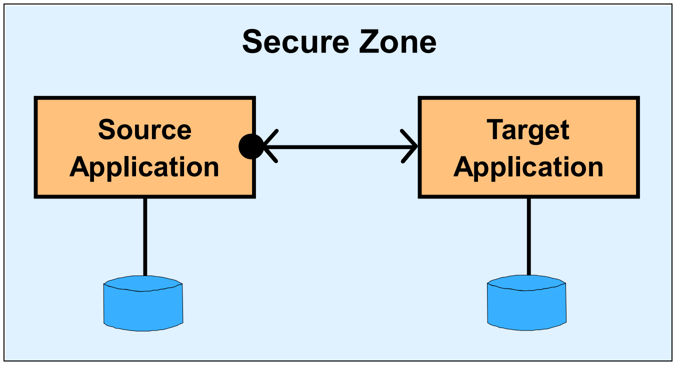


Figure 3-4 Call Connection variation

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

Business and IT drivers

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

- Support real-time request/reply message flows

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

Putting the Application pattern to use

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

3.3.2 Broker application pattern

The Broker application pattern (also known as Aggregator/Broker), shown in Figure 3-5, is based on a 1-to-N topology that separates distribution rules from the applications. It allows a single interaction from the source application to be distributed to multiple target applications concurrently.

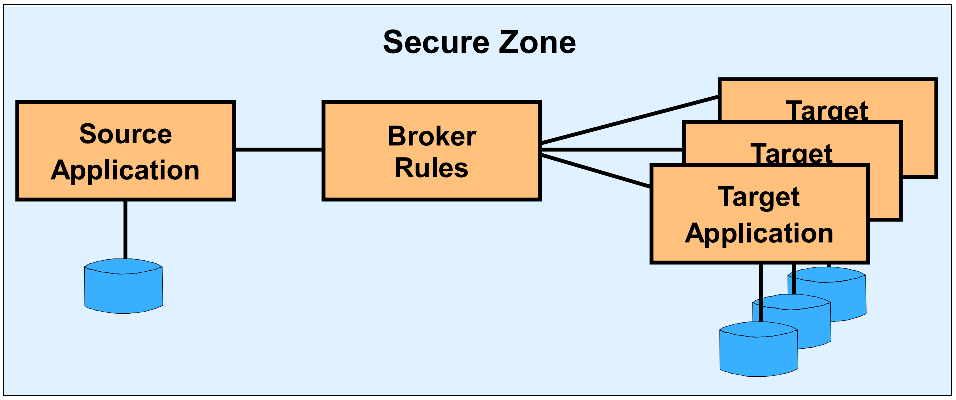


Figure 3-5 Broker application pattern

The Broker application pattern applies to solutions where the source application initiating the operation starts an interaction that is distributed to multiple target applications that are within the organization. It separates the application logic from the distribution logic based on broker rules. The decomposition/recomposition of the interaction is managed by the broker runtime component using these broker rules.

The Broker application pattern was previously known as the *Aggregator* application pattern for read intent and the *Broker* application pattern for update intent.

Look for complete details on the revised Broker application pattern in a future redbook. Until then, refer to the Application Integration::Aggregator or Broker application pattern discussion on the IBM Patterns for e-business Web site:

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

Router variation

The Router variation of the Broker application pattern, shown in Figure 3-6, applies to solutions where the source application initiates an interaction that is forwarded to only one of multiple target applications. The selection of the target application is controlled by the distribution rules that govern the functioning of the connector component.

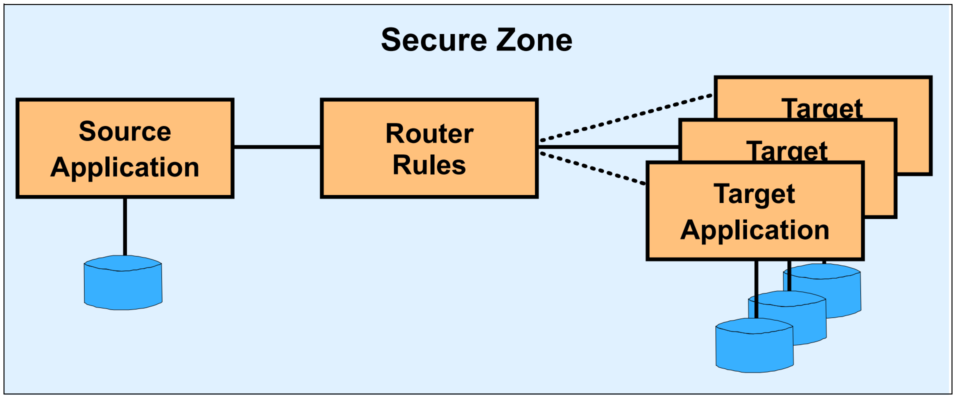


Figure 3-6 Router variation

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

3.3.3 Serial Process application pattern

The Serial Process application pattern, shown in Figure 3-7, is based on a 1-to-N topology where serial process rules are separated from the applications. It allows a single interaction from the source application to execute a sequence of target applications.

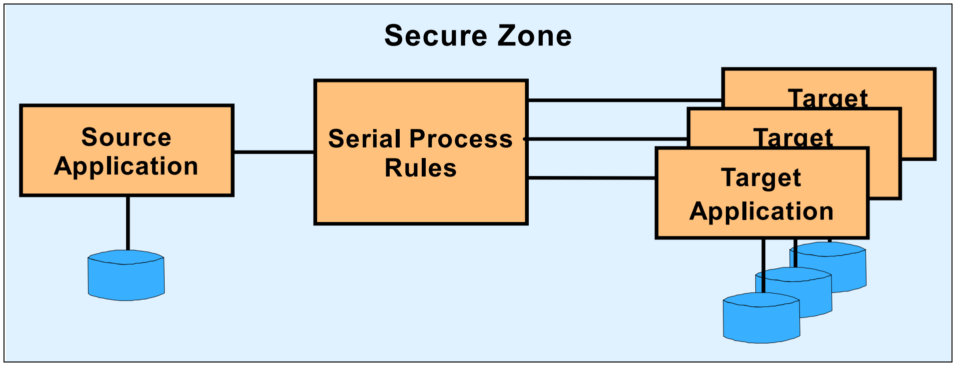


Figure 3-7 Serial Process application pattern

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

Look for complete details on the Serial Process application pattern in a future redbook. Until then, refer to the Application Integration::Managed Process application pattern discussion on the IBM Patterns for e-business Web site:

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

Serial Workflow variation

The Serial Workflow variation of the Serial Process application pattern, shown in Figure 3-8, allows for routable activities (operations requiring human interaction, for example) to be routed to a suitable resource. In addition to the serial process rules, the serial workflow flow rules are supplemented with resource definitions and task-resource relationships. In this context:

- Tasks are activities or pieces of work.
- Resources execute tasks. People, departments, applications, and so forth can all be resources capable of executing particular tasks.
- The task-resource relationship defines which resources are capable of executing which tasks.

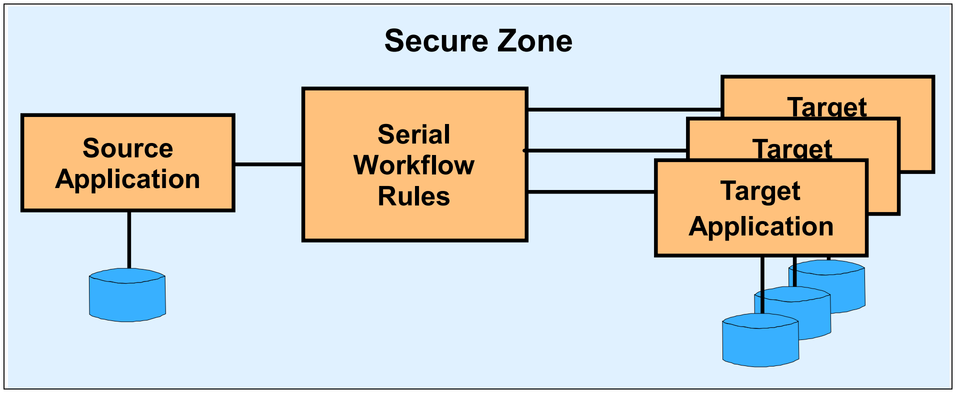


Figure 3-8 Serial Workflow variation

3.3.4 Parallel Process application pattern

The Parallel Process application pattern, shown in Figure 3-9, is a combination of the Serial Process application pattern and the Broker application pattern. The interaction initiated by the source application may control concurrent (parallel) activities on multiple target applications. Each activity may consist of a sequence of operations executed in succession on a target application.

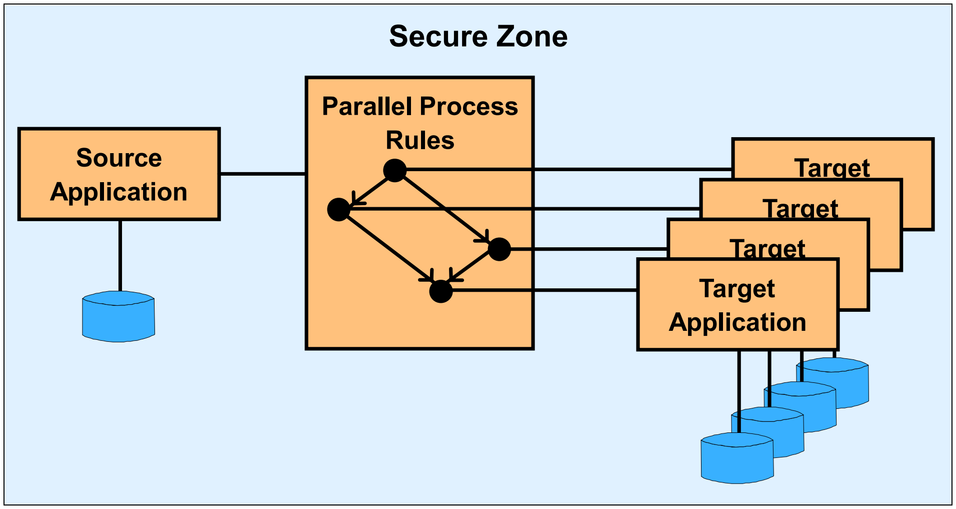


Figure 3-9 Parallel Process application pattern

Similar to the Serial Process application pattern, the Parallel Process application pattern separates process logic from application logic. The parallel process rules must additionally allow for definitions of start and join conditions for activities executing in parallel. A runtime component is required that provides for the start, join, and management of these parallel activities as a unit.

Look for complete details on the Parallel Process application pattern in a future redbook. Until then, refer to the Application Integration::Managed Process application pattern discussion on the IBM Patterns for e-business Web site:

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

Parallel Work Flow variation

Similar to the Serial Work Flow variation, the Parallel Work Flow variation of the Parallel Process application pattern, shown in Figure 3-10, represents an extension of the Parallel Process application pattern to account for routable activities.

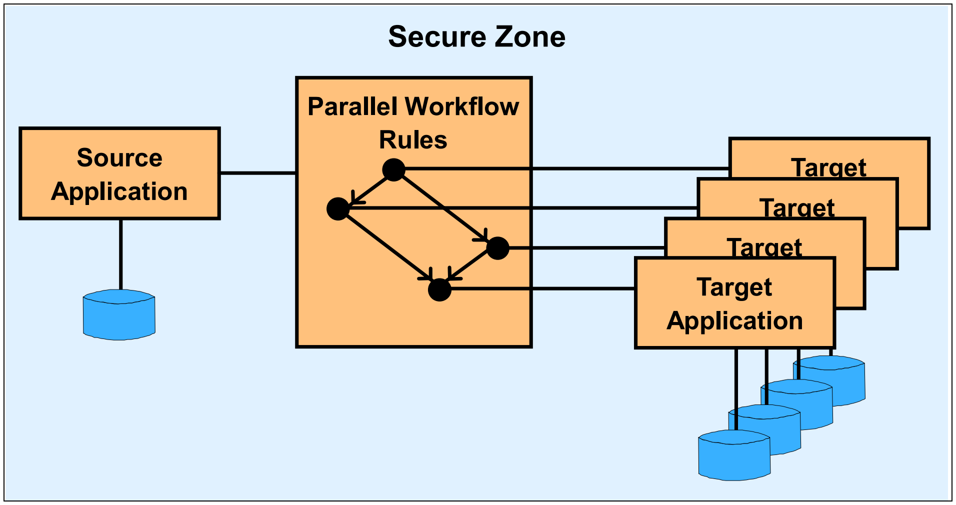


Figure 3-10 Parallel Workflow variation

3.4 Data-focused Application patterns

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

In delineating Data-focused Application Integration patterns, two key environmental questions should be asked:

- Is the enterprise data topology centralized or decentralized?
  - Centralized: This integration effort will bring about centralized access to all or a subset of the enterprise data model.
  - Decentralized: Applications will retain their isolated repositories but now with cohesion based on data integration.
- What is the database affinity type?
  - Homogeneous: all repositories are of the same type.
  - Multi-vender Relational: all repositories are relational with ODBC/JDBC support for interoperability but are from different vendors.

- – Heterogeneous Structured: repositories are not all relational but all have a structured layout.
- – Structured/Non-Structured: the need to integrate non-structured (for example, free-form text) with structured data sources.

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

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

3.5 Runtime patterns

The next step is to choose Runtime patterns that most closely match the requirements of the application. A Runtime pattern uses nodes to group functional and operational components. The nodes are interconnected to solve a business problem. Each Application pattern leads to one or more underpinning Runtime patterns.

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

To understand the Runtime pattern, you will need to review the node definitions provided in 5.1, “Node types” on page 98.

> **Note:** We cover Runtime patterns for Direct Connection in this section. Look for details on the Broker runtime pattern or the Serial/Parallel Process runtime patterns in a future redbook.

3.5.1 Runtime patterns for Direct Connection

When using the Direct Connection runtime pattern, shown in Figure 3-11, the source application uses a connector to access the target application.

The connector itself may be explicitly or implicitly modeled. If the connector is explicitly modeled, the modeler can use decomposition and abstraction techniques to expand the connector to the appropriate level of detail.

The term *Connector* may be qualified by both the connector variation and by the interaction variation. Some examples are:

- Adapter Connector
- Path Connector
- Message Connector

- Call Connector
- Call Adapter Connector

The source and target applications both rely on services provided by their respective hosting servers. These are modeled using the *Application Server/Services* component.

The Rules Directory and Domain QoS Providers may or may not exist. If they do exist, it is a modeling decision as to whether they need to be shown in the Runtime pattern. For example, analysis may determine that connection rules are not an important part of the solution, so the Rules Directory may be left off the Runtime pattern.

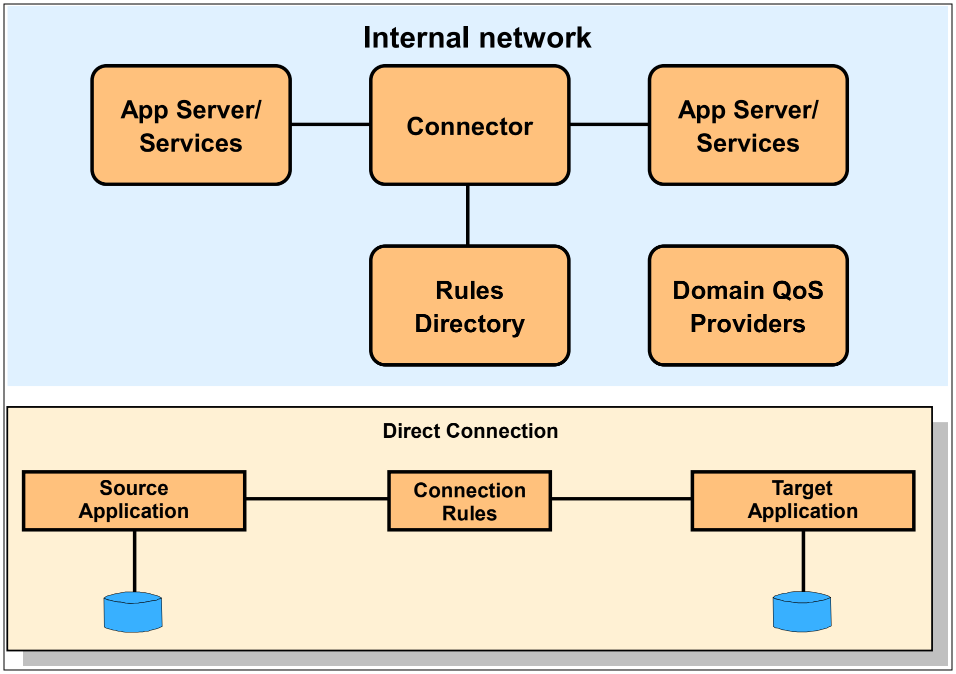


Figure 3-11 Direct Connection runtime pattern

The basic Direct Connection runtime pattern allows integration between a source and target application that use different protocols using a single adapter connector. Direct Connection using a single adapter connector is shown in Figure 3-12.

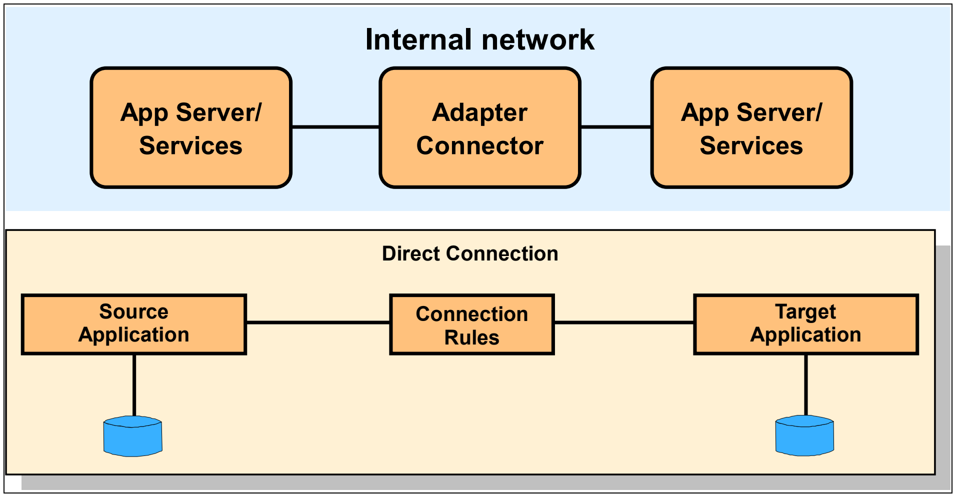


Figure 3-12 Direct Connection using single adapter

Direct Connection can also be implemented using coupling adapter connectors, as shown in Figure 3-13, to improve reuse potential in multiple point to point scenarios. It supports conversion of the request and response into a common protocol between the adapters.

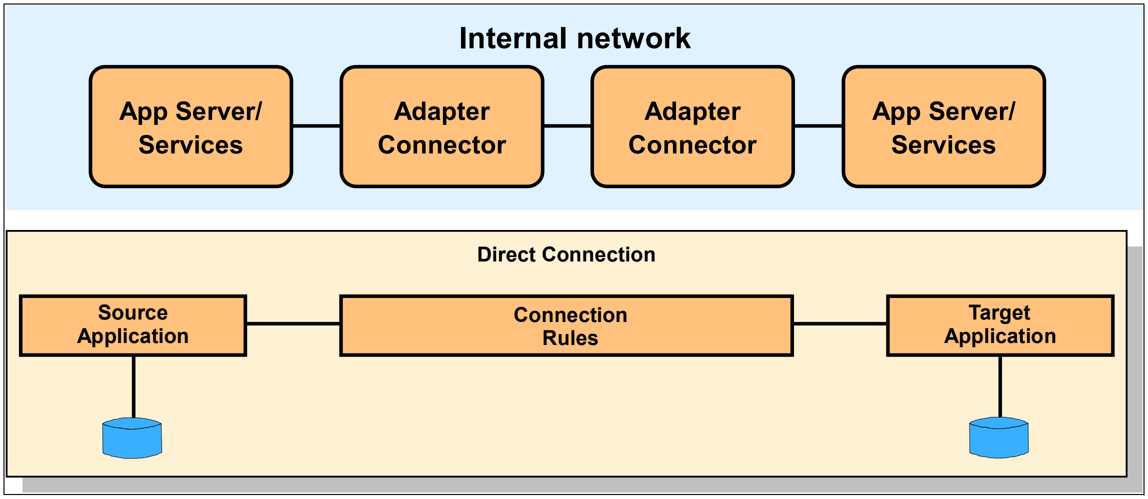


Figure 3-13 Direct Connection using coupling adapters

You may notice that we don't have separate Runtime patterns for the message and call variations of the Direct Connection application pattern. It is still important to identify that your business scenario requires a message or call application pattern, because you can use this knowledge as a consideration when selecting a Product mapping. In the next section we highlight Product mappings that have a more natural fit to the Application pattern message variation or to the Application pattern call variation.

3.6 Product mappings

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

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

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

This section introduces the major products used in the application and provides an overview of the products as they apply to the Direct Connection runtime patterns.

Our sample application, based on the Direct Connection application patterns, has been implemented using IBM WebSphere Application Server V5.0.2 on the Microsoft Windows 2000 platform.

Refer to 5.2, “Product descriptions” on page 101 for descriptions of the products used in these Product mappings.

Note: Although we developed these product mappings from our sample scenarios on the Windows 2000 operating system, there are a number of other options because IBM WebSphere products run on a wide range of platforms (for example, Windows 2000, Linux, AIX®, OS/400®, z/OS®, and so forth).

3.6.1 Product mappings for Direct Connection: Message variation

This section presents Product mappings for the Message Connection variation of the Direct Connection pattern using:

- Web services
- Web Services Gateway
- Java Message Service

Web services

Figure 3-14 shows a Product mapping based on IBM WebSphere Application Server V5.0.2 and the Message Connection variation of the Direct Connection pattern that uses a one-way Web service invocation.

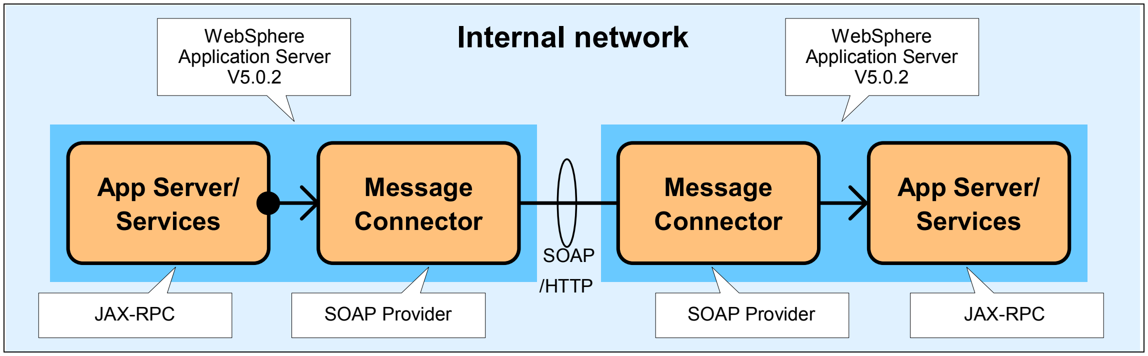


Figure 3-14 Direct Connection::Message Connection: Web services Product mapping

We use coupling adapter connectors to model Web services application integration. This emphasizes the use of adapter connectors to convert the request into the common SOAP/HTTP protocol.

In this case, the source application uses the JAX-RPC API to send a one-way request via the WebSphere V5.0.2 SOAP provider. The target application uses JAX-RPC API to receive the request from the source via its WebSphere V5.0.2 SOAP provider.

We used this combination of runtime nodes and products to implement the sample scenario described in Chapter 8, “Using RPC style Web services” on page 147 and Chapter 9, “Using document style Web services” on page 183.

As shown in Table 3-3, this Product mapping for the Message Connection variation of the Direct Connection pattern can use either of the SOAP messaging styles: RPC style or document style. Table 3-3 also shows that the Web service transmission style is generally one-way for the Message Connection variation; however, request-response may be needed when transport reliability is an issue.

Table 3-3 Direct Connection variation versus Web service type

	RPC style		Document style	
	One-way	Request-response	One-way	Request-response
Message variation	✓		✓	
Call variation		✓		✓

Note: When integrating between J2EE application servers, RMI/IIOP is generally the preferred approach. The intention of this product mapping is to demonstrate that WebSphere V5.0.2 can be used to implement either a Web service requester or a Web service provider.

Web Services Gateway

Figure 3-15 shows another Product mapping based on IBM WebSphere Application Server V5.0.2 and the Message Connection variation of the Direct Connection application pattern that uses a one-way Web service invocation. This time we introduce the Web Services Gateway packaged with IBM WebSphere Application Server Network Deployment V5.0.2.

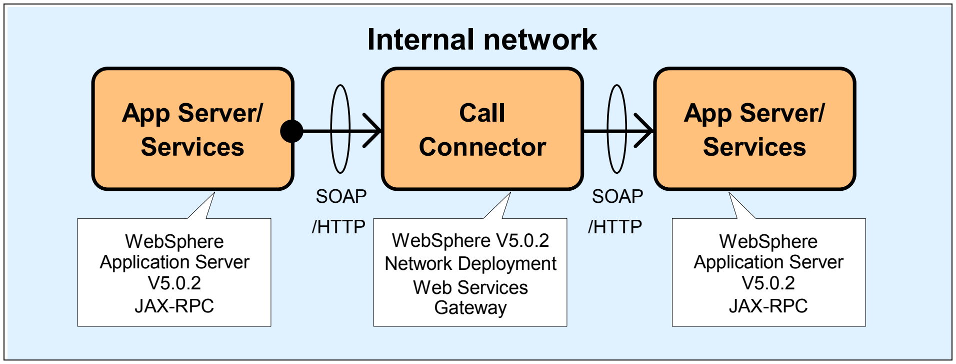

Internal network
App Server/ Services
SOAP /HTTP
Call Connector
SOAP /HTTP
App Server/ Services
WebSphere Application Server V5.0.2 JAX-RPC
WebSphere V5.0.2 Network Deployment Web Services Gateway
WebSphere Application Server V5.0.2 JAX-RPC

Figure 3-15 Direct Connection::Message Connection: Web Services Gateway Product mapping

This product mapping uses connection rules provided by the Web Services Gateway to allow greater control over the point to point connection between the source and target applications. The gateway provides access control and a common access point for internal Web services. It can also protect client applications from changes in the Web services they access.

We used this combination of runtime nodes and products to implement the sample scenario described in Chapter 10, “Using the Web Services Gateway” on page 215.

Java Message Service

Figure 3-16 shows a Product mapping based on the Message Connection variation of the Direct Connection application pattern that uses JMS to send a message from the source application and to receive the sent message at the target application.

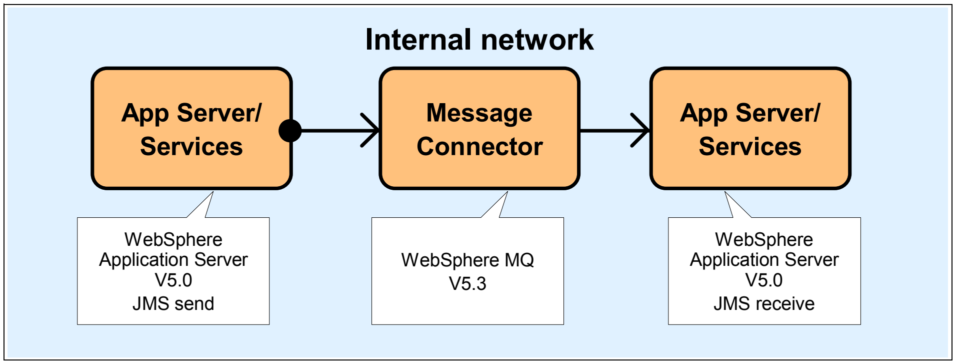


Figure 3-16 Direct Connection::Message Connection: JMS Product mapping

This product mapping uses WebSphere MQ as the transport mechanism for JMS messages. The product mapping uses a WebSphere MQ queue manager on each server to transport the messages. The source application uses JMS to place messages on a local queue. WebSphere MQ is then responsible for ensured delivery of this message to the proper destination, in our case, the WebSphere MQ queue manager on the target application server.

We used this combination of runtime nodes and products to implement the sample scenario described in Chapter 13, “Using Java Message Service” on page 279.

3.6.2 Product mappings for Direct Connection: Call variation

This section presents Product mappings for the Call Connection variation of the Direct Connection pattern using:

- Web services
- Web services to .NET
- Web Services Gateway
- Web Services Gateway with protocol change

- J2EE Connector
- WebSphere Business Integration Adapters

Web services

Figure 3-17 shows a Product mapping based on IBM WebSphere Application Server V5.0.2 and the Call Connection variation of the Direct Connection pattern that uses a request-response Web service invocation.

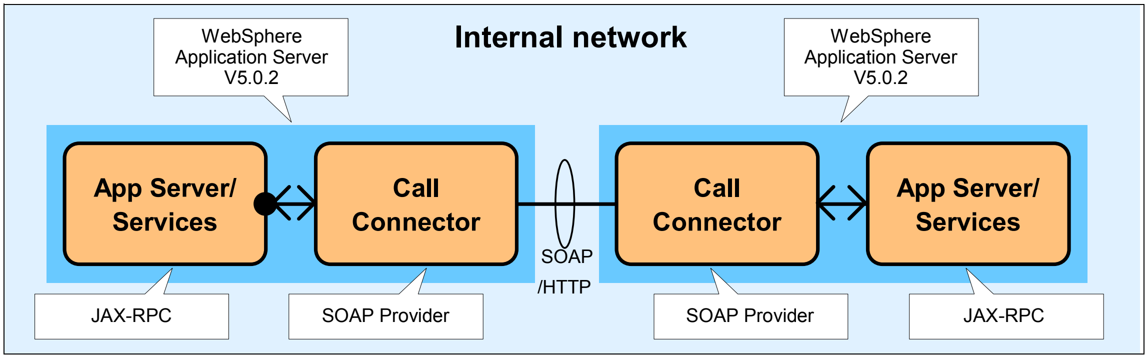


Figure 3-17 Direct Connection::Call Connection: Web services Product mapping

We use coupling adapter connectors to model Web services application integration. This emphasizes the use of adapter connectors to convert the request and response into the common SOAP/HTTP protocol.

In this case, the source application uses the JAX-RPC API to initiate a request-response operation via the WebSphere V5.0.2 SOAP provider. The target application uses JAX-RPC API to receive the request from the source via its WebSphere V5.0.2 SOAP provider.

We used this combination of runtime nodes and products to implement the sample scenario described in Chapter 8, “Using RPC style Web services” on page 147 and Chapter 9, “Using document style Web services” on page 183.

As shown in Table 3-3 on page 59, this Product mapping for the Call Connection variation of the Direct Connection pattern can use either of the SOAP messaging styles: RPC style or document style. Table 3-3 on page 59 also shows that the Web service transmission style is generally request-response for the Call Connection variation.

Note: When integrating between J2EE application servers, RMI/IIOP is generally the preferred approach. The intention of this product mapping is to demonstrate that WebSphere V5.0.2 can be used to implement either a Web service requester or a Web service provider.

Web services to .NET

Figure 3-18 shows a Product mapping providing connectivity between IBM WebSphere Application Server V5.0.2 and Microsoft .NET using the Call Connection variation of the Direct Connection pattern. The source and target applications communicate using a request-response Web service invocation.

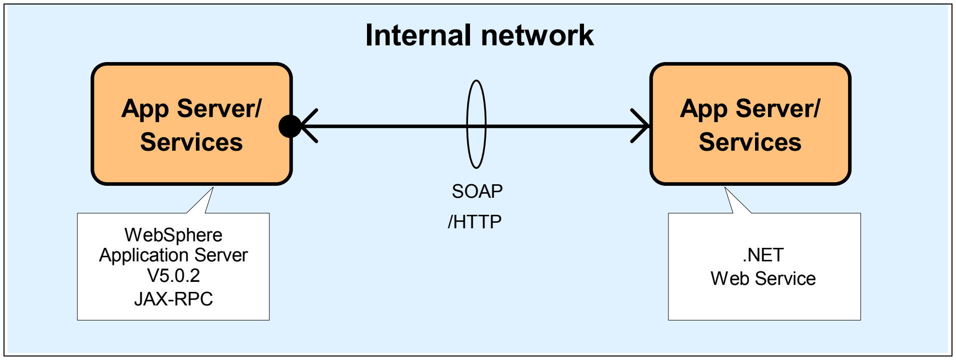


Figure 3-18 Direct Connection::Call Connection: Web services to .NET Product mapping

We chose not to model the connection using coupling adapter connectors in this case. This product mapping focuses on the two platforms and the SOAP/HTTP connection between them.

We used this combination of runtime nodes and products to implement the sample scenario described in 9.5, “Integration with .NET-based Web services” on page 205.

Web Services Gateway

Figure 3-19 shows another Product mapping based on IBM WebSphere Application Server V5.0.2 and the Call Connection variation of the Direct Connection application pattern that uses a request-response Web service invocation. This time we introduce the Web Services Gateway packaged with IBM WebSphere Application Server Network Deployment V5.0.2.

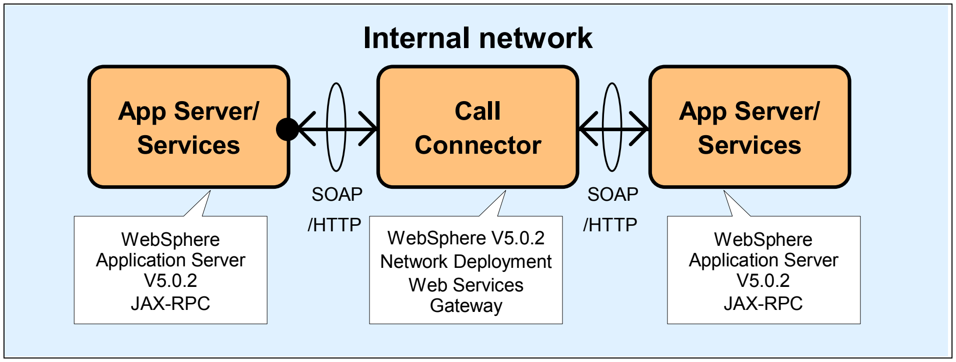

Internal network
App Server/ Services
Call Connector
App Server/ Services
SOAP /HTTP
SOAP /HTTP
WebSphere Application Server V5.0.2 JAX-RPC
WebSphere V5.0.2 Network Deployment Web Services Gateway
WebSphere Application Server V5.0.2 JAX-RPC

Figure 3-19 Direct Connection::Call Connection: Web Services Gateway Product mapping 1

This product mapping uses connection rules provided by the Web Services Gateway to allow greater control over the point to point connection between the source and target applications. The gateway provides access control and a common access point for internal Web services. It can also protect client applications from changes in the Web services they access.

We used this combination of runtime nodes and products to implement the sample scenario described in Chapter 10, “Using the Web Services Gateway” on page 215.

Web Services Gateway with protocol change

Figure 3-20 shows a Product mapping based on IBM WebSphere Application Server V5.0.2 and the Call Connection variation of the Direct Connection application pattern that uses a request-response Web service invocation. In this product mapping the Web Services Gateway provides a protocol change between the source and target applications.

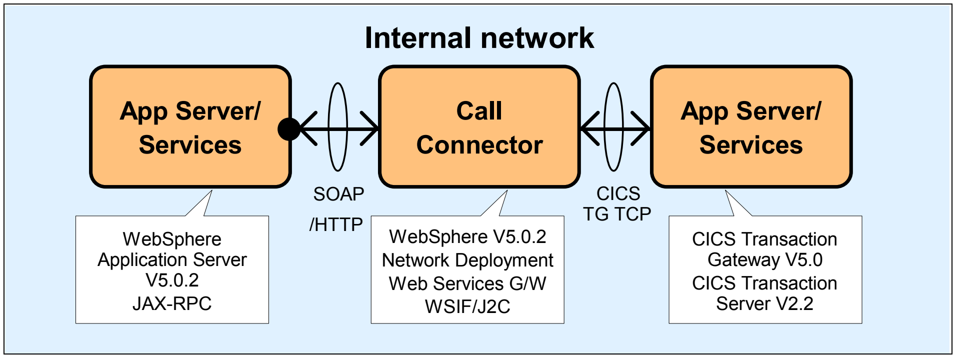

Internal network
App Server/ Services
Call Connector
App Server/ Services
SOAP /HTTP
CICS TG TCP
WebSphere Application Server V5.0.2 JAX-RPC
WebSphere V5.0.2 Network Deployment Web Services G/W WSIF/J2C
CICS Transaction Gateway V5.0 CICS Transaction Server V2.2

Figure 3-20 Direct Connection::Call Connection: Web Services Gateway Product mapping 2

In addition to the connection rules capabilities described in “Web Services Gateway” on page 62, the gateway provides adapter connector capabilities. This product mapping allows a Web service client application to invoke a CICS® target application using SOAP/HTTP. The gateway converts the SOAP/HTTP call to the CICS Transaction Gateway TCP protocol using the Web Services Invocation Framework and the CICS ECI J2EE Connector.

In addition to J2EE Connectors, the Web Services Gateway can be used to connect Web service client applications with target applications that are accessed via JMS or RMI/IIOP.

We used this combination of runtime nodes and products to implement the sample scenario described in Chapter 11, “Using the Web Services Gateway with J2EE Connectors” on page 237.

J2EE Connector

Figure 3-21 shows a Product mapping based on the Call Connection variation of the Direct Connection application pattern where the source application uses a J2EE Connector to call the target application.

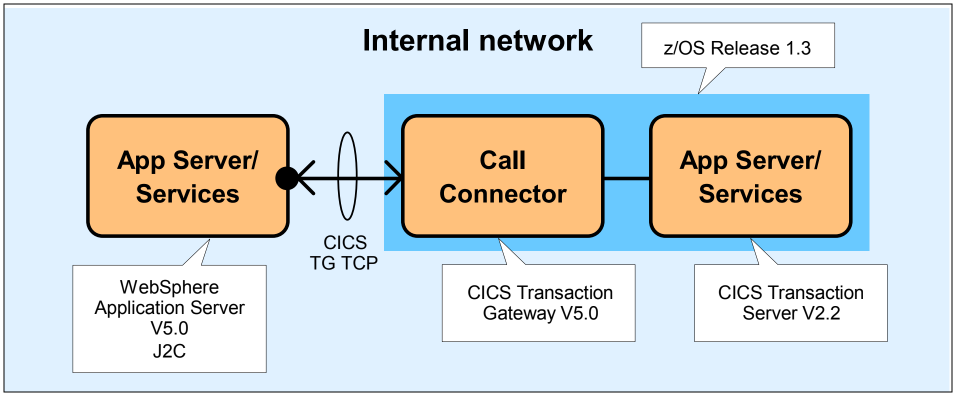


Figure 3-21 Direct Connection::Call Connection: J2EE Connector Product mapping

This product mapping uses the CICS Transaction Gateway TCP protocol to communicate with the CICS Transaction Gateway on the zSeries® enterprise system. The source J2EE application uses the CICS ECI J2EE Connector to access the existing CICS enterprise application, via the CICS Transaction Gateway.

We used this combination of runtime nodes and products to implement the sample scenario described in Chapter 12, “Using J2EE Connectors” on page 263.

WebSphere Business Integration Adapters

Figure 3-22 shows a Product mapping based on IBM WebSphere Application Server Enterprise V5.0.2 and the Call Connection variation of the Direct Connection application pattern. In this product mapping the WebSphere Business Integration Adapter provides adapter connector capabilities between the source and target applications. This product mapping allows a WebSphere Enterprise application to invoke a target application using the Web Services Invocation Framework, JMS and IBM WebSphere MQ V5.3.1.

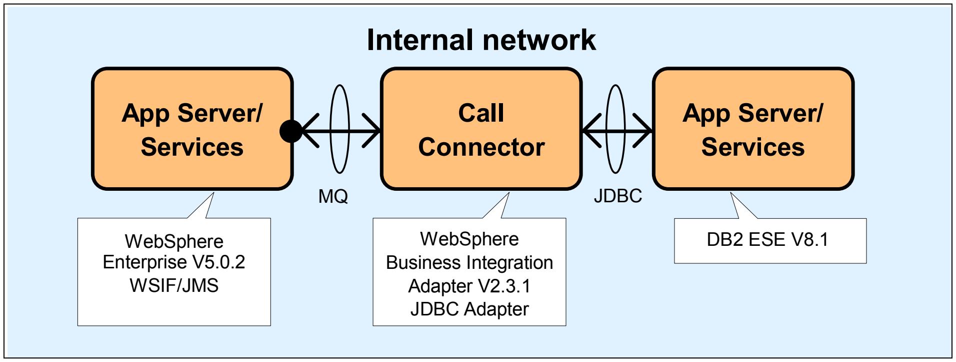

Internal network
App Server/ Services
Call Connector
App Server/ Services
MQ
JDBC
WebSphere Enterprise V5.0.2 WSIF/JMS
WebSphere Business Integration Adapter V2.3.1 JDBC Adapter
DB2 ESE V8.1

Figure 3-22 Direct Connection::Call Connection: WebSphere Business Integration Adapter Product mapping

The product mapping uses the WebSphere Business Integration Adapter JDBC adapter to access a DB2® V8.1 database. You can use a similar approach to integrate a WebSphere Enterprise application with a range of target applications, such as CICS, IMS™, PeopeSoft, SAP, and Siebel. For further details see *Using Web Services for Business Integration*, SG24-6583 (to be released late in 2003).

3.7 Previous Application Integration patterns

Table 3-4 provides an overview of the relationship between the previous Process-focused Application Integration patterns and the revised Process-focused Application Integration patterns presented in this chapter. The differences between the old and new definitions are summarized as follows:

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

Table 3-4 Relationship to old Process-focused Application Integration patterns

	Old Pattern		**New Pattern**
	Information Request (R/O)	**Processing Request (R/W)**	
Application Coordinated	Direct Connection	Direct Connection	Same
Transactional Coordinated	Not applicable	Transactional	Now a Quality of Service
Broker Coordinated	Aggregator	Broker	Broker
Process Managed Coordinated	Not applicable	Managed Process	Split into: ► Serial Process ► Parallel Process

4

Extended Enterprise

The Extended Enterprise business pattern, which is also known as the Business-to-Business or B2B pattern, addresses the interactions and collaborations between business processes in separate enterprises. This pattern can be observed in solutions that implement programmatic interfaces to connect inter-enterprise applications. In other words, it does not cover applications that are directly invoked via a user interface by business partners across organizational boundaries.

In Table 4-1 you can see some cross-industry examples of the Extended Enterprise pattern.

Table 4-1 Cross-industry examples

Service	**Examples**
Buy Side	► Direct Procurement (SCM) ► Indirect Procurement (MRO) ► Supply chain execution
Sell Side	► B2B e-commerce (Distributors)
Trading Partner Modernization	► EDI Modernization
Exchange Participation	► Private e-exchanges ► Public e-exchanges

In Table 4-2 we list some industry-specific example applications that can be implemented though the Extended Enterprise pattern.

Table 4-2 Industry-specific examples

Industry	Example applications
Manufacturing	► Supply chain planning ► Supply chain execution ► Vendor-Managed Inventory
Travel	► Checking flight or room availability ► Making or modifying reservations
Retail	► Checking supplier inventory ► Placing replenishment orders ► Paying suppliers automatically
Financial	► Transferring payments ► Checking account balances ► Obtaining credit information ► Loan Origination ► Processing securities
Telecommunication	► OSS Integration ► Cross organization order management ► Managed service provider interconnect

Note: There are broad similarities between the Application Integration patterns and the Extended Enterprise patterns. The differentiation is mainly in the way Quality of Services aspects affect the Runtime patterns.

What's next

If you are not yet sure that your business problem can be solved by the functionality enabled through an Extended Enterprise solution design, the guidelines in the next section provide additional information on choosing this Business pattern. Business and IT drivers, the e-business context appropriate for this solution type, and additional solution details are discussed here.

If you have determined that the Extended Enterprise business pattern can provide an appropriate solution design for your business needs, the next step is to select an Application pattern. The Extended Enterprise business pattern can be implemented using any one of three Application patterns (see 4.2, “Application patterns” on page 74), providing solution flexibility so that the selected Business pattern can address the specific needs of the business process being automated.

4.1 General guidelines

To help you determine if the Extended Enterprise business pattern is appropriate for the design of your inter-enterprise application integration scenario, this section details the business and IT scenario into which an Extended Enterprise solution fits.

4.1.1 Business and IT drivers

Businesses developing a solution needing the following characteristics should consider using the Extended Enterprise business pattern:

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

4.1.2 Context

The general problem addressed by this pattern is illustrated in Figure 4-1 on page 72. Interactions between partners form a public process, or potentially, multiple distinct public processes. Each of these must be integrated into the private business process flows implemented by each partner. Such integration might be as simple as passing data to a particular application, or as sophisticated as initiating or resuming a multi-step workflow involving several applications and user interactions. For example, Partner A (source application) and Partner B (target application) agree upon sharing specific business processes and a process flow. Partner A invokes a public process flow that in turn may invoke a specific private internal process flow within Partner B's organization. Partner A is not concerned with the details of Partner B's private process flow. Instead, Partner A cares only about the results it expects in response to the invoked public process.

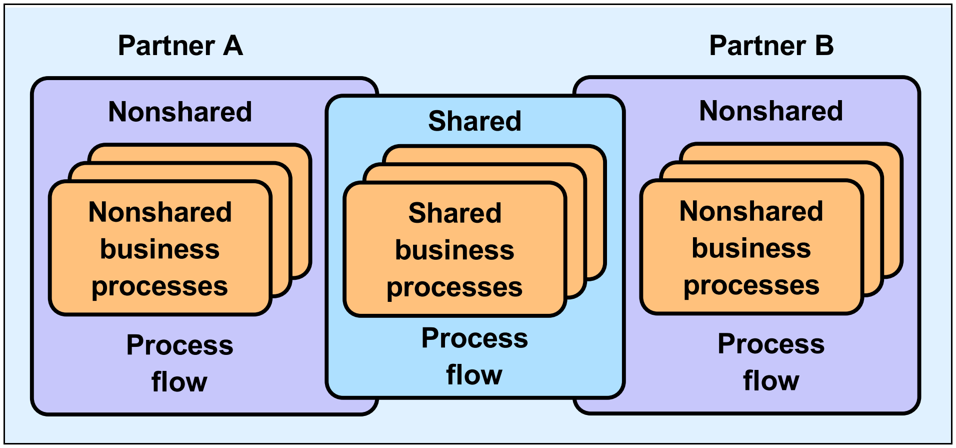

Partner A
Partner B
Nonshared
Shared
Nonshared
Nonshared business processes
Shared business processes
Nonshared business processes
Process flow
Process flow
Process flow

Figure 4-1 Extended Enterprise context

The “golden rule” of business-to-business integration is the less you know about the business partner's private processes and the implementation details of their applications the better off you are. This loose coupling enables organizations to evolve their applications without affecting business partner's applications.

Obviously, specific functionality supported by these applications depends on the particular details of the trading partner agreements and service level agreements between the organizations involved. Yet a survey of such applications in multiple industries reveals certain common approaches that have been successful. These commonalities of success are harvested as the various Application patterns that can be used to implement this Business pattern.

In our sample business scenario, described in Chapter 6., “Business scenarios used in this book” on page 111, ITSO Electronics wants to integrate their wholesale organization with diverse external resellers. The Extended Enterprise pattern will improve organizational efficiency and reduce the latency of business events by integrating the external resellers with the inventory replenishment system and reducing the likelihood of unfilled orders. The Extended Enterprise pattern also applies a structured exchange with business partners and supports real-time access to and from applications. This will allow the resellers to receive the benefits of an updated inventory and receive real-time response of any out of stock items.

The Extended Enterprise pattern also benefits ITSO Electronics by minimizing application complexity and allowing them to integrate their applications with resellers that have unique infrastructure designs. ITSO Electronics will be able to leverage their current skills and legacy investments, eliminating the need for extensive retraining and infrastructure investments.

4.1.3 Solution

The Extended Enterprise pattern might consist of all or some of the following elements:

- Business Entities, which typically:
  - Are programs, applications, or databases that exist within an organization
  - Access and connect to other business entities across the network
- A network which:
  - Is based on TCP/IP and other Internet technologies
  - Can be a dedicated Wide Area Network (WAN) connection
- Business rules that:
  - Manage the integration between the business entities
  - Describe Trading Partner Agreements
  - Use Workflow rules to determine the sequence of steps and the data flow that needs to be used to facilitate the integration. These rules:
    - Describe the sequence of steps that a message needs to go through before being transferred to the other business entity
    - Specify how and where the message should be delivered
  - Use Transformation rules to specify format and protocol transformations that need to be applied to messages that flow between the business entities
- A set of interactions that includes the execution of a jointly-agreed business process
- Patterns based on the those described in Chapter 2., “Fundamental concepts in Process Integration” on page 17.

4.1.4 Putting the pattern to use

This pattern can be observed in solutions such as:

- An electronics retailer/wholesaler, ITSO Electronics from our sample scenario, enabling external resellers to place orders to the ITSO inventory management system.
- Extended Value Chain functions within e-Marketplaces that support cross-enterprise processes such as demand planning and collaborative design.

4.1.5 What's next

If you have determined that the Extended Enterprise business pattern can provide an appropriate solution design for the application you are developing, next select an Application pattern.

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

4.2 Application patterns

The Extended Enterprise application patterns are presented here in order of increasing flexibility and sophistication. As the Application patterns build on each other, their capabilities and reliance on middleware increase, and they require less application development effort. From the following Application patterns, select the one that best fits your requirements:

- Exposed Direct Connection application pattern
  - Message/Call Connection variations
- Exposed Broker application pattern
  - Router variation
- Exposed Serial Process application pattern

Note: The Exposed Parallel Process application pattern is a further possibility, but it is not currently being observed in the Extended Enterprise domain. It is expected to appear at some later stage. In the Process-focused Application Integration domain, see 3.3.4, “Parallel Process application pattern” on page 51.

In this redbook we focus on the Exposed Direct Connection application pattern. A brief overview of the other Extended Enterprise patterns is provided. For full details on the other Extended Enterprise patterns, see the IBM Patterns for e-business Web site:

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

The diagram conventions shown in Figure 4-2 on page 75 are used to describe these successful approaches in the following Application patterns.

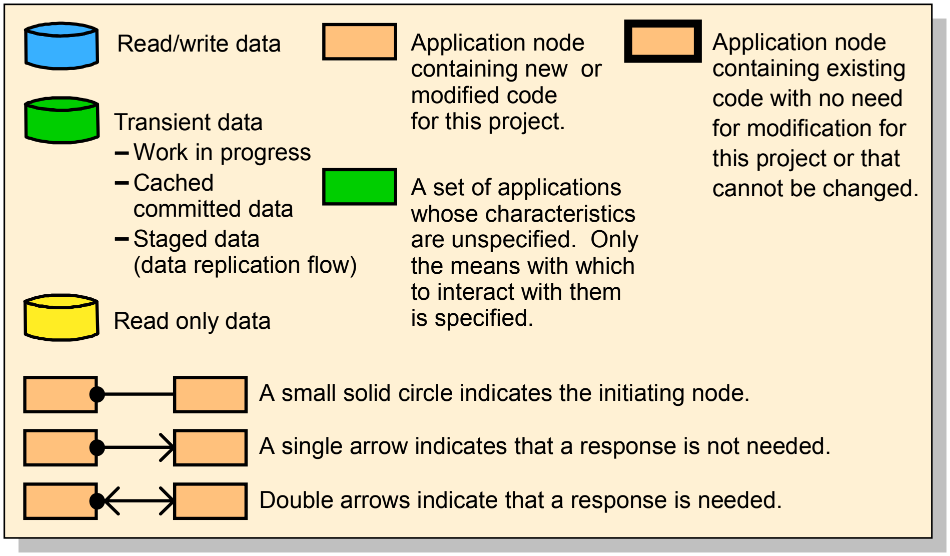

Read/write data
Transient data
– Work in progress
– Cached committed data
– Staged data (data replication flow)
Read only data
Application node containing new or modified code for this project.
A set of applications whose characteristics are unspecified. Only the means with which to interact with them is specified.
Application node containing existing code with no need for modification for this project or that cannot be changed.
A small solid circle indicates the initiating node.
A single arrow indicates that a response is not needed.
Double arrows indicate that a response is needed.

Figure 4-2 Application pattern diagram conventions

Business and IT drivers

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

Table 4-3 Business drivers

Business drivers	Exposed Direct Connection Message variation	Exposed Direct Connection Call variation	Exposed Broker Router variation	Exposed Broker	Exposed Serial Process
Improve the organizational efficiency	✓	✓	✓	✓	✓
Reduce the latency of business events	✓	✓	✓	✓	✓
Support a structured exchange with business partners	✓	✓	✓	✓	✓
Support real-time one-way message flows to partner processes	✓		✓	✓	✓

Business drivers	**Exposed Direct Connection Message variation**	**Exposed Direct Connection Call variation**	**Exposed Broker Router variation**	**Exposed Broker**	**Exposed Serial Process**
Support real-time request/reply message flows to partner processes		✓	✓	✓	✓
Support dynamic routing of messages to one of many target partners’ processes			✓	✓	✓
Support dynamic distribution of messages to multiple target partners’ processes				✓	✓
Support shared public process flows with partners					✓

Table 4-4 IT drivers

IT Drivers	**Exposed Direct Connection Message variation**	**Exposed Direct Connection Call variation**	**Exposed Broker Router variation**	**Exposed Broker**	**Exposed Serial Process**
Leverage existing skills	✓	✓	✓	✓	✓
Leverage the legacy investment	✓	✓	✓	✓	✓
Enable back-end application integration	✓	✓	✓	✓	✓
Minimize application complexity	✓	✓	✓	✓	✓
Minimize enterprise complexity			✓	✓	✓

QoS concerns

This section highlights Quality of Service capabilities that are of particular concern in the Extended Enterprise domain.

In 2.3, “QoS capabilities framework” on page 26, we describe a QoS capabilities framework for Process Integration based on the following general concerns:

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

The following QoS concerns are of particular importance when working in the Extended Enterprise domain.

> **Important:** This profile is intended as a very rough preliminary guide to QoS concerns which differentiate this domain, suitable for high-level architectural design. They are not a substitute for thorough analysis at a later design stage.

Availability

High availability can be a particularly significant issue in the inter-enterprise integration domain. It is important that careful availability management be used to provide acceptable levels of customer service or, in some cases, to meet contractual obligations regarding the availability of the application service being provided.

Federation

To avoid overlap and inconsistencies in the implementation and management of an inter-enterprise application integration scenario, it is crucial to clearly define and agree to the responsibilities of each partner. In particular, agreed mechanisms are needed for passing resource and user authentication and authorization information between domains.

Performance

With inter-enterprise application integration, components of the end-to-end solution are outside the enterprise boundaries and cannot be directly influenced. As a client to an external application, it is difficult to control variables such as response time, workload, and availability.

To minimize such dependencies, loosely coupled and reliable communications should be considered.

Security

This topic includes a range of complex issues. To discuss each of them explicitly is far beyond the scope of this book. For the purpose of this discussion, we assume that the communication channel is secured by using firewalls as well as proper authentication, authorization, and so forth.

In addition, we have to secure the exchange of the data itself. In the case of an intra-enterprise scenario, it may be sufficient to use a trustworthy network. For inter-enterprise communication, this is not longer valid. There is a need to protect (encrypt) the data and be sure about the sender's identity (signature).

Standards compliance

To enable interoperability between enterprises, standards compliance is a key capability in the inter-enterprise integration domain. Widely accepted public standards normally are required in order to have agreement between partners. There is also usually a need for compatibility with standard firewalls when communicating between trusted private networks and untrusted networks, such as the Internet.

4.2.1 Exposed Direct Connection application pattern

The Exposed Direct Connection application pattern represents the simplest interaction type based on a 1-to-1 topology. It allows a pair of applications to directly communicate with each other across organization boundaries. Interactions between a source and a target application can be arbitrarily complex. Generally, complexity can be addressed by breaking down interactions into more elementary interactions.

More complex point-to-point connections will have modeled connection rules such as business rules associated with them, as shown in Figure 4-3. Connection rules are generally used to control the mode of operation of a connector depending on external factors. Examples of connection rules are:

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

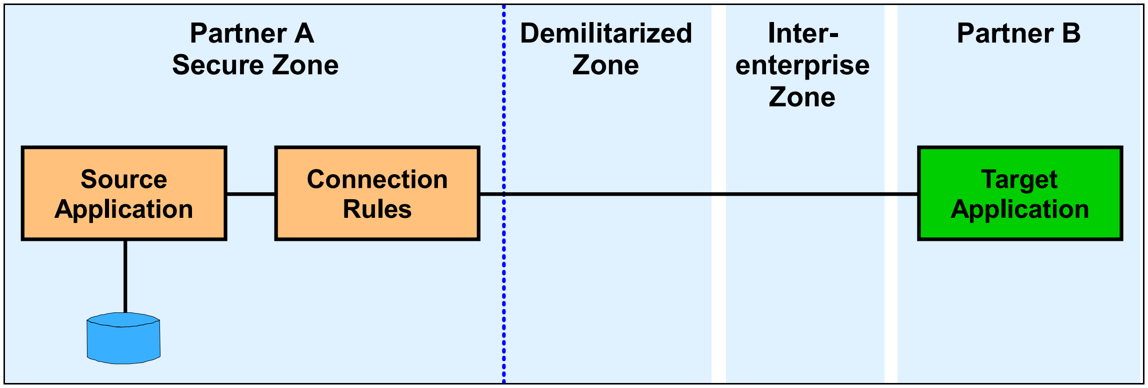


Figure 4-3 Exposed Direct Connection application pattern

> **Note:** The Connection Rules component is not needed when there are no modeled rules associated with the connection.

The Exposed Direct Connection application pattern has two variations:

- Message Connection variation
- Call Connection variation

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

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

We examine these two variations in more detail later in this section.

Business and IT drivers

The business and IT drivers for choosing the Exposed Direct Connection application pattern are to:

- Improve the organizational efficiency
- Reduce the latency of business events
- Support a structured exchange with business partners
- Support real-time one-way message flows to partner processes

- Support real-time request/reply message flows to partner processes
- Leverage existing skills
- Leverage the legacy investment
- Enable back-end application integration
- Minimize application complexity

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

Solution

This Application pattern, as shown in Figure 4-3 on page 79, is divided into a number of logical components:

- The Source Application represents one or more applications that are interested in initiating an interaction with the target application in another organization.
- The Connection is the line between the source application and the target application representing a point-to-point connection between the two applications.
- The Connection Rules represent any business rules associated with the connection, such as data mapping rules and security rules.
- The Target Application represents a new application, a modified existing application, or an unmodified existing application. This application is responsible for implementing the necessary business services.

Since this application is directly exposed across organizational boundaries, it must implement or exploit the necessary security features such as authentication, authorization, confidentiality, integrity, and logging for non-repudiation purposes.

Guidelines for use

Direct integration between applications can be inflexible, in that any changes to one application may have knock-on effects on other applications. This is especially dangerous when integrating across organizational boundaries. Any changes to the exposed target application may require changes to many partner applications. Such changes can be both expensive and time consuming.

Such knock-on effects can be minimized using document-based adapters that wrapper the applications in the exposed connection. Document-based adapters are small programs that convert the mutually agreed upon messages into API calls to existing or new backend applications. This layering technique isolates the exposed applications from partner applications and increases flexibility. Any changes to these exposed applications would only impact the adapter, provided there is no need to change the mutually agreed upon messages.

Message definition should be generalized to further promote flexibility. In other words, messages should not be tightly coupled with backend application APIs. Rather the message should capture all the necessary information required for that logical interaction across business boundaries. Such generalization will help cope with changes to the backend application API without having to change the agreed upon message format.

Benefits

The use of this pattern allows the complete integration of applications belonging to different companies, assuring a real-time and service-oriented access to external data and processes. Source and target applications are clearly decoupled, as are business logic and communication details. Therefore, it is possible to develop different parts of the whole system in an independent way.

Limitations

This pattern implements a direct connection between the source and target application. Hence, it cannot be used for intelligent routing of requests, decomposition and re-composition of requests, and for invoking complex business process workflow as a result of a request from a partner application. Under such circumstances, you should consider a more advanced Application pattern, such as Exposed Broker or Exposed Serial Process.

Putting the Application pattern to use

With the successful integration of their internal retail and wholesale systems, ITSO Electronics has now decided to integrate with their external business partners. The goal is to integrate the external resellers with the internal wholesale system. As with the internal retail system, orders placed by the external resellers will need to update the wholesale inventory system. To meet these requirements ITSO Electronics chooses the Exposed Direct Connection application pattern.

Message Connection variation

The Message Connection variation (also known as Document Exchange), shown in Figure 4-4, applies to solutions where the business process does not require a response from the exposed target application within the scope of the interaction.

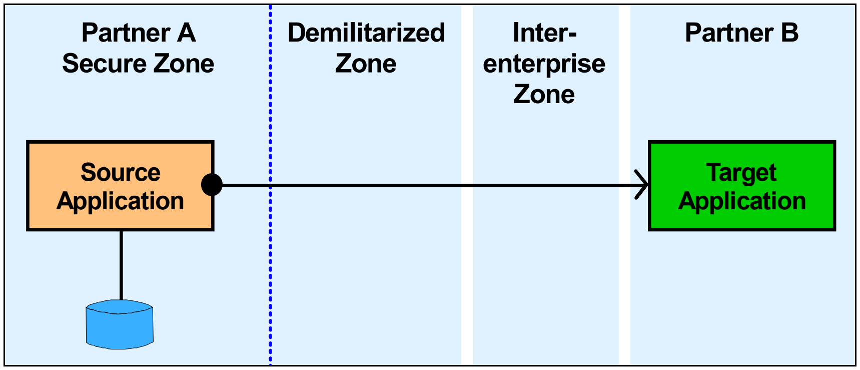


Figure 4-4 Message Connection variation

Note: We chose not to show the connection rules box in Figure 4-4 because we want to focus on the connection itself.

The Message Connection variation of the Exposed Direct Connection application pattern was previously known as the *Document Exchange* application pattern.

Business and IT drivers

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

- Support real-time one-way message flows

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

Putting the Application pattern to use

In our scenario, external business partners of the ITSO Electronics organization need to notify the ITSO wholesale department to update their inventory records when a part needs to be ordered. The external business partners do not require any acknowledgement of the request. ITSO Electronics chooses the Message Connection variation of the Exposed Direct Connection application pattern to meet this requirement.

Call Connection variation

The Call Connection variation (also known as Exposed Application), shown in Figure 4-5, applies to solutions where the business process depends on the exposed target application to process a request and return an response within the scope of the interaction.

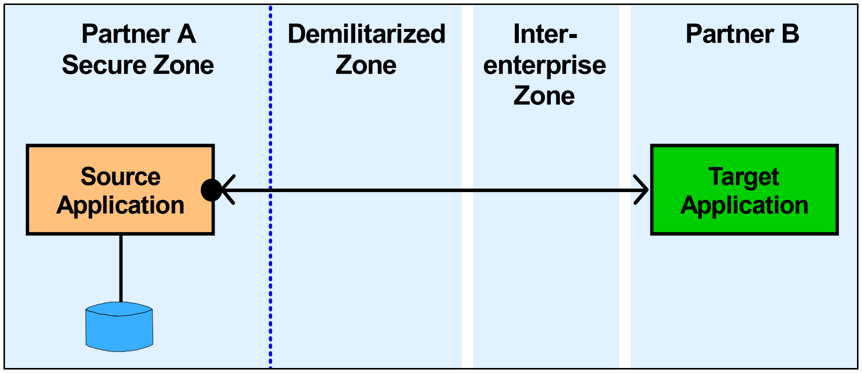


Figure 4-5 Call Connection variation

> **Note:** We chose not to show the connection rules box in Figure 4-5 because we want to focus on the connection itself.

The Call Connection variation of the Exposed Direct Connection application pattern was previously known as the *Application Integration::Exposed Application* application pattern.

Business and IT drivers

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

- Support real-time request/reply message flows

The main driver for selecting this variation is when the business process requires a result message in the interaction.

Putting the Application pattern to use

The final stage of the scenario is addressing any out of stock situations with the external resellers. As with the internal retail system, the external resellers require an immediate notice on any out of stock situations and a delivery date indicating when the order can be filled. To meet these requirements, ITSO Electronics chooses the Call Connection variation of the Exposed Direct Connection application pattern.

4.2.2 Exposed Broker application pattern

The Exposed Broker application pattern (also known as Exposed Business Services), shown in Figure 4-6, is based on a 1-to-N topology that separates distribution rules from the applications. It allows a single interaction from a partner's source application to be distributed to multiple target applications concurrently.

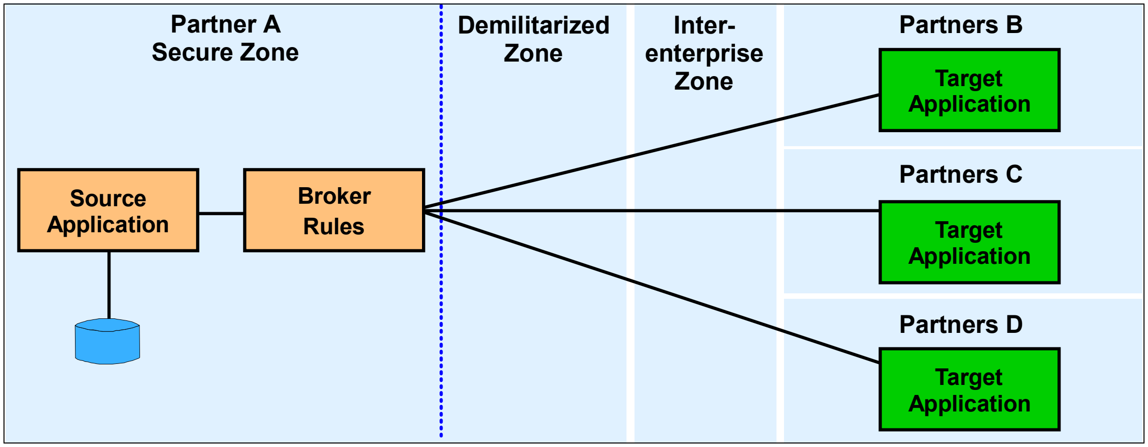


Figure 4-6 Exposed Broker application pattern

The Exposed Broker application pattern applies to solutions where the partner's source application initiating the operation starts an interaction that is distributed to multiple target applications across organization boundaries. It separates the application logic from the distribution logic based on distribution rules. The decomposition/recomposition of the interaction is managed by the connector component using these distribution rules.

The Exposed Broker application pattern was previously known as the *Exposed Business Services* application pattern.

Look for full details on the Exposed Broker application pattern in a future redbook. Until then, refer to the Extended Enterprise::Exposed Business Services application pattern discussion on the IBM Patterns for e-business Web site:

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

Router variation

The Router variation of the Exposed Broker application pattern, shown in Figure 4-7, applies to solutions where the partner's source application initiates an interaction that is forwarded to, at most, one of multiple target applications. The selection of the target application is controlled by the distribution rules that govern functioning of the connector component.

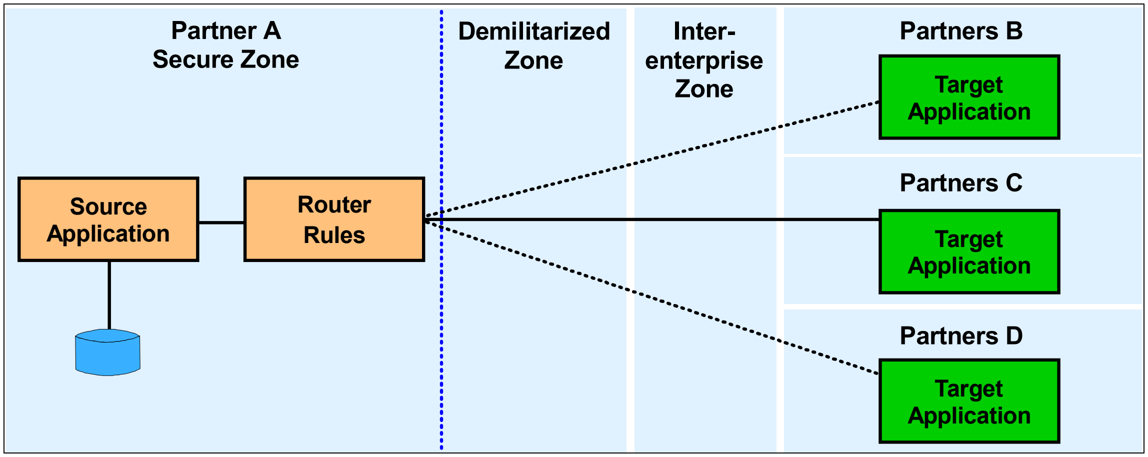


Figure 4-7 Router variation

4.2.3 Exposed Serial Process application pattern

The Exposed Serial Process application pattern (also known as Managed Public Processes), shown in Figure 4-8, is based on a 1-to-N topology where serial process rules are separated from the applications. It allows a single interaction from the partner's source application to execute a sequence of target applications.

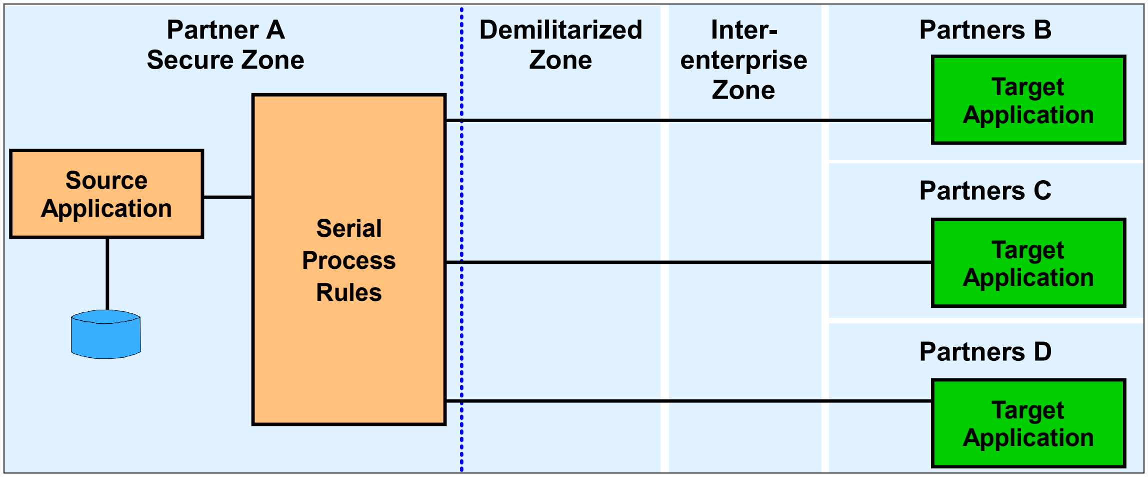


Figure 4-8 Exposed Serial Process application pattern

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

The Exposed Serial Process application pattern was previously known as the *Managed Public Processes* application pattern.

Look for full details on the Exposed Serial Process application pattern in a future redbook. Until then, refer to the Extended Enterprise::Managed Public Processes application pattern discussion on the IBM Patterns for e-business Web site:

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

> **Note:** It is expected that the *Extended Enterprise::Managed Public and Private Processes* application pattern will be reclassified as a composite pattern, based on the public Extended Enterprise::Exposed Serial Process application pattern and the private Application Integration::Serial/Parallel Process application pattern.

4.3 Runtime patterns

The next step is to choose Runtime patterns that most closely match the requirements of the application. A Runtime pattern uses nodes to group functional and operational components. The nodes are interconnected to solve a business problem. Each Application pattern leads to one or more underpinning Runtime patterns.

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

To understand the Runtime pattern, you will need to review the node definitions provided in 5.1, “Node types” on page 98.

> **Note:** We cover Runtime patterns for Exposed Direct Connection in this section. Look for details on the Exposed Broker runtime pattern or the Exposed Serial Process runtime pattern in a future redbook.

4.3.1 Runtime patterns for Exposed Direct Connection

When using the Exposed Direct Connection runtime pattern, shown in Figure 4-9, the source application uses a connector to access the target application.

The connector itself may be explicitly or implicitly modeled. If the connector is explicitly modeled, the modeler can use decomposition and abstraction techniques to expand the connector to the appropriate level of detail.

The term *Connector* may be qualified by both the connector variation and by the interaction variation. Some examples are:

- Adapter Connector
- Path Connector
- Message Connector
- Call Connector
- Call Adapter Connector

The target application relies on services provided by its hosting server. These are modeled using the *Application Server/Services* component.

The Rules Directory and Domain QoS Providers may or may not exist. If they do exist, it is a modeling decision as to whether they need to be shown in the Runtime pattern. For example, analysis may determine that connection rules are not an important part of the solution, so the Rules Directory may be left off the Runtime pattern.

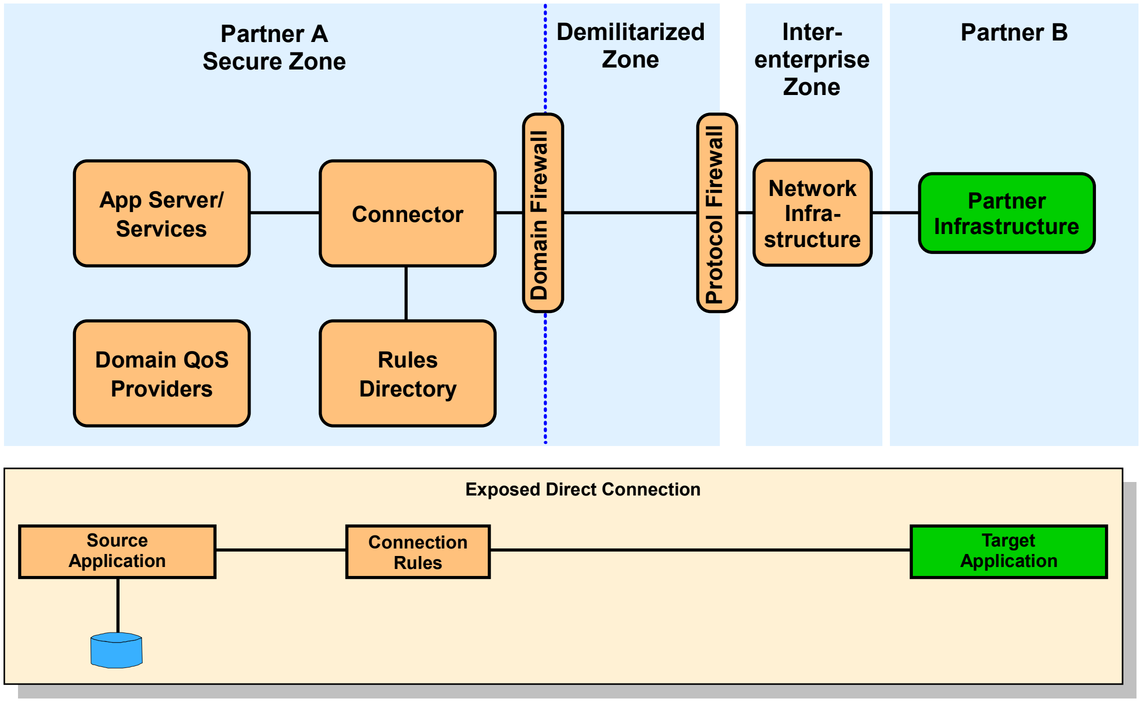


Figure 4-9 Exposed Direct Connection runtime pattern

Figure 4-9 shows a standard pattern of Path Connectors (firewalls and network infrastructure), but other variations do exist with fewer or more firewalls.

The secure zone Connector is primarily concerned with logical connection of the Path Connector to the Application Services, and will therefore often be modeled as an Adapter Connector.

Less secure applications and connectors may be placed within the Demilitarized Zone, depending on local security policies; they are usually placed as shown in Figure 4-9.

This Runtime pattern allows two different organizations to talk to each other with a mutually agreed message format and protocol. Each partner can use their own internal messaging format, using a connector adapter to convert from the internal format to the external format.

You may notice that we don't have separate Runtime patterns for the message and call variations of the Exposed Direct Connection application pattern. It is still important to identify that your business scenario requires a message or call application pattern because you can use this knowledge as a consideration when selecting a Product mapping. In the next section we highlight Product mappings that have a more natural fit to the Application pattern message variation or to the Application pattern call variation.

4.4 Product mappings

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

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

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

This section introduces the major products used in the application and provides an overview of the products as they apply to the Exposed Direct Connection Runtime patterns.

The product mappings shown use the standard pattern of path connectors (firewalls, demilitarized zone, and network infrastructure), although variations exists with fewer or more firewalls.

Our sample application, based on the Exposed Direct Connection application patterns, has been implemented using IBM WebSphere Application Server V5.0.2 on the Microsoft Windows 2000 platform.

Refer to 5.2, “Product descriptions” on page 101 for descriptions of the products used in these Product mappings.

> **Note:** Although we developed these product mappings from our sample scenarios on the Windows 2000 operating system, there are a number of other options because IBM WebSphere products run on a wide range of platforms (for example, Windows 2000, Linux, AIX, OS/400, z/OS, and others).

4.4.1 Product mappings for Exposed Direct Connection: Message variation

This section presents Product mappings for the Message Connection variation of the Exposed Direct Connection pattern using:

- Web services
- Web Services Gateway
- WebSphere Data Interchange

Web services

Figure 4-10 shows a Product mapping based on IBM WebSphere Application Server V5.0.2 and the Message Connection variation of the Exposed Direct Connection pattern that uses a one-way Web service invocation.

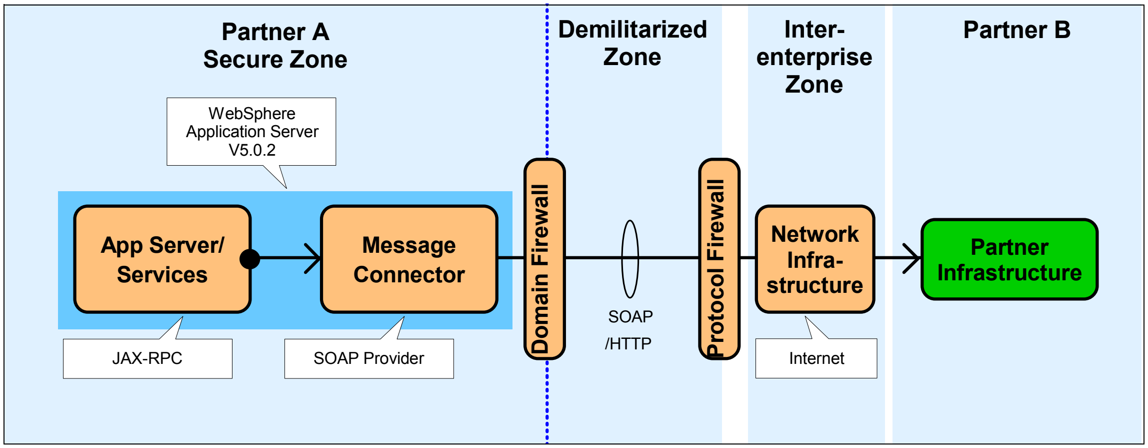


Figure 4-10 Exposed Direct Connection::Message Connection: Web services Product mapping

We use a message adapter connector in Partner A to model Web services application integration. This emphasizes the use of an adapter connector to convert the request into the common SOAP/HTTP protocol.

In this case, the source application uses the JAX-RPC API to send a one-way request via the WebSphere V5.0.2 SOAP provider. Partner B receives the request from the source via its unspecified infrastructure.

We used this combination of runtime nodes and products to implement the sample scenario described in Chapter 14, “Using inter-enterprise Web services” on page 299.

Web Services Gateway

Figure 4-11 shows another Product mapping based on IBM WebSphere Application Server V5.0.2 and the Message Connection variation of the Exposed Direct Connection application pattern that uses a one-way Web service invocation. This time we introduce the Web Services Gateway packaged with IBM WebSphere Application Server Network Deployment V5.0.2.

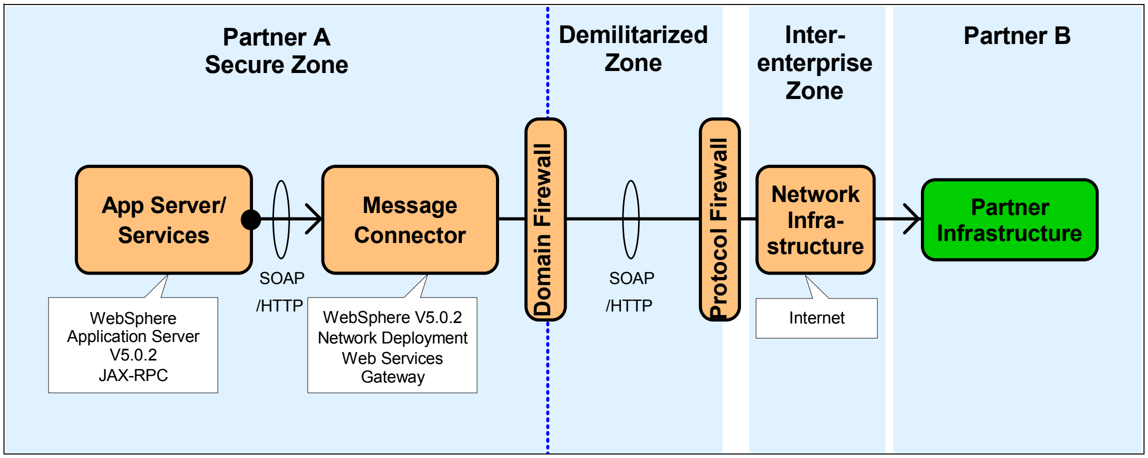


Figure 4-11 Exposed Direct Connection::Message Connection: Web Services Gateway Product mapping

This product mapping uses connection rules provided by the Web Services Gateway to allow greater control over the point-to-point connection between the source application and a business partner’s target application. The gateway provides access control and a common access point for external Web services. It can also protect client applications from changes in the Web services they access.

We used this combination of runtime nodes and products to implement the sample scenario described in Chapter 14, "Using inter-enterprise Web services" on page 299.

WebSphere Data Interchange

Figure 4-12 shows a Product mapping based on WebSphere Data Interchange for Multiplatforms and the Message Connection variation of the Exposed Direct Connection application pattern. This product mapping uses IBM WebSphere MQ V5.3.1 as the transport mechanism between WebSphere Application Server, WebSphere Data Interchange, and iSoft Peer-to-Peer Agent.

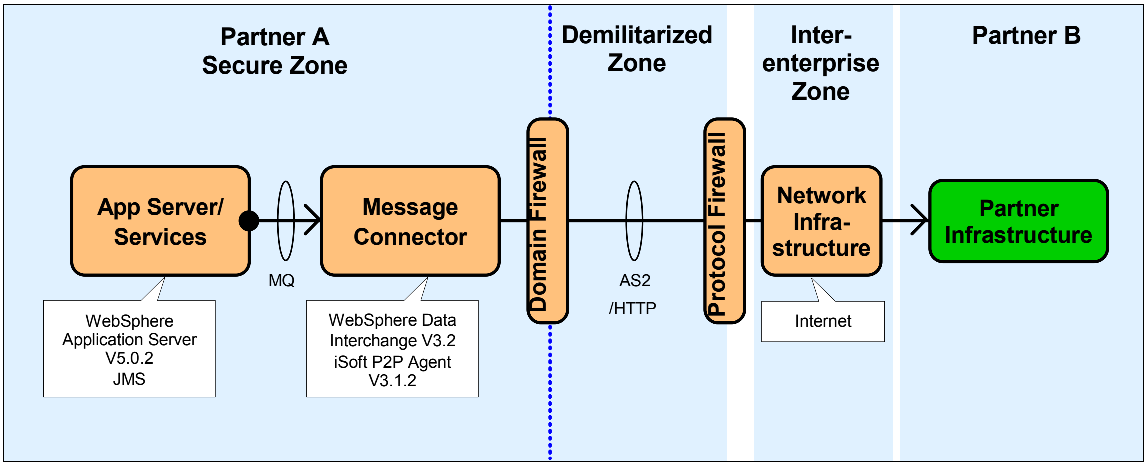


Figure 4-12 Exposed Direct Connection::Message Connection: WebSphere Data Interchange Product mapping

WebSphere Data Interchange V3.2 with CSD1 is used to adapt each type of message or document to partner requirements. The product mapping uses iSoft Peer-to-Peer Agent V3.1.2 to adapt MQ messages and documents to the AS2 EDI protocol for secure and reliable transport of messages and documents with business partners via the Internet.

We discuss this combination of runtime nodes and products in Chapter 15, "Using WebSphere Data Interchange" on page 315. Further details can be found in the following Redpapers:

- *WebSphere Data Interchange Installation and Configuration*, REDP3600
- *Implementation of iSoft and Integration with an EAI solution*, REDP3625

4.4.2 Product mappings for Exposed Direct Connection: Call variation

This section presents Product mappings for the Call Connection variation of the Exposed Direct Connection pattern using:

- Web services
- Web Services Gateway
- Web Services Gateway with protocol change

Web services

Figure 4-13 shows a Product mapping based on IBM WebSphere Application Server V5.0.2 and the Call Connection variation of the Exposed Direct Connection pattern that uses a request-response Web service invocation.

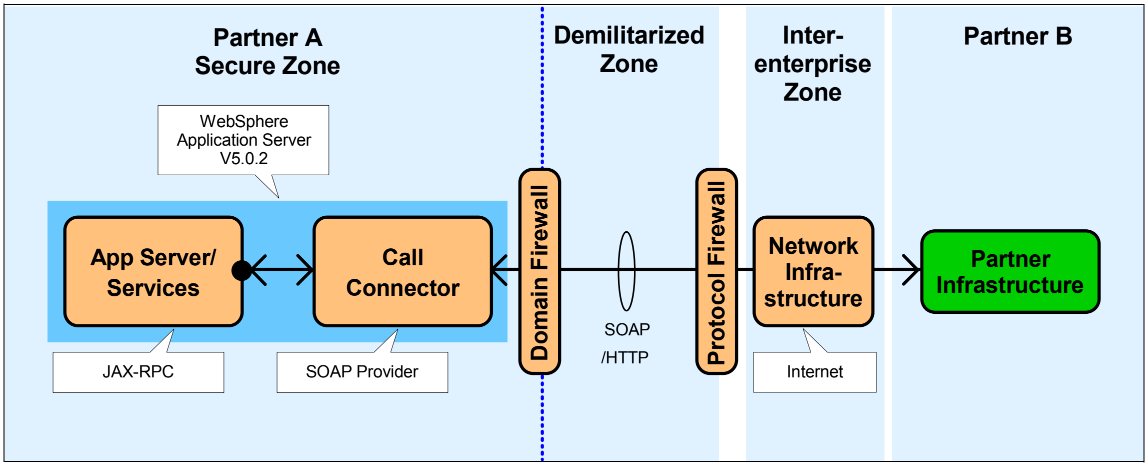


Figure 4-13 Exposed Direct Connection::Call Connection: Web services Product mapping

We use a call adapter connector in Partner A to model Web services application integration. This emphasizes the use of an adapter connector to convert the request and response into the common SOAP/HTTP protocol.

In this case, the source application uses the JAX-RPC API to initiate a request-response operation via the WebSphere V5.0.2 SOAP provider. Partner B receives the request from the source via its unspecified infrastructure.

We used this combination of runtime nodes and products to implement the sample scenario described in Chapter 14, “Using inter-enterprise Web services” on page 299. See also 9.5, “Integration with .NET-based Web services” on page 205 for discussion on integrating with .NET partner infrastructure.

Web Services Gateway

Figure 4-14 shows another Product mapping based on IBM WebSphere Application Server V5.0.2 and the Call Connection variation of the Exposed Direct Connection application pattern that uses a request-response Web service invocation. This time we introduce the Web Services Gateway packaged with IBM WebSphere Application Server Network Deployment V5.0.2.

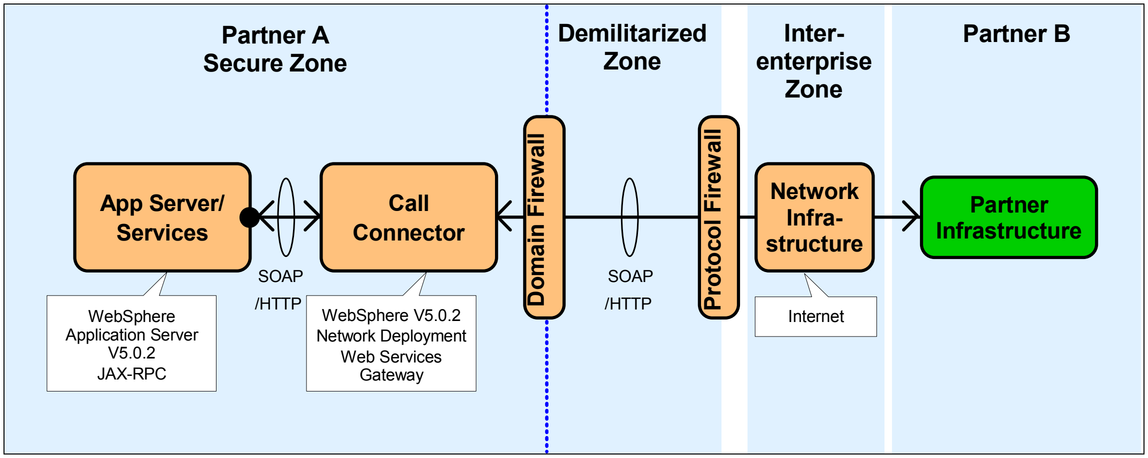


Figure 4-14 Exposed Direct Connection::Call Connection: Web Services Gateway Product mapping 1

This product mapping uses connection rules provided by the Web Services Gateway to allow greater control over the point-to-point connection between the source application and a business partner's target application. The gateway provides access control and a common access point for external Web services. It can also protect client applications from changes in the Web services they access.

We used this combination of runtime nodes and products to implement the sample scenario described in Chapter 14, "Using inter-enterprise Web services" on page 299.

Web Services Gateway with protocol change

Figure 4-15 shows a Product mapping based on IBM WebSphere Application Server V5.0.2 and the Call Connection variation of the Exposed Direct Connection application pattern that uses a request-response Web service invocation. In this product mapping the Web Services Gateway provides a protocol change between Partner B and the target application in Partner A.

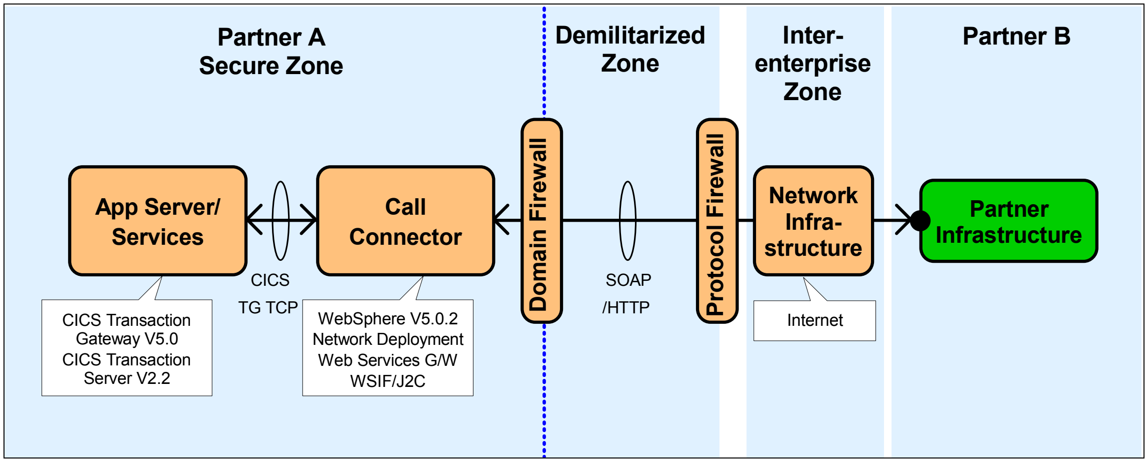


Figure 4-15 Exposed Direct Connection::Call Connection: Web Services Gateway Product mapping 2

In addition to the connection rules capabilities described in “Web Services Gateway” on page 93, the gateway provides adapter connector capabilities. This product mapping allows a Web service client application in Partner B to invoke a CICS target application in Partner A using SOAP/HTTP. The gateway converts the SOAP/HTTP call to the CICS Transaction Gateway TCP protocol using the Web Services Invocation Framework and the CICS ECI J2EE Connector.

In addition to J2EE Connectors, the Web Services Gateway can be used to connect Web service client applications with target applications that are accessed via JMS or RMI/IIOP.

This combination of runtime nodes and products is based on the sample scenarios described in Chapter 14, “Using inter-enterprise Web services” on page 299 and Chapter 11, “Using the Web Services Gateway with J2EE Connectors” on page 237.

4.5 Previous Extended Enterprise patterns

Table 4-5 provides an overview of the relationship between the previous Extended Enterprise patterns and the revised Extended Enterprise patterns presented in this chapter. The definition changes are summarized as follows:

- The revised Extended Enterprise patterns use the same names as the revised Process-focused Application Integration patterns, with the added prefix of “Exposed”. This provides consistency between the intra-enterprise and inter-enterprise pattern names.

- The Managed Public and Private Process pattern will be reclassified as a composite pattern, based on the public Extended Enterprise::Exposed Serial Process application pattern and the private Application Integration::Serial/Parallel Process application pattern.

Table 4-5 Relationship to old Extended Enterprise patterns

Old Pattern	**New Pattern**
Document Exchange	Exposed Direct Connection ► Exposed Message Connection variation
Exposed Application	Exposed Direct Connection ► Exposed Call Connection variation
Exposed Business service	Exposed Broker ► Exposed Router variation
Managed Public Process	Exposed Serial Process
Managed Public & Private Process	Composite pattern based on Process-focused Application Integration and Extended Enterprise

5

Node types and Product descriptions

This chapter provides definitions of the nodes used in the intra-enterprise and inter-enterprise Runtime patterns that are described in 3.5, “Runtime patterns” on page 54 and 4.3, “Runtime patterns” on page 86.

It also provides Product definitions for the products used in the intra-enterprise and inter-enterprise Product mappings that are described in 3.6, “Product mappings” on page 57 and 4.4, “Product mappings” on page 88.

5.1 Node types

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

Application server/Services

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

Applications may also rely on services provided by their hosting server to interact with other applications. Examples of services provided by the Application Server/Services node include:

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

Connector

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

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

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

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

See also:

- “Adapter connector” on page 98
- “Path connector” on page 99

Adapter connector

Adapter connectors are concerned with enabling logical connectivity by bridging the gap between the context schema and protocols used by the source and target components. An adapter connector is one that supports the transformation of data and protocols.

Path connector

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

Rules directory

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

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

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

Domain QoS providers

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

> Integration pattern = topology pattern + QoS providers

The QoS capabilities framework can be used to address the particular QoS concerns for the domain:

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

The domain QoS providers may or may not exist. If they do exist, they can still be left off the Runtime pattern, for example, when analysis determines that domain QoS providers are not an important part of the solution.

Protocol firewall node

A firewall is a hardware/software system that manages the flow of information between the Internet and an organization's private network. Firewalls can prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets, and can block some virus attacks (as long as those viruses are coming from the Internet). A firewall can separate two or more parts of a local network to control data exchange between departments. Components of firewalls include filters or screens, each of which controls transmission of certain classes of traffic. Firewalls provide the first line of defense for protecting private information, but comprehensive security systems combine firewalls with encryption and other complementary services, such as content filtering and intrusion detection.

Firewalls control access from a less trusted network to a more trusted network. Traditional implementations of firewall services include:

- Screening routers (the protocol firewall)
- Application gateways (the domain firewall)

A pair of firewall nodes provides increasing levels of protection at the expense of increasing computing resource requirements. The protocol firewall is typically implemented as an IP router.

Domain firewall node

The domain firewall is typically implemented as a dedicated server node.

A domain firewall is usually used to separate a secure zone, such as the internal network, from a demilitarized zone. This provides added security protection from the un-secure zone, such as the Internet.

Partner infrastructure

Partner infrastructure includes the partner's installed applications, data, computing, and network infrastructure. Partner infrastructure has unspecified internal characteristics; only the means with which to interact with it is specified.

Inter-enterprise network infrastructure

Inter-enterprise network infrastructure includes the network infrastructure allowing connectivity between enterprises. Inter-enterprise network infrastructure has unspecified internal characteristics; only the means with which to interact with it is specified.

Local Area Network

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

Wide Area Network

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

5.2 Product descriptions

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

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

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

Our sample application, based on the Application Integration pattern and Extended Enterprise business pattern, has been implemented using IBM WebSphere Application Server V5.0 in a Microsoft Windows 2000 environment.

The following alternatives are detailed for implementation of the point-to-point connection between applications:

- For the Application Integration pattern:
  - Web services using the Web services support provided with IBM WebSphere Application Server base V5.0.2
  - Web services using the Web Services Gateway provided with IBM WebSphere Application Server Network Deployment V5.0.2
  - J2EE Connectors using IBM CICS
  - Java Message Service (JMS) using IBM WebSphere MQ
  - WebSphere Business Integration Adapters JDBC adapter
- For the Extended Enterprise application pattern:
  - Web services using the Web services support provided with IBM WebSphere Application Server base V5.0.2

  - Web services using the Web Services Gateway provided with IBM WebSphere Application Server Network Deployment V5.0.2
  - Electronic Data Interchange using WebSphere Data Interchange

This section introduces the major products used in the application and provides an overview of the products as they apply to these two Runtime patterns.

> **Note:** You only need a subset of the products detailed in this chapter, depending on the application connectivity needed. Refer to the product mapping diagrams to determine which products are needed for specific patterns.

5.2.1 IBM WebSphere Application Server

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

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

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

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

IBM WebSphere Application Server Express V5.0

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

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

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

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

IBM WebSphere Application Server base V5.0

IBM WebSphere Application Server base V5.0 provides a robust application deployment environment for single-server light production situations.

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

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

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

IBM WebSphere Application Server Network Deployment V5.0

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

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

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

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

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

IBM WebSphere Application Server Enterprise V5.0

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

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

WebSphere MQ is bundled with the package (except on z/OS).

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

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

5.2.2 IBM WebSphere MQ

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

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

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

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

WebSphere MQ supports a variety of application programming interfaces (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 a number of connectors and gateways to a variety of other products, such as Microsoft Exchange, Lotus® Domino®, SAP/R3, CICS, and IMS, to name just a few.

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

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

5.2.3 IBM CICS

IBM Customer Information Control System (CICS) is a family of application servers and connectors that provides industrial-strength, online transaction management and connectivity for mission-critical applications. Existing CICS installations process more than $1 trillion in transactions each day.

Our J2EE Connector scenario makes use of the following IBM CICS components:

- IBM CICS Transaction Server
- CICS Transaction Gateway (CICS TG)

More information can be found at the IBM CICS Web site:

`http://www.ibm.com/software/ts/cics`

IBM CICS Transaction Server

The IBM CICS Transaction Server is an online transaction processing (OLTP) environment for hosting business applications. It provides transaction management services, operating system services, database services, security services, and a variety of client access services. It allows resource managers to join transactions, and provides them with notification of transaction events, such as commit or rollback.

It supports numerous application development environments and models, including COBOL, PL/I, Java, EJB, and object-oriented (OO) development, in any combination. CICS Transaction Server runs on z/OS, OS/390®, and VSE/ESA™.

CICS Transaction Gateway

The CICS Transaction Gateway (CICS TG) has a long heritage as a Java connector for CICS, originally being provided as the CICS Gateway for Java for use with the CICS Client. Since then, CICS TG has advanced along with the Java world and now provides three principal interfaces for communication with CICS:

- Base classes
- Common Connector Framework API
- J2EE Common Client Interface

CICS TG is a well-proven and established product, available on multiple platforms, including AIX, Windows, z/OS, Solaris, HP-UX, and Linux on zSeries.

5.2.4 WebSphere Business Integration Adapters

IBM WebSphere Business Integration Adapters V2.2 are part of the WebSphere Business Integration family of products and offerings. They can be thought of as the “spokes” in a hub-and-spoke architecture radiating out to packaged applications, legacy systems, mainframe and e-business systems. They enable the exchange of data and transactions between systems. Their capability to transform data can be used to access databases and integrate disparate data formats.

There are more than 60 systems and versions of systems that are currently supported by WebSphere Business Integration Adapters. They are categorized into four strategic areas:

- Technology adapters provide various ways to access data, technologies, and protocols that enhance an integration infrastructure.
- Application adapters extract data and transaction information from both leading and industry-specific packaged applications.
- Mainframe adapters provide access to application data in z/OS systems and to OS/400.
- e-business adapters for securely connecting over the firewall to customer desktops, trading partner internal applications, and online marketplaces and exchanges.

More information about IBM WebSphere Business Integration Adapters can be found at:

`http://www.ibm.com/websphere/integration/wbiadapters`

5.2.5 WebSphere Data Interchange

WebSphere Data Interchange for Multiplatforms V3.2 provides advanced translation, validation, and batched information exchange capabilities for Electronic Data Interchange (EDI) standards and for XML. WebSphere Data Interchange V3.2 electronically translates EDI format data, such as invoices, purchase orders, and billing forms, for exchange with trading partners.

WebSphere Data Interchange V3.2 supports industry implementations of the ANSI X12, EDIFACT, VICS, UCS and Rail standards. Translation can take place between any combination of EDI, XML, or structured Application Data Format.

WebSphere Data Interchange V3.2 provides advanced data validation and standards compliance functions that allow the functional acknowledgments, defined by some standards, to be generated in response to inconsistencies in the data content. WebSphere Data Interchange V3.2 can be configured to both construct and de-construct envelopes of EDI format data that contain batches of related EDI items such as invoices or purchase orders.

WebSphere Data Interchange V3.2 is available on the Windows 2000, AIX, and z/OS platforms.

WebSphere Data Interchange V3.2 supports integration with WebSphere MQ, enabling inter-operation with a wide range of enterprise applications, business process engines (such as the IBM CrossWorlds® InterChange Server), information brokers (such as WebSphere MQ Integrator), and ERP systems (such as SAP R3).

WebSphere Data Interchange V3.2 provides for communication with trading partners via both value added networks (VANs) or Internet B2B gateways.It provides an easy-to-use, configurable interface that enables connection to leading VAN and Internet gateways. The WebSphere Business Connection offerings that provide AS1 and AS2 support, and the IBM e-business hosting Expedite VAN gateway, are two examples of supported gateways from IBM.

More information about IBM WebSphere Data Interchange can be found at:

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

Part 2

Scenarios and guidelines

Part 2 presents guidelines for applying the Patterns approach to a sample business scenario and for selecting application integration technologies.

Included in Part 2 are the following chapters:

- Chapter 6, “Business scenarios used in this book” on page 111
- Chapter 7, “Technology options” on page 127

6

Business scenarios used in this book

To demonstrate the use of the Patterns for e-business presented in this redbook, we came up with the business scenarios outlined in this chapter for our imaginary customer, ITSO Electronics. Of course, simple business scenarios like these don’t exist in reality, but they help us to explain how the Patterns for e-business approach can be applied.

6.1 Customer overview

This section provides some background information on our imaginary client, ITSO Electronics, including:

- Business profile and goals
- Existing environment
- Non-functional requirements, or Quality of Service requirements

6.1.1 Business profile

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

6.1.2 Business goals

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

- Reduce costs by reducing the staff workload associated with placing stock replenishment orders with the wholesale department. This should be achieved by integrating the ordering system with the wholesale inventory system.
- Increase customer satisfaction by reducing latency between the retail ordering process and the wholesale inventory process, thereby decreasing the likelihood of an item being out-of-stock.

After the internal processes have been integrated, ITSO Electronics plans to enable their external resellers to place stock replenishment orders with the wholesale department.

6.1.3 Existing environment

This section describes the existing environment at ITSO Electronics.

Business perspective

From the business perspective, the existing environment includes:

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

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

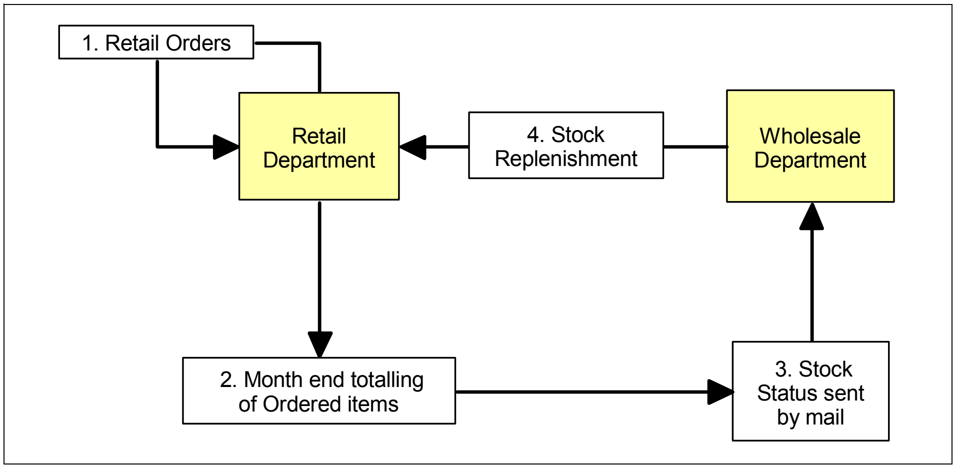


Figure 6-1 ITSO Electronics wholesale ordering process flow

IT perspective

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

- An existing wholesale inventory system. This important legacy system implements the core business processes of the wholesale department.
- An existing retail ordering system. This system is used by retail staff and has recently been upgraded to a Self-Service browser-based J2EE application.
- External reseller’s with their own, heterogeneous IT infrastructures.
- Limited existing application integration infrastructure.

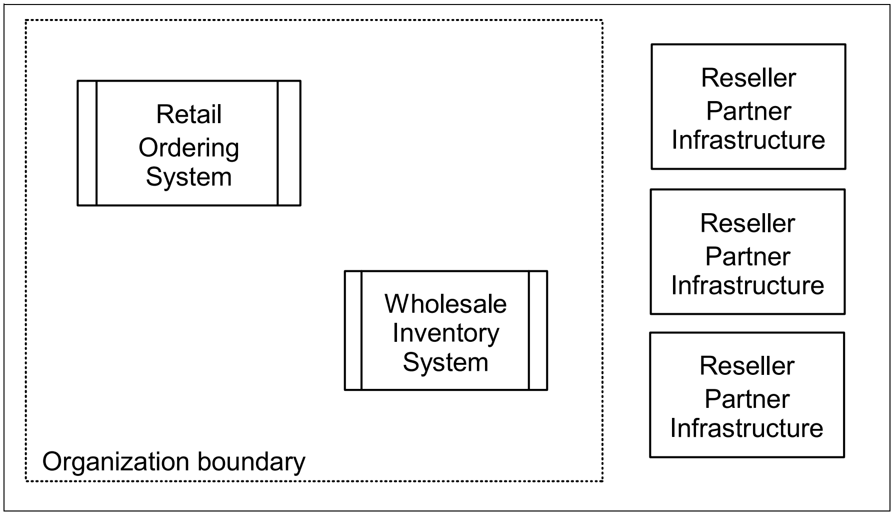


Figure 6-2 ITSO Electronics current IT infrastructure

6.1.4 Non-functional requirements

ITSO Electronics requires that all solutions provide a standard Quality of Service (QoS) set. The following specific criteria must be met:

- Autonomic
  - Solutions provide suitable system management tools, procedures, and logs.
- Availability
  - Solutions meet both the defined unplanned and planned downtime requirements.
  - Meaningful messages are provided to system users during downtime.
- Federation
  - The responsibilities of the stakeholders are clearly defined and agreed to by all parties.
- Performance
  - Solutions meet the defined throughput and response times.
  - Solutions scale to provide for future growth.
- Security
  - Sensitive systems and data is protected from unauthorized access.

- Non-repudiation of the end user for all commercial transactions is provided.

- Standards compliance
  - Appropriate standards are identified and applied.
  - Compatibility with existing internal systems and partners is considered.

It is beyond the scope of this redbook to define such requirements in real, measurable terms for our sample scenarios. Of course you would do so in a real-world implementation to ensure that the delivered solution meets the demands of the organization.

6.2 Intra-enterprise scenarios

In the first phase of implementation, ITSO Electronics wants to integrate their retail and wholesale departments. Currently, both organizations have proven IT infrastructures but have no interconnectivity. The first process ITSO Electronics wants to focus on is the inventory and order replenishment process. Currently, the items sold are tallied at the end of the month by the retail ordering process and delivered to the wholesale organization by internal mail. This creates a lag in the inventory replenishment process and causes many out of stock situations. A primary business goal is to minimize the loss of sales due to items being out of stock.

Selecting a Business/Integration pattern

In 3.1.1, “Business and IT drivers” on page 35, the following drivers are listed for selecting the Application Integration pattern:

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

Both drivers apply to ITSO Electronics. The business processes of the retail department and the wholesale department need to be integrated by integrating the existing retail business system with the existing wholesale business system. The pattern would integrate the retail order information with the wholesale inventory information, eliminating the lag and providing an up-to-date inventory.

The Application Integration pattern can be applied in our intra-enterprise scenarios, as shown in Figure 6-3.

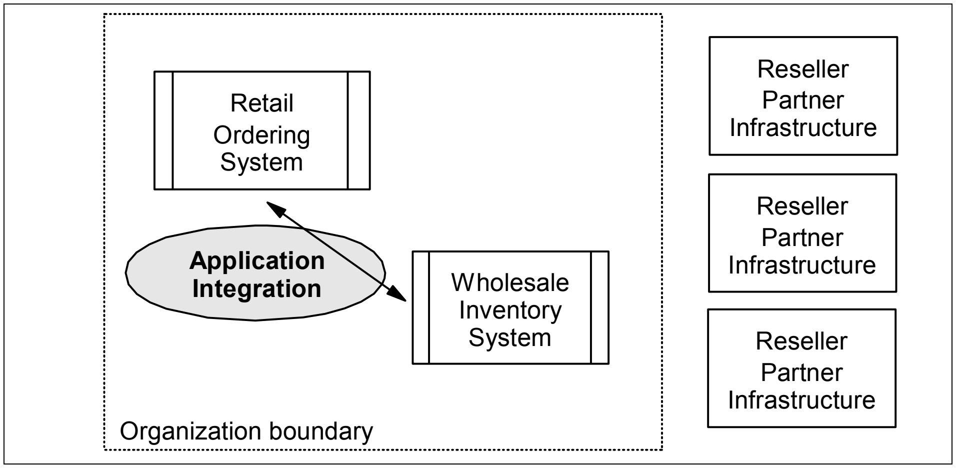


Figure 6-3 ITSO Electronics: Stage I and II architecture overview diagram

6.2.1 Stage I: Internal ordering on demand

In the first stage of the internal implementation, ITSO Electronics wants to integrate the retail system and the wholesale system. The primary goal is to integrate the internal retail ordering system with the internal wholesale system to *update inventory* as replenishment orders are placed. The wholesale group will then be able to monitor inventory levels and deliver replacement parts as needed. There is no business requirement for confirmation when the inventory is updated, but there is a requirement for an audit trail.

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

- The retail system
- The wholesale system

We can also identify a *use case*:

- Update inventory

Actors

Table 6-1 provides details on the retail system actor and Table 6-2 provides details on the wholesale system actor.

Table 6-1 Retail system actor details

Actor name	Retail system
Brief description	The retail system implements the retail ordering business process
Status	Primary
Relationships	001 Update Inventory, 002 Get Delivery Date
Associations to use cases	

Table 6-2 Wholesale system actor details

Actor name	Wholesale system
Brief description	The wholesale system implements the wholesale inventory management business process
Status	Primary
Relationships	
Associations to use cases	

Use case 001: Update inventory

Table 6-3 provides details on the update inventory use case.

Table 6-3 Use case 001: Update inventory

Use case name	001 Update Inventory.
Subject area	Wholesale ordering.
Business event	An item sold by the retail division needs to be replaced from the wholesale inventory.
Actors	Retail system, Inventory system.
Use case overview	The retail system places a replenishment order for a sold part with the inventory system.
Preconditions	The retail system supplies a part number for the item to be ordered.
Termination outcome 1	The wholesale inventory system logs the order to the audit trail and schedules delivery of the required part.
Notes	

The Stage I use case model is shown in Figure 6-4.

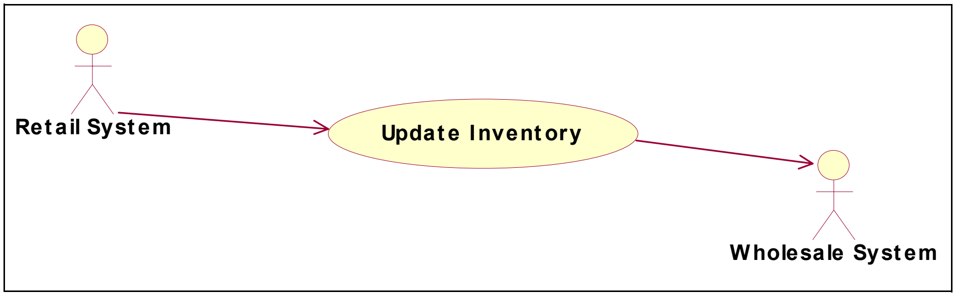


Figure 6-4 Stage I use case model

Selecting an Application pattern

In Table 3-1 on page 41 and Table 3-2 on page 42, the following drivers are listed for selecting the Message variation of the Process-focused Application Integration::Direct Connection application pattern:

- Improve the organizational efficiency
- Reduce the latency of business events
- Support a structured exchange within the organization
- Support real-time one-way message flows
- Leverage existing skills
- Leverage the legacy investment
- Enable back-end application integration
- Minimize application complexity

These drivers are a good match for the Stage I scenario. If we were undertaking large-scale integration between numerous applications, the Broker or Serial/Parallel Process application pattern would probably be a better choice. In this example, we are only concerned with point-to-point integration between two applications, so the Direct Connection application pattern is a reasonable choice.

The Message variation of the Process-focused Application Integration::Direct Connection application pattern can be applied in our Stage I scenario. See the following Application Integration scenarios for our sample implementations:

- Chapter 8, “Using RPC style Web services” on page 147
- Chapter 9, “Using document style Web services” on page 183
- Chapter 10, “Using the Web Services Gateway” on page 215
- Chapter 13, “Using Java Message Service” on page 279

6.2.2 Stage II: Internal ordering on demand with delivery date

In the second stage of the internal implementation, ITSO Electronics wants to further integrate their internal retail and wholesale systems. They now require a real-time notification for out of stock situations with a *delivery date* when the order can be filled. To improve customer service, the retail department wants to be able to quickly provide their customers with an expected delivery date for items that are not in stock with the retail department.

For Stage II of our business scenario we identify an additional *use case*:

- Get delivery date

Use case 002: Get delivery date

Table 6-3 provides details on the Get delivery date use case.

Table 6-4 Use case 002: Get delivery date

Use case name	002 Get delivery date.
Subject area	Wholesale ordering.
Business event	A delivery date for an item out of stock with the retail division needs to be obtained from the wholesale system.
Actors	Retail system, Inventory system.
Use case overview	The retail system requests a delivery date for an out of stock part from the inventory system.
Preconditions	The retail system supplies a part number for the out of stock item.
Termination outcome 1	The wholesale inventory system logs the request to the audit trail and returns the expected delivery date of the required part to the retail system.
Notes	

The Stage II use case model is shown in Figure 6-5.

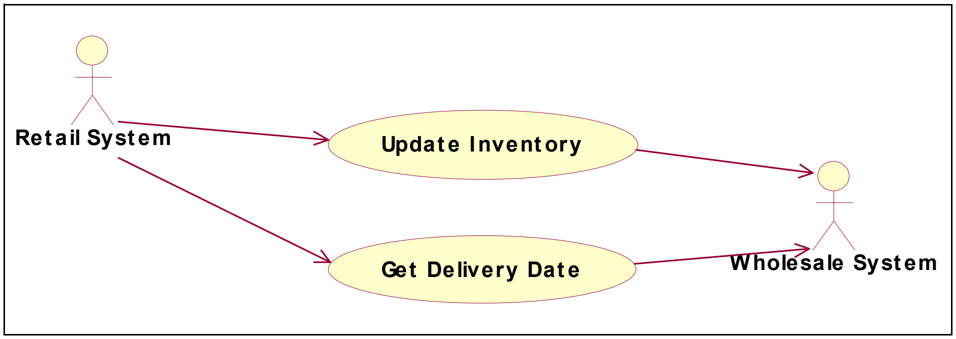


Figure 6-5 Stage II use case model

Selecting an Application pattern

In Table 3-1 on page 41 and Table 3-2 on page 42, the following drivers are listed for selecting the Call variation of the Process-focused Application Integration::Direct Connection application pattern:

- Improve the organizational efficiency
- Reduce the latency of business events
- Support a structured exchange within the organization
- Support real-time request/reply message flows
- Leverage existing skills
- Leverage the legacy investment
- Enable back-end application integration
- Minimize application complexity

All drivers apply to the Stage II scenario. In this scenario, the business process requires that a delivery date be returned in real time, so support for real-time calls between applications is a key driver.

The Call variation of the Process-focused Application Integration::Direct Connection application pattern can be applied in our Stage II scenario. See the following Application Integration scenarios for our sample implementations:

- Chapter 8, “Using RPC style Web services” on page 147
- Chapter 9, “Using document style Web services” on page 183
- Chapter 10, “Using the Web Services Gateway” on page 215
- Chapter 11, “Using the Web Services Gateway with J2EE Connectors” on page 237
- Chapter 12, “Using J2EE Connectors” on page 263

6.3 Inter-enterprise scenarios

In the second phase of the implementation, ITSO Electronics wishes to enable their external resellers to place orders to the wholesale group. Currently, the resellers fill out an order form and mail it to the wholesale organization. Difficulties arise when orders cannot be filled because of latency in the manual process and outdated inventory.

Selecting a Business/Integration pattern

In 4.1.1, "Business and IT drivers" on page 71, the following drivers are listed for selecting the Extended Enterprise business pattern:

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

Both drivers apply to ITSO Electronics. The business processes of the reseller partners and the wholesale department need to be integrated by integrating the existing partner systems with the existing wholesale business system. The pattern would integrate the wholesale inventory information with processes and information that exist at partner organizations.

The Extended Enterprise business pattern can be applied in our inter-enterprise scenarios, as shown in Figure 6-6.

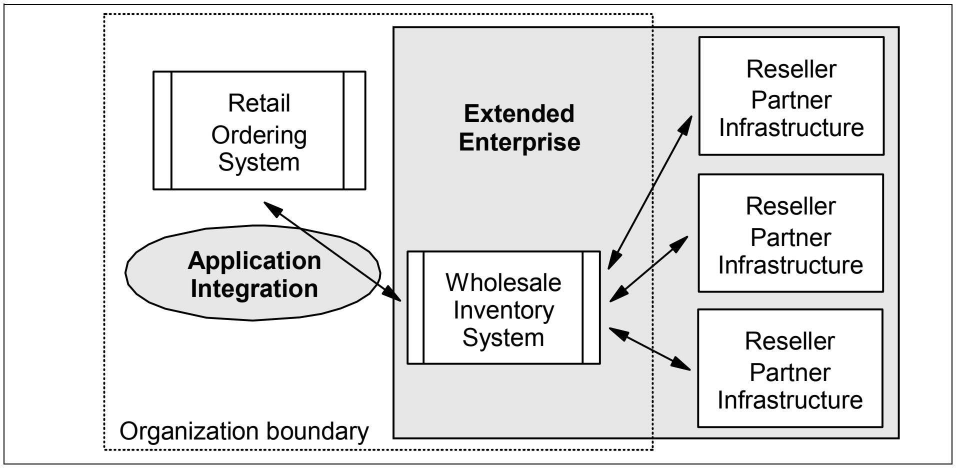

Retail Ordering System
Application Integration
Extended Enterprise
Wholesale Inventory System
Reseller Partner Infrastructure
Reseller Partner Infrastructure
Reseller Partner Infrastructure
Organization boundary

Figure 6-6 ITSO Electronics: Stage III and IV architecture overview diagram

6.3.1 Stage III: External ordering on demand

With the success of the internal retail and wholesale systems, ITSO Electronics has now decided to integrate their external reseller business partners. The goal of the third stage of implementation is to integrate the external *partner systems* with the internal wholesale system. As with the internal retail system, orders placed by the external partner systems will need to update the wholesale inventory system.

For Stage III of our business scenario, we can identify an additional actor:

- The partner system or systems

We also need to provide the Partner System actor with access to the Update Inventory use case.

Actors

Table 6-5 provides details on the partner system actor.

Table 6-5 Retail system actor details

Actor name	Partner system
Brief description	The partner system implements the partner ordering business process
Status	Primary
Relationships	001 Update Inventory, 002 Get Delivery Date
Associations to use cases	
Notes	

The Stage III use case model is shown in Figure 6-7.

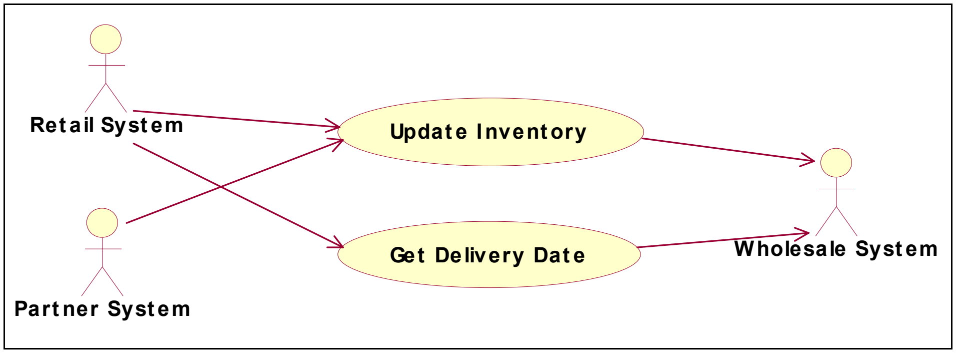


Figure 6-7 Stage III use case model

Selecting an Application pattern

In Table 4-3 on page 75 and Table 4-4 on page 76, the following drivers are listed for selecting the Message variation of the Extended Enterprise::Exposed Direct Connection application pattern:

- Improve the organizational efficiency
- Reduce the latency of business events
- Support a structured exchange with business partners
- Support real-time one-way message flows to partner processes
- Leverage existing skills
- Leverage the legacy investment
- Enable back-end application integration
- Minimize application complexity

These drivers are a good match for the Stage III scenario. If we were undertaking large-scale integration between numerous business partners, the Exposed Broker or Exposed Serial Process application pattern might be a better choice. In this example, we are only concerned with point-to-point integration between ITSO Electronics and its partners, so the Exposed Direct Connection application pattern is a reasonable choice.

The Message variation of the Extended Enterprise::Exposed Direct Connection application pattern can be applied in our Stage III scenario. See the following Extended Enterprise scenarios for our sample implementations:

- Chapter 14, “Using inter-enterprise Web services” on page 299
- Chapter 15, “Using WebSphere Data Interchange” on page 315

6.3.2 Stage IV: External on demand ordering with delivery date

The final stage of the scenario addresses out of stock situations with external resellers. As with the internal retail system, the external resellers require an immediate notice on any out of stock situations and a delivery date when the order can be filled.

For the Stage IV scenario, we need to provide the Partner System actor with access to the Get Delivery Date use case.

The Stage IV use case model is shown in Figure 6-8.

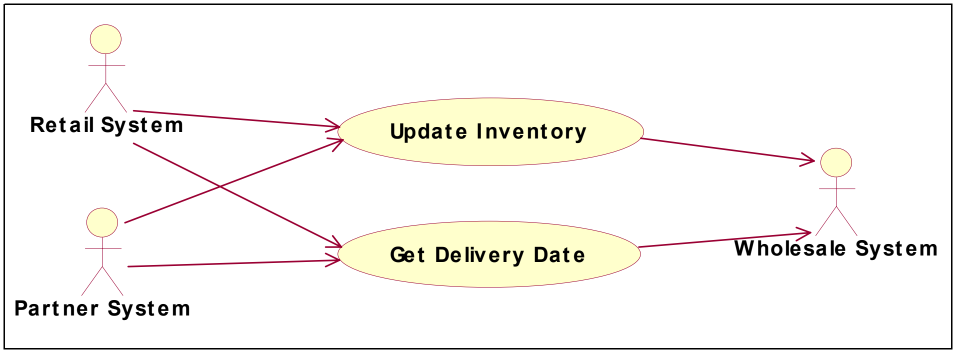


Figure 6-8 Stage IV use case model

Selecting an Application pattern

In Table 4-3 on page 75 and Table 4-4 on page 76, the following drivers are listed for selecting the Call variation of the Extended Enterprise::Exposed Direct Connection application pattern:

- Improve the organizational efficiency
- Reduce the latency of business events
- Support a structured exchange with business partners
- Support real-time request/reply message flows to partner processes
- Leverage existing skills
- Leverage the legacy investment
- Enable back-end application integration
- Minimize application complexity

All drivers apply to the Stage IV scenario. In this scenario, the business process requires that a delivery date be returned to the partner application in real time, so support for real-time calls between applications is a key driver.

The Call variation of the Extended Enterprise::Exposed Direct Connection application pattern can be applied in our Stage IV scenario. See the following Extended Enterprise scenarios for our sample implementations:

- Chapter 14, “Using inter-enterprise Web services” on page 299

7

Technology options

We take a look at the application integration technology options you should consider in this chapter. The recommendations are guided by the demands of reuse, flexibility, and interoperability, and subsequently are based on the open industry standards outlined by Java 2 Platform, Enterprise Edition (J2EE). Many of the choices continue to evolve and expand as the J2EE specification matures to include a broader view of the enterprise architecture. These recommendations are based on the J2EE1.3 specification and parts of the J2EE1.4 specification.

Our discussion of technology options is focused on the connection between applications, as shown in Figure 7-1 on page 128. The topics covered are:

- How to select the appropriate integration technology
- XML as a technology for data exchange
- Web services
- J2EE Connector Architecture
- Asynchronous messaging with JMS
- Other integration technologies

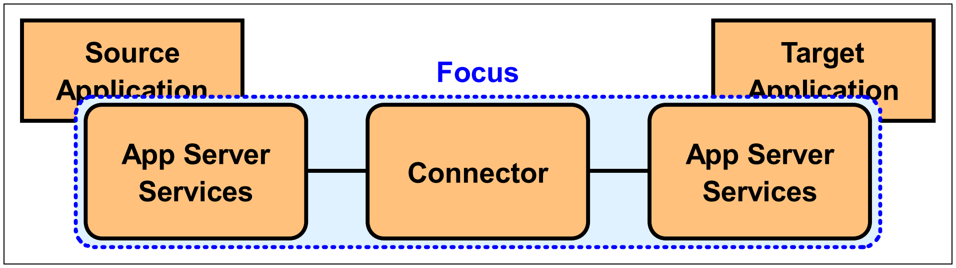


Figure 7-1 Technology options focus

7.1 Selecting an integration technology

With the continuous progress of enterprise computing, more and more enterprises are finding the need to quickly adopt new technologies and integrate with existing applications. Furthermore, due to costs and human resource limitations, it is often not feasible for enterprises to completely discard their existing infrastructure.

Enterprise application integration (EAI) and Extended Enterprise (EE) allow disparate applications to communicate with each other. Both domains are very similar; their differences are limited mainly to the required Quality of Services capabilities. Some points to consider while deciding on the integration technology between applications are the following:

- The current infrastructure

  Do you already have a messaging system on the enterprise tier? Then it makes sense to go for JMS. Or, if you have a legacy system such as CICS or IMS, J2EE Connectors might be the better choice.

- Time to market

  Web service enabling an application is relatively fast with the Web services development tools available.

- Future expansion plans

  If you plan to expand your enterprise systems in the future, you need to keep in mind the integration with your current infrastructure and your planned infrastructure. Web services may provide the most cost-effective migration path in such a case.

- Reliability

  JMS with WebSphere MQ, for example, can be used to provide assured transfer of data, even when the enterprise application is unavailable.

- Transaction support

  Web services currently do not offer support for transactions. If your application needs transactional management, it might be worthwhile considering technologies that do, such as JMS or J2EE Connectors.

7.2 XML

Extensible Markup Language (XML) allows you to specify your own markup language with tags defined in a Document Type Definition (DTD) or XML Schema. XML can be used as a means to specify the content of messages between servers, whether the two servers are within an enterprise or represent a business-to-business connection. The critical factor here is the agreement between parties on the message schema, which is specified as an XML DTD or Schema. An XML parser is used to extract specific content from the message stream. Your design will need to consider whether to use an event-based approach, for which the SAX API is appropriate, or to navigate the tree structure of the document using the DOM API.

The IBM XML4J XML parser was made available through the Apache open source organization under the Xerces name. For open source XML frameworks, see:

`http://xml.apache.org/`

7.2.1 Defining XML documents

XML documents are defined using DTDs or XML Schemas.

DTDs are a basic XML definition language, inherited from the SGML specification. The DTD specifies what markup tags can be used in the document and what their structures are.

DTDs have two major problems:

- Poor data typing: In DTDs, elements can only be specified as EMPTY, ANY, element content, or mixed element-and-text content, and there is no standard way to specify null values for elements.

  Data typing like date formats, numbers, or other common data types cannot be specified in the DTD, so an XML document may comply with the DTD but still have data type errors that can only be detected by the application.
- Not defined in XML: DTD uses its own language to define XML syntax that is not compliant with the XML specification. This makes it difficult to manipulate a DTD.

To solve these problems, the World Wide Web Consortium (W3C) specified a new standard to define XML documents called XML Schema. XML Schema provides the following advantages over DTDs:

- Strong typing for elements and attributes
- Standardized way to represent null values for elements
- Key mechanism that is directly analogous to relational database foreign keys
- Defined as XML documents, making them programmatically accessible

Even though XML Schema is a more powerful technology to define XML documents, it is also a lot harder to work with, so DTDs are still widely used to define XML documents. Additionally, simple, non-hard-typified documents can be easily defined using DTDs with similar results to using XML Schema.

Whether to use one or the other will depend on the complexity of the messages and the validation requirements of the application. Actually, in many cases both a DTD and an XML Schema are provided, so they can be used by the application depending on its requirements.

> **Note:** Remember that the validation process of an XML document using XML Schemas is an *expensive process*. Validation should be performed only when it is necessary.

7.2.2 XSLT

Extensible Stylesheet Language Transformations (XSLT) is a W3C specification for transforming XML documents into other XML documents. The XSLT is built on top of the Extensible Stylesheet Language (XSL), a stylesheet language for XML (such as CSS2 for HTML). Unlike CSS2, XSL is also a transformation language.

A transformation expressed in the XSLT language defines a set of rules for transforming a source tree to a result tree, and it is expressed in the form of a stylesheet.

An XSLT processor is used for transforming a source document to a result document. There are currently a number of XSLT processors available on the market. DataPower has introduced an XSL just-in-time (JIT) compiler, which speeds up the time taken for the XSL transformation.

The XSLT processor has a performance overhead, so online processing of larger documents can be slow.

7.2.3 XML security

XML security is an important issue, particularly where XML is being used by organizations to interchange data across the Internet. Several new XML security specifications are working their way through three standards bodies—the World Wide Web Consortium (W3C), Internet Engineering Task Force (IETF), and Organization for the Advancement of Structured Information Standards (OASIS). We highlight a few of them here:

- XML Signature Syntax and Processing is a specification for digitally signing electronic documents using XML syntax. According to the W3C, “XML Signatures provide integrity, message authentication, and/or signer authentication services for data of any type, whether located within the XML that includes the signature or elsewhere.”

  A key feature of the protocol is the ability to sign parts of an XML document rather than the document in its entirety. This is necessary because an XML document might contain elements that will change as the document is passed along, or various elements that will be signed by different parties.

  WebSphere Studio provides you with the ability to create (using a wizard) and verify XML digital signatures.

- XML encryption will allow encryption of digital content, such as Graphical Interchange Format (GIF) images or XML fragments. XML Encryption allows parts of an XML document to be encrypted while leaving other parts open, encryption of the XML itself, or the super-encryption of data (that is, encrypting an XML document when some elements have already been encrypted).
- XML Key Management Specification (XKMS) establishes a standard for XML-based applications to use Public Key Infrastructure (PKI) when handling digitally signed or encrypted XML documents. XML signature addresses message and user integrity, but not issues of trust that key cryptography ensures.
- Security Assertion Markup Language (SAML) is the first industry standard for secure e-commerce transactions using XML. It aims to standardize the exchange of user identities and authorizations by defining how this information is to be presented in XML documents, regardless of the underlying security systems in place.

For further discussion, see the Sun ONE article *Riddle Me This: Is Your XML Data Safe?* by Brett Mendel:

```
http://sunonedev.sun.com/building/tech_articles/xmldata.html
```

7.2.4 Advantages of XML

XML has many advantages over other technologies. Some of the factors that have influenced the wide acceptance of XML are:

- Acceptability of use for data transfer

  XML is a standard way of putting information in a format that can be processed and exchanged across different hardware devices, operating systems, software applications, and the Web.

- Uniformity and conformity

  XML gives you a common format that can be developed upon and is accepted industry-wide.

- Simplicity and openness

  Information coded in XML is human readable.

- Separation of data and display

  The representation of the data is separated from the presentation and formatting of the data for display in a browser or other device.

- Industry acceptance

  XML has been accepted widely by the information technology and computing industry. Numerous tools and utilities are available, along with new products for parsing and transforming XML data to other data, or for display.

7.2.5 Disadvantages of XML

Some XML issues to consider are:

- Complexity

  While XML tags can allow software to recognize meaningful content within documents, this is only useful to the extent that the software reading the document knows what the tagged content means in human terms, and knows what to do with it.

- Standardization

  When multiple applications use XML to communicate with each other they need to agree on the tag names they are using. While industry-specific standard tag definitions often do exist, you can still declare your own non-standard tags.

- Large size

  XML documents tend to be larger in size than other forms of data representation.

7.3 Web services

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

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

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

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

Some of the key features of Web services are the following:

- Web services are self-contained.

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

- Web services are self-describing.

  Neither the client nor the server knows or cares about anything besides the format and content of request and response messages (loosely coupled application integration).

  The definition of the message format travels with the message. No external metadata repositories or code generation tools are required.

- Web services are modular.

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

- Web services can be published, located, and invoked across the Web.

  The standards required to do so are:

  - Simple Object Access Protocol (SOAP), also known as service-oriented architecture protocol, an XML-based RPC and messaging protocol
  - Web Service Description Language (WSDL), a descriptive interface and protocol binding language
  - Universal Description, Discovery, and Integration (UDDI), a registry mechanism that can be used to look up Web service descriptions

- Web services are language independent and interoperable.

  The interaction between a service provider and a service requester is designed to be completely platform and language independent. This interaction requires a WSDL document to define the interface and describe the service, along with a network protocol (usually HTTP). Because the service provider and the service requester have no idea what platforms or languages the other is using, interoperability is a given.

- Web services are inherently open and standards-based.

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

- Web services are dynamic.

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

- Web services are composable.

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

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

For detailed information on Web services, check out the following:

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

- The World Wide Web Consortium (W3C) Web site at:

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

7.3.1 Static and dynamic Web services

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

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

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

7.3.2 Web Services Invocation Framework

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

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

WSIF is WSDL-driven and provides a uniform interface to invoke services using WSDL documents. So if a SOAP service you are using becomes available as an EJB, for example, you can change to RMI/IIOP by just modifying the WSDL service description, without needing to modify your applications that use the service.

This API is used by tools such as WebSphere Studio Integration Edition and runtimes such as WebSphere Application Server to construct and manipulate services defined in WSDL documents. The architecture allows new bindings to be added at runtime.

WSIF has the following advantages:

- Multiple bindings can be offered for services, and bindings can be decided at runtime.

- Services can be used either by a set of stub classes (static) or by a dynamic interface invocation (dynamic).
- You can switch protocols, location, and so forth, without having to recompile your client code.

For more details on the Web Services Invocation Framework see:

`http://ws.apache.org/wsif/`

7.3.3 Web services and the service-oriented architecture

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

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

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

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

`http://www.ibm.com/software/solutions/webservices/resources.html`

This Web site provides a collection of IBM resources on this topic. For example, there is an introduction to the SOA in a white paper titled *Web Services Conceptual Architecture (WSCA 1.0)*.

7.3.4 Web services security

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

The Web services security specification provides a broad set of specifications that cover security technologies, including authentication, authorization, privacy, trust, integrity, confidentiality, secure communications channels, federation, delegation, and auditing across a wide spectrum of application and business topologies. These specifications provide a framework that is extensible and flexible, and that maximizes existing investments in security infrastructure. By leveraging the natural extensibility that is at the core of the Web services model, the specifications build upon foundational technologies such as SOAP, WSDL, XML Digital Signatures, XML Encryption, and SSL/TLS.

As shown in Figure 7-2, this set includes a message security model (WS-Security) that provides the basis for the other security specifications. Layered on this, we have a policy layer that includes a Web service endpoint policy (WS-Policy), a trust model (WS-Trust), and a privacy model (WS-Privacy). Together these initial specifications provide the foundation upon which we can work to establish secure interoperable Web services across trust domains.

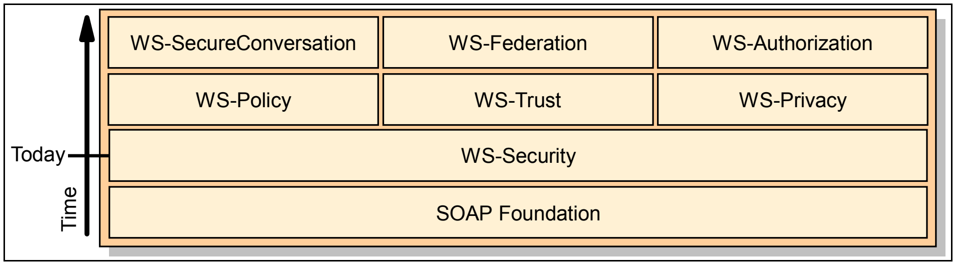


Figure 7-2 The evolving WS-Security roadmap

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

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

WS-Security

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

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

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

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

7.3.5 Advantages of Web services

Web services technology enables businesses to:

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

7.3.6 Disadvantages of Web services

Some Web services issues to consider are:

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

7.3.7 Comparing Web services with CORBA and RMI

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

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

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

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

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

7.4 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 API (CCI). 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 WebSphere Application Server V5.0 supports the J2EE Connector Architecture 1.0, as required by the J2EE 1.3 specification. The administrative console supports J2EE Connector resource adapter configuration. The administrative console allows the association of connection factories for the resource adapter that encapsulate the pooling attributes. Component providers request a connection for an enterprise information system (EIS) from the connection factory through the JNDI lookup mechanism. IBM supplies resource adapters for enterprise systems such as CICS, HOD, IMS, SAP, and Crossworlds as separate products.

IBM WebSphere Studio Application Developer V5.0 supports application development using J2EE Connectors, and development of custom J2EE Connectors.

7.4.1 CICS resource adapter

The CICS Transaction Gateway (CICS TG) V5 is a set of client and server software components that allow a Java application to invoke services in a CICS region.

The CICS TG offers three basic interfaces for Java clients:

- External Call Interface (ECI) for COMMAREA-based CICS applications
- External Presentation Interface (EPI) for 3270-based transactions
- External Security Interface (ESI) for password management to verify and change user IDs and passwords

The CICS resource adapter is covered in detail in the following chapters.

7.4.2 IMS resource adapter

The IMS Connector for Java provides a way to create Java applications that can access IMS transactions. The IMS Connector for Java uses IMS Connect to access IMS. IMS Connect is a facility that runs on the host IMS machine and supports TCP/IP and Local Option communication to IMS. A Java application or servlet accesses IMS Open Transaction Manager Access (OTMA) through IMS Connect. IMS Connect accepts messages from its TCP/IP clients and routes them to IMS OTMA using the Cross-System Coupling Facility (XCF).

The runtime component of IMS Connector for Java is provided as a component of IMS Connect, Version 1 Release 2 (Program Number 5655-E51). The J2EE Connector implementation of this runtime component is also referred to as the IBM WebSphere Adapter for IMS. It is packaged as a RAR file, imsico.rar, for deployment into a WebSphere Application Server. The RAR file is installed to a target directory from the IBM IMS Connect, Version 1 Release 2.

7.4.3 Advantages of J2EE Connectors

Some reasons to use J2EE Connectors are:

- The common client interface simplifies application integration with diverse EISs. This common interface makes it easy to plug third-party or home-grown resource adapters into your applications.

- Each EIS requires just one implementation of the resource adapter since there is no need to custom develop an adapter for every application.
- J2EE Connectors facilitate scalability and provide Quality of Service features transparently to the client application.
- J2EE Connector Architecture-compliant resource adapters are portable across J2EE application servers. If a vendor provides a resource adapter for WebLogic, for example, it should also work with the WebSphere Application Server.
- J2EE Connectors have low intrusion on the enterprise system because native client interfaces are utilized.

7.4.4 Disadvantages of J2EE Connectors

Some J2EE Connector issues to consider are:

- J2EE Connector Architecture has support only for synchronous communication. (The CICS adapter does offer support for non-blocking calls.) Support for asynchronous communications is expected in the J2EE Connector Architecture 1.5 specification.
- The J2EE Connectors standard is still relatively new, and performance compared with previous alternatives has not been firmly established. For example, some customers may prefer to continue with the well-proven non-J2EE Connector CICS TG base classes.
- Though J2EE Connector Architecture promises an abstraction to access any legacy system, with J2EE Connector Architecture 1.0, parts of the client application need to have resource adapter-specific implementation. This means that if you have to change the resource adapter (move to a different enterprise system, which provides a different adapter), the client application will be affected.

For more information on J2EE Connectors and CICS, refer to the following redbooks:

- *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401
- *Revealed! Architecting Web Access to CICS*, SG24-5466

7.5 Java Message Service

Messaging middleware is a popular choice for accessing existing enterprise systems in an asynchronous manner. It is one of the options if you are implementing a solution based on the Message variation of the Direct Connection pattern in an intra-enterprise scenario.

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

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

According to the JMS 1.0 specification, a message provider is integrated in an application server. As shown in Figure 7-3, the integrated message provider makes it possible to communicate asynchronously with other WebSphere applications, without installing separate messaging software like IBM WebSphere MQ. WebSphere's integrated JMS server is based on IBM WebSphere MQ.

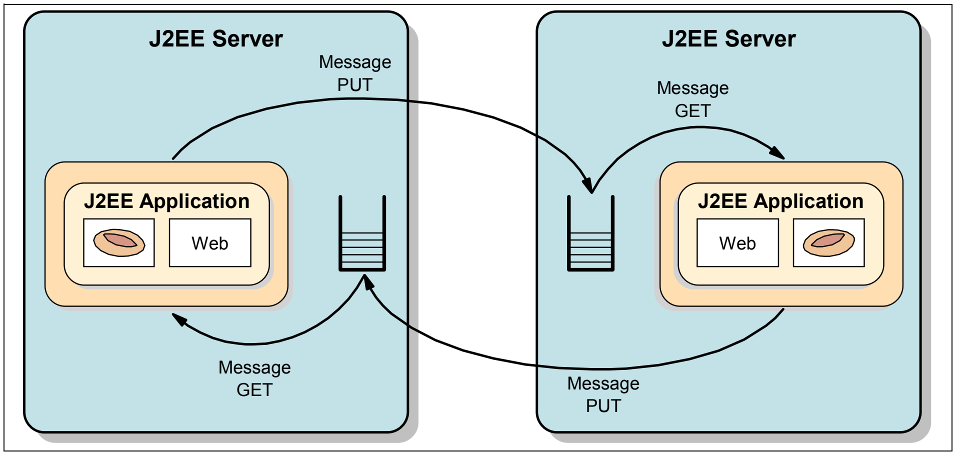


Figure 7-3 Integrated JMS Provider

An important new feature of EJB 2.0 is message-driven beans (MDB). Message-driven beans are designed specifically to handle incoming JMS messages. Further information on message-driven beans can be found in the IBM Redbook *EJB 2.0 Development with WebSphere Studio Application Developer*, SG24-6819.

7.5.1 What messaging is

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

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

- One-to-one, or point-to-point model
- Publish/subscribe model

7.5.2 JMS and IBM WebSphere MQ

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

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

7.5.3 Advantages of JMS

The JMS standard is important because:

- It is the first enterprise messaging API that has achieved wide cross-industry support.
- It simplifies the development of enterprise applications by providing standard messaging concepts and conventions that apply across a wide range of enterprise messaging systems.

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

7.5.4 Disadvantages of JMS

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

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

Further information on JMS can be found in the IBM Redbook *MQSeries Programming Patterns*, SG24-6506.

7.6 Other integration technologies

In this section we briefly touch on a few other integration technologies, including:

- RMI/IIOP
- CORBA

7.6.1 RMI/IIOP

Remote Method Invocation (RMI) APIs allow developers to build distributed applications in the Java programming language. They enable an object running in one Java Virtual Machine to access another object running in a different Java Virtual Machine.

The Internet Inter-ORB (Object Request Broker) Protocol (IIOP) is a protocol used for communication between CORBA object request brokers. An object request broker is a library that enables CORBA objects to locate and to communicate with one another.

RMI/IIOP is an implementation of the RMI API over IIOP that allows developers to write remote interfaces in the Java programming language.

7.6.2 CORBA

Common Object Request Broker Architecture (CORBA) is a platform-, language-, and vendor-neutral standard for writing distributed object systems. The CORBA standard was developed by the Object Management Group (OMG), a consortium of companies founded in 1989. CORBA offers a broad range of middleware services, including naming service, relationship service, and so on.

CORBA can be used for integration with legacy applications. This is done by creating a CORBA wrapper for the existing application, which can then be invoked by other applications.

CORBA is just a specification, and there are a number of vendors (such as IONA or Borland) that implement it. Each vendor will provide additional value-added services such as persistence, security, and so on, which can be leveraged by CORBA developers.

The disadvantage of CORBA is in the steep learning curve involved. Also, CORBA is slow-moving; it takes a long time for the OMG to adopt a new feature.

7.7 Where to find more information

For more information on topics discussed in this chapter, see:

- *WebSphere Version 5 Web Services Handbook*, SG24-6891
- *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401
- *Revealed! Architecting Web Access to CICS*, SG24-5466
- *MQSeries Programming Patterns*, SG24-6506
- IBM CICS

  `http://www.ibm.com/software/ts/cics`
- IBM WebSphere MQ

  `http://www.ibm.com/software/ts/mqseries`
- Java APIs and technology

  `http://java.sun.com/products`
- World Wide Web Consortium (W3C) site

  `http://www.w3.org/`
- Open source XML frameworks

  `http://xml.apache.org/`

- Sun ONE article, *Riddle Me This: Is Your XML Data Safe?* by Brett Mendel:

  http://sunonedev.sun.com/building/tech_articles/xmldata.html
- Service-oriented architecture and Web services:

  http://www.ibm.com/software/solutions/webservices/resources.html

8

Using RPC style Web services

This chapter discusses using RPC style Web services in an intra-enterprise integration scenario. We are using Web services for J2EE as provided with IBM WebSphere Application Server base V5.0.2 in this scenario.

This chapter describes the following:

- An overview of the ITSO Electronics sample application, which we use to demonstrate the different scenarios.
- Design issues to be considered when using RPC style Web services and the design applied to our RPC based Web services.
- Creating a Web service provider and requester using Web Services for J2EE.
- Quality of Service capabilities for Web services.
- Best practices for Web services.

8.1 Business scenario

As described in 6.2.1, “Stage I: Internal ordering on demand” on page 116, ITSO Electronics plans to integrate their retail and wholesale departments. Both departments have existing IT infrastructures, but currently there is no connectivity between their applications.

The retail order information is to be integrated with the wholesale inventory system, eliminating the lag that occurs with the current manual process and providing up-to-date inventory information.

The Application Integration pattern applies to this situation. This pattern can be applied for intra-enterprise integration activities. This use of the Application Integration pattern will provide a number of benefits to ITSO Electronics. Not only does it provide a means to integrate existing business processes, information and systems, but it also improves organizational efficiency and reduces the latency of business events. Integrating existing systems will leverage existing skills in the organization and maximize the investment already made in the legacy systems.

When the retail department needs to notify the wholesale department that a part must be ordered, the Message variation of Direct Connection application pattern can be applied. This Application pattern is suitable because the retail department will be notifying the wholesale department that a part must be ordered, but the retail department does not require any response as part of this process.

When the retail department needs to know the expected delivery date for a part on order, the Call variation of the Direct Connection application pattern can be applied. In this instance, the retail department requires a response from the wholesale department, advising them of the expected delivery date for the part in question.

The RPC style Web service can be applied to both processes. It provides a relatively simple development effort in integrating the existing systems in the retail and wholesale departments. It has the advantages of providing heterogeneous platform support and loose-coupling between the two systems.

8.2 System design overview

The Application Integration pattern focuses on the mechanisms selected to achieve communication between two applications within the domain of a single organization.

To illustrate the various runtime patterns applicable when using the Direct Connection application pattern, we use a very simple Web application as the source application. It comprises a welcome page, a servlet, a factory class, command beans, a view bean and a JSP to display the response of the interaction.

We use an enterprise bean as the target application, which is exposed as a Web service. Selecting an EJB can simplify a business application because the EJB container provides services that handle complexities such as security, resource pooling, persistence, concurrency, and transactional integrity. The business logic provided by our EJB is also available to any Java source application using RMI.

Figure 8-1 highlights the ability to establish a Direct Connection between the source and target applications as the focus of our discussion. It also shows the major components constructed for the source and target applications used in our sample solution.

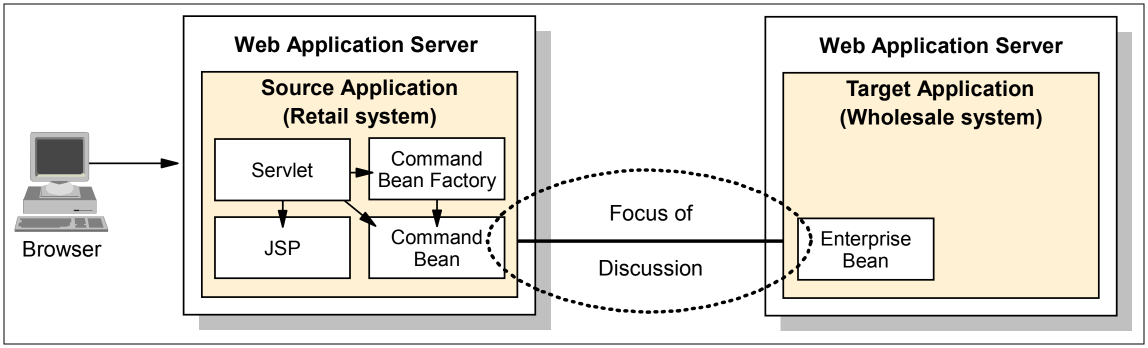
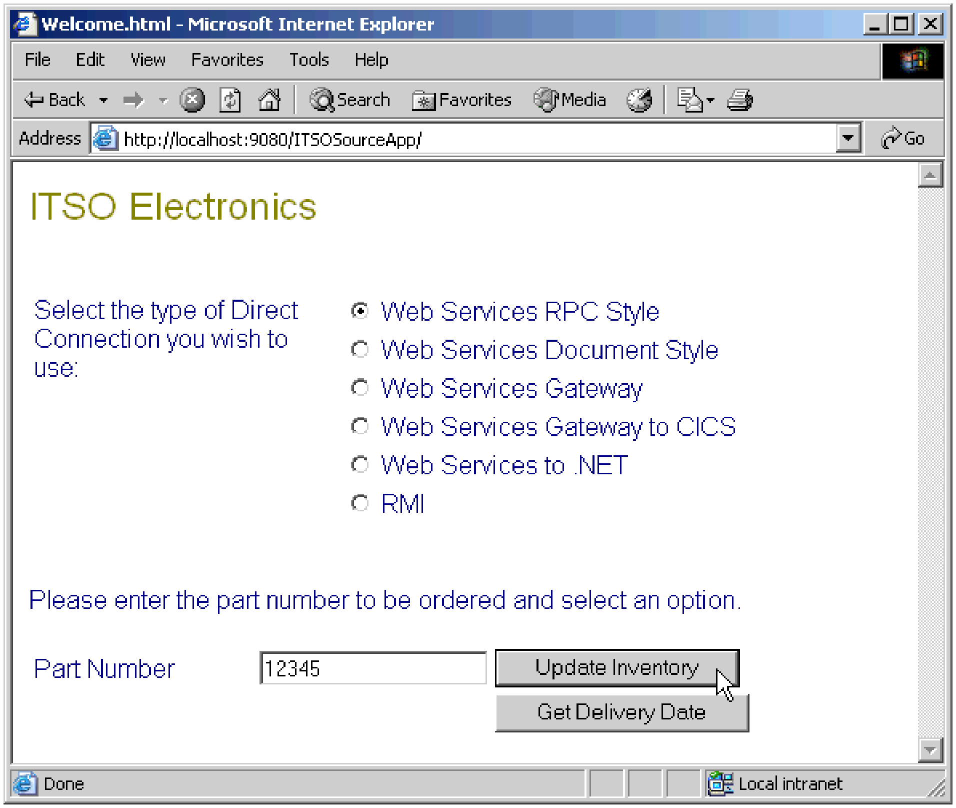


Figure 8-1 High level design for source and target application

A Welcome page, shown in Figure 8-2, enables the user to exercise a Direct Connection using a variety of different technologies by selecting the required option. The Update Inventory button invokes a Message-style interaction, where a reply is not needed. The Get Delivery Date button uses a Call-style interaction, where a reply providing an expected delivery date is required.


Welcome.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/ITSOSourceApp/
Go
Update Inventory
Get Delivery Date
Done
Local intranet

ITSO Electronics

Select the type of Direct Connection you wish to use:

- ● Web Services RPC Style
- ○ Web Services Document Style
- ○ Web Services Gateway
- ○ Web Services Gateway to CICS
- ○ Web Services to .NET
- ○ RMI

Please enter the part number to be ordered and select an option.

Part Number 12345

Figure 8-2 Welcome page

Our source application is a simple implementation of the Model-View-Controller (MVC) design pattern. The InventoryServlet acts as the *Controller* by parsing the request from the browser.

The CommandBean acts as the *Model* in our source application. This class is responsible for implementing the business logic of our application. A CommandBean is created using a singleton CommandBeanFactory class which instantiates the relevant CommandBean based on the option selected. The implementation of each specific CommandBean makes use of generated classes specific to the option selected. Each CommandBean implements an updateInventory() and a getDeliveryDate() method. The InventoryServlet calls the execute() method on the generic CommandBean, passing a parameter indicating which push button has been selected. The execute() method on the base class then invokes the required method. These methods interact with the target Web service application.

The *View* represents the result of the interaction with the target application. This is provided by an InventoryViewBean, which is created by the InventoryServlet and used in a JSP which is displayed to the user in the browser.

8.3 Web services for J2EE

Web services provide communication between programs based on industry standards, and are platform and language independent. Each Web service has an interface description that encapsulates its details in a form suitable for use by other applications. This interface description may be published in a repository, so that applications can find and utilize the Web service. The description is constructed using Web Services Description Language (WSDL).

Web services are typically based on SOAP (Simple Object Access Protocol), which is an XML-based message protocol that is transport protocol independent. Our sample solution uses the Web Services for J2EE support provided in IBM WebSphere Application Server V5.0.2; Web Services for J2EE support is provided using a standard defined in Java Specification Request (JSR) 109.

In our sample solution, the target EJB application is wrapped as a JSR 109-based Web service. The source application accesses the target Web service using proxy classes generated from the WSDL file for the target Web service.

A new set of Java Specification Requests

The technologies used by J2EE application servers to provide Web services facilities are evolving very quickly. The Java community has recently adopted a set of standards to define the different aspects of how Web services can be supported in a J2EE-compliant application server. These standards are described in Java Specification Requests (JSRs).

The main JSR for Web services is JSR-109, *Implementing Enterprise Web Services* (also known as Web services for J2EE). It reached the final release status in November 2002.

The aim of JSR-109 is to define the programming model and runtime architecture for implementing Web services in Java. It federates the work done on several other JSRs. This JSR was led by IBM.

The Web services for J2EE Version 1.0 specification is an addition to J2EE 1.3. J2EE 1.4 requires support for Web services for J2EE Version 1.1. There are minor differences between the J2EE 1.3 Version (JSR-109 Version 1.0) and the J2EE 1.4 Version (JSR-109 Version 1.1).

Specifications have also been opened for defining APIs for specific parts of the Web services stack:

- JSR 67: *Java APIs for XML Messaging (JAXM)*

  JAXM provides an API for packaging and transporting business transactions using on-the-wire protocols being defined by ebXML.org, OASIS, W3C, and IETF.

- JSR 93: *Java APIs for XML Registry*

  JAXR provides an API for a set of distributed registry services that enables business-to-business integration between business enterprises, using the protocols being defined by ebXML.org, OASIS, and ISO 11179.

- JSR 101: *Java APIs for XML-Based RPC*

  JAX-RPC defines APIs to support emerging industry XML-based RPC standards.

- JSR 110: *Java APIs for WSDL*

  This JSR provides a standard set of APIs for representing and manipulating services described by WSDL (Web Services Description Language) documents. These APIs define a way to construct and manipulate models of service descriptions.

See the Java Community Process Web site for the JSR details:

`http://www.jcp.org/`

8.4 Design guidelines

Figure 8-3 shows the Runtime pattern and Product mapping we used to demonstrate the Direct Connection application pattern within the business domain of an organization, using Web services technology. It illustrates the logical connection between the source and target applications provided using SOAP.

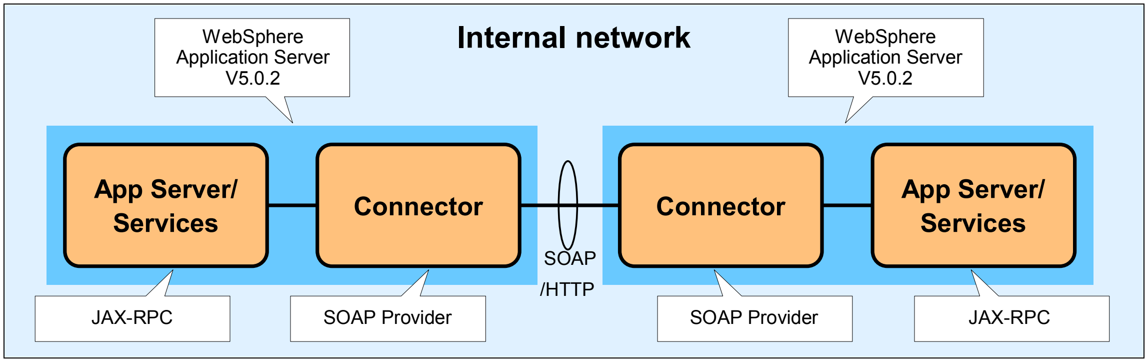


Figure 8-3 Web services product mapping for Application Integration::Direct Connection

This view of the solution can be further expanded to obtain a “stack” of coupling adapter pairs over “virtual” connections between the source and target, as shown in Figure 8-4. In this diagram the dotted line represents a virtual connection which is actually implemented at lower levels.

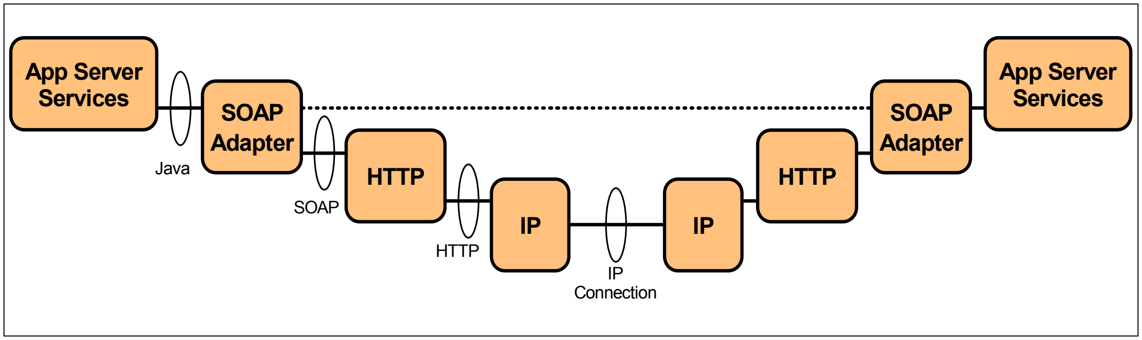


Figure 8-4 Web services coupling adapter connector stack

8.4.1 Design considerations

A number of factors affect the design of a Web service. In this section we discuss some of the factors we considered when building our sample application.

SOAP messaging style

SOAP provides support for both RPC style Web services and document-style Web services. In RPC style, Web Services are invoked using remote method calls. In document-style messaging, Web Services are invoked by sending a complete well-formed document describing the task to be performed, and possibly, some parametric data.

The advantages of SOAP RPC-based Web services are:

- Simpler to develop than SOAP message-oriented messaging
- Uses a higher level API
- Supports strong typing: all calls must conform to the method signature

The disadvantages of RPC-based SOAP messaging are:

- Higher coupling between the Web service requester and provider because the requester to provider binding includes both methods and parameters.

We used the RPC-based mechanism in our sample solution to demonstrate the Direct Connection pattern because of the simpler development effort involved. The RPC-based mechanism can be used for both the Call and Message styles of communication.

Transmission style

Web services support a number of transmission operations. These are the different types of operations that can be defined in a WSDL file:

- One-way - client sends a message to service
- Request-response - client sends a message to service, and awaits response
- Solicit-response - service sends a message to the client, and awaits response
- Notification operation - service sends a message to the client

The Direct Connection Call variation is a Request-response operation, while the Message variation is one-way.

SOAP transport protocol

In principle, SOAP is transport protocol independent and can therefore potentially use a variety of protocols (such as HTTP, JMS, SMTP, and others) to connect the Web service requestor and the provider.

HTTP is currently the most popular transport protocol for SOAP. The reasons for this mainly result from the following advantages:

- HTTP is the de facto standard on the Internet
- HTTP is wide spread
- HTTP is supported from most programming languages
- HTTP has a simple extension for security, HTTPS
- HTTP needs no complex infrastructure

But there are also some important limitations:

- HTTP is optimized for use in browser and end-user scenarios
- HTTP is a stateless communication protocol
- HTTP does not provide reliable communication
- HTTP establishes a point-to-point connection

These limitations are usually acceptable for human-to-machine communication using a Web browser. This may not be valid when switching to machine-to-machine communication. The requirements in machine-to-machine communication are usually more complex and other transport protocols may be more suitable.

For example, if you need assured, once-only delivery, message-oriented middleware may be more suitable. In Java you can use Java Messaging Service (JMS) with IBM WebSphere MQ. The advantages are:

- Reliable messaging
- Option of asynchronous communication

On the other hand you have the following disadvantages:

- More complex infrastructure
- Not generally available for all potential customers
- Message-oriented middleware skills are necessary
- At this time most vendors only have limited support for SOAP over JMS

With the increasing adoption of Web services, the portfolio of supported transport protocols will also increase.

We use the HTTP transport protocol in our sample solution to demonstrate the Direct Connection pattern because HTTP can provide the point-to-point connection we need between source application and target application without the need for added infrastructure.

Static versus dynamic Web services discovery

The source (requester) application can use either static or dynamic discovery of available Web services. The requester has to begin with the WSDL file that describes the interface and implementation specification of the Web service to be invoked. This WSDL file can be retrieved dynamically using a service registry, or statically, as shown in Figure 8-5.

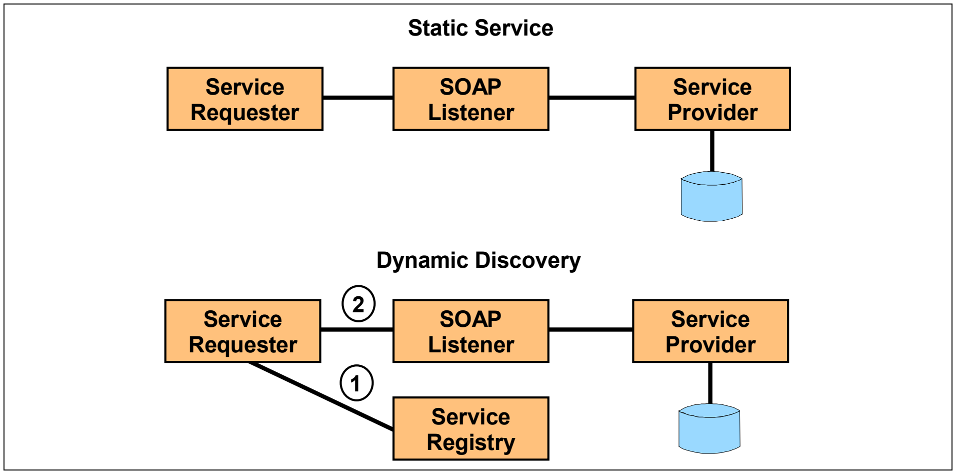


Figure 8-5 Web services discovery methods

Three types of discovery methods for requesters can be identified:

- Static service with no public, private, or shared UDDI registry. The service description is obtained through a proprietary channel (an e-mail, for example).
- Provider-dynamic, where the service interface is dynamically discovered using a UDDI registry, and proxy code is generated from the interface at build time. The service implementation is dynamically discovered at runtime using the same or another UDDI registry.
- Type-dynamic, where the service requester dynamically discovers both the service implementation and interface at runtime. The requester uses generic SOAP APIs to bind to the service provider and invoke the Web service, rather than using generated proxy code.

Although early thinking on Web services asserted that discovery using UDDI would play a significant role in the implementation of the technology, at this time the vast majority of business solutions do not make use of dynamic discovery mechanisms. For this reason, our sample application uses the static service discovery method to access the WSDL file. For further details on dynamic discovery with examples, refer to the following redbook:

- *WebSphere Version 5 Web Services Handbook*, SG24-6891

Message structure

The Web services specification does not mandate any particular message structure. The message structure is defined by the service provider. Message structures can be anything from simple strings to complex XML documents. In our example we use a simple message structure passing a string representing the part number being ordered in the request message and receiving another string representing the expected delivery date in the response.

For an example using an XML message see “Message structure” on page 187.

8.4.2 Object model

In this section we provide an object model for our RPC-based Web services scenario. We focus on how the source application accesses the target application using the JAX-RPC API in WebSphere V5.0.2.

Class diagram

Figure 8-6 is a class diagram of the main source and target application classes directly involved in initiating the direct connection between the source and target applications.

Only the WebServiceBean class and the Inventory interface were developed by hand. The Inventory interface is developed as part of the target EJB application. It is based on the Inventory interface of the InventoryBean session EJB, but is modified to extend java.rmi.Remote rather than javax.ejb.EJBObject, as discussed below in 8.5.1, “Web service enabling the target application” on page 161. This interface is used to generate the target WSDL file, which we use later when generating the client bindings. This is discussed in 8.5.2, “Web service-enabling the source application” on page 166.

The InventoryService, InventoryServiceLocator, and InventorySoapBindingStub shown in Figure 8-6 were generated from the WSDL file. The WebServicesServlet is provided in the WebSphere V5.0.2 Web services runtime.

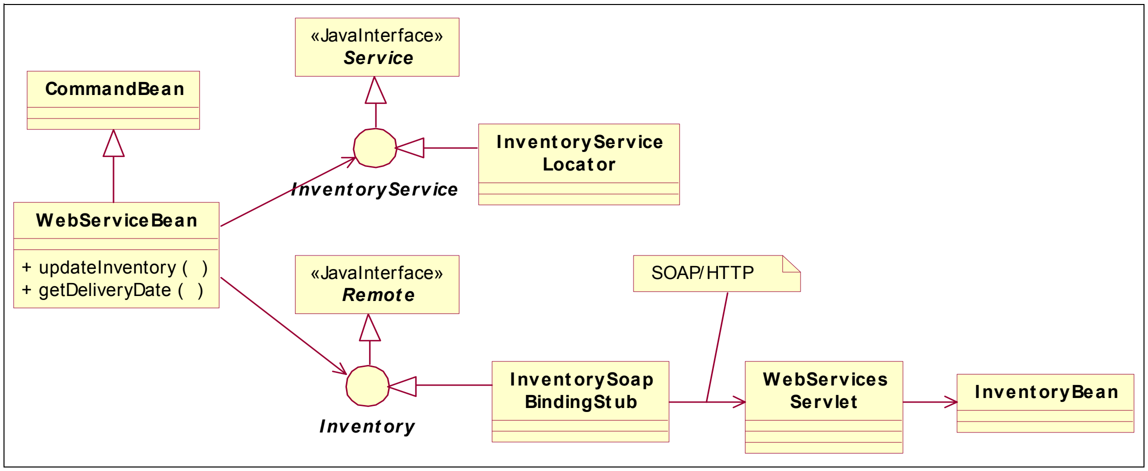


Figure 8-6 Class diagram of the Web service requester and provider (source and target)

Interaction diagrams

In this section we provide interaction diagrams for the Message variation and Call variation of our RPC-based Web services scenario.

Message variation

Our implementation of the Update Inventory use case follows the Message variation, where the source application does not expect or wait for a response from the target application.

The interaction diagram in Figure 8-7 shows the sequence of steps performed when the source application uses the WebServiceBean to perform the updateInventory() method:

1. WebServiceBean in the source application creates an InitialContext and uses it to perform a JNDI lookup to locate the InventoryService. This lookup finds the value set in the webservicesclient.xml deployment descriptor.
2. WebServiceBean invokes the getInventory method of the InventoryService interface (implemented by InventoryServiceLocator) to get the target service endpoint interface, Inventory.
3. WebServiceBean calls the updateInventory method of the Inventory interface (implemented by InventorySoapBindingStub) to invoke the Web service.
4. The stub implementing the Inventory interface performs a one-way invocation of the remote Web service via SOAP and HTTP using the WebSphere Web services runtime.

5. The WebServicesServlet in the target application server invokes the getDeliveryDate method on the InventoryBean, which is the stateless session bean on the target application.

The one-way invocation of the Web service does not block the client application. Execution control returns back through the chain to the WebServiceBean at step 4, without having to wait for the updateInventory call to the InventoryBean to complete.

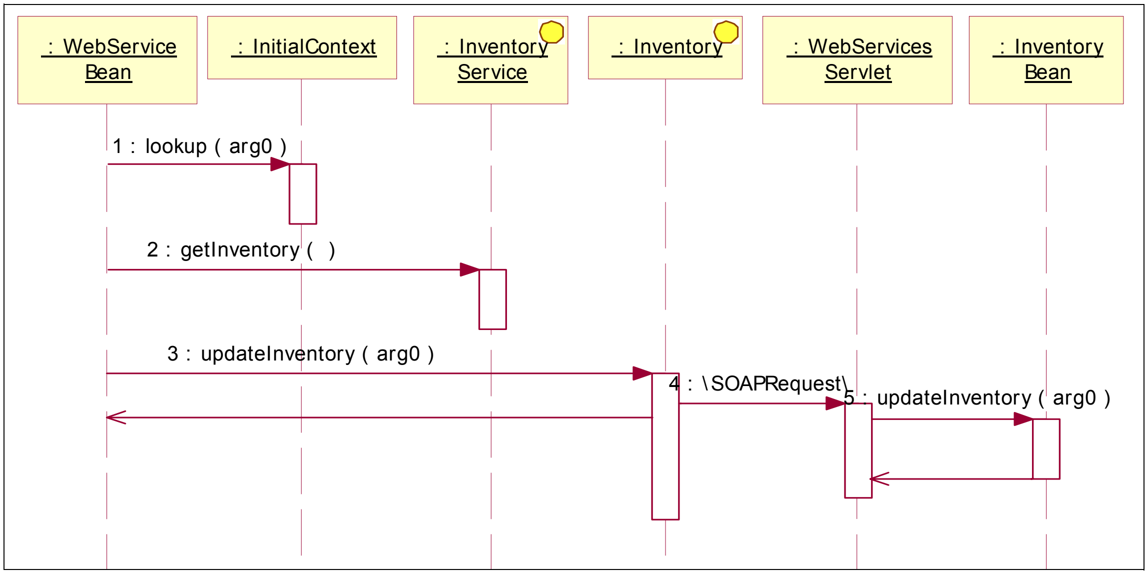


Figure 8-7 Interaction diagram for Web services Message variation

Call variation

Our implementation of the Get Delivery Date use case follows the Call variation, where the source application expects and waits for a response from the target application.

The interaction diagram in Figure 8-8 shows the sequence of steps performed when the source application uses the WebServiceBean to perform the getDeliveryDate() method:

1. WebServiceBean in the source application creates an InitialContext and uses it to perform a JNDI lookup to locate the InventoryService. This lookup finds the value set in the webservicesclient.xml deployment descriptor.
2. WebServiceBean invokes the getInventory method of the InventoryService interface (implemented by InventoryServiceLocator) to get the target service endpoint interface, Inventory.

3. WebServiceBean calls the getDeliveryDate method of the Inventory interface (implemented by InventorySoapBindingStub) to invoke the Web service.
4. The stub implementing the Inventory interface invokes the remote Web service via SOAP and HTTP using the WebSphere Web services runtime.
5. The WebServicesServlet in the target application server invokes the getDeliveryDate method on the InventoryBean, which is the stateless session bean on the target application.

The delivery date then gets passed back through the chain to the requester side and to the WebServiceBean that made the request to begin with.

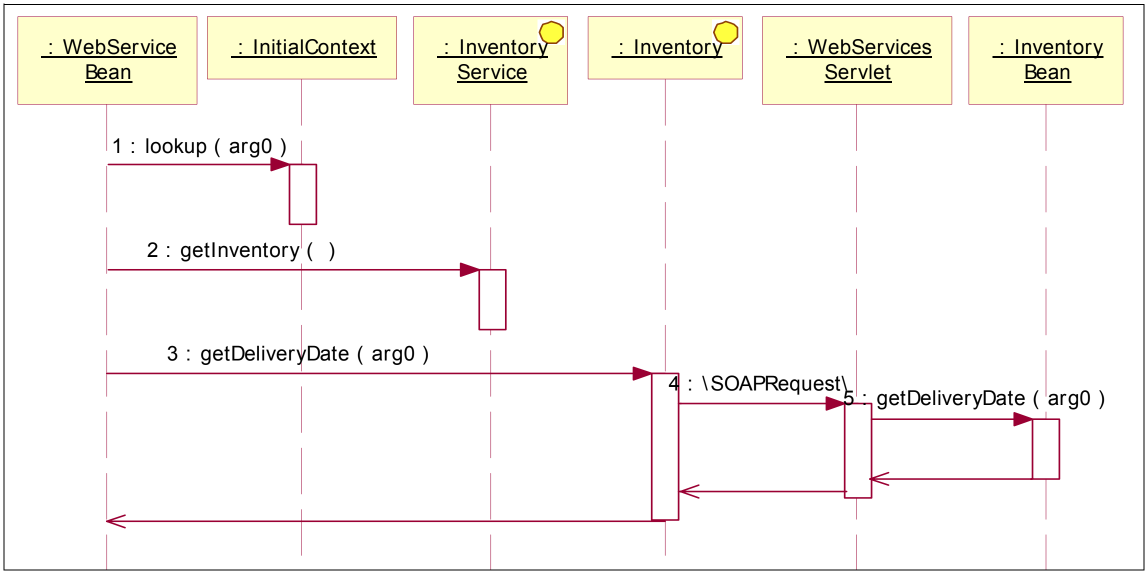


Figure 8-8 Interaction diagram for Web services Call variation

8.5 Development guidelines

In this section we describe:

- How to enable an EJB target application as an RPC style Web service provider.
- How to enable a Web source application as an RPC style Web service client.

We are using the Web Services for J2EE support provided with WebSphere V5.0.2 to implement our solution, which includes command line tools to generate the Web services classes and deployment descriptors. WebSphere Studio

Application Developer V5.1 also provides full support for the creation of Web services for J2EE.

Although all the specifications are human readable (XML), there is a strong need for tools supporting development because many documents with overlapping content are involved. It would be cumbersome and error prone to define all these files without tools.

8.5.1 Web service enabling the target application

Figure 8-9 shows the steps involved in creating the Web services enabled target application using the stateless session bean from our sample code.

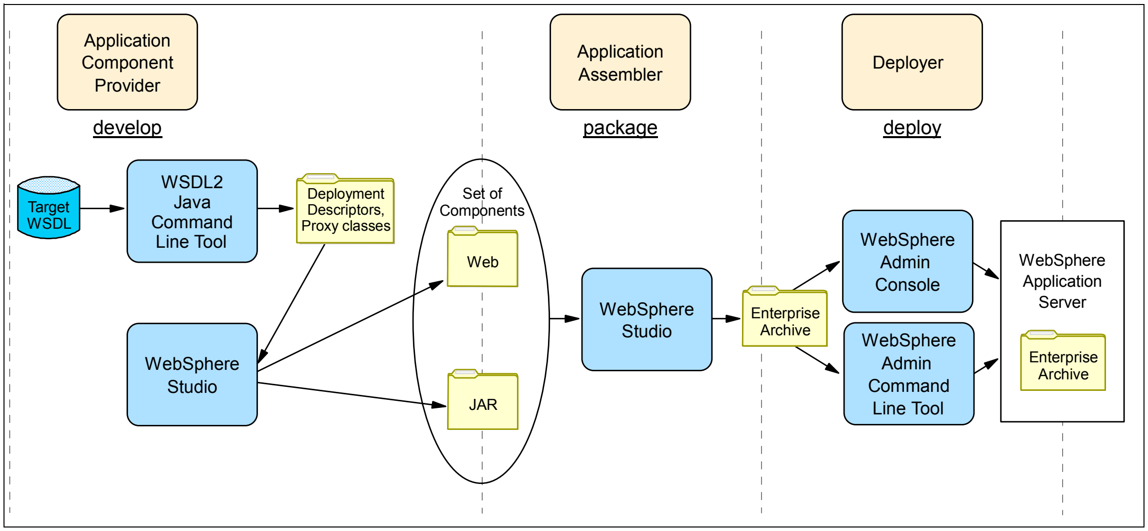


Figure 8-9 Web service development for target application

Let’s walk through the process shown in Figure 8-9 for our target application.

We have already created a simple EJB application using IBM WebSphere Studio Application Developer. We can create an RPC style Web service using this EJB as follows:

1. Create a service endpoint interface for the Web service in a new package. (This step has already been done for you in our sample application.)
   a. Using WebSphere Studio, switch to the J2EE Perspective and click the **Project Navigator** tab.
   b. Navigate to the **ITSOTargetAppEJB** → **ejbModule** folder.

c. Right-click the **ejbModule** folder and select **New** → **Package**. Set the package name to **com.ibm.itso.ws.inventory**.

d. Create the service endpoint interface by copying the Inventory EJB Remote Interface, Inventory.java, from the com.ibm.itso.ejb.inventory package to Inventory.java in the new com.ibm.itso.ws.inventory package.

e. Edit the new Inventory.java file, so that it extends java.rmi.Remote, as shown in Example 8-1.

Example 8-1 Inventory service endpoint interface

```
package com.ibm.itso.ws.inventory;

public interface Inventory extends java.rmi.Remote {
   public void updateInventory(String partNo)
      throws java.rmi.RemoteException;
   public String getDeliveryDate(String partNo)
      throws java.rmi.RemoteException;
}
```

2. Open a command window and change to ITSOTargetAppEJB \ejbModule in your Studio workspace folder.

3. Generate a Web Services Description Language (WSDL) file from the service endpoint interface. We used the command shown in Example 8-2.

Example 8-2 Generating WSDL using Java2WSDL

```
C:\WebSphere\AppServer\bin\Java2WSDL -verbose
   -implClass com.ibm.itso.ejb.inventory.InventoryBean
   -location http://target.itso.ral.ibm.com:9080/ITSOTargetApp/services/Inventory
   -output C:\workspace\ITSOTargetAppEJB\ejbModule\META-INF\wsdl\Inventory.wsdl
   -style rpc -use literal -voidReturn ONEWAY com.ibm.itso.ws.inventory.Inventory

WSWS3010I: Info: Generating portType {http://inventory.ws.itso.ibm.com}Inventory
WSWS3010I: Info: Generating message {http://inventory.ws.itso.ibm.com}getDeliveryDateRequest
WSWS3010I: Info: Generating message {http://inventory.ws.itso.ibm.com}getDeliveryDateResponse
WSWS3010I: Info: Generating message {http://inventory.ws.itso.ibm.com}updateInventoryRequest
WSWS3010I: Info: Generating message {http://inventory.ws.itso.ibm.com}updateInventoryResponse
WSWS3010I: Info: Generating binding {http://inventory.ws.itso.ibm.com}InventorySoapBinding
WSWS3010I: Info: Generating service {http://inventory.ws.itso.ibm.com}InventoryService
WSWS3010I: Info: Generating port Inventory
```

You can see the result of the **-voidReturn ONEWAY** option we specified with **Java2WSDL** by examining the generated WSDL, Inventory.wsdl. As shown in Example 8-3, the getDeliveryDate operation has input and output messages, so it is a request-response operation where a reply is required. The updateInventory operation, however, only has an input message, so it is a one-way operation that does not require a response.

Note: One-way invocations will not throw any exceptions if the target application is not available.

Example 8-3 Generated portTypes for Inventory service

```
...
<wsdl:portType name="Inventory">
   <wsdl:operation name="getDeliveryDate" parameterOrder="partNo">
      <wsdl:input message="intf:getDeliveryDateRequest"
         name="getDeliveryDateRequest"/>
      <wsdl:output message="intf:getDeliveryDateResponse"
         name="getDeliveryDateResponse"/>
   </wsdl:operation>
   <wsdl:operation name="updateInventory" parameterOrder="partNo">
      <wsdl:input message="intf:updateInventoryRequest"
         name="updateInventoryRequest"/>
   </wsdl:operation>
</wsdl:portType>
...
```

4. Using the WSDL file created in the previous step, generate the Web services deployment descriptors and classes using the WSDL2Java tool. We used the command shown in Example 8-4.

Example 8-4 Generating server deployment descriptors and classes using WSDL2Java

```
C:\WebSphere\AppServer\bin\WSDL2Java -verbose -role server -container ejb
   -output C:\workspace\ITSOTargetAppEJB\ejbModule
   C:\workspace\ITSOTargetAppEJB\ejbModule\META-INF\wsdl\Inventory.wsdl

WSWS3185I: Info: Parsing XML file:
C:\workspace\ITSOTargetAppEJB\ejbModule\META-INF\wsdl\Inventory.wsdl
WSWS3330I: Info:
C:\workspace\ITSOTargetAppEJB\ejbModule\com\ibm\itso\ws\inventory\Inventory.java already
exists, WSDL2Java will not overwrite it.
WSWS3282I: Info: Generating
C:\workspace\ITSOTargetAppEJB\ejbModule\com\ibm\itso\ws\inventory\InventorySoapBindingImpl.java
.
WSWS3282I: Info: Generating
C:\workspace\ITSOTargetAppEJB\ejbModule\com\ibm\itso\ws\inventory\Inventory_RI.java.
WSWS3282I: Info: Generating
C:\workspace\ITSOTargetAppEJB\ejbModule\com\ibm\itso\ws\inventory\InventoryHome.java.
WSWS3282I: Info: Generating C:\workspace\ITSOTargetAppEJB\ejbModule\META-INF\webservices.xml.
WSWS3282I: Info: Generating
C:\workspace\ITSOTargetAppEJB\ejbModule\META-INF\ibm-webservices-bnd.xmi.
WSWS3282I: Info: Generating
C:\workspace\ITSOTargetAppEJB\ejbModule\META-INF\ibm-webservices-ext.xmi.
```

```
WSWS3282I: Info: Generating
C:\workspace\ITSOTargetAppEJB\ejbModule\META-INF\Inventory_mapping.xml.
```

> **Note:** The deployment descriptors and class files will not be regenerated when the tool is re-run, unless the existing files have been removed first.

5. Right-click the ITSOTargetAppEJB project and select **Refresh**. You should see the generated files shown in Figure 8-10.

   The webservices.xml deployment descriptor defines the set of Web services that are being deployed in the Web service-enabled J2EE container.

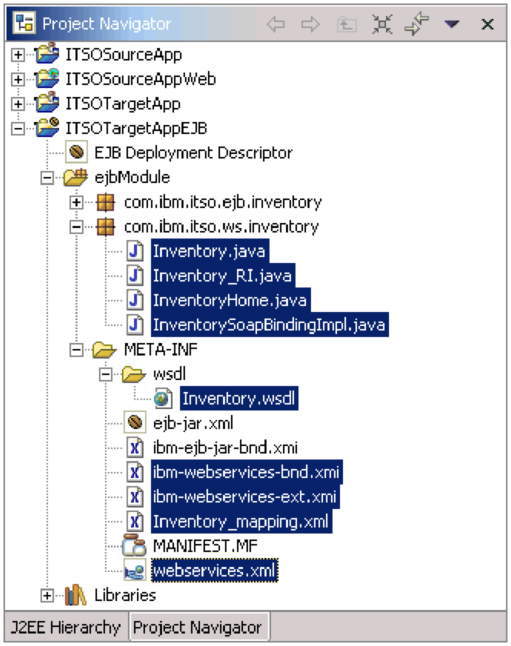


Figure 8-10 WSDL2Java generated files

6. Navigate to **ITSOTargetAppEJB** → **ejbModule** → **META-INF** and edit webservices.xml. Set the **ejb-link** element to **`Inventory`**, as shown in Example 8-5.

   The ejb-link element corresponds to the ejb-name element of the required EJB, as defined in ejb-jar.xml.

Example 8-5 Updating webservices.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE webservices PUBLIC "-//IBM Corporation, Inc.//DTD J2EE Web services 1.0//EN"
"http://www.ibm.com/webservices/dtd/j2ee_web_services_1_0.dtd">
<webservices>
  <webservice-description>
    <webservice-description-name>InventoryService</webservice-description-name>
    <wsdl-file>META-INF/wsdl/Inventory.wsdl</wsdl-file>
    <jaxrpc-mapping-file>META-INF/Inventory_mapping.xml</jaxrpc-mapping-file>
    <port-component>
      <port-component-name>Inventory</port-component-name>
      <wsdl-port>
        <namespaceURI>http://inventory.ws.itso.ibm.com</namespaceURI>
        <localpart>Inventory</localpart>
      </wsdl-port>

<service-endpoint-interface>com.ibm.itso.ws.inventory.Inventory</service-endpoint-interface>
      <service-impl-bean>
        <ejb-link>Inventory</ejb-link>
      </service-impl-bean>
    </port-component>
  </webservice-description>
</webservices>
```

7. Export the ITSOTargetApp project to an EAR file:
   a. Right click the **ITSOTargetApp** project and select **Export...**.
   b. Select **EAR file** and click **Next**.
   c. Select the destination you want the EAR file to be exported to, for example:

      C:\WebSphere\AppServer\installableApps\ITSOTargetApp.ear
   d. Click **Finish**.
8. Run the endptEnabler command line tool to add an HTTP router module to the EAR file. (This step has already been done for you in our sample application.) We used the command shown in Example 8-6.

   > **Tip:** This step is only required if the Web service is implemented in an EJB module.

Example 8-6 Running the endpoint enabler tool

```
C:\WebSphere\AppServer\bin\endptEnabler

WSWS2004I: IBM WebSphere Application Server Release 5
WSWS2005I: Web Services Enterprise Archive Endpoint Enabler Tool.
WSWS2007I: (C) COPYRIGHT International Business Machines Corp. 1997, 2003.
WSWS2006I: Please enter the name of your EAR file:  ITSOTargetApp.ear
```

```
WSWS2003I: Backing up EAR file to: ITSOTargetApp.ear~

WSWS2016I: Loading EAR file: ITSOTargetApp.ear
WSWS2017I: Found EJB Module: ITSOTargetAppEJB.jar

WSWS2029I: Enter http router name for EJB Module ITSOTargetAppEJB
[ITSOTargetAppEJB_HTTPRouter.war]: ITSOTargetAppWeb.war

WSWS2030I: Enter http context root for EJB Module ITSOTargetAppEJB
[/ITSOTargetAppEJB]: ITSOTargetApp
WSWS2024I: Adding http router for EJB Module ITSOTargetAppEJB.jar.
WSWS2036I: Saving EAR file ITSOTargetApp.ear...
WSWS2037I: Finished saving the EAR file.
WSWS2018I: Finished processing EAR file ITSOTargetApp.ear.
```

The endptEnabler tool makes the following changes to the EAR file:

- Adds a Web module to the EAR file that contains the HTTP router for the EJB module. Also sets the context root for the Web module in application.xml.
- Adds servlet and servlet-mapping elements to the Web module deployment descriptor. These elements map the Web service endpoint URL to the Web services router servlet, and are added for each Web service in the module.
- Adds a routerModules element to ibm-webservices-bnd.xmi in the EJB module.

> **Tip:** If you are using WebSphere Studio V5.1, you can right-click the EAR project, and select **Web Services** → **Endpoint Enabler** from the pop-up menu.

The EAR file is now ready to deploy in the IBM WebSphere Application Server V5.0.2 runtime.

> **Note:** The WebSphere V5.0.2 Web service deployment tools will not append new Web services to existing Web services deployment descriptors. If you need to deploy more than one Web service in a module, you will need to manually merge the Web service deployment descriptors.

8.5.2 Web service-enabling the source application

Figure 8-11 shows the steps involved in Web services-enabling the source (client) application.

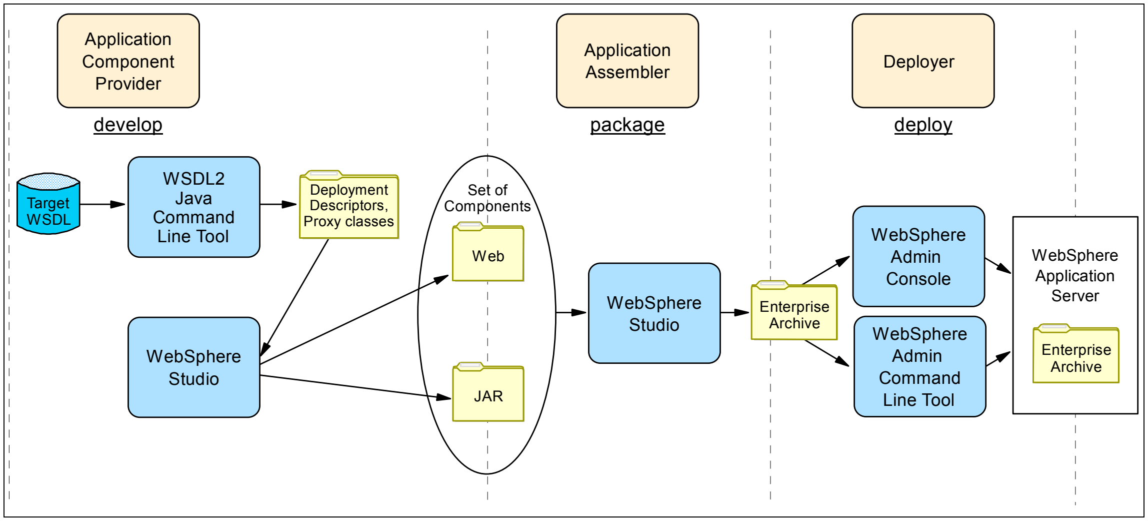


Figure 8-11 Web service development for source application

Let's walk through the process shown in Figure 8-11 for our source application.

When developing a Web services client, you must have access to a server application. This could be developed by a separate department within your organization, which will supply the WSDL file defining their server. In our example, we use the target application described in 8.5.1, "Web service enabling the target application" on page 161.

Web service-enabling our source application is simply a matter of obtaining the WSDL file for the target Web service, and running the WSDL2Java tool to generate the required deployment descriptors and proxy classes:

1. Copy the target application WSDL file to the source application Web module:
   a. Using WebSphere Studio, switch to the J2EE Perspective and click the **Project Navigator** tab.
   b. Navigate to the **ITSOSourceAppWeb** → **WebContent** → **WEB-INF** folder.
   c. Right-click the **WEB-INF** folder and select **New** → **Folder**. Set the folder name to `wsdl`.
   d. Copy the target application WSDL file, Inventory.wsdl, from the ITSOTargetAppEJB/ejbModule/META-INF/wsdl folder to Inventory.wsdl in the new ITSOSourceAppWeb/WebContent/WEB-INF/wsdl folder.
2. Open a command window.

3. Using the WSDL file created in step 1, generate the Web service client deployment descriptors and classes using the WSDL2Java tool. We used the command shown in Example 8-7.

Example 8-7 Generating client deployment descriptors and classes using WSDL2Java

```
C:\WebSphere\AppServer\bin\WSDL2Java -verbose -role client -container web
   -output C:\workspace\ITSOSourceAppWeb\WebContent
   C:\workspace\ITSOSourceAppWeb\WebContent\WEB-INF\wsdl\Inventory.wsdl

WSWS3185I: Info: Parsing XML file:
C:\workspace\ITSOSourceAppWeb\WebContent\WEB-INF\wsdl\Inventory.wsdl
WSWS3282I: Info: Generating
C:\workspace\ITSOSourceAppWeb\WebContent\com\ibm\itso\ws\inventory\InventoryService.java.
WSWS3282I: Info: Generating
C:\workspace\ITSOSourceAppWeb\WebContent\com\ibm\itso\ws\inventory\InventoryServiceLocator.java
.
WSWS3282I: Info: Generating
C:\workspace\ITSOSourceAppWeb\WebContent\com\ibm\itso\ws\inventory\Inventory.java.
WSWS3282I: Info: Generating
C:\workspace\ITSOSourceAppWeb\WebContent\com\ibm\itso\ws\inventory\InventorySoapBindingStub.jav
a.
WSWS3282I: Info: Generating
C:\workspace\ITSOSourceAppWeb\WebContent\WEB-INF\webservicesclient.xml.
WSWS3282I: Info: Generating
C:\workspace\ITSOSourceAppWeb\WebContent\WEB-INF\ibm-webservicesclient-bnd.xmi.
WSWS3282I: Info: Generating
C:\workspace\ITSOSourceAppWeb\WebContent\WEB-INF\ibm-webservicesclient-ext.xmi.
WSWS3282I: Info: Generating
C:\workspace\ITSOSourceAppWeb\WebContent\WEB-INF\Inventory_mapping.xml.
```

4. In WebSphere Studio, move the generated Java source files from the Web module's WebContent folder to its JavaSource folder:

   a. Right-click the **ITSOSourceAppWeb** project and select **Refresh** from the pop-up menu. The generated files should now appear in the Studio workspace.

   b. Move the com.ibm.itso.ws.inventory package in the ITSOSourceAppWeb\WebContent folder to the ITSOSourceAppWeb\JavaSource folder.

   The generated files are highlighted in Figure 8-12.

   The webservicesclient.xml deployment descriptor defines the JNDI name for accessing the Web service and the associated service endpoint interface to be used.

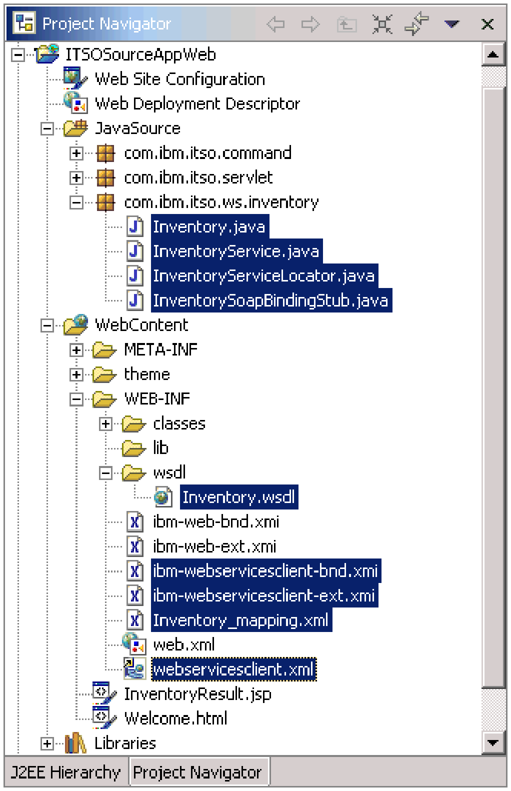


Figure 8-12 Generated client binding files and deployment descriptors

5. Add client application code to invoke the Web service on the target application.

   To invoke getDeliveryDate on the target application, we added the code shown in Example 8-8. We added this code to the com.ibm.itso.command.WebServiceBean command bean in our ITSOSourceAppWeb module.

Example 8-8 Web service client code for getDeliveryDate

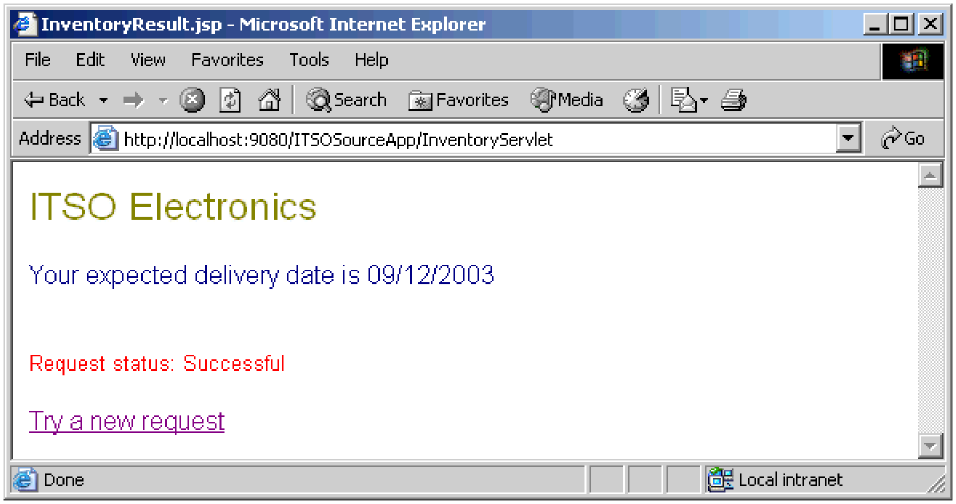
```
public String getDeliveryDate(String partNumber) throws Exception {

   String deliveryDate = null;

   try {

      Context ctx = new InitialContext();

      InventoryService service = (InventoryService) ctx.lookup(
            "java:comp/env/service/InventoryService");

         // Request the Service Endpoint from the Service
         Inventory port = service.getInventory();

         // Get the quote
         deliveryDate = port.getDeliveryDate(partNumber);

      } catch (Exception e) {
         //...
      }

      return deliveryDate;
   }
```

6. Test the source and target applications in the WebSphere Studio Application Developer V5.1 test environment.

> **Note:** If you want to go ahead and test the RPC style Web service before you implement the document style Web service (described in Chapter 9, “Using document style Web services” on page 183) you need to comment out the following imports and methods in com.ibm.itso.ejb.inventory.Inventory and com.ibm.itso.ejb.inventory.InventoryBean in the ITSOTargetAppEJB module in the sample application:
>
> ```
>    import com.ibm.itso.ws.inventory.reply.InventoryReply
>    import com.ibm.itso.ws.inventory.request.InventoryRequest
>    public void updateInventory(InventoryRequest reqMsg)
>    public InventoryReply getDeliveryDate(InventoryRequest reqMsg)
> ```

7. Deploy the source and target applications in your IBM WebSphere Application Server V5.0.2 runtime environment to try the applications on separate machines.

Figure 8-13 shows the results page for our Get Delivery Date use case.


InventoryResult.jsp - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Media
Address
http://localhost:9080/ITSOSourceApp/InventoryServlet
Go

ITSO Electronics

Your expected delivery date is 09/12/2003

Request status: Successful

Try a new request

Figure 8-13 Get delivery date results page

Changing the Web service endpoint URL

By default, the endpoint proxy class (InventoryServiceLocator in our example) will use the endpoint address specified in the WSDL file when the proxy class was originally generated. It is likely that the endpoint address used will change over time, as the client application moves though the development, test, and production environments, for example. You can set the endpoint address in several ways:

1. When getting an instance of the generated endpoint class from the Service interface, you can optionally pass an endpoint address that will override the default address obtained from the WSDL file. Compare the following approach with that used in Example 8-8 on page 169:

   ```
   // Request the Service Endpoint from the Service
   //  overriding the default endpoint address
   Inventory port = service.getInventory(new java.net.URL(
      "http://localhost:9080/ITSOTargetApp/services/Inventory"));
   ```

2. When deploying your client application to IBM WebSphere Application Server V5.0.2, you can specify the **Deploy WebServices** option, as shown in Figure 8-14.

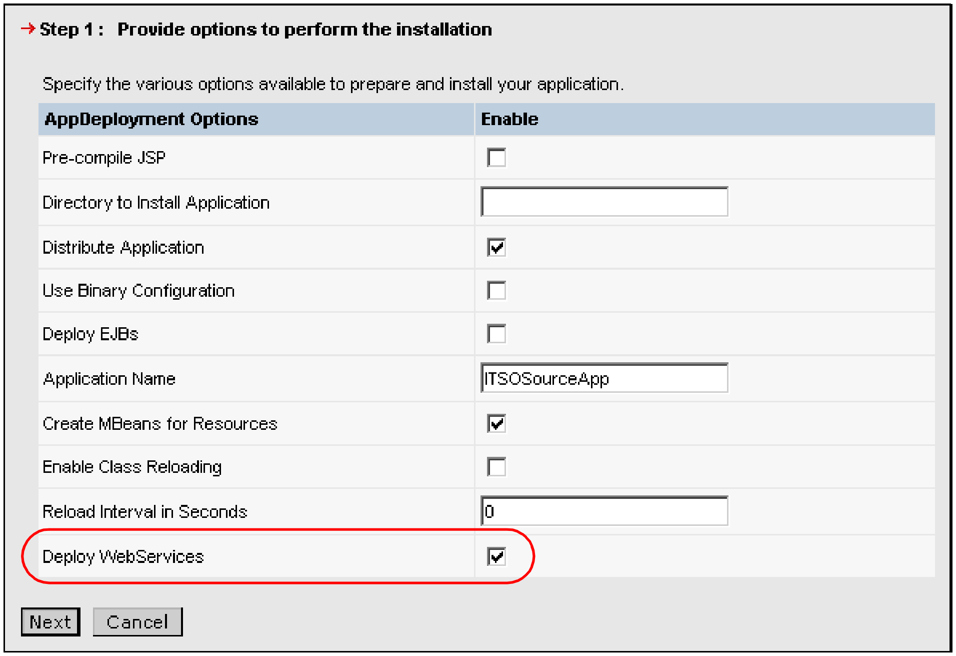
   WebSphere will regenerate the deployment code based on the Web services client deployment descriptors, updating it with the current endpoint address from the WSDL file. Similar to EJB deployment, this only needs to be performed when the deployment details have changed.

   Example 8-9 shows how the endpoint address is specified in the WSDL file for our Inventory service.

The second option is recommended in most cases. Using this method, the endpoint address can be specified at application packaging/deployment time using the Web services deployment features of the application server.

Example 8-9 Setting the endpoint address in the WSDL file

```
...
<wsdl:service name="InventoryService">
   <wsdl:port binding="intf:InventorySoapBinding" name="Inventory">
      <wsdlsoap:address location="http://localhost:9080/ITSOTargetApp/services/Inventory"/>
   </wsdl:port>
</wsdl:service>
...
```



→ **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	ITSOSourceApp
Create MBeans for Resources	☑
Enable Class Reloading	☐
Reload Interval in Seconds	0
Deploy WebServices	☑

Figure 8-14 Deploying Web services using the WebSphere administrative console

8.5.3 Monitoring SOAP messages

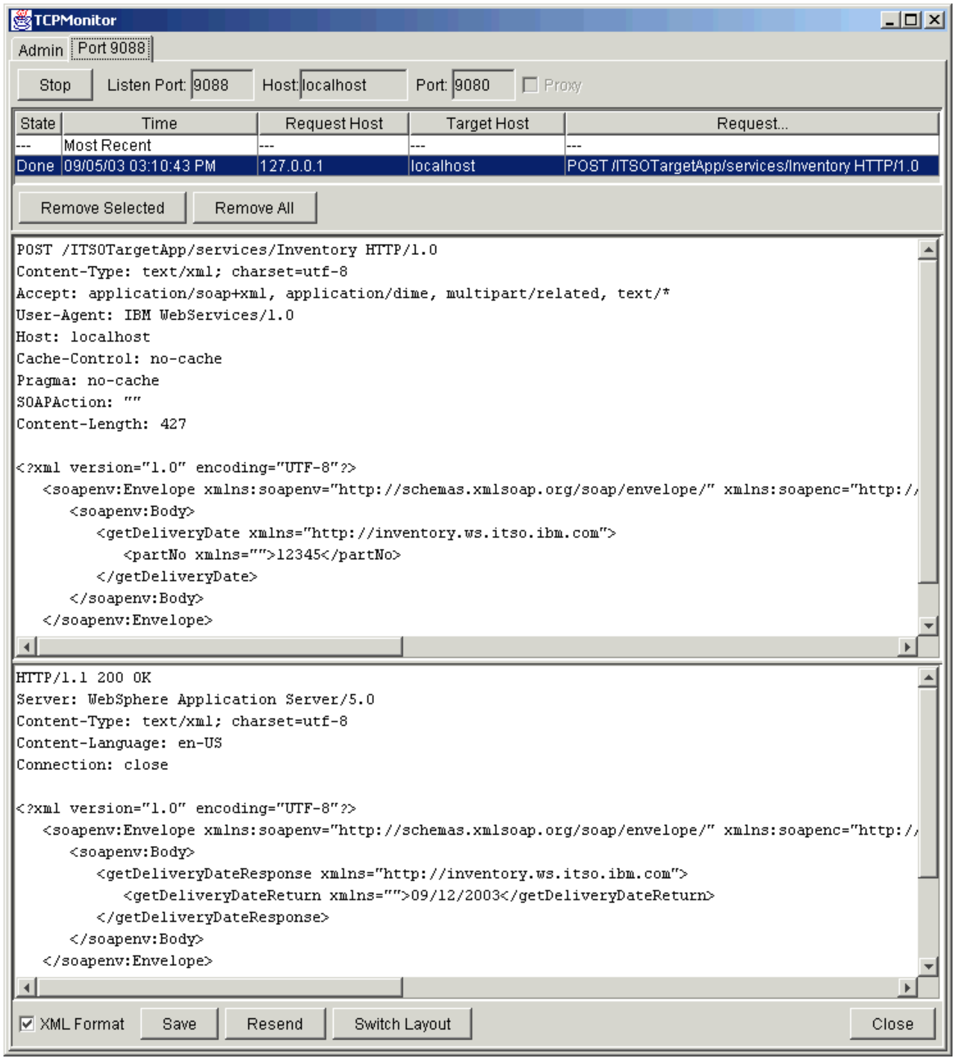
You can trace the XML messages exchanged between a Web service client and the server. In this section we look at two tools:

- The TCPMon tool provided with IBM WebSphere Application Server V5.0
- The TCP/IP Monitor Server provided with WebSphere Studio Application Developer

WebSphere TCPMon tool

The TCPMon tool allows SOAP messages to be traced by redirecting messages from one port to another, displaying the contents as they go. The WebSphere application server normally listens on port 9080. To trace messages 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 IBM WebSphere Application Server V5.0.2. It allows you to view the contents of the SOAP messages exchange between the source and target applications, as shown in Figure 8-15.



Listen Port: 9088 Host: localhost Port: 9080 □ Proxy

State	Time	Request Host	Target Host	Request...
---	Most Recent	---	---	---
Done	09/05/03 03:10:43 PM	127.0.0.1	localhost	POST /ITSOTargetApp/services/Inventory HTTP/1.0

```
POST /ITSOTargetApp/services/Inventory HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: IBM WebServices/1.0
Host: localhost
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 427

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soapenc="http:/
      <soapenv:Body>
         <getDeliveryDate xmlns="http://inventory.ws.itso.ibm.com">
            <partNo xmlns="">12345</partNo>
         </getDeliveryDate>
      </soapenv:Body>
   </soapenv:Envelope>
```

```
HTTP/1.1 200 OK
Server: WebSphere Application Server/5.0
Content-Type: text/xml; charset=utf-8
Content-Language: en-US
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soapenc="http:/
      <soapenv:Body>
         <getDeliveryDateResponse xmlns="http://inventory.ws.itso.ibm.com">
            <getDeliveryDateReturn xmlns="">09/12/2003</getDeliveryDateReturn>
         </getDeliveryDateResponse>
      </soapenv:Body>
   </soapenv:Envelope>
```

☑ XML Format

Figure 8-15 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 further details on TCPMon, see the InfoCenter article *Tracing Web services messages* at:

http://www.ibm.com/software/webservers/appserv/infocenter.html

The SOAP request for our one-way updateInventory() method is shown in Example 8-10.

Example 8-10 SOAP request for updateInventory

```
POST /ITSOTargetApp/services/Inventory HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: IBM WebServices/1.0
Host: localhost
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 427

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <updateInventory xmlns="http://inventory.ws.itso.ibm.com">
            <partNo xmlns="">12345</partNo>
         </updateInventory>
      </soapenv:Body>
   </soapenv:Envelope>
```

The HTTP response for our updateInventory() method is shown in Example 8-11. There is no SOAP response envelope for the one-way invocation of updateInventory.

A response appears because the HTTP protocol requires an acknowledgement to be returned to the sender. However, this acknowledgement simply confirms that the one-way call was successfully received by the HTTP transport mechanism, for “out-bound” transmission. There is no guarantee that the call was successfully delivered to the final recipient, nor can there be one, by definition.

Example 8-11 SOAP response for updateInventory

```
HTTP/1.1 200 OK
Server: WebSphere Application Server/5.0
Content-Type: text/html; charset=ISO-8859-1
Content-Language: en-US
Content-Length: 0
Connection: close
```

The SOAP request for our getDeliveryDate() method is shown in Example 8-12.

Example 8-12 SOAP request for getDeliveryDate

```
POST /ITSOTargetApp/services/Inventory HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: IBM WebServices/1.0
Host: localhost
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 427

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <getDeliveryDate xmlns="http://inventory.ws.itso.ibm.com">
            <partNo xmlns="">12345</partNo>
         </getDeliveryDate>
      </soapenv:Body>
   </soapenv:Envelope>
```

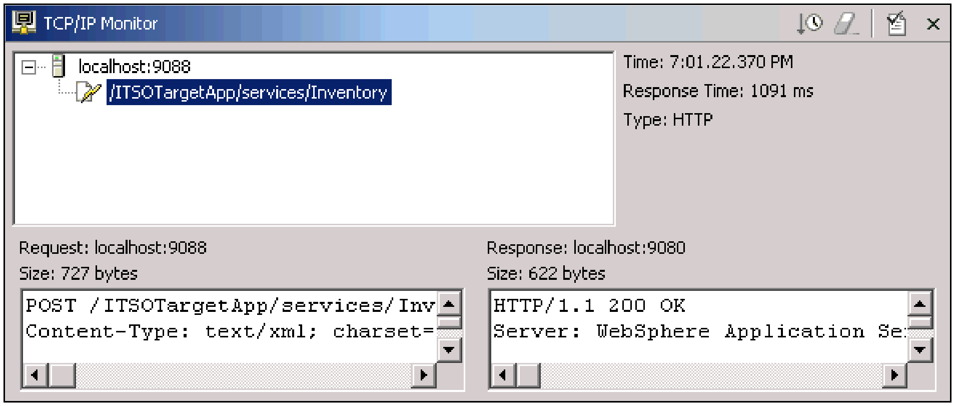
The SOAP response for the getDeliveryDate() method is shown in Example 8-13.

Example 8-13 SOAP response for getDeliveryDate

```
HTTP/1.1 200 OK
Server: WebSphere Application Server/5.0
Content-Type: text/xml; charset=utf-8
Content-Language: en-US
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <getDeliveryDateResponse xmlns="http://inventory.ws.itso.ibm.com">
            <getDeliveryDateReturn xmlns="">09/12/2003</getDeliveryDateReturn>
         </getDeliveryDateResponse>
      </soapenv:Body>
   </soapenv:Envelope>
```

TCP/IP Monitor Server

The TCP/IP Monitor Server provided with WebSphere Studio Application Developer (shown in Figure 8-16) 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 8-16 Tracing SOAP messages using WebSphere Studio TCP/IP Monitor Server

8.6 Quality of Service capabilities

In this section we discuss Quality of Service capabilities and considerations specific to Web services.

For further discussion on Quality of Services see the IBM developerWorks article *Understanding quality of service for Web services*:

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

8.6.1 Autonomic

Log and trace facilities are important for fault monitoring and isolation. WebSphere Application Server provides a number of log files. JVM logs are located in the <WAS_HOME>/logs/<applicationServerName> directory, and by default are named SystemOut.log and SystemErr.log.

The Diagnostic Trace Service can be used to enable tracing of application server components. The following trace specification can be used when diagnosing Web services problems:

```
com.ibm.ws.webservices.*=all=enabled
```

The TCPMon tool described in 8.5.3, “Monitoring SOAP messages” on page 172 allows you to view the contents of the SOAP messages being generated by the interaction between the source and target applications.

8.6.2 Availability

The Patterns for e-business define a set of high-availability patterns that provide the node redundancy and failover capabilities needed to eliminate single-point-of-failure for the end-to-end production system. The following Non-Functional Requirements::High Availability: Runtime patterns are defined:

- High Availability: Basic Runtime pattern
- High Availability: Runtime pattern: Single load balancer
- High Availability: Runtime pattern: Load balancer hot standby
- High Availability: Runtime pattern: Mutual high availability
- High Availability: Runtime pattern: Wide area load balancing

Many of the same principles that apply to any Web application can be applied to Web services when it comes to availability.

See the IBM Redbook *Patterns for the Edge of Network*, SG24-6822, for details on how these Non-Functional Requirements custom designs are applied to Web applications.

8.6.3 Performance

The Patterns for e-business also define a set of high performance patterns that provide the scalability and workload management capabilities needed to meet performance and throughput requirements. The following Non-Functional Requirements::High Performance: Runtime patterns are defined:

- High Performance: Basic Runtime pattern
- High Performance: Runtime pattern: Redirectors
- High Performance: Runtime pattern: Separation
- High Performance: Runtime pattern: Caching proxy

Many of the same principles that apply to any Web application can be applied to Web services when it comes to performance.

See the IBM Redbook *Patterns for the Edge of Network*, SG24-6822, for details on how these Non-Functional Requirements custom designs are applied to Web applications.

XML parsing

The WebSphere V5.0.2 Web services runtime uses SAX (event-based) parsing, achieving improved performance over the earlier versions of Apache SOAP, which used DOM parsing. Still, in order to minimize the effect on performance of XML parsing, some steps can be taken:

- Avoid chaining services if possible, since this will increase path lengths.
- Design for coarse-grained, document-based interactions.
- Balance architecting service re-use with number of invocations per transaction.

Usually the performance loss from using an XML parser such as SAX is negligible compared to the effort necessary to build up the communication itself. The highest performance results can be expected from using the appropriate transport protocol. For example, consider using RMI instead, if appropriate. You can always expose the EJB as Web services later, if needed.

8.6.4 Security

Web services security for IBM WebSphere Application Server V5.0.2 is based on standards included in the Web services security (WS-Security) specification. Web services security is a message-level standard, based on securing SOAP messages through XML digital signatures, confidentiality through XML encryption and credential propagation through security tokens.

Transport-level security is based on the Secured Sockets Layer (SSL) or Transport Layer Security (TLS) mechanisms across the HTTP protocol. SSL and TLS provide security features including authentication, data protection, and cryptographic token support for secure HTTP connections. To run with HTTPS, the service endpoint address must be in the form of https://.

For further details on Web services security for WebSphere V5.0.2, see the InfoCenter article *Securing Web services* at:

`http://www.ibm.com/software/webservers/appserv/infocenter.html`

UDDI security

One reason for the delay in widespread adoption of UDDI is the lack of security standards that would allow companies to restrict Web services access information to trusted partners. The third version of the UDDI standard will include security specifications.

8.6.5 Standards compliance

By utilizing open standards, Web services can, in theory, enable any two software components to communicate—no matter what technologies or platforms are used to create or deploy them. Interoperability across heterogeneous platforms is one of the key value propositions of Web services.

Unfortunately, there is still no common, agreed-upon definition of what a Web service is, many needed standards are still in their infancy, and some are still competing against each other. To address the potential problems, the Web Services Interoperability (WS-I) Group released the Web Services Basic Profile 1.0 on October 17, 2002.

See the WS-I Basic Profile Version 1.0 specification for more details:

http://www.ws-i.org/Profiles/Basic/2002-10/BasicProfile-1.0-WGD.htm

8.6.6 Transactionality

Almost every party agrees that we need a standard that accommodates both classical ACID (XA or database-style transactions) and long-running, compensating transactions. But there is still sharply divided opinion on where such standards fit in the Web services stack.

The Business Transaction Protocol (BTP) from OASIS was backed by a number of smaller vendors (BEA, HP, Choreology, Oracle) and Version 1.0 was released in May 2002. BTP tries to adopt XML-based technology for business transactions on the Internet and tackles such challenges as transactions that span multiple enterprises and long-lasting transactions. BTP has been criticized as being too complex, and still lacks backing from an industry heavyweight (like IBM or Microsoft).

In August 2002, IBM, Microsoft, and BEA published two draft specifications:

- WS-Coordination is a general purpose and extensible framework for providing protocols that coordinate the actions of distributed transactions. The defined framework enables an application service to create a context needed to propagate an activity to other services and to register for coordination protocols. The framework also enables existing transaction processing, workflow, and other systems for coordination to hide their proprietary protocols and to operate in a heterogeneous environment. It can be used with message sequencing and state machine synchronization.

  See http://www.ibm.com/developerworks/library/ws-coor/ for the published specification.

- WS-Transaction includes support for the two types of transactions. It describes coordination types that are used with the extensible coordination framework as described in WS-Coordination. Two coordination types are defined: Atomic Transaction (AT) and Business Activity (BA). WS-Transaction is a building block used with other specifications (for example, WS-Coordination, WS-Security) and application-specific protocols that are able to accommodate a wide variety of coordination protocols related to the coordination actions of distributed applications.

See http://www.ibm.com/developerworks/library/ws-transpec/ for the published specification.

While these proposals and specifications are still evolving, it is recommended that we, as architects and developers, actively participate in, review, comment on, and help improve the specifications. Also evaluate early implementations for inclusion in corporate architecture standards and possible application implementation. If there is urgent need for designing and implementing a Web services-based transactional infrastructure and related business services, we recommend using the principles behind the new specifications.

8.7 Best practices

In this section, we focus on best practices for Web services development and deployment within a J2EE environment, that is, Web services that are built using servlets, JSP pages, EJB architecture, and all the other standards that are part of the J2EE technology.

These best practices include:

- Apply distributed computing principles

  Think of Web services as another technology for developing distributed systems.

- Design systems that are layered.

  It is especially important in Web services applications where we do not have control over some components (services) that we access in our application.

- Design coarse-grained Web services.

  Requesting a service from a machine over the network is more expensive than a local operation. Keep the request as coarse grained as possible when requesting a Web service from a remote machine, thereby avoiding unnecessary network traffic and overhead on the communication stack.

- Design for “loosely coupled” components.

  Because a Web service is by definition an interface to a loosely coupled component on a remote system, define clear contracts between layers and services, but utilize the “Parameter List” paradigm where possible.

- Limit dependency on other components.

  Common dependencies to be avoided are Call flow dependency and Object association dependency. Implement all cross “domain” business processes in a “control” or “workflow” layer.

- Utilize standard XML structures to pass data, where possible using a standardized structure and meaning.

- Use existing Web services tools, such as WebSphere Studio Application Developer.

  This allows you to expose assets and services using WSDL and proxy-generation tools, which shield you from the underlying XML messages in Web services.

9

Using document style Web services

This chapter discusses using document style Web services in an intra-enterprise integration scenario. We are using Web services for J2EE as provided with IBM WebSphere Application Server base V5.0.2 in this scenario.

This chapter describes the following:

- Using document style Web services in the context of our ITSO Electronics business scenario.
- Design issues to be considered when using document style Web services and the design applied to our document-based Web services.
- Creating a document style Web service provider and requester using Web Services for J2EE.
- An example of integrating WebSphere with .NET-based Web services.
- Quality of Service capabilities for document style Web services.
- Best practices for document style Web services.
- An overview of ebXML.

9.1 Business scenario

The document style Web service is equally applicable to the business scenario we described in Chapter 8, “Using RPC style Web services” on page 147. The development effort involved is more complex than RPC style, but document style provides some other advantages that make it an important alternative to RPC style. Document style provides lower coupling between the Web service requester and provider because requester and provider binding is reduced from stricter methods and parameters to more flexible documents.

9.2 Document style Web services

While our discussion of Web service implementation has so far been based on the RPC communication style, the SOAP and WSDL specifications allow for another kind of Web service binding style: the document style.

Document style Web services are document-driven and more loosely coupled than RPC style Web service. RPC style assumes that the provider is a procedure, and the call to the service is a remote procedure call, where the requester provides the call parameters in the SOAP body. The call parameters are wrapped in an element that specifies the procedure’s name. The document style makes no assumptions about how the provider will process the service call, which leaves more flexibility in the definition of the messages consumed by and produced by the service.

Advantages of document style include:

- Document style is more naturally suited to asynchronous processing and one-way scenarios.
- Changes to the message schema are less likely to break service consumers. Adding elements to messages, and reordering sequences in message definitions, are less likely to impact both consumers and providers.
- It is possible to add information to the XML document that is being exchanged between the consumer and provider for purposes other than the invocation of the service. In the RPC style, the message must be validated in order to invoke the target procedure. With document style, validation of additional elements can be deferred to a component of the implemented service.
- Microsoft tools for the creation of Web services tend to use the document style, so interoperability with service providers or requesters built for compatibility with Microsoft platforms is more likely.
- Document style services tend to perform better than RPC style services. This is because the SOAP server has to perform validation with RPC style,

whereas for document style the XML request is sent straight to the processing application.

Disadvantages of document style include:

- It is more complex to write and provide services for document style. RPC services are more rigid in the message formats they use, and therefore less parsing and message analysis code is required in the application code than with document style, which may have to cater for more data and more combinations of document structure.
- Development tool support for document style Web services has not been strong. WebSphere Studio Application Developer V5.1 and WebSphere Application Server V5.0.2 provide new development and deployment tools supporting document style Web services.

A detailed discussion of the merits of document style is available from:

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

9.3 Design guidelines

Figure 9-1 shows the Runtime pattern and Product mapping we used to demonstrate the Direct Connection application pattern within the business domain of an organization, using Web services technology. It is the same Runtime pattern and Product mapping used in Chapter 8, “Using RPC style Web services” on page 147. It supports both RPC and document style Web services.

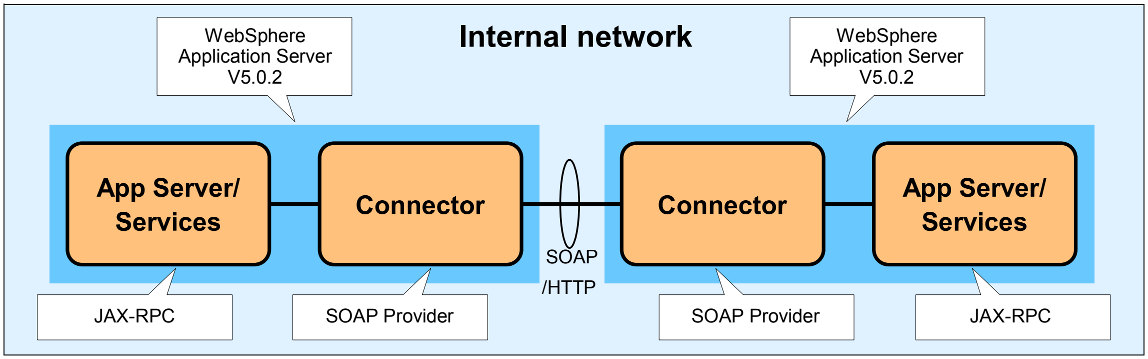


Figure 9-1 Web services product mapping for Application Integration::Direct Connection

See Figure 8-4 on page 153 for an expanded view of this solution, showing the “stack” of coupling adapter pairs over “virtual” connections between the source and target.

9.3.1 Design considerations

A number of factors affect the design of a Web service. In this section we discuss some of the factors we considered when using document style Web services in our sample application.

SOAP messaging style

SOAP provides support for both RPC style Web services and document style Web services. As discussed previously, the advantages of document style Web services include:

- Looser coupling between the Web service requester and provider
- Better interoperability between Web service requesters and providers
- Better fit with asynchronous messaging scenarios

The disadvantages of document style include:

- Higher development effort than RPC style

We used document style Web services in our sample solution to demonstrate the Direct Connection pattern because of the improved interoperability and looser coupling provided. The document-based mechanism can be used for both the Call and Message styles of communication.

Transmission style

As with RPC style Web services, document style Web services can be operated in synchronous (request-reply, solicit-response) or asynchronous (one-way, notification operation) modes. Deciding which mode to use depends on several factors:

- Quality of Service requirements

  Improving the performance, reliability, and scalability of the business process may be a major issue when discussing the decoupling of applications. Services using asynchronous mode are usually less prone to execution delays or failures of the target application.

- Implementation effort

  Where the source application expects a response from the target application, or requires confirmation of successful execution on the target side, it may be easier and more appropriate to provide services using a synchronous mode, such as request-reply. Using an asynchronous mode would require additional effort to handle the receipt of responses and notifications or to implement error handling. This also has an impact on QoS aspects such as maintainability or complexity of architecture.

- Transport protocol

  The choice of underlying transport protocol, and the resulting transport capabilities, will affect the ease of using one or other asynchronous mode. For example, using a message transport infrastructure facilitates asynchronous operations while HTTP rather favors synchronous operations.

Message structure

The Web services specification does not mandate any particular message structure; the message structure is defined by the service provider. Message structures can be anything from simple strings to complex XML documents. In our document style example we use an XML request document containing the part number being ordered in the request message, and in reply receive an XML response document containing the expected delivery date.

We define the required message structures using schema files that are imported into the WSDL file for our Web service provider. There are a number of considerations when importing your message and type definitions, including:

- Defining messages separately from the interface or implementation files facilitates reuse of the message definitions in other service definitions.
- Changes to schema files may impact services that use those definitions, particularly if the implementer of the service or consumer has generated serialization code directly from the WSDL files. This means that once a data schema is published, altering it may have a big impact on users of services that use the schema.
- For any imports that refer to a network location, the network location must be available when performing operations such as designing WSDL files, starting services, and so on.
- Both run-time and development-time products must support the use of imported schemas.

9.3.2 Object model

In this section we provide an object model for our document style Web services scenario. We focus on how the source application is accessing the target application using Axis.

Class diagram

Figure 9-2 is a class diagram of the main application classes involved in initiating the direct connection between the source and target applications via document style Web services.

The WebServiceDocBean class was developed by hand. The InventoryDoc, InventoryDocService, InventoryDocServiceLocator, and InventoryDocSoapBindingStub shown in Figure 9-2 were generated from the WSDL file. The WebServicesServlet is provided in the WebSphere V5.0.2 Web services runtime.

The InventoryRequest and InventoryReply classes were also generated from from the WSDL file. These classes provide application access to the XML request and reply documents. We developed the InventoryRequestBuilder and InventoryReplyBuilder classes by hand as constructor helpers for request and reply messages.

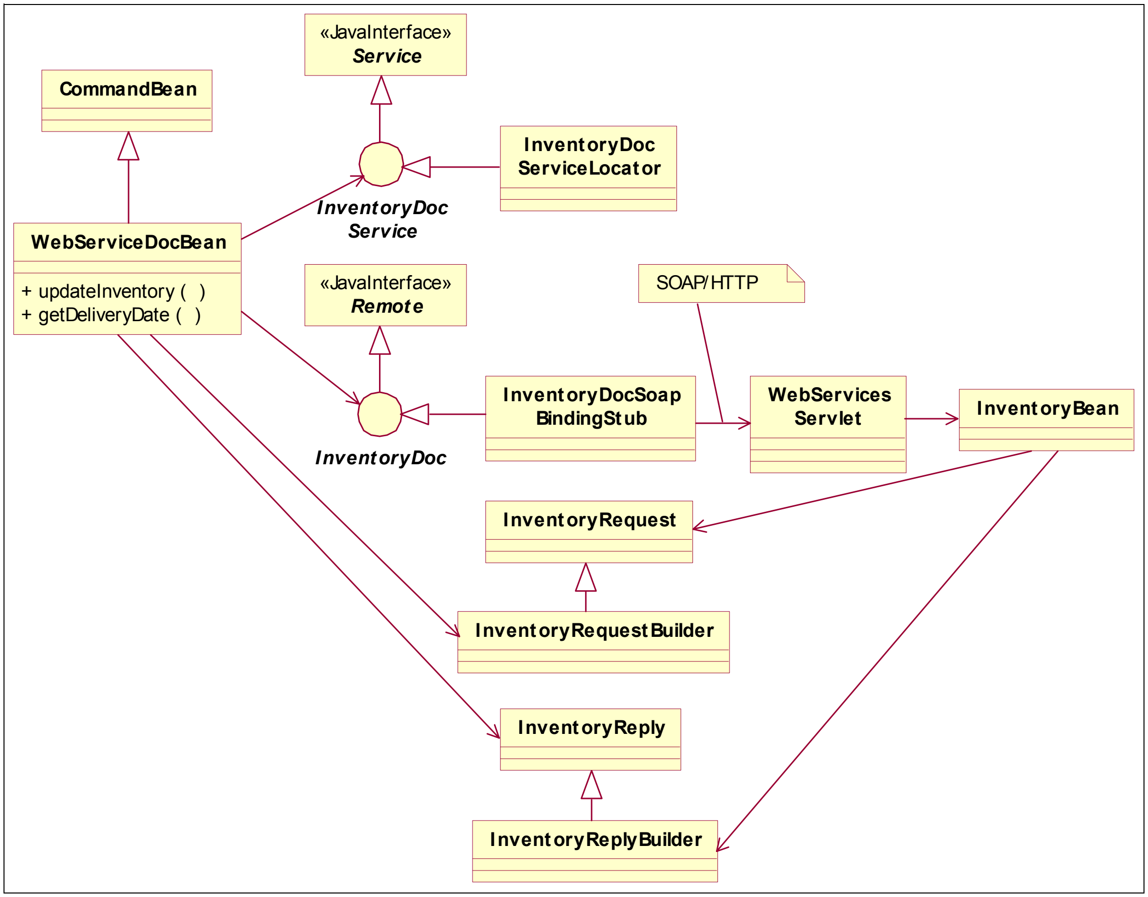


Figure 9-2 Class diagram of the document style Web service requester and provider (source and target)

Interaction diagrams

The interaction diagrams in Figure 9-3 and Figure 9-4 show the sequence of steps performed when the source application uses the WebServiceDocBean to perform the getDeliveryDate() method. In Figure 9-3:

1. WebServiceDocBean in the source application creates an InitialContext and uses it to perform a JNDI lookup to locate the InventoryDocService. This lookup finds the value set in the webservicesclient.xml deployment descriptor.
2. WebServiceDocBean invokes the getInventoryDoc method of the InventoryDocService interface (implemented by InventoryDocServiceLocator) to get the target service endpoint interface, InventoryDoc.
3. The WebServiceDocBean uses the InventoryRequestBuilder to create an InventoryRequest message.
4. WebServiceDocBean calls the getDeliveryDate method of the InventoryDoc interface (implemented by InventoryDocSoapBindingStub) to invoke the Web service. WebServiceDocBean passes the InventoryRequest message as a parameter of getDeliveryDate.

   The stub implementing the InventoryDoc interface invokes the remote Web service via SOAP and HTTP using the WebSphere Web services runtime.

   The stub returns an InventoryReply message to WebServiceDocBean.
5. The WebServiceDocBean calls the getBody method of the returned InventoryReply object. It can then get the delivery date from the message Body.

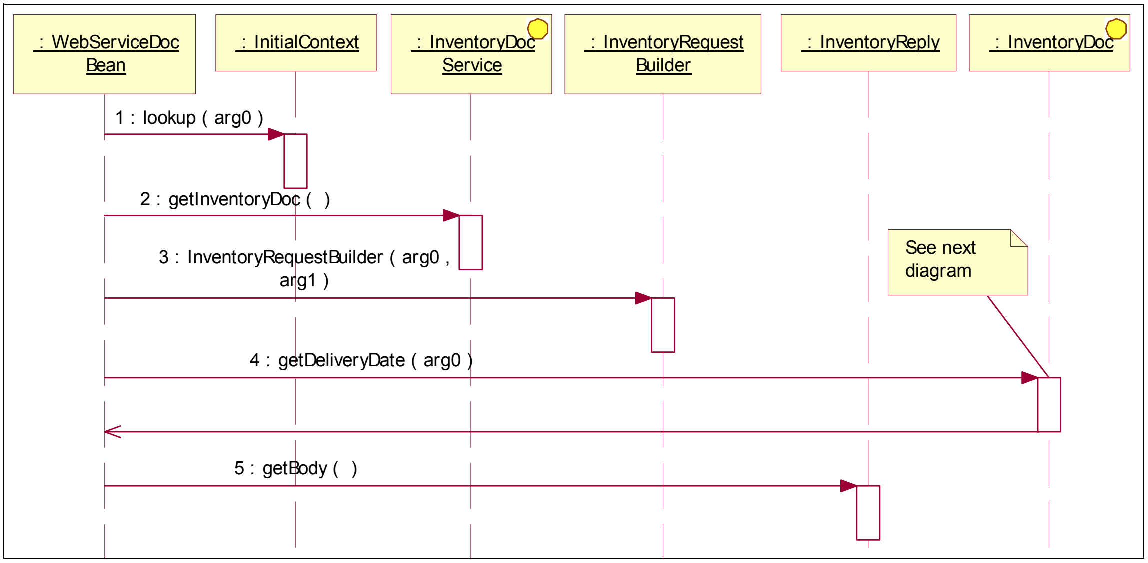


Figure 9-3 Interaction diagram for document style Web services: Part 1

In Figure 9-4:

1. The stub implementing the InventoryDoc interface (InventoryDocSoapBindingStub) invokes the remote Web service via SOAP and HTTP using the WebSphere Web services runtime.
2. The WebServicesServlet in the target application server invokes the getDeliveryDate method on the InventoryBean, which is the stateless session bean on the target application.
3. The InventoryBean calls the getBody method of the passed InventoryRequest message. It can then get the part number from the message Body.
4. The InventoryBean determines the delivery date for the part number and uses the InventoryReplyBuilder to create an InventoryReply message.

The InventoryReply message then gets passed back through the chain to the requester side and to the WebServiceDocBean that made the request to begin with.

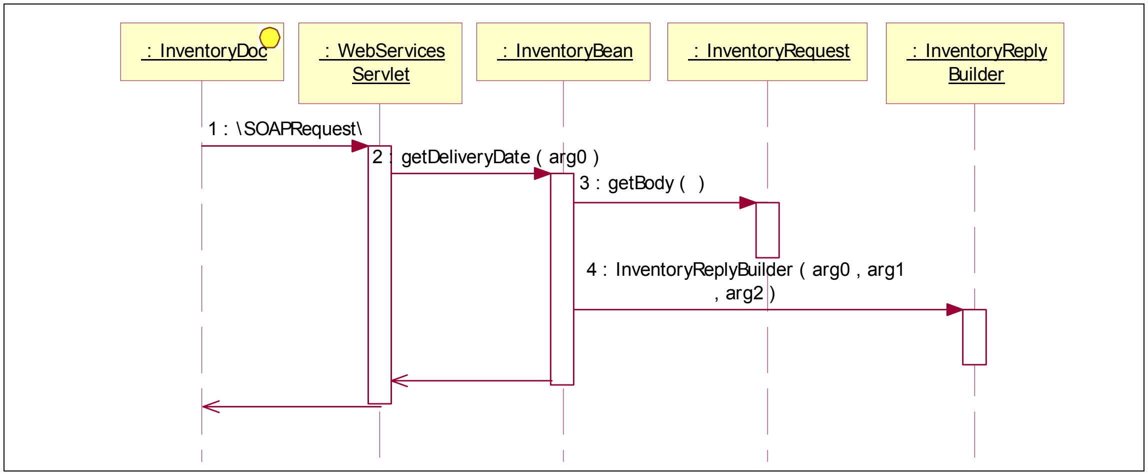


Figure 9-4 Interaction diagram for document style Web services - part 2

These interaction diagrams cover our implementation of the Get Delivery Date use case using document style Web services. Although we have not provided interaction diagrams covering the Update Inventory use case, this use case has been implemented as a one-way document style Web service. Its interaction diagrams are similar, except the InventoryReply message is not used. Also, the one-way invocation of the Web service does not block the client application. Execution control returns back to the WebServiceDocBean, without having to wait for the updateInventory call to the InventoryBean to complete.

See also Figure 8-7 on page 159 for our interaction diagram for one-way RPC style Web services.

9.4 Development guidelines

In this section we describe:

- How to enable an EJB target application as a document style Web service provider.
- How to enable a Web source application as a document style Web service client.

We are using the Web Services for J2EE support provided with WebSphere V5.0.2 to implement our solution, which includes command line tools to generate the Web services classes and deployment descriptors. IBM WebSphere Studio Application Developer V5.1 also provides full support for the creation of Web services for J2EE.

9.4.1 Web service enabling the target application

See Figure 8-9 on page 161 for an overview of the Web service development process for a target application. Let's walk through this process for our target application.

We have already created a simple EJB application using WebSphere Studio Application Developer. We can create a document style Web service using this EJB as follows:

1. Create a service endpoint interface for the Web service:
   a. Using WebSphere Studio, switch to the J2EE Perspective and click the **Project Navigator** tab.
   b. Navigate to the **ITSOTargetAppEJB** → **ejbModule** folder.
   c. Create the service endpoint interface by copying the Inventory EJB Remote Interface, **Inventory.java**, from the com.ibm.itso.ejb.inventory package to **InventoryDoc.java** in the com.ibm.itso.ws.inventory package.
   d. Edit the new InventoryDoc.java file so that it extends java.rmi.Remote, as shown in Example 9-1.

Example 9-1 InventoryDoc service endpoint interface

```
package com.ibm.itso.ws.inventory;

public interface InventoryDoc extends java.rmi.Remote {
   public void updateInventory(String partNo)
```

```
      throws java.rmi.RemoteException;
   public String getDeliveryDate(String partNo)
      throws java.rmi.RemoteException;
}
```

2. Open a command window.
3. Generate a Web Services Description Language (WSDL) file from the service endpoint interface. We used the command shown in Example 9-2.

   Note the **-style document** option. It specifies that we want to generate WSDL for a document style Web service.

Example 9-2 Generating WSDL using Java2WSDL

```
C:\WebSphere\AppServer\bin\Java2WSDL -verbose
   -implClass com.ibm.itso.ejb.inventory.InventoryBean
   -location http://target.itso.ral.ibm.com:9080/ITSOTargetApp/services/InventoryDoc
   -output C:\workspace\ITSOTargetAppEJB\ejbModule\META-INF\wsdl\InventoryDoc.wsdl
   -style document -use literal -voidReturn ONEWAY com.ibm.itso.ws.inventory.InventoryDoc

WSWS3010I: Info: Generating portType {http://inventory.ws.itso.ibm.com}InventoryDoc
WSWS3010I: Info: Generating message {http://inventory.ws.itso.ibm.com}getDeliveryDateRequest
WSWS3010I: Info: Generating message {http://inventory.ws.itso.ibm.com}getDeliveryDateResponse
WSWS3010I: Info: Generating message {http://inventory.ws.itso.ibm.com}updateInventoryRequest
WSWS3010I: Info: Generating message {http://inventory.ws.itso.ibm.com}updateInventoryResponse
WSWS3010I: Info: Generating binding {http://inventory.ws.itso.ibm.com}InventoryDocSoapBinding
WSWS3010I: Info: Generating service {http://inventory.ws.itso.ibm.com}InventoryDocService
WSWS3010I: Info: Generating port InventoryDoc
```

4. Create XML schemas defining the required messages structures. We created two XML schemas:
   - InventoryRequest.xsd as the request message for both updateInventory and getDeliveryDate
   - InventoryReply.xsd as the reply message for getDeliveryDate

   See Example 9-3 for the listing of InventoryReply.xsd. InventoryRequest.xsd is the same except it doesn't have the DeliveryDate element. We placed both schema files in the ITSOTargetAppEJB\ejbModule\META-INF\wsdl folder along with the WSDL file.

Example 9-3 InventoryReply.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://reply.inventory.ws.itso.ibm.com"
      xmlns:reply="http://reply.inventory.ws.itso.ibm.com"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="InventoryReply">
        <xsd:sequence>
```

```
            <xsd:element name="Header">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="SourceName" type="xsd:string"/>
                        <xsd:element name="Version" type="xsd:int"/>
                        <xsd:element name="CreateDate" type="xsd:dateTime"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="Body">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="PartNumber" type="xsd:string"/>
                        <xsd:element name="DeliveryDate" type="xsd:date"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

5. Edit the generated WSDL file so it imports the required XML schemas into the WSDL types element.

   Example 9-4 shows the WSDL types definitions in the generated WSDL file *before* editing.

Example 9-4 Generated WSDL types from InventoryDoc.wsdl

```
...
<wsdl:types>
 <schema elementFormDefault="qualified"
   targetNamespace="http://inventory.ws.itso.ibm.com"
   xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="getDeliveryDate">
   <complexType>
    <sequence>
     <element name="partNo" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
  </element>
  <element name="getDeliveryDateResponse">
   <complexType>
    <sequence>
     <element name="getDeliveryDateReturn" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
  </element>
  <element name="updateInventory">
```

```
   <complexType>
    <sequence>
     <element name="partNo" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
  </element>
 </schema>
</wsdl:types>
...
```

Looking at Example 9-5, you can see that we have imported the InventoryRequest.xsd and InventoryReply.xsd XML schemas. The getDeliveryDate and updateInventory elements now reference InventoryRequest rather than string. The getDeliveryDateResponse element references InventoryReply rather than string. Our XML types will now be used in the WSDL message definitions, instead of string types.

Example 9-5 Imported WSDL types from InventoryDoc.wsdl

```
...
<wsdl:types>
 <xsd:schema elementFormDefault="qualified"
   targetNamespace="http://inventory.ws.itso.ibm.com"
   xmlns:reply="http://reply.inventory.ws.itso.ibm.com"
   xmlns:request="http://request.inventory.ws.itso.ibm.com">
  <xsd:import namespace="http://reply.inventory.ws.itso.ibm.com"
   schemaLocation="InventoryReply.xsd"/>
  <xsd:import namespace="http://request.inventory.ws.itso.ibm.com"
   schemaLocation="InventoryRequest.xsd"/>
  <xsd:element name="getDeliveryDate">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="InventoryRequest" type="request:InventoryRequest"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="getDeliveryDateResponse">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="InventoryReply" type="reply:InventoryReply"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="updateInventory">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="InventoryRequest" type="request:InventoryRequest"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:schema>
</wsdl:types>
...
```

6. Delete the service endpoint interface we created previously, com.ibm.itso.ws.inventory.InventoryDoc in ITSOTargetAppEJB\ejbModule. We have changed the WSDL for our service so it is no longer valid.
7. Using the WSDL file we just created, generate the Web services deployment descriptors and classes using the **WSDL2Java** tool. We used the command shown in Example 9-6.

Example 9-6 Generating server deployment descriptors and classes using WSDL2Java

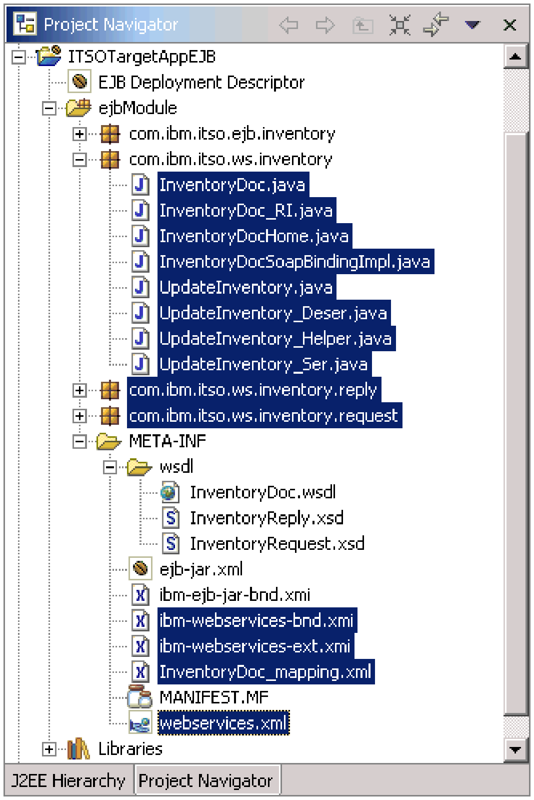
```
C:\WebSphere\AppServer\bin\WSDL2Java -verbose -role server -container ejb
   -output C:\workspace\ITSOTargetAppEJB\ejbModule
   C:\workspace\ITSOTargetAppEJB\ejbModule\META-INF\wsdl\InventoryDoc.wsdl

WSWS3185I: Info: Parsing XML file:  C:\...\wsdl\InventoryDoc.wsdl
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\InventoryDoc.java.
```

```
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\InventoryDocSoapBindingImpl.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\InventoryDoc_RI.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\InventoryDocHome.java.
WSWS3282I: Info: Generating C:\...\META-INF\webservices.xml.
WSWS3282I: Info: Generating C:\...\META-INF\ibm-webservices-bnd.xmi.
WSWS3282I: Info: Generating C:\...\META-INF\ibm-webservices-ext.xmi.
WSWS3282I: Info: Generating C:\...\META-INF\InventoryDoc_mapping.xml.
```

Note: The deployment descriptors and class files will not be regenerated when the tool is re-run, unless the existing files have been removed first.

8. Right-click the ITSOTargetAppEJB project and select **Refresh**. You should see the generated files shown in Figure 9-5.

   The webservices.xml deployment descriptor defines the set of Web services that are being deployed in the Web service-enabled J2EE container.



```
ITSOTargetAppEJB
  EJB Deployment Descriptor
  ejbModule
    com.ibm.itso.ejb.inventory
    com.ibm.itso.ws.inventory
      InventoryDoc.java
      InventoryDoc_RI.java
      InventoryDocHome.java
      InventoryDocSoapBindingImpl.java
      UpdateInventory.java
      UpdateInventory_Deser.java
      UpdateInventory_Helper.java
      UpdateInventory_Ser.java
    com.ibm.itso.ws.inventory.reply
    com.ibm.itso.ws.inventory.request
    META-INF
      wsdl
        InventoryDoc.wsdl
        InventoryReply.xsd
        InventoryRequest.xsd
      ejb-jar.xml
      ibm-ejb-jar-bnd.xmi
      ibm-webservices-bnd.xmi
      ibm-webservices-ext.xmi
      InventoryDoc_mapping.xml
      MANIFEST.MF
      webservices.xml
  Libraries
```

Figure 9-5 WSDL2Java generated files

Attention: We found an error in the InventoryDoc_mapping.xml file generated by the WSDL2Java tool that resulted in the following exception when starting the application server hosting the Web service:

```
java.io.IOException: WSWS3097E: Error: Emitter failure.  All input parts
must be listed in the parameterOrder attribute of updateInventory
```

To correct this problem, we added the following line to the mapping file to identify the updateInventory operation as wrapped:

```
...
<wsdl-operation>updateInventory</wsdl-operation>
<wrapped-element></wrapped-element>
<method-param-parts-mapping id="MethodParamPartsMapping_...">
...
```

Contact WebSphere technical support for details on P169766.

9. Navigate to ITSOTargetAppEJB → ejbModule → META-INF and edit webservices.xml. Set the ejb-link element to Inventory, as shown in Example 9-7.

   The ejb-link element corresponds to the ejb-name element of the required EJB, as defined in ejb-jar.xml.

Example 9-7 Updating webservices.xml

```
...
<service-impl-bean>
   <ejb-link>Inventory</ejb-link>
</service-impl-bean>
...
```

Next we need to add methods to the EJB in our target application that will process the document style Web service requests.

10. Open the generated service endpoint interface, **com.ibm.itso.ws.inventory.InventoryDoc** in the ITSOTargetAppEJB\ejbModule folder.

    Notice that the input parameter of the getDeliveryDate and updateInventory methods is now type InventoryRequest, and getDeliveryDate returns type InventoryReply, as shown in Example 9-8.

    The WSDL2Java tool has generated the InventoryRequest and InventoryReply classes so our application can access the XML request and reply documents.

Example 9-8 Generated InventoryDoc.java

```
package com.ibm.itso.ws.inventory;

import com.ibm.itso.ws.inventory.reply.InventoryReply;
import com.ibm.itso.ws.inventory.request.InventoryRequest;

public interface InventoryDoc extends java.rmi.Remote {
   public InventoryReply getDeliveryDate(InventoryRequest inventoryRequest)
      throws java.rmi.RemoteException;
   public void updateInventory(InventoryRequest inventoryRequest)
      throws java.rmi.RemoteException;
}
```

We also developed two additional classes, InventoryRequestBuilder and InventoryReplyBuilder, as constructor helpers for request and reply messages. See Example 9-9 for the source listing for InventoryRequestBuilder. These classes provide a reusable and streamlined interface for creating InventoryRequest and InventoryReply messages.

Example 9-9 InventoryReplyBuilder.java

```
package com.ibm.itso.ws.inventory;

import com.ibm.itso.ws.inventory.reply.InventoryReply;
import java.util.Calendar;
import java.util.Date;

public class InventoryReplyBuilder extends InventoryReply {

   // message version number for Version element
   private final int VERSION = 1;

   public InventoryReplyBuilder(
      String msgSource, String msgPartNumber, Date msgDeliveryDate) {

      super();
      com.ibm.itso.ws.inventory.reply.Header header =
         new com.ibm.itso.ws.inventory.reply.Header();
      header.setSourceName(msgSource);
      header.setVersion(VERSION);
      // set create date to the current date
      header.setCreateDate(Calendar.getInstance());
      super.setHeader(header);
      com.ibm.itso.ws.inventory.reply.Body body =
         new com.ibm.itso.ws.inventory.reply.Body();
      body.setPartNumber(msgPartNumber);
      body.setDeliveryDate(msgDeliveryDate);
      super.setBody(body);
   }
}
```

11. Add the new getDeliveryDate and updateInventory methods from com.ibm.itso.ws.inventory.InventoryDoc to our target application EJB, **com.ibm.itso.ejb.inventory.InventoryBean**. See Example 9-10 for the source listing for getDeliveryDate.

Example 9-10 Document style Web service server code for getDeliveryDate

```
public InventoryReply getDeliveryDate(InventoryRequest reqMsg) {
   InventoryReply repMsg = null;

   String partNo = (reqMsg.getBody()).getPartNumber();

   Date deliveryDate = getDeliveryDateObject(partNo);
   repMsg = new InventoryReplyBuilder(APP_NAME, partNo, deliveryDate);
   return repMsg;
}
```

12. Promote the new EJB methods to the EJB Remote interface and regenerate the EJB deployment code.
13. Export the ITSOTargetApp project to an EAR file, then run the **endptEnabler** command line tool to add an HTTP router module to the EAR file.

    You can use the same procedure we used for our RPC style Web service. See step 7 on page 165 to export the EAR file and step 8 on page 165 to run the **endptEnabler** tool.
14. The HTTP router module also needs access to the XML schemas used, so add InventoryReply.xsd and InventoryRequest.xsd to the WEB-INF/wsdl folder in the ITSOTargetAppWeb module. You can do this using WebSphere Studio or the WebSphere Application Assembly Tool.

The EAR file is now ready to deploy in the IBM WebSphere Application Server V5.0.2 runtime.

Note: The WebSphere V5.0.2 Web service deployment tools will not append new Web services to existing Web services deployment descriptors. If you need to deploy more than one Web service in a module, you will need to manually merge the Web service deployment descriptors.

9.4.2 Web service enabling the source application

See Figure 8-11 on page 167 for an overview of the Web service development process for a source application. Let's walk through this process for our source application.

Web service-enabling our source application is simply a matter of obtaining the WSDL and XML schema files for the target Web service, and running the **`WSDL2Java`** tool to generate the required deployment descriptors and proxy classes:

1. Copy the target application WSDL and XML schema files to the source application Web module:
   a. Using WebSphere Studio, switch to the J2EE Perspective and click the **Project Navigator** tab.
   b. Copy the target application WSDL and XML schema files, **InventoryDoc.wsdl**, **InventoryReply.xsd**, and **InventoryRequest.xsd** from the ITSOTargetAppEJB/ejbModule/META-INF/wsdl folder to the ITSOSourceAppWeb/WebContent/WEB-INF/wsdl folder.
2. Open a command window.
3. Using the WSDL file created in step 1, generate the Web service client deployment descriptors and classes using the **WSDL2Java** tool. We used the command shown in Example 9-11.

Example 9-11 Generating client deployment descriptors and classes using WSDL2Java

```
C:\WebSphere\AppServer\bin\WSDL2Java -verbose -role client -container web
   -output C:\workspace\ITSOSourceAppWeb\WebContent
   C:\workspace\ITSOSourceAppWeb\WebContent\WEB-INF\wsdl\InventoryDoc.wsdl

WSWS3185I: Info: Parsing XML file:  C:\...\wsdl\InventoryDoc.wsdl
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body_Deser.java.
```

```
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\InventoryDocService.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\InventoryDocServiceLocator.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\InventoryDoc.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\InventoryDocSoapBindingStub.java.
WSWS3282I: Info: Generating C:\...\WEB-INF\webservicesclient.xml.
WSWS3282I: Info: Generating C:\...\WEB-INF\ibm-webservicesclient-bnd.xmi.
WSWS3282I: Info: Generating C:\...\WEB-INF\ibm-webservicesclient-ext.xmi.
WSWS3282I: Info: Generating C:\...\WEB-INF\InventoryDoc_mapping.xml.
```

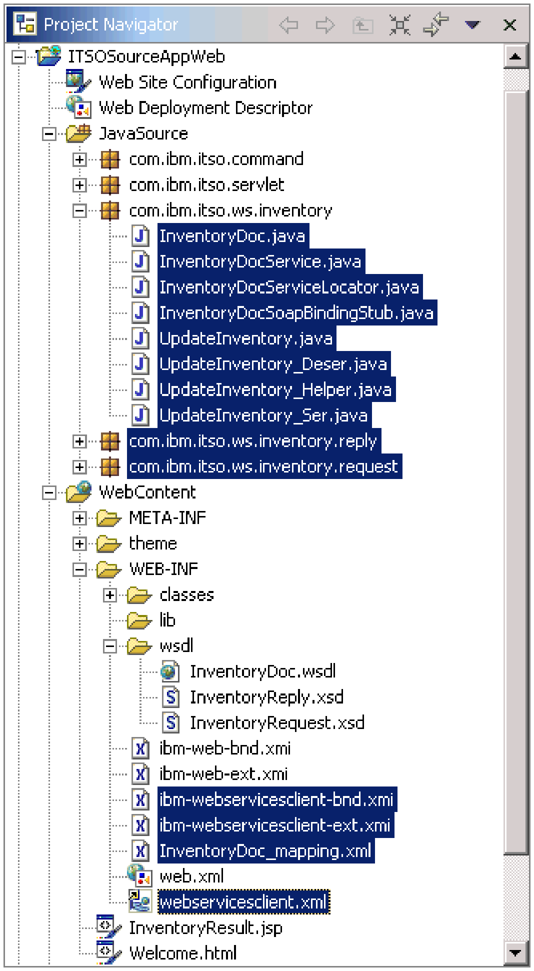
4. In WebSphere Studio, move the generated Java source files from the Web module's WebContent folder to its JavaSource folder:

   a. Right-click the **ITSOSourceAppWeb** project and select **Refresh** from the pop-up menu. The generated files should now appear in the Studio workspace.

   b. Move the **com.ibm.itso.ws.inventory** package in the ITSOSourceAppWeb\WebContent folder to the ITSOSourceAppWeb\JavaSource folder.

   The generated files are highlighted in Figure 9-6.

   The webservicesclient.xml deployment descriptor defines the JNDI name for accessing the Web service and the associated service endpoint interface to be used.



```
Project Navigator
ITSOSourceAppWeb
    Web Site Configuration
    Web Deployment Descriptor
    JavaSource
        com.ibm.itso.command
        com.ibm.itso.servlet
        com.ibm.itso.ws.inventory
            InventoryDoc.java
            InventoryDocService.java
            InventoryDocServiceLocator.java
            InventoryDocSoapBindingStub.java
            UpdateInventory.java
            UpdateInventory_Deser.java
            UpdateInventory_Helper.java
            UpdateInventory_Ser.java
        com.ibm.itso.ws.inventory.reply
        com.ibm.itso.ws.inventory.request
    WebContent
        META-INF
        theme
        WEB-INF
            classes
            lib
            wsdl
                InventoryDoc.wsdl
                InventoryReply.xsd
                InventoryRequest.xsd
            ibm-web-bnd.xmi
            ibm-web-ext.xmi
            ibm-webservicesclient-bnd.xmi
            ibm-webservicesclient-ext.xmi
            InventoryDoc_mapping.xml
            web.xml
            webservicesclient.xml
        InventoryResult.jsp
        Welcome.html
```

Figure 9-6 Generated client binding files and deployment descriptors

5. Add client application code to invoke the Web service on the target application.

   To invoke getDeliveryDate on the target application, we added the code shown in Example 9-12. We added this code to the com.ibm.itso.command.WebServiceDocBean command bean in our ITSOSourceAppWeb module.

Example 9-12 Web service client code for getDeliveryDate

```
public String getDeliveryDate(String partNumber) throws Exception {

   String deliveryDate = null;
   try {

      Context ctx = new InitialContext();
      InventoryDocService service = (InventoryDocService) ctx.lookup(
         "java:comp/env/service/InventoryDocService");

      // Request the Service Endpoint from the Service
      InventoryDoc port = service.getInventoryDoc();

      // Generate XML document for update inventory request
      InventoryRequest reqMsg =
         new InventoryRequestBuilder(APP_NAME, partNumber);

      // Get the quote
      InventoryReply repMsg = port.getDeliveryDate(reqMsg);

      // Get the delivery date from the XML reply document
      Date date = (repMsg.getBody()).getDeliveryDate();
      SimpleDateFormat dateFormatter = new SimpleDateFormat(DATE_PATTERN);
      deliveryDate = dateFormatter.format(date);

   } catch (Exception e) {
      //...
   }

   return deliveryDate;
}
```

6. Test the source and target applications in the IBM WebSphere Studio Application Developer V5.1 test environment.
7. Deploy the source and target applications in your IBM WebSphere Application Server V5.0.2 runtime environment to try the applications on separate machines.

The document style SOAP request for getDeliveryDate is shown in Example 9-13.

Example 9-13 SOAP request for getDeliveryDate

```
POST /ITSOTargetApp/services/InventoryDoc HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: IBM WebServices/1.0
Host: localhost
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
```

```
Content-Length: 649

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <getDeliveryDate xmlns="http://inventory.ws.itso.ibm.com">
            <InventoryRequest xmlns="http://request.inventory.ws.itso.ibm.com">
               <Header>
                  <SourceName>ITSOSourceApp</SourceName>
                  <Version>1</Version>
                  <CreateDate>2003-09-07T22:26:19.113Z</CreateDate>
               </Header>
               <Body>
                  <PartNumber>12345</PartNumber>
               </Body>
            </InventoryRequest>
         </getDeliveryDate>
      </soapenv:Body>
   </soapenv:Envelope>
```

The SOAP response for getDeliveryDate is shown in Example 9-14.

Example 9-14 SOAP response for getDeliveryDate

```
HTTP/1.1 200 OK
Server: WebSphere Application Server/5.0
Content-Type: text/xml; charset=utf-8
Content-Language: en-US
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <getDeliveryDateResponse xmlns="http://inventory.ws.itso.ibm.com">
            <InventoryReply xmlns="http://reply.inventory.ws.itso.ibm.com">
               <Header>
                  <SourceName>ITSOTargetApp</SourceName>
                  <Version>1</Version>
                  <CreateDate>2003-09-07T22:26:20.946Z</CreateDate>
               </Header>
               <Body>
                  <PartNumber>12345</PartNumber>
                  <DeliveryDate>2003-09-14</DeliveryDate>
               </Body>
            </InventoryReply>
         </getDeliveryDateResponse>
      </soapenv:Body>
   </soapenv:Envelope>
```

9.5 Integration with .NET-based Web services

In this section we take a brief look at interoperability between WebSphere V5.0.2 and Microsoft .NET Web services, which use the wrapped document style. Figure 9-7 shows a Runtime pattern and Product mapping providing connectivity between IBM WebSphere Application Server V5.0.2 and Microsoft .NET using the Direct Connection pattern.

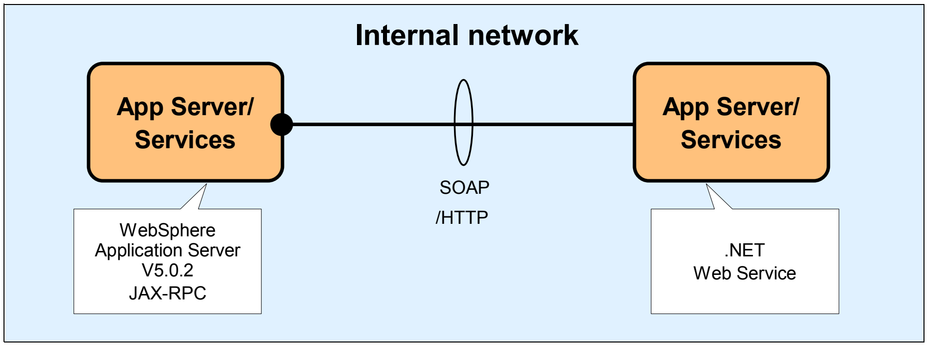


Figure 9-7 Direct Connection::Call Connection: Web services to .NET Product mapping

You can find a number of .NET Web service providers on the Internet, with a search on "web service directory" in popular search engines. One of the sites, WebserviceX.NET provides a number of .NET Web services for general consumption. You can access the WebserviceX.NET Web site at:

http://www.webservicex.net/

To demonstrate interoperability with .NET Web services, we downloaded the WSDL for the WebserviceX.NET SendEmail service from:

http://www.webservicex.net/SendEmail.asmx?WSDL

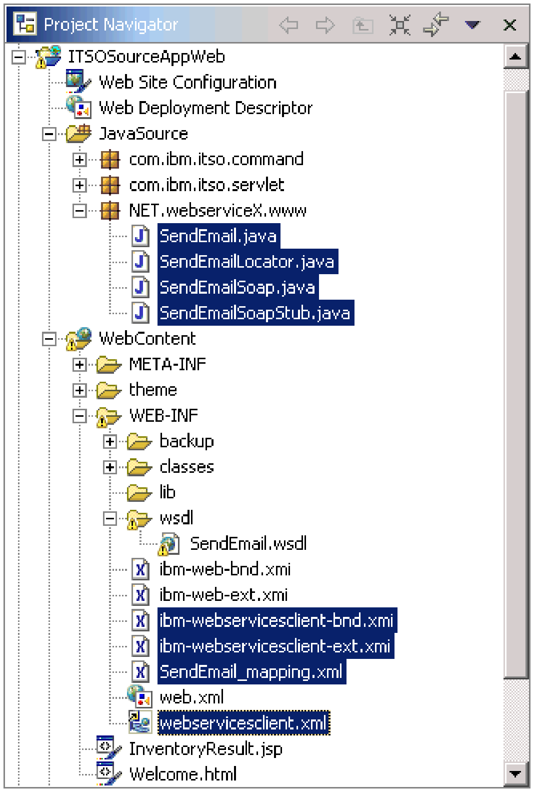
Using this WSDL file, we generated the Web service client deployment descriptors and classes using the WebSphere V5.0.2 WSDL2Java tool. We used the command shown in Example 9-15.

Example 9-15 Generating client deployment descriptors and classes using WSDL2Java

```
C:\WebSphere\AppServer\bin\WSDL2Java -verbose -role client -container web
   -output C:\workspace\ITSOSourceAppWeb\WebContent
   C:\workspace\ITSOSourceAppWeb\WebContent\WEB-INF\wsdl\SendEmail.wsdl

WSWS3185I: Info: Parsing XML file:  C:\...\WEB-INF\wsdl\SendEmail.wsdl
WSWS3282I: Info: Generating C:\...\NET\webserviceX\www\SendEmailSoap.java.
WSWS3282I: Info: Generating C:\...\NET\webserviceX\www\SendEmailSoapStub.java.
WSWS3282I: Info: Generating C:\...\NET\webserviceX\www\SendEmail.java.
WSWS3282I: Info: Generating C:\...\NET\webserviceX\www\SendEmailLocator.java.
WSWS3282I: Info: Generating C:\...\WEB-INF\webservicesclient.xml.
WSWS3282I: Info: Generating C:\...\WEB-INF\ibm-webservicesclient-bnd.xmi.
WSWS3282I: Info: Generating C:\...\WEB-INF\ibm-webservicesclient-ext.xmi.
WSWS3282I: Info: Generating C:\...\WEB-INF\SendEmail_mapping.xml.
```

We then refreshed the ITSOSourceAppWeb project in WebSphere Studio and moved the generated **NET.webserviceX.www** package in the ITSOSourceAppWeb\WebContent folder to the ITSOSourceAppWeb\JavaSource folder.

The generated files are highlighted in Figure 9-8.



```
ITSOSourceAppWeb
   Web Site Configuration
   Web Deployment Descriptor
   JavaSource
      com.ibm.itso.command
      com.ibm.itso.servlet
      NET.webserviceX.www
         SendEmail.java
         SendEmailLocator.java
         SendEmailSoap.java
         SendEmailSoapStub.java
   WebContent
      META-INF
      theme
      WEB-INF
         backup
         classes
         lib
         wsdl
            SendEmail.wsdl
         ibm-web-bnd.xmi
         ibm-web-ext.xmi
         ibm-webservicesclient-bnd.xmi
         ibm-webservicesclient-ext.xmi
         SendEmail_mapping.xml
         web.xml
         webservicesclient.xml
      InventoryResult.jsp
      Welcome.html
```

Figure 9-8 Generated client binding files and deployment descriptors

Next we added client application code to invoke the Web service on the target application. To create an e-mail message containing the part number for the Update Inventory use case, we added the code shown in Example 9-16. We added this code to the com.ibm.itso.command.WebServiceNetBean command bean in our ITSOSourceAppWeb module.

Example 9-16 Web service client code for updateInventory using .NET service

```
public void updateInventory(String partNumber) throws Exception {

   try {

      // Send the order
      Context ctx = new InitialContext();
      String emailSender =
         (String) ctx.lookup("java:comp/env/EmailSender");
      String emailReceiver =
         (String) ctx.lookup("java:comp/env/EmailReceiver");
      SendEmail service =
         (SendEmail) ctx.lookup("java:comp/env/service/SendEmail");

      // Request the Port Object from it
      SendEmailSoap endPoint = service.getSendEmailSoap();

      endPoint.sendEmails(emailSender, emailReceiver, "ITSO Electronics
order",
         "Please order: " + partNumber);

   } catch (Exception e) {
      //...
   }

   return;
}
```

We tested the source application in the IBM WebSphere Studio Application Developer V5.1 test environment and in the IBM WebSphere Application Server V5.0.2 runtime environment. The .NET SOAP request for updateInventory is shown in Example 9-17.

Example 9-17 SOAP request for updateInventory

```
POST /SendEmail.asmx HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: IBM WebServices/1.0
Host: www.webservicex.net
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "http://www.webserviceX.NET/SendEmails"
Content-Length: 556

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <SendEmails xmlns="http://www.webserviceX.NET">
            <ToEmailAddress>aaa@bbb.ccc</ToEmailAddress>
            <FromEmailAddress>aaa@bbb.ccc</FromEmailAddress>
            <Subject>ITSO Electronics order</Subject>
            <Body>Please order: 12345</Body>
         </SendEmails>
      </soapenv:Body>
   </soapenv:Envelope>
```

The .NET SOAP response for updateInventory is shown in Example 9-18.

Example 9-18 SOAP response for updateInventory

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Mon, 08 Sep 2003 01:22:16 GMT
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 389

<?xml version="1.0" encoding="utf-8"?>
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <soap:Body>
         <SendEmailsResponse xmlns="http://www.webserviceX.NET">
            <SendEmailsResult>Your Message send successfully</SendEmailsResult>
         </SendEmailsResponse>
      </soap:Body>
   </soap:Envelope>
```

Example 9-19 shows the delivered e-mail.

Example 9-19 The e-mail delivered by the SendEmail service

```
From:    aaa@bbb.ccc
To:      aaa@bbb.ccc
Subject: ITSO Electronics order
Date:    Sun, 7 Sep 2003 21:22:15 -0400

Please order: 12345
```

Unfortunately, we can’t guarantee that your WebSphere to .NET Web services integration experience will be this smooth!

9.6 Quality of Service capabilities

In this section we discuss Quality of Service capabilities and considerations specific to document style Web services. For further discussion on Quality of Services for Web services in general, see 8.6, “Quality of Service capabilities” on page 177.

9.6.1 Transactionality

As discussed in 8.6.6, “Transactionality” on page 180, standards for Web services transactionality are still evolving at the time of writing. Document style Web services can provide benefits in this area.

A complex business process may require a series of calls to finer-grained RPC style Web services. It may be appropriate to use courser-grained document style Web services instead, where the contents of an entire transaction can be passed within one XML document. The business transaction can then be performed within one service request. See also “Use document style when maintaining application state” on page 211.

9.7 Best practices

The decision on using RPC style or document style Web services should be based on the nature of underlying business process to be implemented. Since document style involves extra development effort, it should be considered only when RPC style imposes serious restrictions or the benefits of document style justify the extra effort. In order to reach a decision, the following questions might be helpful:

- Is the service accessed through an interface based on procedure calls, and unlikely to be affected by future API changes?
- Does the service require a request/response architecture?
- Does the service exceed organization boundaries?
- Is there any state maintenance required?
- Is one of the parameters passed to the target application an XML document?
- Do the parameters represent complex structures that may benefit from an XML document schema for validation?

Use document style to ease validation and use of documents

In cases where Web services need to pass or return XML documents (for example, a high-level business document) or objects with a complex structure, document style may be more appropriate. Passing the XML document as a string parameter in an RPC call postpones the validation of the XML document to the target application. This may cause valid calls with invalid parameters. Usage of document style allows for publishing an XML document schema and validation against that schema prior to calling the service. In the case of a high-level business document, the document’s XML schema may be used to enforce high-level business rules.

When dealing with complex structures, a remote procedure service may have to deal with custom marshaling code while the application is still responsible for meticulously validating each element of the structure. If document messaging is used, then the application programmer can delegate validation to the document designer using an XML schema, and no custom marshaling code is required.

Use document style when maintaining application state

Consider document style Web services when a particular sequence of multiple procedure calls is involved to accomplish a complex business service. If a service consists of multiple procedure calls, then the service normally is not stateless and must maintain application state. Maintaining state between successive Web service requests may be difficult since most client platforms do not generate client stubs that support state information. In order to maintain state information beyond the scope of the request, additional logic has to be implemented in the source and target applications. This erodes the clear separation between Web service requester and Web service provider.

Alternatively, when a document style Web service is used, the contents of an entire transaction may be passed within one XML document. In this case the service is responsible for ensuring the proper sequence of procedure calls. State information only has to be maintained within the scope of this transaction.

It is also important to balance this approach against the need to keep the business objects to a reasonable size.

Use document style when changes to target API are likely

The rules of good design dictate that the method signature of an RPC style Web service interface should never change. In using document style there are less rigid rules concerning enhancements and changes to the XML schema without compromising the calling application. Therefore, document style provides more flexibility if there needs to be future modification to the target application API.

Use document style when late binding is required

RPC style Web services require rigid type specification agreed upon at service implementation. Document style Web services allow for run-time type specification of the object to be passed via the call. This allows for variation of the passed business object type depending on the context of the actual request. Validation of a passed document against an XML schema may be done either before or during the execution of a service request.

Use document style to publish external services

If a service implemented by an enterprise application is being published outside of the organization, the publisher has very little control over who is relying on the service. Since the consequences of any changes to the Web service interface (that means changes to the WSDL file) are not traceable outside of the organization, it may be better to use document style and support a common exchange protocol such as ebXML. See the next section for further details.

9.8 Overview of ebXML

Discussion of document transfer automatically leads to the question of a universal standard for transmitted documents. The most popular standard up to now is the EDI standard. The disadvantage of EDI is that it is not an XML document and therefore in a Web services scenario a special treatment is needed. ebXML aims to address this problem.

ebXML stand for Electronic Business using XML. It provides a modular suite of specifications that enables enterprises to conduct business over the Internet. Using ebXML, companies now have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes.

It is a joint development effort between OASIS (Organization for the Advancement of Structured Information Standards) and the UN/CEFACT (United Nations Centre for Trade Facilitation and Electronic Business). OASIS (formerly known as SGML group) has brought XML expertise while UN/CEFACT, who was the main sponsor of Electronic Data Transmission (EDI) has brought business expertise.

For further information see the ebXML Web site at:

http://www.ebxml.org/

Part 3

Application Integration scenarios

Part 3 provides detailed design, development, and runtime guidelines for intra-enterprise integration solutions. It teaches you by example, using IBM WebSphere Application Server V5.0 with Web services, J2EE Connectors, and JMS.

Included in Part 3 are the following chapters:

10

Using the Web Services Gateway

This chapter discusses using the Web Services Gateway in an intra-enterprise integration scenario. We are using the Web Services Gateway packaged with IBM WebSphere Application Server Network Deployment V5.0.2 in this scenario.

This chapter describes the following:

- Using the Web Services Gateway in the context of our ITSO Electronics business scenario.
- An overview of the Web Services Gateway.
- The high-level design applied to our gateway scenario.
- Development guidelines for integrating a Web service provider and requester using the Web Services Gateway.
- Quality of Service capabilities for the Web Services Gateway.

10.1 Business scenario

In this scenario, we focus on the intra-enterprise interaction between the Retail and Wholesale departments. Introducing the IBM Web Services Gateway provides ITSO Electronics with a number of advantages:

- Central access point for all services inside the enterprise

  The gateway provides a single, well-known point to access Web services within the enterprise.

- Decoupling the deployment of Web services from clients

  The gateway isolates any changes in the deployment of services within the enterprise from consumers of the services. The location of services also becomes transparent to clients of the service.

- Central security control point

  Access control can be applied to Web services so only authorized clients are allowed to access services.

- Protocol conversion between Web service requesters and providers

  Access to the services of applications that use protocols other then HTTP is planned for the near future. Therefore, access to the Web services has to be open for different protocols.

We also look at using the Web Services Gateway in an inter-enterprise integration scenario in Chapter 14, “Using inter-enterprise Web services” on page 299.

10.2 IBM Web Services Gateway

The IBM Web Services Gateway is a runtime component that provides configurable mapping between Web service providers and requesters. Services defined with WSDL can be mapped to available transport channels. The Web Services Gateway is included with IBM WebSphere Application Server Network Deployment V5.0.2.

The basic gateway components are:

- Channels that define the entry-points into the gateway and carry the Web service request and response through the gateway.
- Filters that are used to intercept service invocations which come into the gateway and act upon the services.
- Services that are described with the help of a Web Services Description Language (WSDL) document.

- UDDI references to manage the publishing of an exposed Web service to a private or public UDDI registry.

Figure 10-1 shows the relationship between the first three components. The entry point to the gateway is defined by a channel. A channel is a piece of software that defines the protocol you can use to access the gateway. The incoming message is assessed on arrival through the channel to determine which service is required. Each service (defined in a WSDL document) 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. The WSDL service definition specifies the provider service interface and implementation used to access the target service.

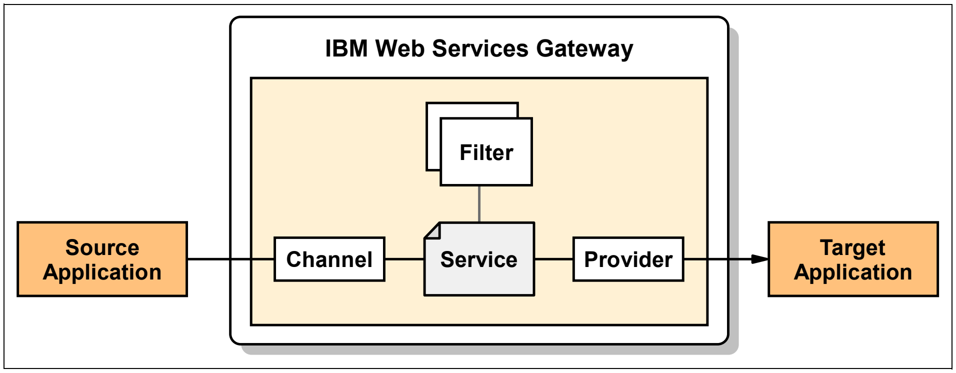


Figure 10-1 IBM Web Services Gateway

A request to the Web Services Gateway arrives through a channel and is translated into an internal representation of the service. With the help of filters for the request, a request can be logged, intercepted, or generally manipulated. After filtering the request, an appropriate provider is used to communicate with the target service. The provider in the gateway acts as a client for the target Web service.

The response from the target service flows along the exact same path back to the provider. There is no extra channel for an immediate response. In this sense the layout of the gateway is asymmetric. However, one or more response filters can be deployed independently of the request filters.

The process of deploying a target service into a 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 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 being used. Having obtained the implementation definition, the client can then access the WSDL interface definition produced by gateway, which provides full information about the target service (as presented externally by the gateway).

The Web Services Gateway uses the Web Services Invocation Framework (WSIF) API from Apache to decouple invocation from deployment within the gateway. Over time, the location of the Web service target application and the bindings may change, but these details are handled by the gateway. The Web Services Gateway separates the actual implementation of a service from how it is accessed by another service for:

- Inbound requests: To Web services created and deployed within the organization.
- Outbound requests: To Web services created and deployed outside the organization.
- Process abstraction: The service invocation approach must be flexible enough to cope with events such as switching frequently between external providers of a similar service without requiring changes to the application.
- Flexibility: As a service provider, you need the flexibility to change your deployment infrastructure without notifying all the service requestors. For example, consider a Web service deployed in a machine that later fails during operation. There needs to be a process to route the invocations to an alternate service in your infrastructure.

WSIF is used within Web Services Gateway as shown in Figure 10-2. It demonstrates the WSIF transformation from a SOAP message to a target service:

1. The SOAP message arrives at the gateway and the channel listener accepts the message.
2. The channel converts the SOAP message into a WSIF message format.
3. Elements within the message are used to locate the appropriate target service, which is bound to the channel within the gateway.
4. The target WSDL associated with the gateway service is then processed by WSIF.
5. WSIF dynamically generates a Java proxy class.
6. The target Web service is called.

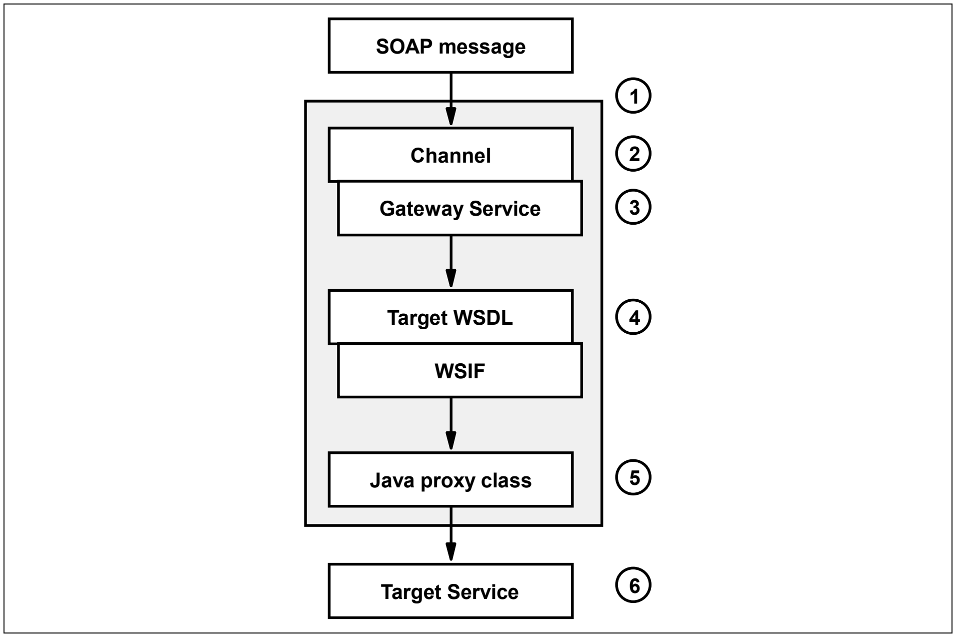


Figure 10-2 WSIF transformation

Refer to the following IBM developerWorks articles for further details:

- *Applying the Web services invocation framework*

  http://www.ibm.com/developerworks/webservices/library/ws-appwsif.html
- *An introduction to Web Services Gateway*

  http://www.ibm.com/developerworks/webservices/library/ws-gateway/
- *Business process integration with IBM CrossWorlds, Part 3: Automatically externalize Web services with WebSphere Business Connection*

  http://www.ibm.com/developerworks/ibm/library/i-cross3

10.3 Design guidelines

Figure 10-3 shows the Runtime pattern and Product mapping used in this scenario. It is based on IBM WebSphere Application Server V5.0.2 and the Direct Connection application pattern. It also includes the Web Services Gateway packaged with IBM WebSphere Application Server Network Deployment V5.0.2.

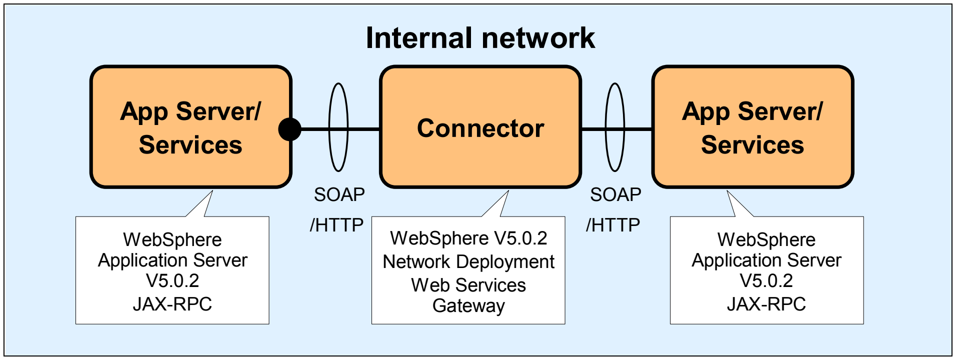

Internal network
App Server/ Services
Connector
App Server/ Services
SOAP /HTTP
SOAP /HTTP
WebSphere Application Server V5.0.2 JAX-RPC
WebSphere V5.0.2 Network Deployment Web Services Gateway
WebSphere Application Server V5.0.2 JAX-RPC

Figure 10-3 Direct Connection::Call Connection: Web Services Gateway Product mapping 1

This pattern models the connector between the source application node on the left and the gateway node in the middle, and the connector between the gateway and target application node on the right, as simple lines. Both connectors could also be modeled using coupling adapter connectors if further detail is need, similar to the basic Web services product mapping shown in Figure 9-1 on page 185.

10.4 Development guidelines

In this section, we start with a brief look at installing and configuring the Web Services Gateway. After that we walk through the process of implementing a gateway solution, based on three simple steps:

- Deploying the Web Services Gateway service
- Exporting the WSDL service implementation file
- Creating the Web service client

10.4.1 Installing and configuring the Web Services Gateway

Web Services Gateway is essentially a J2EE application installed in the WebSphere Application Server runtime. For this scenario, we installed the gateway on a stand-alone IBM WebSphere Application Server base V5.0.2 server. Details for completing the installation can be found in the WebSphere InfoCenter article, *Installing the gateway into a stand-alone application server* at:

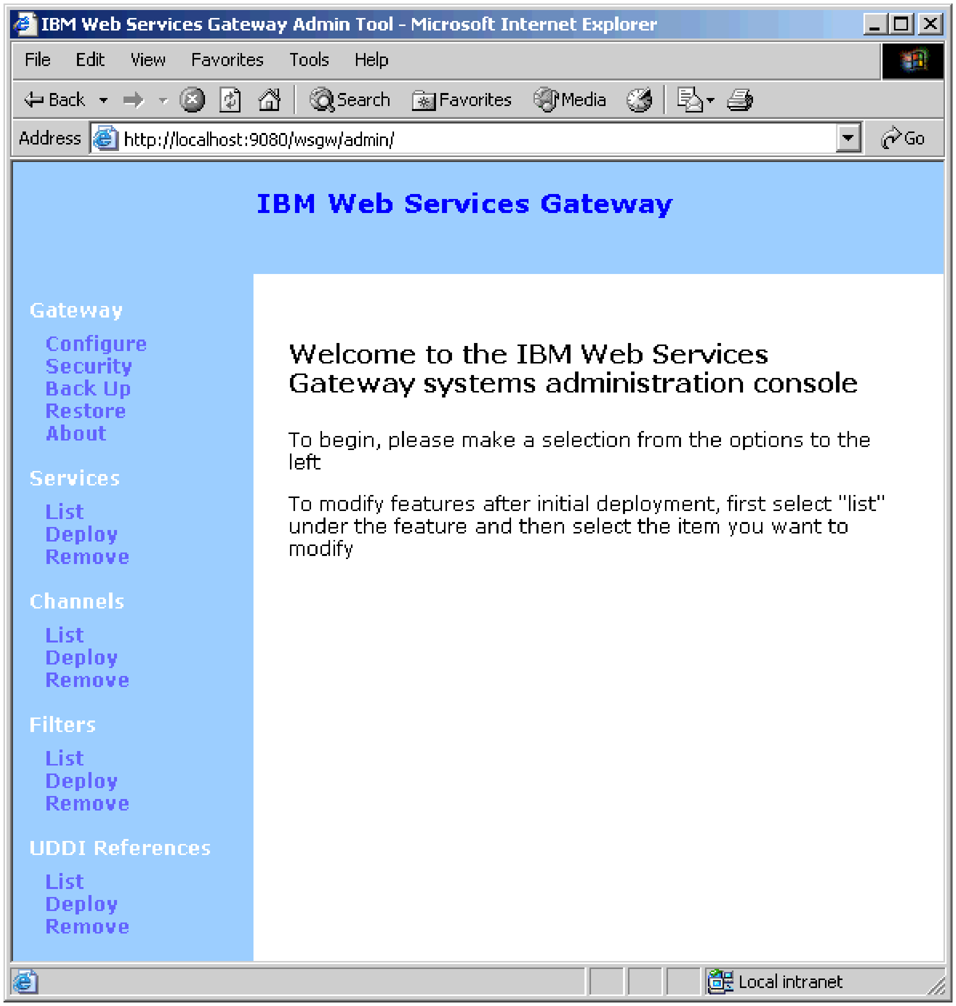
`http://www.ibm.com/software/webservers/appserv/infocenter.html`

Configuring the gateway

After the gateway has been installed and started it must be configured. To configure the Web Services Gateway:

1. Open the IBM Web Services Gateway systems administration console, shown in Figure 10-4. For the default installation on server1, the URL for accessing the console will be:

   ```
   http://<hostname>:9080/wsgw/admin/
   ```



IBM Web Services Gateway Admin Tool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/wsgw/admin/
Go
Local intranet

IBM Web Services Gateway

Gateway
- Configure
- Security
- Back Up
- Restore
- About

Services
- List
- Deploy
- Remove

Channels
- List
- Deploy
- Remove

Filters
- List
- Deploy
- Remove

UDDI References
- List
- Deploy
- Remove

Welcome to the IBM Web Services Gateway systems administration console

To begin, please make a selection from the options to the left

To modify features after initial deployment, first select "list" under the feature and then select the item you want to modify

Figure 10-4 IBM Web Services Gateway systems administration console

2. Click **Gateway** → **Configure** in the navigation panel on the left to configure the gateway.

3. In the Configure Gateway window, set the following properties:

   – Namespace URI for services

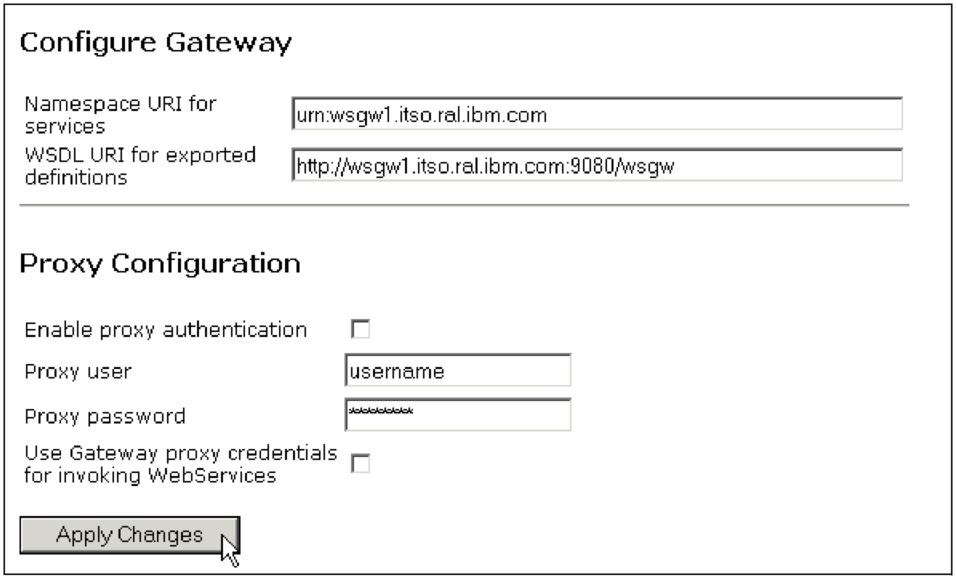
     The gateway namespace URI will appear in WSDL files exported from the gateway. Keep in mind that Java clients generated from the WSDL need to convert the gateway namespace to a Java package.

     > **Note:** Take care when specifying the namespace URI for services. If you need to change the namespace URI, you will need to redeploy all of your deployed services.

   – WSDL URI for exported definitions

     This is the URI that Web service clients will use to access the WSDL file and the exposed Web service.

   Our gateway configuration settings are shown in Figure 10-5.



Configure Gateway

Namespace URI for services: urn:wsgw1.itso.ral.ibm.com

WSDL URI for exported definitions: http://wsgw1.itso.ral.ibm.com:9080/wsgw

Proxy Configuration

Enable proxy authentication □

Proxy user: username

Proxy password: ********

Use Gateway proxy credentials for invoking WebServices □

Figure 10-5 Configuring the Web Services Gateway

4. Click **Apply Changes** to configure the gateway.

Deploying a channel

There are two types of channels provided with the Web Services Gateway:

- Apache SOAP Channel
- SOAP/HTTP Channel

Both channel types support SOAP 1.1 compatible Web services that use the RPC SOAP messaging style. The SOAP/HTTP Channel adds support for document messaging style, and for passing attachments in a MIME message.

Two versions of each channel type are supplied with the gateway, so you can set up separate channels for inbound and outbound requests. This also provides a simple way to grant different access rights to users within your organization from those outside your organization.

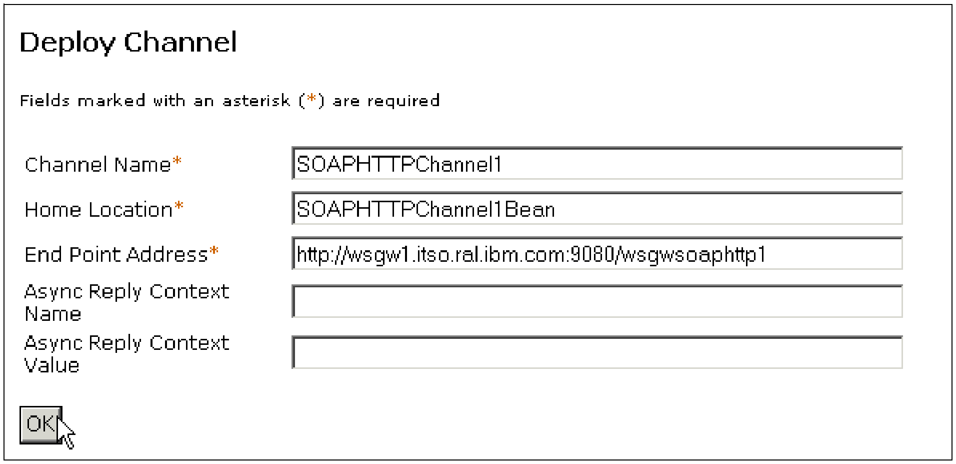
In our scenario, we are using the SOAP/HTTP Channel because we are using the document style SOAP message format.

To deploy a SOAP/HTTP Channel:

1. Click **Channels** → **Deploy** in the navigation panel on the left to deploy a channel.
2. In the Deploy Channel window, 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)

   See the InfoCenter article, *Web services gateway - Channel deployment details* if you need channel settings for other channel types, or for clustered environments.

   Our channel settings are shown in Figure 10-6.



Deploy Channel

Fields marked with an asterisk (*) are required

Channel Name* SOAPHTTPChannel1

Home Location* SOAPHTTPChannel1Bean

End Point Address* http://wsgw1.itso.ral.ibm.com:9080/wsgwsoaphttp1

Async Reply Context Name

Async Reply Context Value

Figure 10-6 Deploying a gateway channel

3. Click **OK** to deploy the channel.

10.4.2 Deploying the Web Services Gateway service

Once the gateway is configured and the required channel deployed, you can deploy the service. There are two steps involved:

- Provide gateway access to the WSDL file (and any files it imports) for the target Web service you want to deploy.
- Use the gateway systems administration console to deploy the service.

Accessing the target WSDL from the gateway

The WSDL file for the service you want to deploy needs to be accessible by the gateway. The gateway and Web service client applications will also need access to any WSDL files or XML schemas imported by the service WSDL.

If your service WSDL includes imports, our recommendation is to make the WSDL and any imports available via an HTTP URL. This way both the gateway and clients can access the required files from the same location. Ideally, these HTTP URLs should point to documents on a related Web server.

If your service WSDL doesn't import other files, you can place the WSDL on the local file system of the gateway node, since clients will be able to access all the required service definitions via the gateway.

For this example, we use the InventoryDoc service we created in Chapter 9, "Using document style Web services" on page 183. It imports two XML schema files. To provide access to the InventoryDoc service description:

1. Create a new folder called `wsdl` under the <WAS_HOME>\installedApps\<node_name>\wsgw.server1.<node_name>.ear\wsgw.war folder on your Web Services Gateway node.
2. Locate the following files in the ITSOTargetAppEJB\ejbModule\META-INF\wsdl folder under your WebSphere Studio workspace for the ITSO Electronics sample:
   - InventoryDoc.wsdl
   - InventoryReply.xsd
   - InventoryRequest.xsd
3. Copy these files to the new wsdl folder on your gateway node. In our case, the WSDL file will now be accessible in the gateway administrative console Web module with the following URL:

```
http://wsgw1.itso.ral.ibm.com:9080/wsgw/wsdl/InventoryDoc.wsdl
```

4. Edit the WSDL to make sure any imports will be accessible from client applications. As shown in Example 10-1, our InventoryDoc.wsdl contains two relative imports that need to be updated. We changed:

   – `schemaLocation="InventoryReply.xsd"` to `"http://wsgw1.itso.ral.ibm.com:9080/wsgw/wsdl/InventoryReply.xsd"`

   – `schemaLocation="InventoryRequest.xsd"` to `"http://wsgw1.itso.ral.ibm.com:9080/wsgw/wsdl/InventoryRequest.xsd"`

Example 10-1 InventoryDoc.wsdl before changes to import schemaLocations

```
...
<wsdl:types>
  <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://inventory.ws.itso.ibm.com"
      xmlns:reply="http://reply.inventory.ws.itso.ibm.com"
      xmlns:request="http://request.inventory.ws.itso.ibm.com">
   <xsd:import namespace="http://reply.inventory.ws.itso.ibm.com"
      schemaLocation="InventoryReply.xsd"/>
   <xsd:import namespace="http://request.inventory.ws.itso.ibm.com"
      schemaLocation="InventoryRequest.xsd"/>
   <xsd:element name="getDeliveryDate">
...
```

5. Save your changes.

> **Note:** We deployed our service WSDL and XML schema files to the gateway administrative console Web module for simplicity only. In a production environment, it would make more sense to deploy these files to an appropriate Web server.

Deploying the gateway service

To deploy InventoryDoc.wsdl as a gateway service:

1. Open the Web Services Gateway systems administration console and click **Services** → **Deploy** in the navigation panel on the left.
2. In the Deploy Gateway Service window, we set the following fields:

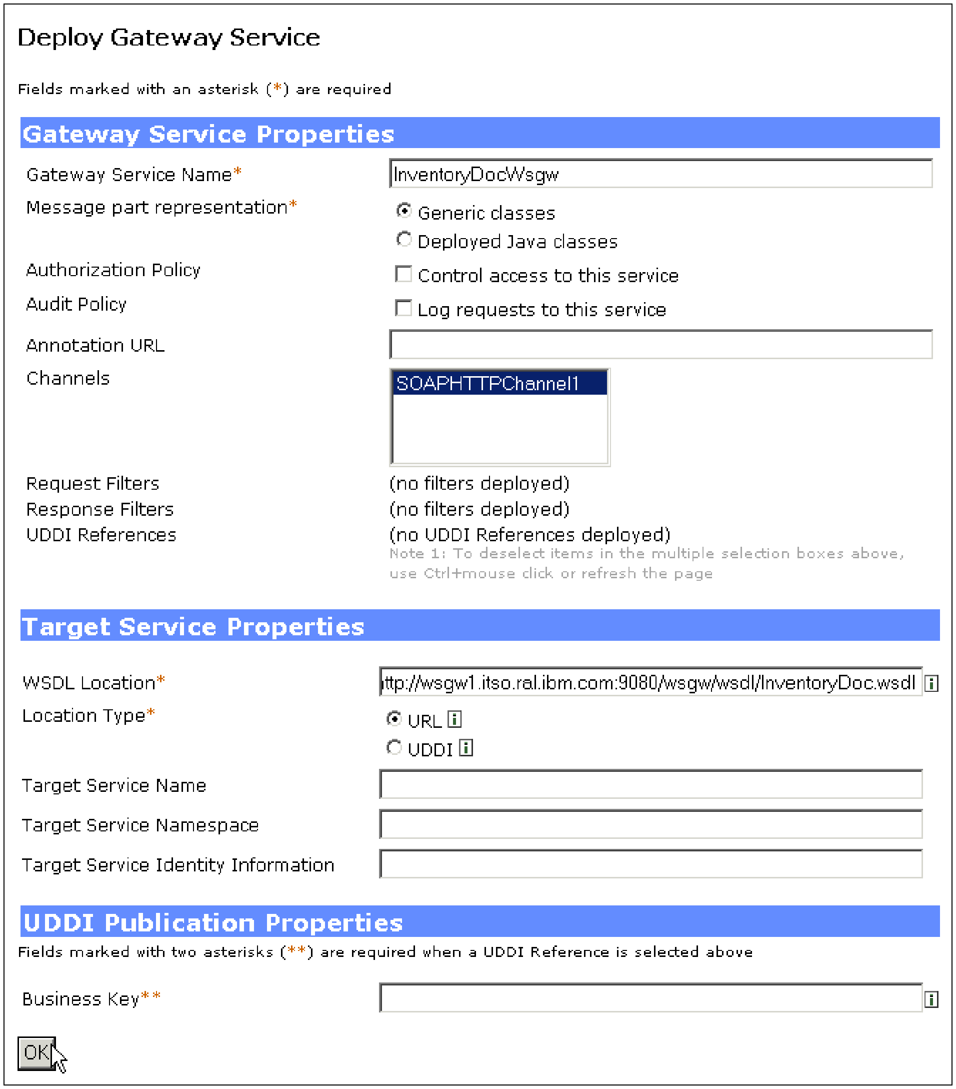
   – Gateway Service Name: `InventoryDocWsgw`

   – Channels: click to select **`SOAPHTTPChannel1`**

   – WSDL Location: `http://wsgw1.itso.ral.ibm.com:9080/wsgw/wsdl/InventoryDoc.wsdl`

   – Location Type: `URL`

   We accepted the defaults for the remaining fields. Our gateway service settings are shown in Figure 10-6.



Deploy Gateway Service

Fields marked with an asterisk (*) are required

Gateway Service Properties

Gateway Service Name*	InventoryDocWsgw
Message part representation*	● Generic classes ○ Deployed Java classes
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*	ttp://wsgw1.itso.ral.ibm.com:9080/wsgw/wsdl/InventoryDoc.wsdl
Location Type*	● URL ○ UDDI
Target Service Name	
Target Service Namespace	
Target Service Identity Information	

UDDI Publication Properties

Fields marked with two asterisks (**) are required when a UDDI Reference is selected above

Business Key**	

Figure 10-7 Deploying a gateway service

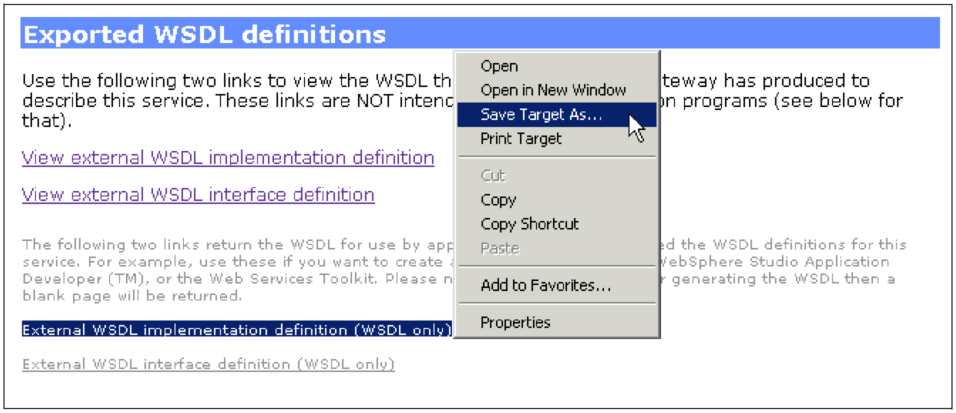
3. Click **OK** to deploy the service.

10.4.3 Exporting the WSDL file

When the service is deployed, the gateway generates new WSDL files that can be shared with clients of your Web service. The gateway-generated WSDL implementation definition file has the gateway as the service end-point, and it imports the WSDL interface definition file that contains bindings and portType information.

To export the WSDL file generated by the Web Services Gateway:

1. Open the Web Services Gateway systems administration console and click **Services** → **List** in the navigation panel on the left.
2. In the List of Gateway Services window, click the required service, **InventoryDocWsgw** in our case.
3. In the Service: InventoryDocWsgw window:
   a. Scroll down to the Exported WSDL definitions section.
   b. Right-click **External WSDL implementation definition (WSDL only)** and select **Save Target As...** from the pop-up menu, as shown in Figure 10-8.
   c. Save the WSDL file to the required location. We saved the file as `InventoryDocWsgw.wsdl` under the `ITSOSourceAppWeb\WebContent\WEB-INF\wsdl` folder in our WebSphere Studio workspace.



Exported WSDL definitions

Use the following two links to view the WSDL th teway has produced to
describe this service. These links are NOT intenc n programs (see below for
that).

View external WSDL implementation definition

View external WSDL interface definition

The following two links return the WSDL for use by app ed the WSDL definitions for this
service. For example, use these if you want to create a WebSphere Studio Application
Developer (TM), or the Web Services Toolkit. Please n generating the WSDL then a
blank page will be returned.

External WSDL implementation definition (WSDL only)

External WSDL interface definition (WSDL only)

Figure 10-8 Exporting the WSDL implementation definition file

We are now ready to Web service-enable our source application using the gateway-generated WSDL implementation definition file for our target service.

Restriction: The WebSphere V5.0.2 Web Services Gateway does not support XML schema-type elements that use the ref attribute. To work around this restriction, you can re-factor any WSDL or XML schema files that use the ref attribute to use nested elements or the type attribute instead.

10.4.4 Web service-enabling the source application

See Figure 8-11 on page 167 for an overview of the Web service development process for a source application. Let's walk through this process for our source application.

To Web service-enable the source application using the gateway-generated WSDL implementation definition file:

4. Open a command window.
5. Using the gateway WSDL file exported in 10.4.3, "Exporting the WSDL file" on page 226, generate the Web service client deployment descriptors and classes using the **WSDL2Java** tool. We used the command shown in Example 10-2.

Example 10-2 Generating client deployment descriptors and classes using WSDL2Java

```
C:\WebSphere\AppServer\bin\WSDL2Java -verbose -role client -container web
   -output C:\workspace\ITSOSourceAppWeb\WebContent
   C:\workspace\ITSOSourceAppWeb\WebContent\WEB-INF\wsdl\InventoryDocWsgw.wsdl

WSWS3185I: Info: Parsing XML file:  C:\...\WEB-INF\wsdl\InventoryDocWsgw.wsdl
Retrieving document at
'http://wsgw1.itso.ral.ibm.com:9080/wsgw/ServiceInterface?name=InventoryDocWsgw', relative to
'C:\workspace\ITSOSourceAppWeb\WebContent\WEB-INF\wsdl\InventoryDocWsgw.wsdl'.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\InventoryRequest_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\InventoryReply_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\UpdateInventory_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Header_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\reply\Body_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Header_Deser.java.
```

```
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\request\Body_Deser.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\InventoryDoc.java.
WSWS3282I: Info: Generating C:\...\itso\ws\inventory\InventoryDocSOAPHTTPBindingStub.java.
WSWS3282I: Info: Generating C:\...\ral\itso\wsgw1\InventoryDocWsgw.java.
WSWS3282I: Info: Generating C:\...\ral\itso\wsgw1\InventoryDocWsgwLocator.java.
WSWS3282I: Info: Generating C:\...\WEB-INF\webservicesclient.xml.
WSWS3282I: Info: Generating C:\...\WEB-INF\ibm-webservicesclient-bnd.xmi.
WSWS3282I: Info: Generating C:\...\WEB-INF\ibm-webservicesclient-ext.xmi.
WSWS3282I: Info: Generating C:\...\WEB-INF\InventoryDocWsgw_mapping.xml.
```

6. In WebSphere Studio, move the generated Java source files from the Web module's WebContent folder to its JavaSource folder:

   a. Right click the **ITSOSourceAppWeb** project and select **Refresh** from the pop-up menu. The generated files should now appear in the Studio workspace.

   b. Move the **com.ibm.itso.ws.inventory** and **com.ibm.ral.itso.wsgw1** packages in the ITSOSourceAppWeb\WebContent folder to the ITSOSourceAppWeb\JavaSource folder.

7. Add client application code to invoke the Web service on the target application.

   To invoke getDeliveryDate on the target application, we added the code shown in Example 10-3. We added this code to the com.ibm.itso.command.GatewayBean command bean in our ITSOSourceAppWeb module.

Example 10-3 Web service client code for getDeliveryDate

```
public String getDeliveryDate(String partNumber) throws Exception {

   String deliveryDate = null;

   try {

      Context ctx = new InitialContext();
      InventoryDocWsgw service = (InventoryDocWsgw) ctx.lookup(
         "java:comp/env/service/InventoryDocWsgw");

      // Request the Service Endpoint from the Service
      InventoryDoc port = service.getInventoryDocSOAPHTTPBindingPort();

      // Generate XML document for update inventory request
      InventoryRequest reqMsg =
         new InventoryRequestBuilder(APP_NAME, partNumber);
      // Get the quote
      InventoryReply repMsg = port.getDeliveryDate(reqMsg);

      // Get the delivery date from the XML reply document
      Date date = (repMsg.getBody()).getDeliveryDate();
      SimpleDateFormat dateFormatter = new SimpleDateFormat(DATE_PATTERN);
      deliveryDate = dateFormatter.format(date);

   } catch (Exception e) {
      //...
   }

   return deliveryDate;
}
```

8. Test, then deploy the source and target applications in your IBM WebSphere Application Server V5.0.2 runtime environment.

The document style SOAP request between the source application and the gateway for getDeliveryDate is shown in Example 10-4.

Example 10-4 SOAP request for getDeliveryDate: source to gateway

```
POST
/wsgwsoaphttp1/soaphttpengine/urn%3Awsgw1.itso.ral.ibm.com%23InventoryDocWsgw
HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: IBM WebServices/1.0
Host: localhost
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 618

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <getDeliveryDate xmlns="http://inventory.ws.itso.ibm.com">
            <InventoryRequest>
               <Header xmlns="">
                  <SourceName>ITSOSourceApp</SourceName>
                  <Version>1</Version>
                  <CreateDate>2003-09-09T00:33:20.981Z</CreateDate>
               </Header>
               <Body xmlns="">
                  <PartNumber>12345</PartNumber>
               </Body>
            </InventoryRequest>
         </getDeliveryDate>
      </soapenv:Body>
   </soapenv:Envelope>
```

The SOAP response between the gateway and the source application for getDeliveryDate is shown in Example 10-5.

Example 10-5 SOAP response for getDeliveryDate: gateway to source

```
HTTP/1.1 200 OK
Server: WebSphere Application Server/5.0
Content-Type: text/xml; charset=utf-8
Content-Language: en-US
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
      <Body>
         <getDeliveryDateResponse xmlns="http://inventory.ws.itso.ibm.com"
             xmlns:ns-445461426="http://inventory.ws.itso.ibm.com"
             xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <InventoryReply>
               <Header xmlns="">
                  <SourceName>ITSOTargetApp</SourceName>
                  <Version>1</Version>
                  <CreateDate>2003-09-09T00:33:28.842Z</CreateDate>
               </Header>
               <Body xmlns="">
                  <PartNumber>12345</PartNumber>
                  <DeliveryDate>2003-09-15</DeliveryDate>
               </Body>
            </InventoryReply>
         </getDeliveryDateResponse>
      </Body>
   </Envelope>
```

The document style SOAP request between the gateway and the target application for getDeliveryDate is shown in Example 10-6.

Example 10-6 SOAP request for getDeliveryDate: gateway to target

```
POST /ITSOTargetApp/services/InventoryDoc HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: IBM WebServices/1.0
Host: localhost
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 676

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <getDeliveryDate xmlns="http://inventory.ws.itso.ibm.com"
            xmlns:ns-445461426="http://inventory.ws.itso.ibm.com">
            <InventoryRequest>
               <Header xmlns="">
                  <SourceName>ITSOSourceApp</SourceName>
                  <Version>1</Version>
                  <CreateDate>2003-09-09T00:33:20.981Z</CreateDate>
               </Header>
               <Body xmlns="">
                  <PartNumber>12345</PartNumber>
               </Body>
            </InventoryRequest>
         </getDeliveryDate>
      </soapenv:Body>
   </soapenv:Envelope>
```

The SOAP response between the target application and the gateway for getDeliveryDate is shown in Example 10-7.

Example 10-7 SOAP response for getDeliveryDate: target to gateway

```
HTTP/1.1 200 OK
Server: WebSphere Application Server/5.0
Content-Type: text/xml; charset=utf-8
Content-Language: en-US
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <getDeliveryDateResponse xmlns="http://inventory.ws.itso.ibm.com">
         <InventoryReply>
            <Header xmlns="">
               <SourceName>ITSOTargetApp</SourceName>
               <Version>1</Version>
               <CreateDate>2003-09-09T00:33:28.842Z</CreateDate>
            </Header>
            <Body xmlns="">
               <PartNumber>12345</PartNumber>
               <DeliveryDate>2003-09-15</DeliveryDate>
            </Body>
         </InventoryReply>
      </getDeliveryDateResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

10.5 Quality of Service capabilities

In this section we discuss Quality of Service capabilities and considerations specific to the Web services Gateway. For further discussion on Quality of Services for Web services in general, see 8.6, “Quality of Service capabilities” on page 177. For document style Web services see 9.6, “Quality of Service capabilities” on page 209.

10.5.1 Autonomic

Log and trace facilities are important for fault monitoring and isolation. WebSphere Application Server provides a number of log files. JVM logs are located in the <WAS_HOME>/logs/<applicationServerName> directory, and by default are named SystemOut.log and SystemErr.log.

The Diagnostic Trace Service can be used to enable tracing of application server components. The following trace specification can be used when diagnosing Web Services Gateway problems:

```
com.ibm.wsgw.*=all=enabled:
org.apache.wsif.*=all=enabled:
com.ibm.ws.webservices.*=all=enabled
```

The following tools can also be helpful when analyzing problems:

- TCPMon

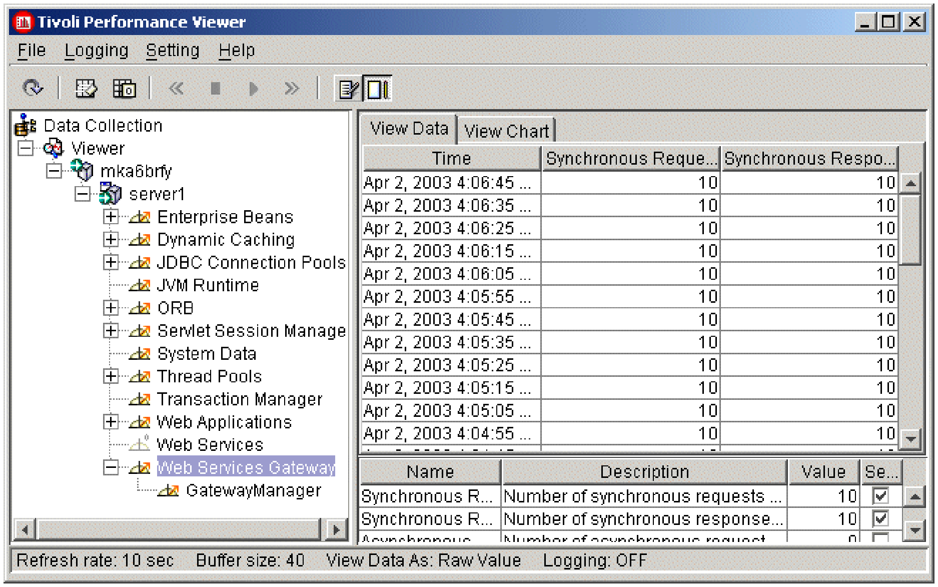
  The TCPMon tool described in 8.5.3, “Monitoring SOAP messages” on page 172 allows you to view the contents of the SOAP messages being generated by the interaction between the source application, gateway, and target application.

- Tivoli® Performance Viewer

  The Tivoli Performance Viewer packaged with IBM WebSphere Application Server V5.0 can monitor Web Services Gateway requests and responses. When the gateway is installed in WebSphere V5.0, counters are added automatically to the Performance Monitoring Service. Gateway monitoring can be enabled by simply starting performance monitoring. See the redbook for details on specific procedures for enabling monitoring:

  – *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195

  Once monitoring is enabled on the WebSphere Application Server, open the Tivoli Performance Monitor and navigate to the Web Services Gateway. The Viewer monitors both synchronous and asynchronous requests and responses. The output can be viewed in either table or graph format. It can also be logged and played back when needed. Figure 10-9 shows the Tivoli Performance Viewer monitoring gateway requests and responses.



Time	Synchronous Reque...	Synchronous Respo...
Apr 2, 2003 4:06:45 ...	10	10
Apr 2, 2003 4:06:35 ...	10	10
Apr 2, 2003 4:06:25 ...	10	10
Apr 2, 2003 4:06:15 ...	10	10
Apr 2, 2003 4:06:05 ...	10	10
Apr 2, 2003 4:05:55 ...	10	10
Apr 2, 2003 4:05:45 ...	10	10
Apr 2, 2003 4:05:35 ...	10	10
Apr 2, 2003 4:05:25 ...	10	10
Apr 2, 2003 4:05:15 ...	10	10
Apr 2, 2003 4:05:05 ...	10	10
Apr 2, 2003 4:04:55 ...	10	10

Name	Description	Value	Se...
Synchronous R...	Number of synchronous requests ...	10	☑
Synchronous R...	Number of synchronous response...	10	☑
Asynchronous	Number of asynchronous request	0	□

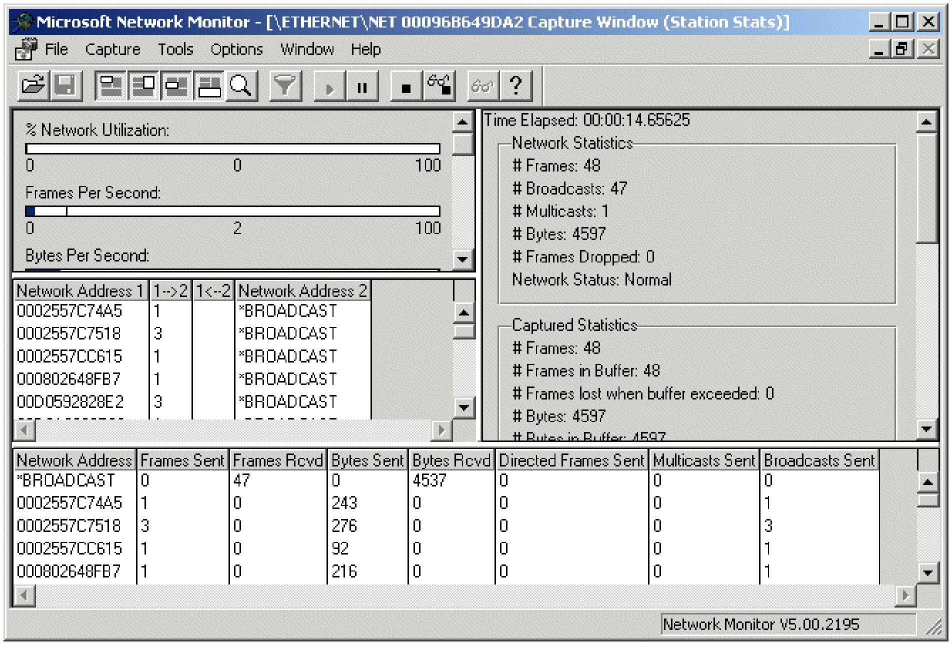
Refresh rate: 10 sec Buffer size: 40 View Data As: Raw Value Logging: OFF

Figure 10-9 Tivoli Performance Viewer

- Microsoft Network Monitor

  Microsoft Network Monitor captures network traffic on local area networks for real-time or post-capture analysis. The Network Monitor captures frames from the network that can be filtered to present only relevant material. It can also be configured to detect specific network conditions and generate events as needed. Figure 10-10 shows the Microsoft Network Monitor. Details concerning the configuration and use of the monitor can be found from Windows Help.

  Of particular interest to the gateway is the Network Monitor's ability to monitor TCP/IP packets from a particular target and source. In this manner, traffic to and from the gateway can be traced from its origin to its destination.


Microsoft Network Monitor - [\ETHERNET\NET 000968649DA2 Capture Window (Station Stats)]
File
Capture
Tools
Options
Window
Help

% Network Utilization:
0 0 100

Frames Per Second:
0 2 100

Bytes Per Second:

Time Elapsed: 00:00:14.65625

Network Statistics
Frames: 48
Broadcasts: 47
Multicasts: 1
Bytes: 4597
Frames Dropped: 0
Network Status: Normal

Captured Statistics
Frames: 48
Frames in Buffer: 48
Frames lost when buffer exceeded: 0
Bytes: 4597
Bytes in Buffer: 4597

Network Address 1	1-->2	1<--2	Network Address 2
0002557C74A5	1		*BROADCAST
0002557C7518	3		*BROADCAST
0002557CC615	1		*BROADCAST
000802648FB7	1		*BROADCAST
00D0592828E2	3		*BROADCAST

Network Address	Frames Sent	Frames Rcvd	Bytes Sent	Bytes Rcvd	Directed Frames Sent	Multicasts Sent	Broadcasts Sent
*BROADCAST	0	47	0	4537	0	0	0
0002557C74A5	1	0	243	0	0	0	1
0002557C7518	3	0	276	0	0	0	3
0002557CC615	1	0	92	0	0	0	1
000802648FB7	1	0	216	0	0	0	1

Network Monitor V5.00.2195

Figure 10-10 Microsoft Network Monitor

10.5.2 Security

The Web Services Gateway provides a basic authentication and authorization mechanism based upon the security features of WebSphere Application Server. Security can be applied at two levels:

- Gateway-level authentication
- Operation-level authorization

Gateway-level authentication

For gateway-level authentication, you set up a role and realm for the gateway on WebSphere's Web server and servlet container, and define the user ID and password that is used by the gateway to access the role and realm. You also modify the gateway's channel applications so that they only give access to the gateway to service requestors that supply the correct user ID and password for that role and realm.

Operation-level authentication

For operation-level authorization, you apply security to individual methods in a Web service. To do this, create an enterprise bean with methods matching the Web service operations. These EJB methods perform no operation and are just dummy entities for applying security. Existing WebSphere Application Server authentication mechanisms can be applied to the enterprise bean. Before any Web service operation is invoked, a call is made to the EJB method. If authorization is granted, the Web service is invoked.

11

Using the Web Services Gateway with J2EE Connectors

This chapter discusses using the Web Services Gateway in an intra-enterprise integration scenario. We are using the Web Services Gateway packaged with IBM WebSphere Application Server Network Deployment V5.0.2 in this scenario.

This chapter extends the scenario covered in the previous chapter by looking at the protocol and message conversion capabilities of the Web Services Gateway. We examine how the gateway can provide Web service access to a CICS application using the CICS ECI J2EE Connector.

11.1 Business scenario

In this scenario, ITSO Electronics has a legacy system running some of its core business processes. The intention is not to replace or re-engineer this system, but to allow more modern and flexible business applications to reuse the important business processes that this core system hosts. The goal is to leverage the legacy system in a way that enables ITSO Electronics to be responsive to their customer's needs.

The integration functionality to be implemented is explained in 6.2.2, "Stage II: Internal ordering on demand with delivery date" on page 119. The retail system must access, in real time, an existing wholesale system to check the availability of a product and obtain its estimated delivery date. For the scenario covered in this chapter, the legacy wholesale system is a CICS server located on the internal network.

An important requirement is to provide a standards-based interface to the system. The aim of this requirement is to allow interoperability with different technologies now and in the future. A solution meeting this requirement is vendor-independent, and easier to maintain and test using standard tools and procedures.

If possible, it is preferable to use off-the-shelf products to provide connectivity among different technologies and protocols because they are usually faster to implement, and have wider support and lower costs.

11.2 Design guidelines

In this scenario, the source application needs to invoke services provided by a target CICS system, as shown in Figure 11-1. Using the Web Services Gateway as an adapter connector makes it possible to decouple the source application from the target application, with all protocol and message conversion handled by the gateway.

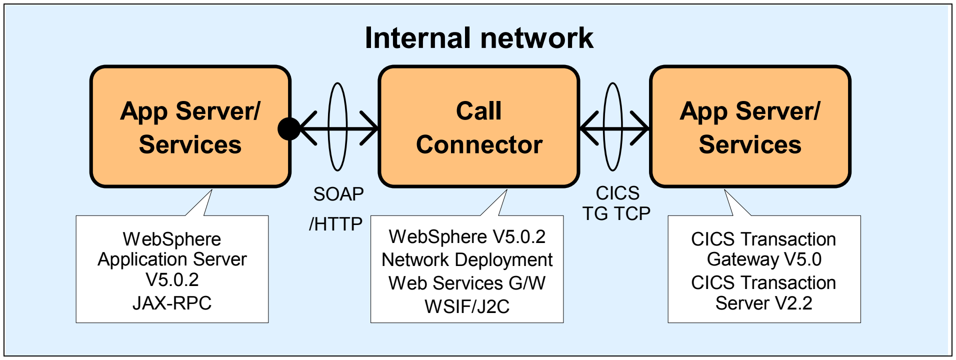

Internal network
App Server/ Services
Call Connector
App Server/ Services
SOAP /HTTP
CICS TG TCP
WebSphere Application Server V5.0.2 JAX-RPC
WebSphere V5.0.2 Network Deployment Web Services G/W WSIF/J2C
CICS Transaction Gateway V5.0 CICS Transaction Server V2.2

Figure 11-1 Direct Connection::Call Connection: Web Services Gateway Product mapping 2

The IBM Web Services Gateway allows our source application to access the target CICS service as an HTTP Web service. An important reason for using the gateway is that it allows a number of different technologies to be accessed as Web services.

This pattern models the connector between the source application node on the left and the gateway node in the middle, and the connector between the gateway and target application node on the right, as simple lines.

If further detail is need, the source application node to gateway node connector can be modeled using coupling adapter connectors. The gateway node to target application node connector can be modeled using an adapter connector. Figure 11-2 shows we are using the WebSphere SOAP provider as coupling adapter connectors from the source, and the CICS Transaction Gateway as adapter connectors to the target.

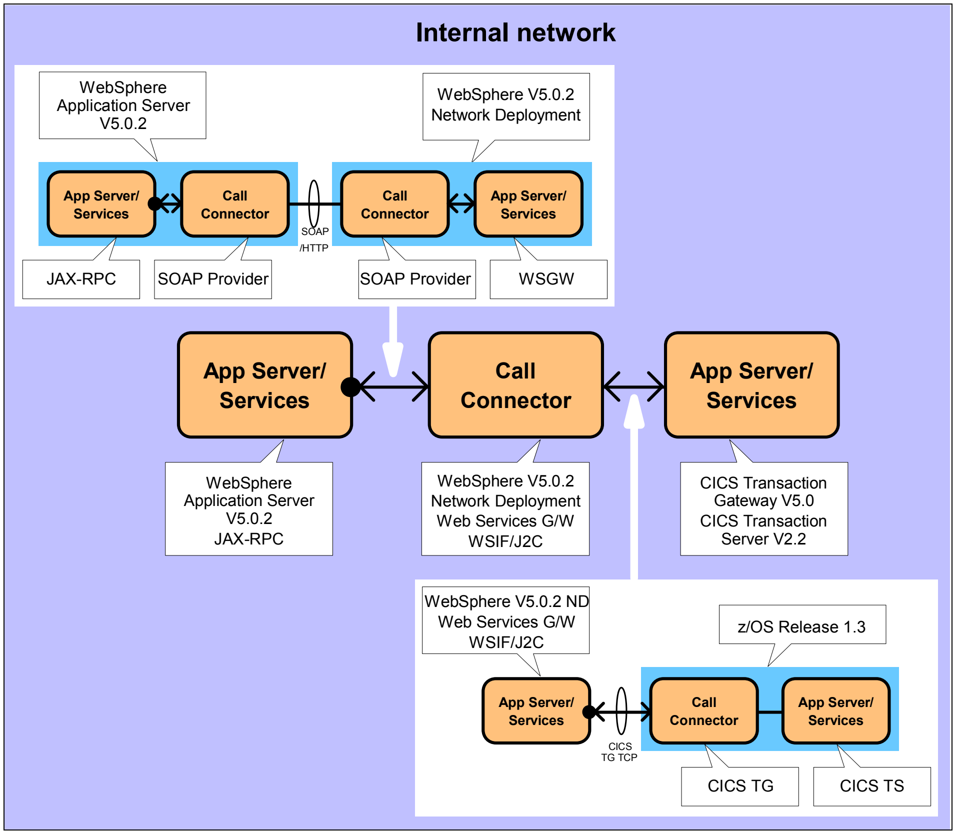


Figure 11-2 Detailed design with product mappings

The IBM CICS Transaction Gateway and the Enterprise Services toolkit in WebSphere Studio Application Developer Integration Edition enable J2EE application integration with CICS applications. More information on J2EE Connectors and the Enterprise Services toolkit can be found in 12.3, “Development guidelines” on page 269.

In Figure 11-3, the structure of the gateway connector adapter is specified in more detail, highlighting four main components:

- IBM WebSphere Application Server V5.0, which provides the J2EE platform implementation.
- IBM Web Services Gateway, which enables Web services connectivity between applications.
- CICS ECI J2EE Connector, which allows J2EE applications to access CICS applications.

- Deployed proxy and data types, which are developed using the Enterprise Services toolkit in IBM WebSphere Studio Application Developer Integration Edition V5.0.

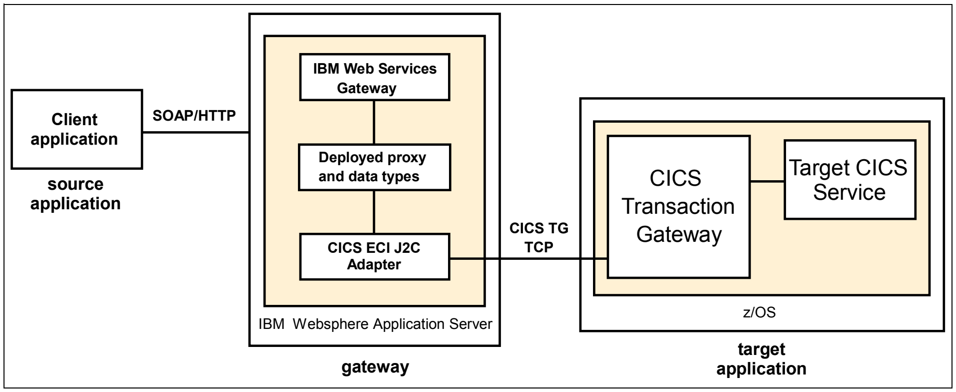


Figure 11-3 Solution architecture overview

This solution provides a Java-based interface and a Web service interface to the target CICS service. This solution can be used to provide a wide range of client platforms with access to the target CICS service, in both intra-enterprise and inter-enterprise integration scenarios. The adoption of a Web service interface provides an open standards-based way of accessing the system.

In the development phase, wizards provided in WebSphere Studio Application Developer Integration Edition can be used to develop the connector solution without requiring you to write code.

11.3 Development guidelines

The main steps for building the integration solution presented in Figure 11-3 are:

1. Create an enterprise service using WebSphere Studio Application Developer Integration Edition V5.0.
2. Test the enterprise service in the WebSphere Studio Application Developer Integration Edition V5.0.
3. Deploy generated Java classes into IBM WebSphere Application Server Enterprise V5.0.
4. Configure a J2EE Connector connection factory in IBM WebSphere Application Server Enterprise V5.0.
5. Configure the service in the IBM Web Services Gateway V5.0.2.

6. Web service-enable your client application.

Note: Our sample implementation code is available on the Web; see Appendix B, “Additional material” on page 335 for details. The application deployed to the gateway is packaged in ITSOConnectorApp.ear. The client application implementation is included in the ITSOSourceApp project.

The configuration of the IBM CICS Transaction Gateway is beyond the scope of this redbook. For further details on the CICS Transaction Gateway and J2EE Connectors refer to the following redbooks:

- *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133
- *Patterns: Self-Service Application Solutions Using WebSphere Application Server V5*, SG24-6591

11.3.1 Creating a CICS enterprise service

This section explains how to use the IBM WebSphere Studio Application Developer Integration Edition V5.0 to create an enterprise service for an existing CICS program. The enterprise service and its deployment classes can be used by applications running in IBM WebSphere Application Server Enterprise V5.0.

The main steps to develop the enterprise service are:

1. Import the resource adapter into the workspace.
2. Create the enterprise service WSDL description.
3. Create the proxy and data types.
4. Generate WSDL description of the proxy.

Import the resource adapter into the workspace

It is only necessary to import the CICS ECI Connector once per workspace. To check if the connector is already present, switch to the J2EE Hierarchy view in the Business Integration perspective. The ECIResourceAdapter should be listed in the Connector Modules folder, as shown in Figure 11-4.

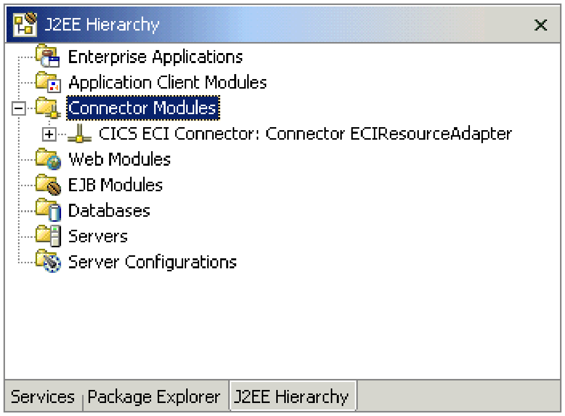


Figure 11-4 CICS ECI Connector in the J2EE Hierarchy

If you need to import the CICS ECI Connector:

1. Select **File** → **Import...** → **RAR file** and click **Next**.
2. Browse to the **cicseci.rar** file in the `<STUDIO_HOME>\resource adapters folder`. <STUDIO_HOME> is the installation path of WebSphere Studio Application Developer Integration Edition.

   Select **cicseci.rar** and click **Open**.
3. Check **Standalone connector project**, and select the **New** connector project option.
4. Enter the New project name. We used `CICS ECI Connector`.
5. Click **Finish**.

The CICS ECI Connector should now be correctly listed in the J2EE Hierarchy view of the Business Integration perspective, as shown in Figure 11-4.

Create the enterprise service WSDL description

To create the enterprise service WSDL description, first create a new enterprise application and Web module:

1. To create a new enterprise application project and Web module project, select **File** →**New** →**Project...** →**J2EE** →**Enterprise Application Project**, then click **Next**.
   a. Select **Create J2EE 1.3 Enterprise Application project** and click **Next**.
   b. In the Enterprise Application Project window set the following fields:
      - Enterprise application project name: `ITSOConnectorApp`.
      - Uncheck **Application client module** and **EJB module**.

Click **Finish**.

2. To set the Context Root of the new Web module to **ITSOConnectorApp**, open the J2EE Navigator view of the J2EE Perspective, right-click the **ITSOConnectorAppWeb** project and select **Properties** → **Web**.
3. Add a new package to the Web module by right-clicking the **Java Source** folder in the ITSOConnectorAppWeb project and selecting **New** →**Package**. Set the new package name to `com.ibm.itso.es.inventory`.
4. Import the source file defining the interface for the CICS service:

   a. Right-click the com.ibm.itso.es.inventory package, select **Import...** → **File System**, then click **Next**.

   b. Browse to the source file, **getdate.c** in our case, and click **Finish**.

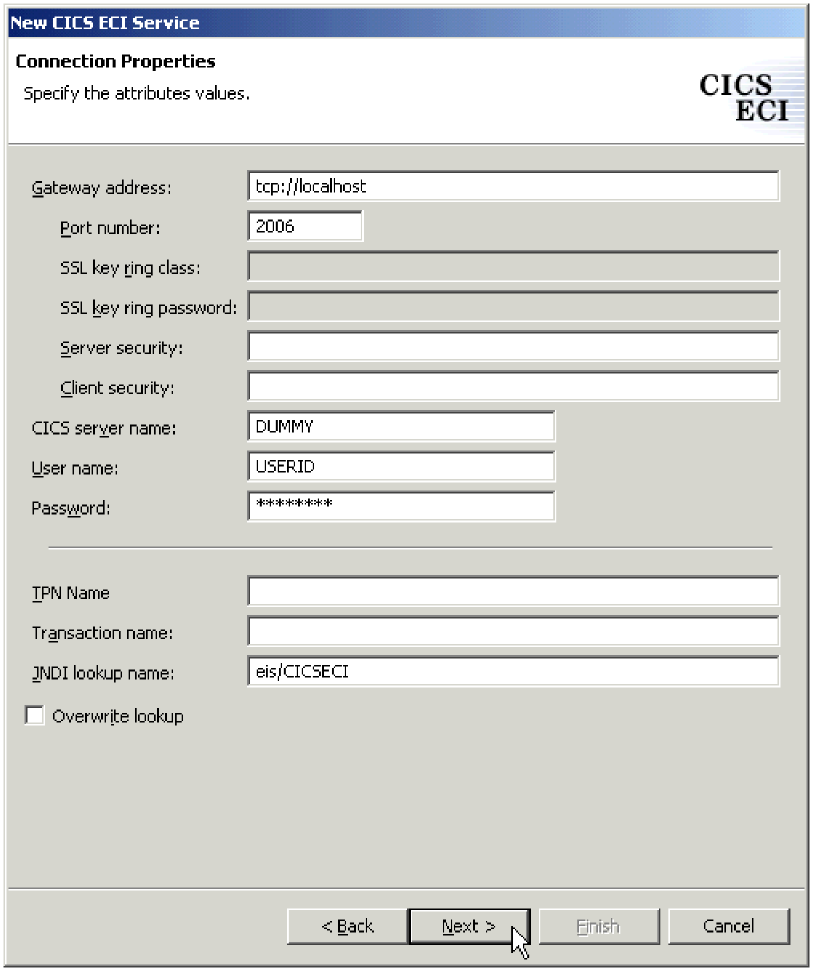
   The getdate.c source file actually contains the full source code for our backend CICS application. To set up this application on CICS, we compiled and linked the source on z/OS, placed the load module in a CICS RPL library, and installed the program definition to CICS.

You are now ready to create the enterprise service. To create the enterprise service:

1. Right-click the **com.ibm.itso.es.inventory** package, select **New** → **Other...** → **Business Integration** → **Service built from...** and click **Next**.
2. In the New Service window select **CICS ECI** in the left panel and click **Next**.
3. In the Connection Properties window, shown in Figure 11-5, enter the details for your CICS Transaction Gateway.

   For our example, the JNDI lookup name should be `eis/CICSECI`. This is the JNDI reference to the J2C connection factory needed in the application server environment.

   Click **Next**.



Connection Properties

Specify the attributes values.

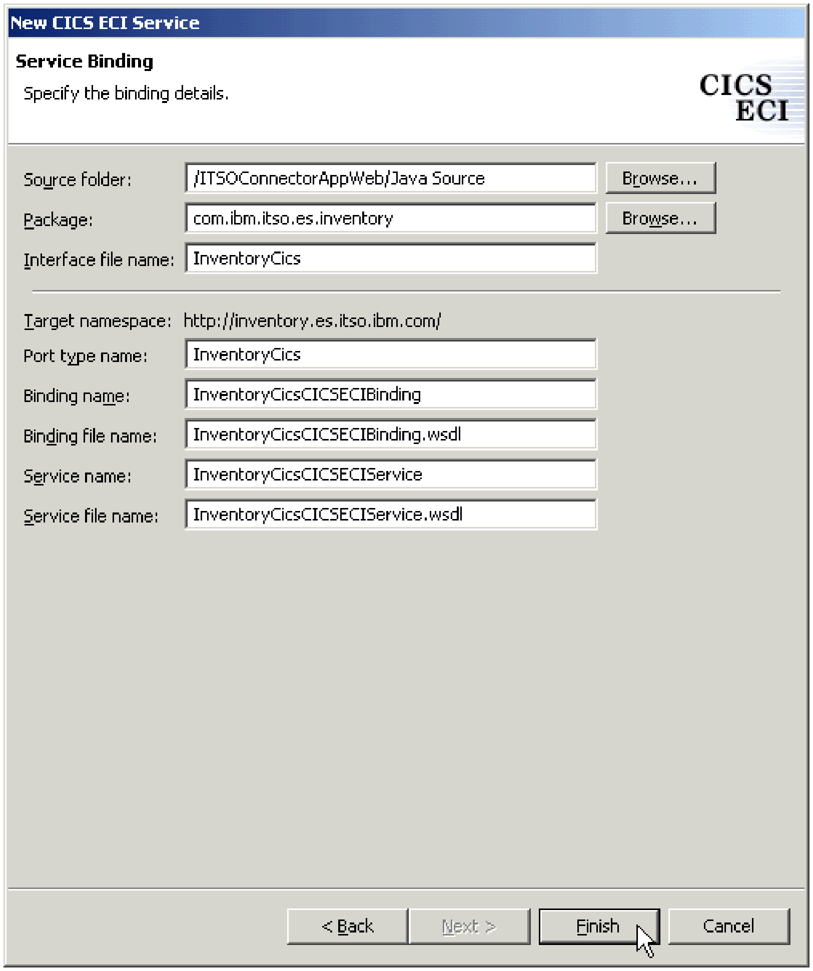
Gateway address:	tcp://localhost
Port number:	2006
SSL key ring class:	
SSL key ring password:	
Server security:	
Client security:	
CICS server name:	DUMMY
User name:	USERID
Password:	********
TPN Name	
Transaction name:	
JNDI lookup name:	eis/CICSECI

□ Overwrite lookup

Figure 11-5 Connection Properties

4. In the Service Binding window specify the package and the interface file name, as shown in Figure 11-6. The other fields will be automatically filled.

   Click **Finish**.



Service Binding

Specify the binding details.

Source folder: /ITSOConnectorAppWeb/Java Source

Package: com.ibm.itso.es.inventory

Interface file name: InventoryCics

Target namespace: http://inventory.es.itso.ibm.com/

Port type name: InventoryCics

Binding name: InventoryCicsCICSECIBinding

Binding file name: InventoryCicsCICSECIBinding.wsdl

Service name: InventoryCicsCICSECIService

Service file name: InventoryCicsCICSECIService.wsdl

Figure 11-6 Service Binding

Notice that three WSDL files are now present in com.ibm.itso.es.inventory package:

- InventoryCics.wsdl
- InventoryCicsCICSECIBinding.wsdl
- InventoryCicsCICESCIService.wsdl

Create the binding operations using the following steps:

1. Double-click **InventoryCicsCICESCIBinding.wsdl** to open it in WSDL editor. Select the **Bindings** tab.

2. In the Port Type and Binding Operations section click **New** to add a new operation.
3. In the Operation Binding window, set the Operation name to `getDeliveryDate` and click **Next**.
4. In the CICS ECI Connector Operation Binding Properties window, set the functionName to `GETDATE` (uppercase). This is the name of the CICS program that will be invoked for the service. Accept the defaults for the other fields and click **Next**.
5. In the Operation Binding window click **Import...** in the Input message section.
6. Navigate to the interface source file, **getdate.c**, in the ITSOConnectorAppWeb/Java Source/com/ibm/itso/es/inventory folder, and click **Next**.
7. In the C Import Properties window, set the platform properties for the environment hosting the CICS service. In our case, we set the Floating point format to **IBM 390 Hexadecimal** and the Code page selection to **Cp037**.

   Click **Next**.
8. In the following window, click **Next** again.
9. In the C Importer window, choose `COMM_AREA` as structure definition. In the XSD type name, set the name for the generated Java class. We used `InventoryData`.

   Click **Finish**.
10. Back in the Operation Binding window, check **Use input message for output** and click **Finish**.

The WSDL definition of the enterprise service is complete, and the J2EE Navigator view should look similar to Figure 11-7.

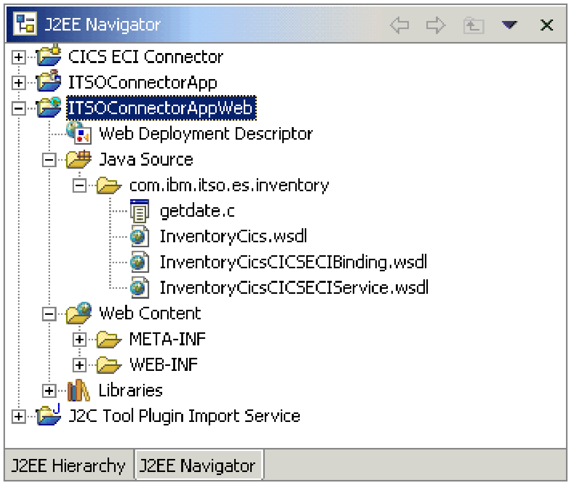


Figure 11-7 J2EE Navigator view after creating the enterprise service WSDL files

The three WSDL files generated provide a complete description of the service, how it is accessed, its methods (getDeliveryDate), and data types (InventoryData). WebSphere Studio's enterprise services toolkit uses the Web Services Description Language as the model for describing any kind of service. Using the toolkit it is possible to build an object for accessing the service from the WSDL description.

Create the proxy and data types

Since the actual implementation of our service exists in a CICS server, we need to build a class to connect to this service, acting as a proxy. The enterprise services toolkit provides a number of options for deploying services, including:

- SOAP
- EJB
- JMS
- JavaBean proxy

For this scenario, we selected a JavaBean proxy, primarily because it is the simplest choice. Our simple scenario does not require the remote access, transaction, and security management capabilities provided by EJBs, so there is really no need to introduce them here.

To build a Java proxy class:

1. Right-click the **InventoryCicsCICSECIService.wsdl** file and choose **Enterprise Services** → **Generate Service Proxy...**.

2. In the first window, accept the default options by clicking **Next**.
3. In the second window, select the operations to be included in the proxy. In this case, select **getDeliveryDate** and click **Finish**.

As shown in Figure 11-8, the following Java files have been created:

- InventoryCicsProxy.java is the proxy class.
- InventoryData.java represents the data type used by the proxy.
- InventoryDataFormatHandler.java provides the binding from the Java representation of the data type to the native format.

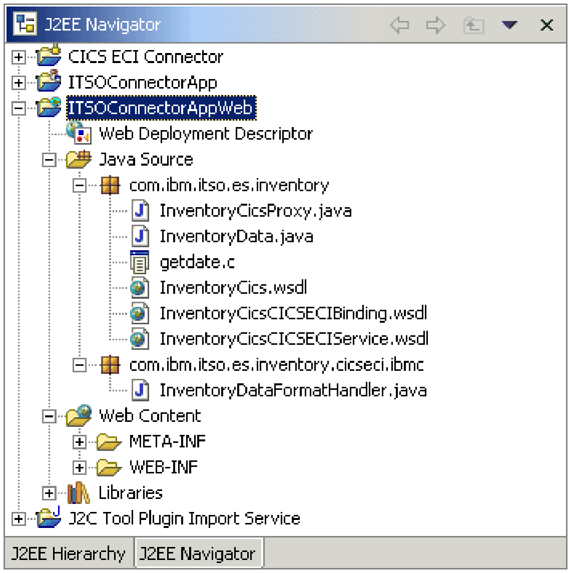

J2EE Navigator
CICS ECI Connector
ITSOConnectorApp
ITSOConnectorAppWeb
Web Deployment Descriptor
Java Source
com.ibm.itso.es.inventory
InventoryCicsProxy.java
InventoryData.java
getdate.c
InventoryCics.wsdl
InventoryCicsCICSECIBinding.wsdl
InventoryCicsCICSECIService.wsdl
com.ibm.itso.es.inventory.cicseci.ibmc
InventoryDataFormatHandler.java
Web Content
META-INF
WEB-INF
Libraries
J2C Tool Plugin Import Service
J2EE Hierarchy
J2EE Navigator

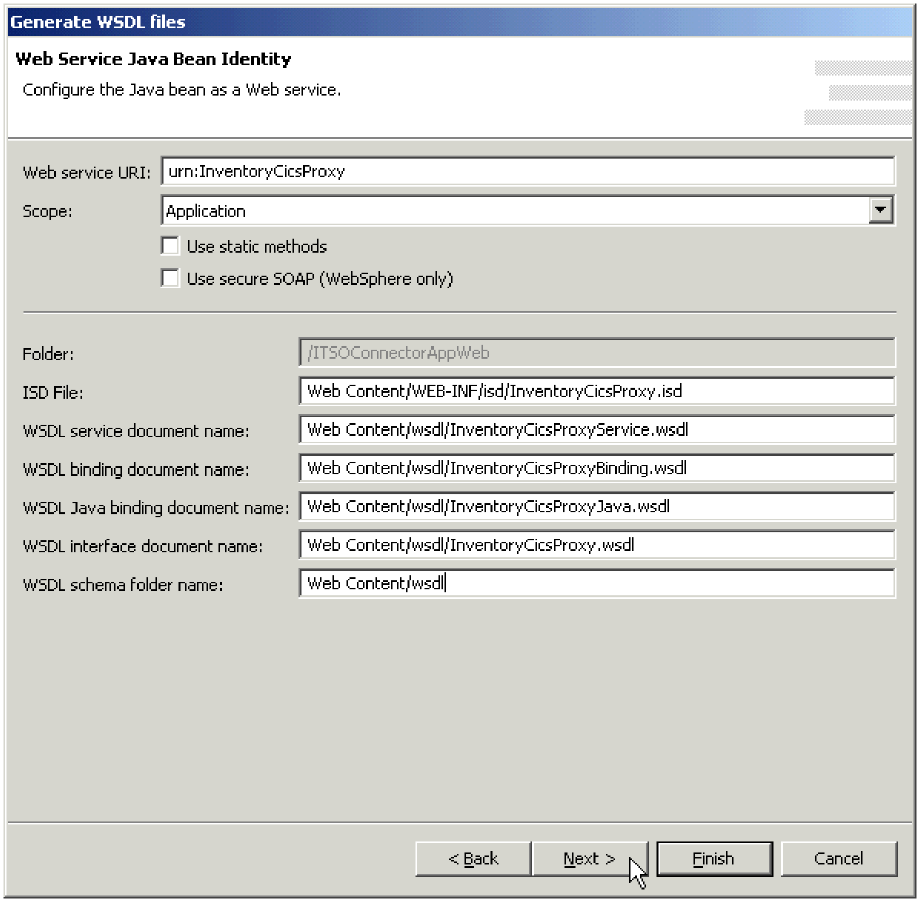
Figure 11-8 J2EE Navigator view after creating the service proxy classes

Generate WSDL description of the proxy

The InventoryCicsProxy.java class provides access to the CICS Transaction Gateway and invokes the GETDATE CICS function. To publish this class as a Web service on the Web Services Gateway, it is necessary to obtain a WSDL description of it by following these steps:

1. Right-click the **InventoryCicsProxy.java** class in the Java Source folder of the ITSOConnectorAppWeb project and select **Web Services** → **Generate WSDL files**.
2. In the Web Service Deployment Settings window, click **Next**.
3. In the Web Service Java Bean Selection window, select **com.ibm.itso.es.inventory.InventoryCicsProxy** as Bean and click **Next**.

4. In the Web Service Java Bean Identity window, abbreviate the Web service URI and default paths for the generated files, as we did, as shown in Figure 11-9. Click **Next**.



Generate WSDL files

Web Service Java Bean Identity

Configure the Java bean as a Web service.

Web service URI: urn:InventoryCicsProxy

Scope: Application

□ Use static methods

□ Use secure SOAP (WebSphere only)

Folder: /ITSOConnectorAppWeb

ISD File: Web Content/WEB-INF/isd/InventoryCicsProxy.isd

WSDL service document name: Web Content/wsdl/InventoryCicsProxyService.wsdl

WSDL binding document name: Web Content/wsdl/InventoryCicsProxyBinding.wsdl

WSDL Java binding document name: Web Content/wsdl/InventoryCicsProxyJava.wsdl

WSDL interface document name: Web Content/wsdl/InventoryCicsProxy.wsdl

WSDL schema folder name: Web Content/wsdl

Figure 11-9 Web Service Java Bean Identity

5. In the Web Service Java Bean Methods window, check only the method **com.ibm.itso.es.inventory.InventoryData getDeliveryDate()** and uncheck all the others. Click **Finish**.

The Web Content folder of the ITSOConnectorAppWeb now contains two sub-folders:

- `wsdl` contains the WSDL service definitions needed to deploy the service in the Web Services Gateway.

- **admin** contains some JSPs for administering Web services deployed directly to WebSphere Application Server. We are using the Web Services Gateway, so these files are not needed and this folder can be deleted.

Next, we need to make some changes so the generated WSDL can be accessed by the gateway and by our client application:

6. Move the **InventoryData.xsd** file from the Web Content\wsdl\com\ibm\itso\es\inventory folder to the Web Content\wsdl folder. Then delete the Web Content\wsdl\com folder and its sub-folders.
7. Edit **InventoryCicsProxy.wsdl** in the Web Content\wsdl folder and change the location attribute of the following import statement (near the start):

   ```
   <import location="com/ibm/itso/es/inventory/InventoryData.xsd" ...
   <import location="InventoryCicsProxy.wsdl" ...
   ```

   Replace the com/ibm/itso/es/inventory/ prefix with the following prefix:

   ```
   http://wsgw2.itso.ral.ibm.com:9080/ITSOConnectorApp/wsdl/
   ```

8. Save and close the file.
9. Edit **InventoryCicsProxyJava.wsdl** in the Web Content\wsdl folder and change the location attribute of the following two import statements (near the start):

   ```
   <import location="com/ibm/itso/es/inventory/InventoryData.xsd" ...
   <import location="InventoryCicsProxy.wsdl" ...
   ```

   Delete the com/ibm/itso/es/inventory/ prefix from the first import, then add the following prefix to both import locations:

   ```
   http://wsgw2.itso.ral.ibm.com:9080/ITSOConnectorApp/wsdl/
   ```

10. Save and close the file.

The WSDL definition of the service proxy is complete.

11.3.2 Testing the enterprise service

In this section, we create a simple JSP to test the generated Java proxy. We also explain how to configure the WebSphere Studio test environment to test the application. Our test JSP (TestProxy.jsp) uses the generated InventoryCicsProxy class to invoke the GETDATE CICS function and display the result.

Creating a JSP to test the enterprise service

To create a JSP to test the deployed enterprise service:

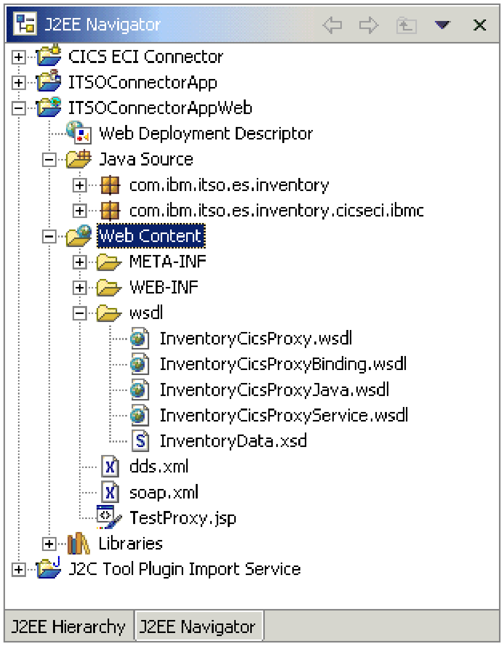
1. Right-click the **Web Content** folder of the ITSOConnectorAppWeb project and select **New** → **JSP File**. In the New JSP File window, set the File Name to TestProxy.jsp and click **Finish**.

2. Add the required code to the test JSP. The listing for our TestProxy.jsp file is shown in Example 11-1. It contains a very simple example of how the service proxy class can be used.

Example 11-1 TestProxy.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<TITLE>TestProxy.jsp</TITLE>
</HEAD>
<BODY>
<h1>Test proxy jsp: ITSOConnectorApp</h1>
<%
   com.ibm.itso.es.inventory.InventoryData indata =
      new com.ibm.itso.es.inventory.InventoryData();
   indata.setPartNumber("1");

   com.ibm.itso.es.inventory.InventoryData outdata = null;

   com.ibm.itso.es.inventory.InventoryCicsProxy proxy =
      new com.ibm.itso.es.inventory.InventoryCicsProxy();

   outdata = proxy.getDeliveryDate(indata);

   out.println("The expect delivery date of part number "
         +indata.getPartNumber()
         +" is "
         +outdata.getDeliveryDate());
%>
</BODY>
</HTML>
```

The enterprise service application is complete, and the J2EE Navigator view should look similar to Figure 11-10.



```
J2EE Navigator
⊞ CICS ECI Connector
⊞ ITSOConnectorApp
⊟ ITSOConnectorAppWeb
    Web Deployment Descriptor
  ⊟ Java Source
    ⊞ com.ibm.itso.es.inventory
    ⊞ com.ibm.itso.es.inventory.cicseci.ibmc
  ⊟ Web Content
    ⊞ META-INF
    ⊞ WEB-INF
    ⊟ wsdl
        InventoryCicsProxy.wsdl
        InventoryCicsProxyBinding.wsdl
        InventoryCicsProxyJava.wsdl
        InventoryCicsProxyService.wsdl
        InventoryData.xsd
      dds.xml
      soap.xml
      TestProxy.jsp
  ⊞ Libraries
⊞ J2C Tool Plugin Import Service
J2EE Hierarchy | J2EE Navigator
```

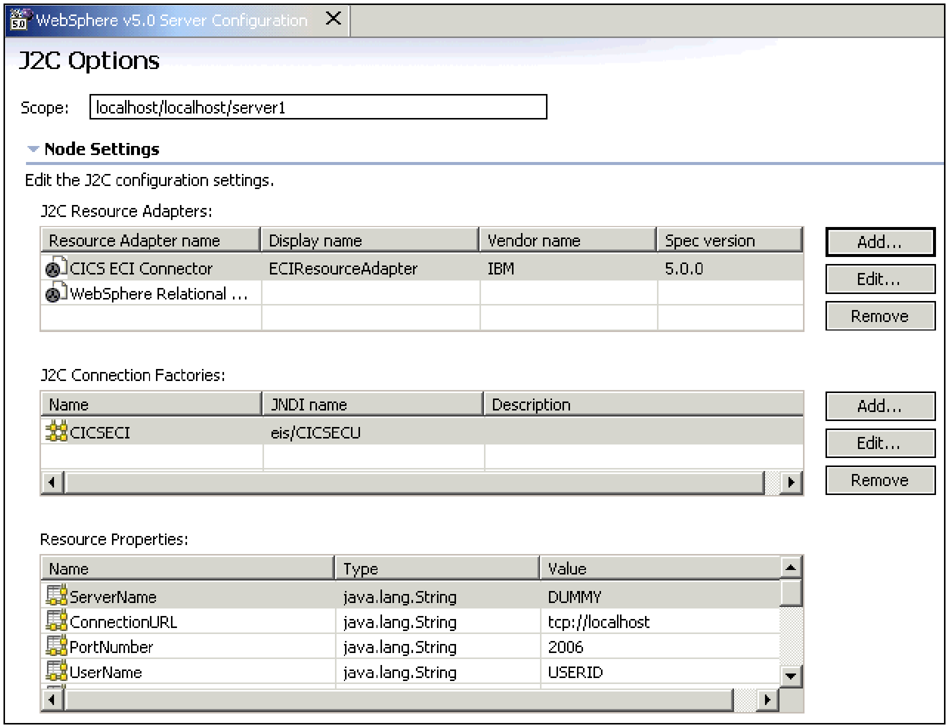
Figure 11-10 J2EE Navigator view after creating the test JSP

Configuring the test environment

It is necessary to define a J2C configuration factory, as shown in Figure 11-11, to configure the WebSphere Studio test environment. Use the following steps to do this:

1. Switch to the J2EE Hierarchy view in the J2EE Perspective.
2. Expand the **Servers** folder and double-click the required server. We are using WebSphere v5.0 Test Environment.
3. When the server configuration editor opens, select the **J2C** tab.
4. In the Node Settings, J2C Resource Adapters section, click **Add**.
5. Select **CICS ECI Connector** for the Resource Adapter Name and click **OK**.
6. Select the **CICS ECI Connector** in the J2C Resource Adapters table and click **Add** in the J2C Connection Factories section.
7. In the Create Connection Factory window, set the Name to CICSECI, set the JNDI name to eis/CICSECI, and click **OK**.
8. In the Resource Properties section, enter the details for your CICS Transaction Gateway. These settings should be the same as those used when creating the enterprise service (see Figure 11-5 on page 245).

9. Save and close the server configuration.



J2C Options

Scope: localhost/localhost/server1

Node Settings

Edit the J2C configuration settings.

J2C Resource Adapters:

Resource Adapter name	Display name	Vendor name	Spec version
CICS ECI Connector	ECIResourceAdapter	IBM	5.0.0
WebSphere Relational ...			

J2C Connection Factories:

Name	JNDI name	Description
CICSECI	eis/CICSECU	

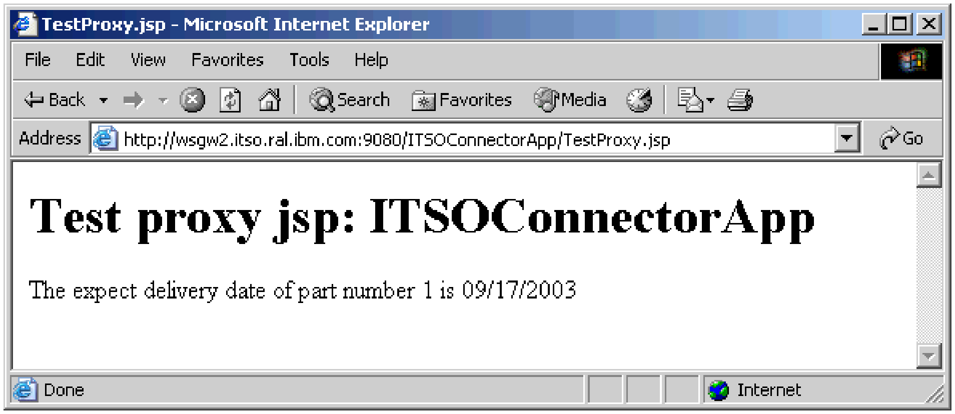
Resource Properties:

Name	Type	Value
ServerName	java.lang.String	DUMMY
ConnectionURL	java.lang.String	tcp://localhost
PortNumber	java.lang.String	2006
UserName	java.lang.String	USERID

Figure 11-11 J2C settings for the WebSphere Studio test environment

Run TestProxy.jsp

To run the test JSP, right-click **TestProxy.jsp** and select **Run on Server...**. The result should look similar to Figure 11-12.


TestProxy.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wsgw2.itso.ral.ibm.com:9080/ITSOConnectorApp/TestProxy.jsp
Go
Done
Internet

Test proxy jsp: ITSOConnectorApp

The expect delivery date of part number 1 is 09/17/2003

Figure 11-12 TestProxy.jsp successful test result

This test checks that the generated Java proxy can invoke the CICS service via the CICS Transaction Gateway and that the data types involved are handled correctly.

11.3.3 Deploying generated Java classes to WebSphere Enterprise

The generated Java proxy classes must be deployed to the application server to put our solution into production. In this section we produce two different packages:

- A JAR file containing Java classes that must be included in the WebSphere Application Server classpath. The Web Services Gateway needs a package to find the proxy and data types classes for invoking the CICS service.
- An EAR file of the ITSOConnectorApp J2EE project. This package contains the WSDL definitions and the test JSP.

To create the required packages:

1. Open the J2EE Perspective.
2. Right-click **ITSOConnectorApp** J2EE project and select **Export...** → **EAR file**. Export the project to ITSOConnectorApp.ear.
3. Right-click the **Java Source** folder of the ITSOConnectorAppWeb project and select **Export...** → **JAR file**. Export the folder to ITSOConnectorApp.jar.
4. Deploy the exported EAR file in the IBM WebSphere Application Server Enterprise V5.0 runtime.
5. Copy the exported JAR file to somewhere in the WebSphere Application Server classpath, such as the <WAS_HOME>\lib\app folder.

11.3.4 Configuring a J2C connection factory in WebSphere

In this section we describe the steps needed to set up a CICS J2C connection factory in WebSphere Application Server for accessing the CICS Transaction Gateway.

Adding the cicsecitools.jar

The WebSphere Studio Integration Edition environment uses a slightly different version of CICS ECI Resource Adapter from the one provided with the CICS Transaction Gateway. The Studio version contains the cicsecitool.jar file that provides the Web Services Invocation Framework support.

To enable the WSIF support in WebSphere Application Server, copy the cicsecitools.jar file from the CICS ECI Connector\connectorModule folder in your Studio workspace to the <WAS_HOME>\lib directory on the application server.

Installing the CICS ECI Resource Adapter

The cicseci.rar file is included in IBM CICS Transaction Gateway, which is installed during the WebSphere Studio installation. It can be found in the <CTG_HOME>\deployable folder, where <CTG_HOME> is the install path of the CICS Transaction Gateway.

To install the CICS ECI Resource Adapter on the production server machine:

1. Start the WebSphere Administrative Console.
2. In the navigation frame on the left, navigate to **Resources** → **Resource Adapters**.
3. Click **Install RAR** in the Resource Adapters form on the right.
4. Browse to the cicseci.rar file and click **Next**.
5. In the Configuration form, click **OK** to accept the default settings.

Adding a connection factory

To add a J2C connection factory using the CICS ECI adapter:

1. In the WebSphere Administrative Console navigation frame, navigate to **Resources** → **Resource Adapters**.
2. Click the newly created resource adapter, **ECIResourceAdapter** in our case, in the Resource Adapters form on the right.
3. Scroll down the right form and click the **J2C Connection Factories** link.
4. In the J2C Connection Factories form, click **New**.
5. In the J2C Connection Factory form, set the Name to CICSECI, set the JNDI name to eis/CICSECI, and click **OK**.

6. Back in the J2C Connection Factories form, click the newly created **CICSECI** connection factory.
7. Scroll down the right form and click the **Custom Properties** link.
8. In the Custom Properties form, enter the details for your CICS Transaction Gateway. These settings should be the same as those used when creating the enterprise service (see Figure 11-5 on page 245).
9. Save your changes and restart the application server.
10. Test application server CICS connectivity using the TestProxy.jsp:

    ```
    http://localhost:9080/ITSOConnectorApp/TestProxy.jsp
    ```

11.3.5 Configuring the service in Web Services Gateway

After developing and deploying the enterprise service, we are ready to deploy the Web Services Gateway service that will expose our enterprise service as a Web service. Notice that the application developed is not a Web service, but only a Java class. The Web Services Gateway creates the actual implementation of the Web service using the Java class to process client requests.

Deploying the gateway service

To deploy InventoryCicsProxy.wsdl as a gateway service:

1. Configure the gateway and deploy the SOAPHTTPChannel1, as described in 10.4.1, “Installing and configuring the Web Services Gateway” on page 220.

   When configuring the gateway, we used the following settings:

   – Namespace URI for services: `urn:wsgw2.itso.ral.ibm.com`

   – WSDL URI for exported definitions: `http://wsgw2.itso.ral.ibm.com:9080/wsgw`

2. Open the Web Services Gateway systems administration console and click **Services** → **Deploy** in the navigation panel on the left.
3. In the Deploy Gateway Service window, we set the following fields:

   – Gateway Service Name: `InventoryCicsWsgw`

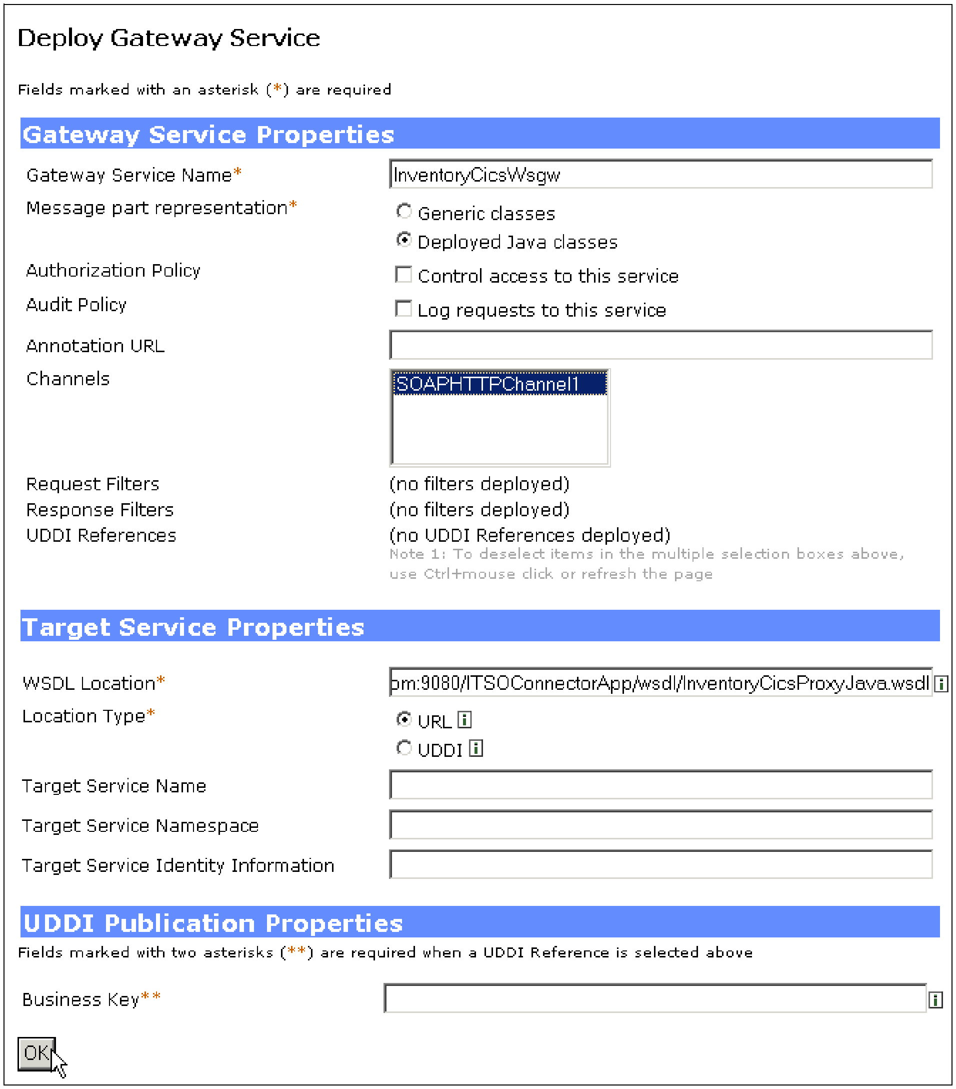
   – Message part representation: `Deployed Java classes`

   – Channels: click to select **`SOAPHTTPChannel1`**

   – WSDL Location: `http://wsgw2.itso.ral.ibm.com:9080/ITSOConnectorApp/wsdl/InventoryCicsProxyJava.wsdl`

   – Location Type: `URL`

We accepted the defaults for the remaining fields. Our gateway service settings are shown in Figure 11-13.



Deploy Gateway Service

Fields marked with an asterisk (*) are required

Gateway Service Properties

Gateway Service Name*	InventoryCicsWsgw
Message part representation*	○ Generic classes ● Deployed Java classes
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*	om:9080/ITSOConnectorApp/wsdl/InventoryCicsProxyJava.wsdl
Location Type*	● URL ○ UDDI
Target Service Name	
Target Service Namespace	
Target Service Identity Information	

UDDI Publication Properties

Fields marked with two asterisks (**) are required when a UDDI Reference is selected above

Business Key**	

Figure 11-13 Deploying a gateway service

4. Click **OK** to deploy the service.

Exporting the WSDL file

When the service is deployed, the gateway generates new WSDL files that can be shared with clients of the enterprise service. The gateway-generated WSDL implementation definition file has the gateway as the service end-point, and it imports the WSDL interface definition file that contains bindings and portType information.

To export the WSDL file generated by the Web Services Gateway:

1. Open the Web Services Gateway systems administration console and click **Services** → **List** in the navigation panel on the left.
2. In the List of Gateway Services window, click the required service, **InventoryDocWsgw** in our case.
3. In the Service: InventoryDocWsgw window:
   a. Scroll down to the Exported WSDL definitions section.
   b. Right-click **External WSDL implementation definition (WSDL only)** and select **Save Target As...** from the pop-up menu.
   c. Save the WSDL file to the required location. We saved the file as InventoryDocWsgw.wsdl under the ITSOSourceAppWeb\WebContent\WEB-INF\wsdl folder in our WebSphere Studio workspace.

You are now ready to Web service-enable your source application using the gateway-generated WSDL implementation definition file for your target service.

11.3.6 Web service enabling the source application

See Figure 8-11 on page 167 for an overview of the Web service development process for a source application. Let's walk through this process for our source application.

To Web service enable the source application using the gateway generated WSDL implementation definition file:

1. Open a command window.
2. Using the gateway WSDL file exported in "Exporting the WSDL file" on page 258, generate the Web service client deployment descriptors and classes using the **WSDL2Java** tool. We used the command shown in Example 11-2.

Example 11-2 Generating client deployment descriptors and classes using WSDL2Java

```
C:\WebSphere\AppServer\bin\WSDL2Java -verbose -role client -container web
   -NStoPkg http://wsdl/InventoryCicsProxyJava/=com.ibm.itso.es.inventory
   -NStoPkg http://wsdl/InventoryCicsProxy/=com.ibm.itso.es.inventory
   -output C:\workspace\ITSOSourceAppWeb\WebContent
   C:\workspace\ITSOSourceAppWeb\WebContent\WEB-INF\wsdl\InventoryCicsWsgw.wsdl
```

```
WSWS3185I: Info: Parsing XML file:  C:\...\WEB-INF\wsdl\InventoryCicsWsgw.wsdl
Retrieving document at
'http://wsgw2.itso.ral.ibm.com:9080/wsgw/ServiceInterface?name=InventoryCicsWsgw', relative to
'C:\...\WEB-INF\wsdl\InventoryCicsWsgw.wsdl'.
Retrieving document at
'http://wsgw2.itso.ral.ibm.com:9080/wsgw/ServiceImport?name=InventoryCicsWsgw&uri=http%3A%2F%2F
wsgw2.itso.ral.ibm.com%3A9080%2FITSOConnectorApp%2Fwsdl%2FInventoryData.xsd', relative to
'http://wsgw2.itso.ral.ibm.com:9080/wsgw/ServiceInterface?name=InventoryCicsWsgw'.
Retrieving document at
'http://wsgw2.itso.ral.ibm.com:9080/wsgw/ServiceImport?name=InventoryCicsWsgw&uri=http%3A%2F%2F
wsgw2.itso.ral.ibm.com%3A9080%2FITSOConnectorApp%2Fwsdl%2FInventoryCicsProxy.wsdl', relative to
'http://wsgw2.itso.ral.ibm.com:9080/wsgw/ServiceInterface?name=InventoryCicsWsgw'.
WSWS3204E: Error: {http://inventory.es.itso.ibm.com/}InventoryData already exists.
WSWS3282I: Info: Generating C:\...\itso\es\inventory\InventoryData.java.
WSWS3282I: Info: Generating C:\...\itso\es\inventory\InventoryData_Helper.java.
WSWS3282I: Info: Generating C:\...\itso\es\inventory\InventoryData_Ser.java.
WSWS3282I: Info: Generating C:\...\itso\es\inventory\InventoryData_Deser.java.
WSWS3282I: Info: Generating C:\...\ral\itso\wsgw2\InventoryCicsWsgw.java.
WSWS3282I: Info: Generating C:\...\ral\itso\wsgw2\InventoryCicsWsgwLocator.java.
WSWS3282I: Info: Generating C:\...\itso\es\inventory\InventoryCicsProxy.java.
WSWS3282I: Info: Generating C:\...\es\inventory\InventoryCicsProxySOAPHTTPBindingStub.java.
WSWS3282I: Info: Generating C:\...\WEB-INF\webservicesclient.xml.
WSWS3282I: Info: Generating C:\...\WEB-INF\ibm-webservicesclient-bnd.xmi.
WSWS3282I: Info: Generating C:\...\WEB-INF\ibm-webservicesclient-ext.xmi.
WSWS3282I: Info: Generating C:\...\WEB-INF\InventoryCicsWsgw_mapping.xml.
```

3. In WebSphere Studio, move the generated Java source files from the Web module's WebContent folder to its JavaSource folder:

   a. Right click the **ITSOSourceAppWeb** project and select **Refresh** from the pop-up menu. The generated files should now appear in the Studio workspace.

   b. Move the **com.ibm.itso.es.inventory** and **com.ibm.ral.itso.wsgw2** packages in the ITSOSourceAppWeb\WebContent folder to the ITSOSourceAppWeb\JavaSource folder.

4. Add client application code to invoke the Web service on the target application.

   To invoke GETDATE CICS function via the Web Services Gateway, we added the com.ibm.itso.command.GatewayCicsBean command bean in our ITSOSourceAppWeb module.

5. Test, then deploy the source application in your IBM WebSphere Application Server V5.0.2 runtime environment.

11.4 Quality of Service capabilities

The components used in this scenario are covered individually in other chapters. The QoS capabilities of the solution presented are therefore closely related to those presented in other chapters.

For QoS capabilities of Web services in general, see 8.6, “Quality of Service capabilities” on page 177.

For QoS related to the IBM Web Services Gateway, see 10.5, “Quality of Service capabilities” on page 233.

For QoS details regarding J2EE Connectors, see 12.4, “Quality of Service capabilities” on page 272.

12

Using J2EE Connectors

This chapter discusses using J2EE Connectors in an intra-enterprise integration scenario. We are using the J2EE Connector support provided with IBM WebSphere Application Server base V5.0 and IBM CICS Transaction Gateway V5.0 in this scenario.

This chapter describes the following:

- Using J2EE Connectors in the context of our ITSO Electronics business scenario.
- Design guidelines for using J2EE Connectors and CICS.
- Development guidelines for using J2EE Connectors and CICS.
- Quality of Service capabilities for J2EE Connectors and CICS.
- Best practices for J2EE Connectors and CICS.

This chapter is based on the J2EE Connector scenario described in redbook *Patterns: Self-Service Application Solutions Using WebSphere Application Server V5*, SG24-6591.

12.1 Business scenario

In this scenario, ITSO Electronics has a legacy system running some of its core business processes. The intention is not to replace or re-engineer this system, but to allow more modern and flexible J2EE applications to reuse the important business processes that this core system hosts. The goal is to leverage legacy systems in a way that enables ITSO Electronics to be responsive to their customer's needs.

The integration functionality to be implemented is explained in 6.2.2, "Stage II: Internal ordering on demand with delivery date" on page 119. The retail system must access, in real time, an existing wholesale system to check the availability of a product and obtain its estimated delivery date. For the scenario covered in this chapter, the legacy wholesale system is a CICS server located on the internal network.

If possible, it is preferable to use off-the-shelf products to provide connectivity between the J2EE application server and CICS because they are usually faster to implement, and have wider support and lower costs.

Note: We didn't implement this solution in the ITSO Electronics sample application provided with this redbook. For a sample J2EE Connector to CICS implementation, see the redbook *Patterns: Self-Service Application Solutions Using WebSphere Application Server V5*, SG24-6591.

12.2 Design guidelines

Figure 12-1 shows the Runtime pattern and Product mapping for the Call Connection variation of the Direct Connection application pattern within the business domain of an organization, using a J2EE Connector.

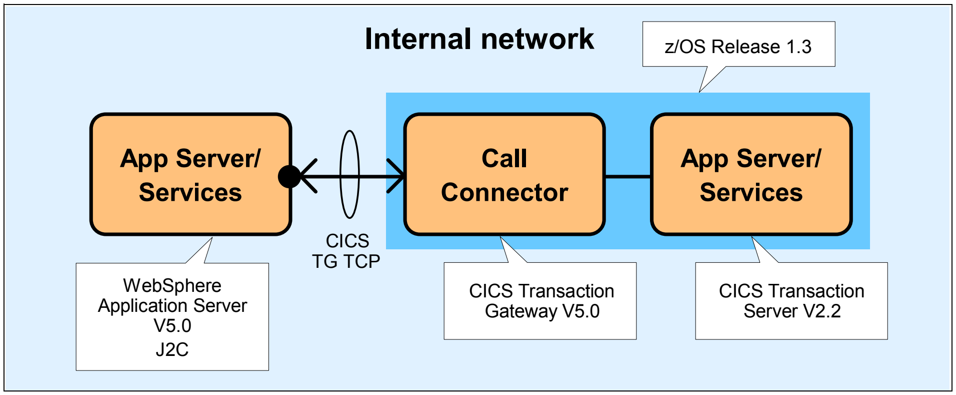


Figure 12-1 Direct Connection::Call Connection: J2EE Connector Product mapping

This product mapping uses the CICS Transaction Gateway TCP protocol to communicate with the CICS Transaction Gateway on the zSeries enterprise system. The source J2EE application uses the CICS ECI J2EE Connector to access the existing CICS enterprise application via the CICS Transaction Gateway.

The following sections outline design guidelines for application integration using the J2EE Connector Architecture. We discuss the Common Connector Interface (CCI) API, the CICS resource adapters, and the IBM WebSphere Application Server V5.0 J2EE Connector environment.

12.2.1 Components of J2EE Connector Architecture

As shown in Figure 12-2 on page 266, Version 1.0 of the J2EE Connector Architecture defines a number of components and interfaces that make up this architecture:

- Common Client Interface (CCI)

  The CCI defines a common API for interacting with resource adapters. It is independent of a specific EIS. A Java developer communicates to the resource adapter using this API.

- System contracts

  A set of system-level contracts between an application server and EIS. These extend the application server to provide:

  - Connection management
  - Transaction management
  - Security management

These system contracts are transparent to the application developer, meaning that they do not implement these services themselves.

- Resource adapter deployment and packaging

  A resource adapter provider develops a set of Java interfaces/classes as part of its implementation of a resource adapter. The Java interfaces/classes are packaged together with a deployment descriptor to create a Resource Adapter Archive (represented by a file with an extension of .rar). This Resource Adapter Archive is used to deploy the resource adapter into the application server.

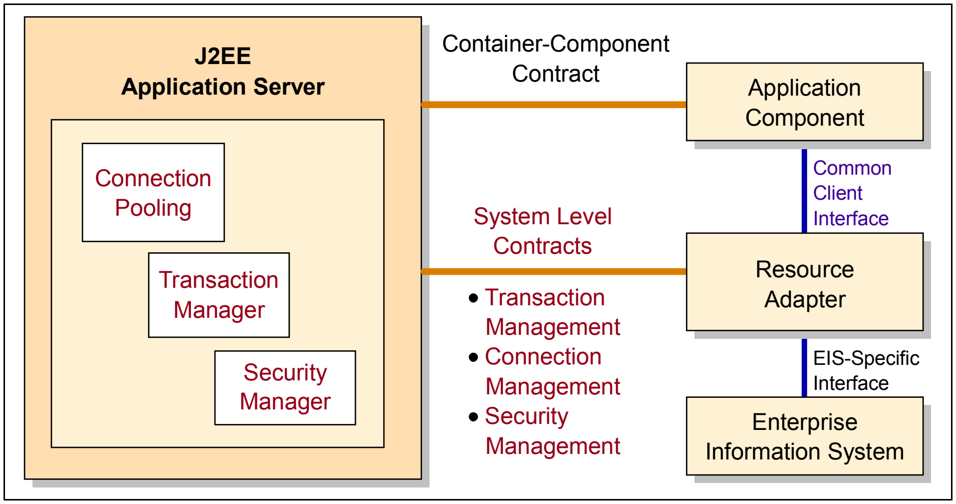


Figure 12-2 J2EE Connector Architecture components

12.2.2 Design considerations

A number of factors affect the design of systems using J2EE Connectors. In this section we discuss some of the issues that should be considered when building an application.

Managed and non-managed environments

There are two different types of environments that a Java application using J2EE Connectors can run in:

- Managed environment: Management of connections, transactions, and security is provided by an application server. The Java application developer does not have to code this management manually.

  To use the managed environment, you need to define a J2C connection factory in the J2EE application server. At deployment time you can then map a resource reference in your J2EE application to the JNDI name of the J2C connection factory.

- Non-managed environment: The Java application directly uses the resource adapter to access an EIS. Management of connections, transactions, and security must be handled manually by the application.

  You can find further details in the WebSphere Developer Domain article *Using J2EE Resource Adapters in a Non-managed Environment* at:

  `http://www7b.boulder.ibm.com/wsdd/library/techarticles/0109_kelle/0109_kelle.html`

CICS resource adapters

The CICS Transaction Gateway (CTG) is a set of client and server software components that allow a Java application to invoke services in a CICS region. The Java application can be an applet, a servlet, an enterprise bean, or any other Java application.

Two J2EE Connector CICS resource adapters are provided with the IBM CICS Transaction Gateway (CTG):

- ECI (External Call Interface) is a call interface to COMMAREA-based CICS applications. The J2EE Connector RAR file is cicseci.rar.
- EPI (External Presentation Interface) is an API to invoke 3270-based transactions. The J2EE Connector RAR file is cicsepi.rar.

Selecting a CICS resource adapter

Characteristics of the two CICS resource adapters and the situations in which each would be selected are:

- External Call Interface: ECI uses COMMAREA as an interface to a CICS enterprise application. If the enterprise application is not using COMMAREA as an interface, it needs to be modified to use COMMAREA. ECI has a simple calling type interface rather than the screen-oriented, conversational type interface of EPI. For this reason, we recommend that ECI be used for new enterprise applications that will be Web-enabled.
- External Presentation Interface: EPI uses 3270 data stream as an interface to a 3270 CICS application. If the enterprise application is a 3270 CICS application, EPI should be used for the resource adapter. There is no need to change the enterprise 3270 application at all. Using J2EE Connector Architecture CCI, the EPI application can use the same interface as ECI, but the underlying interface is conversational.

Table 12-1 summarizes the characteristics of ECI and EPI.

Table 12-1 CICS ECI and EPI characteristics

	ECI	EPI
Protocol type	Remote call	Conversational
Interface	COMMAREA	3270 data stream
Max data length	32 KB	Screen size (eg 24x80) plus control characters
CTG JCA support	Distributed or z/OS	Distributed only
Recommendation	Use with new applications or existing COMMAREA based applications	Use with existing 3270 applications only

Synchronous versus asynchronous calls

You can avoid a blocking call when you invoke an enterprise application using asynchronous CCI calls. This allows your Java application to call a CICS program without blocking while waiting for the response from CICS.

A non-blocking call type option can be specified for the CCI InteractionSpec using setInteractionVerb as follows:

```
myInteractionSpec.setInteractionVerb(ECIInteractionSpec.SYNC_SEND);
```

Synchronous calls can be specified as follows:

```
myInteractionSpec.setInteractionVerb(ECIInteractionSpec.SYNC_SEND_RECEIVE);
```

CICS ECI design considerations

Some application design considerations when selecting the CICS ECI resource adapter are:

- If your legacy CICS application does not use COMMAREA interface, it must be changed to use COMMAREA.
- The COMMAREA size and interaction complexity. For performance reasons, the size of COMMAREA and the number of interactions between the Web application and enterprise application should be minimized. The maximum COMMAREA size is 32 KB.
- DPL considerations. In the CICS world, ECI calls are treated as Distributed Program Link (DPL) calls. Refer to *CICS Application Programming Guide*, SC33-1687 for details on DPL considerations.

12.3 Development guidelines

In this section we take a look at development guidelines for J2EE Connectors. Using IBM WebSphere Studio Application Developer V5.0, there are two ways to develop J2EE Connector applications:

- Using CCI: An application component uses CCI (Common Client Interface), which is provided by a resource adapter. This is a standard way of developing J2EE Connector applications, regardless of the development tools.
- Using the Enterprise services toolkit: Using WebSphere Studio Integration Edition or WebSphere Studio Enterprise Developer, you can develop a J2EE Connector application as an Enterprise Service.

12.3.1 Creating a J2EE Connector application using native CCI

You can implement J2EE Connector connectivity using native CCI in your application with the following steps:

1. Configure your J2EE Connector resource adapter and connection factory in your integrated development environment.
2. Create an input and output Record class.

   These records implement javax.resource.cci.Record and perform the input and output conversions between application Java data structures and enterprise tier data structures. You can manually code classes using the CCI record framework or you can use a tool such as the VisualAge for Java Enterprise Access Builder. The Enterprise Access Builder allows you to create a CCI record from an existing C or COBOL structure.

   > **Note:** The basic configuration of WebSphere Studio Application Developer does not provide a tool to import a C or COBOL structure into a CCI record. With the Enterprise Services toolkit in WebSphere Studio Application Developer Integration Edition, you can import a C or COBOL structure into an enterprise service definition.

3. Get an instance of the required J2EE Connector ConnectionFactory (usually through a JNDI lookup).
4. Get an instance of J2EE Connector Connection from the connection factory.
5. Create an instance of input and output Record using classes developed in step 2.
6. Create an Interaction instance from the Connection.
7. Create an InteractionSpec and set the required properties.

8. Execute the Interaction, passing the InteractionSpec, input Record, and output Record.
9. Close the Interaction and Connection.

12.3.2 Enterprise Services toolkit

WebSphere Studio Application Developer Integration Edition includes a set of tools and wizards, collectively referred to as the Enterprise Services toolkit. The Enterprise Services toolkit is a fully service-oriented development environment for business and enterprise application integration.

At the heart of the Enterprise Services toolkit programming model are Enterprise Services, or Services for short. Services are used to model different kinds of service providers in a consistent way. Figure 12-3 shows the currently supported providers. Note that in the Enterprise Services world a Web service is just one form of service provider. J2EE Connectors are another.

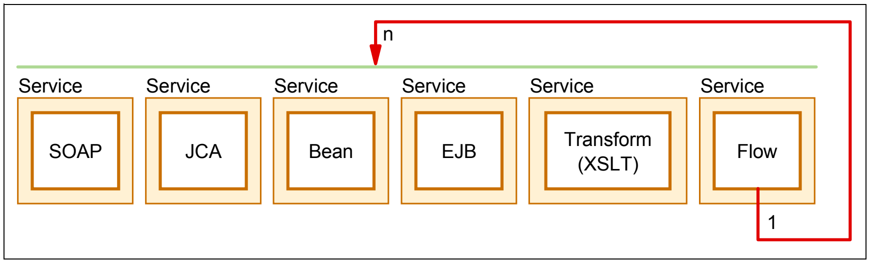


Figure 12-3 Services supported by WebSphere Studio Integration Edition

The Web Services Invocation Framework (WSIF) provides a standard API to invoke services, no matter how or where the service is provided, as long it is described in WSDL. This API is used by tools such as WebSphere Studio Integration Edition, and runtimes such as IBM WebSphere Application Server V5.0, to construct and manipulate services defined in WSDL documents. The architecture allows new bindings to be added at runtime.

The J2EE Connector Tool Plug-in makes it possible to plug a J2EE Connector-compliant EIS resource adapter, such as the CICS ECI adapter, into the Enterprise Services toolkit provided with WebSphere Studio Integration Edition.

Using the Enterprise Services toolkit, there’s no need to write any J2EE Connector code because the Service Definition wizard in WebSphere Studio Integration Edition guides you through the service definition. The wizard generates a WSDL file providing the WSIF with all the information needed to connect to the enterprise tier and to invoke the enterprise application.

12.3.3 Using Enterprise Services toolkit

You can implement J2EE Connector connectivity in your application using the Enterprise Services toolkit (provided in WebSphere Studio Integration Edition) with the following steps:

1. Create a J2EE Connector service definition from the corresponding resource adapter in the Service Provider Browser. In the Service Definition wizard you define the following:

   - Port and portType
   - Operation
   - Binding
   - Input and output message(s)

     A message corresponds to a J2EE Connector Record. You can import the message definition from an existing C or COBOL structure.

2. Deploy the J2EE Connector service. The Service Deployment wizard allows you to deploy services into WebSphere. By default, the wizard deploys the service as a session EJB. It can also deploy the service as a SOAP service.

3. Build your application client for accessing the J2EE Connector service. Create an application EJB or servlet, for example, to access the deployed J2EE Connector enterprise service.

12.3.4 Migration to other J2EE Connector resource adapters

The J2EE Connector Architecture CCI provides a common programing interface to application component developers. When you migrate the application component to a different adapter, you only need to change the method calls that are specific to that resource adapter. The classes that are specific to a resource adapter are:

- Input and output record classes
- ConnectionSpec
- InteractionSpec

For example, IMSConnectionSpec has getGroupName and setGroupName, which are unique to the IMS resource adapter for specifying the IMS Group. Refer to the resource adapter documentation for details.

When you are developing an enterprise service using WebSphere Studio Integration Edition, all the code is generated by the tool. Regenerate the code using a new resource adapter to migrate your application.

12.4 Quality of Service capabilities

In this section we discuss Quality of Service capabilities and considerations specific to J2EE Connectors and CICS.

12.4.1 Autonomic

This section briefly discusses some of the tools important for fault monitoring and isolation.

Tivoli Performance Viewer

The Tivoli Performance Viewer is a graphical performance monitor for IBM WebSphere Application Server V5.0. You can use the Performance Viewer to retrieve performance data from application servers. Data is collected continuously by application servers and retrieved as needed from within the Viewer.

The JCA Connection Pools resource category provides information about J2EE Connectors, such as the number of managed connections (ManagedConnections) and the number of connections handles (Connections).

Logging and tracing

It is often helpful to examine log and trace files when your application experiences J2EE Connector errors or problems:

- Application logging: It is always important for applications to record their activity to a logging facility. When you write a log to the standard output file or standard error file, the application server will record it to the corresponding log files.
- Connection factory trace: Connection factory classes can be traced using the WebSphere Application Server trace service. The trace level can also be set as a connection factory property in WebSphere Administrative Console. Connection factory tracing is often not particularly helpful when debugging the CICS interaction. CTG tracing is usually the better option.
- CICS Transaction Gateway (CTG) trace: CTG trace records detailed activities of the CTG gateway process, such as processing of ECI requests from clients. Of the four levels of CTG tracing, JNI tracing between the CTG and the native client is usually the most useful. The application can enable CTG tracing programmatically. It can also be enabled dynamically in the Gateway daemon using the TCPAdmin protocol handler, or statically as a start option. CTG trace is recorded in the standard output file or standard error file.
- External CICS Interface (EXCI) trace: The EXCI provides a programming interface for the non-CICS address space to invoke CICS programs. CTG

utilizes EXCI to communicate with CICS program. The CTG writes trace entries to the EXCI trace when it issues an EXCI request. The trace entries in a dump can be printed using standard z/OS utilities (GTF).

- CICS trace: CICS Transaction Server provides a facility for recording CICS activity. In CICS for MVS™, there are three destinations for trace entries: internal trace, auxiliary trace, and generalized trace facility (GTF).

12.4.2 Availability

The approaches discussed in “Scalability and availability considerations” on page 274 also provide the node redundancy and failover capabilities needed to eliminate single-point-of-failure for end-to-end production systems.

12.4.3 Performance

The connection management contract defined by the J2EE Connector Architecture gives an application component a connection to an EIS. To deliver performance and scalability, the connection management contract should support connection pooling and management.

The J2EE Connector Architecture Specification V1.0 explains in detail how the connection contact is implemented by various connection management components of the application server and the resource adapter.

There are several parameters you can set to optimize connection pooling properties using WebSphere Administrative Console:

- Connection timeout
- Maximum connections
- Minimum connections
- Reap time
- Unused timeout

See the WebSphere InfoCenter for further details.

The CICS ECI resource adapter supports connection pooling of the connections (and underlying objects) from the J2EE application server to the CICS Transaction Gateway daemon.

Scalability and availability considerations

As shown in Figure 12-4 on page 275, there are several scalability and availability options when using J2EE Connectors to access CICS enterprise applications:

- EJB workload management, as provided by WebSphere Application Server.

  EJBs deployed in IBM WebSphere Application Server Network Deployment V5.0 can automatically take advantage of the WebSphere workload management (WLM) facility for EJBs. Refer to the redbook *IBM WebSphere V5.0 Applications: Ensuring High Performance and Scalability*, SG24-6198 for details.

- Inbound CTG requests (TCP or HTTP) can be workload managed using various methods. CTG for z/OS options for workload managing requests across LPARs include:

  - IBM Load Balancer (a component of IBM WebSphere Edge Server) can be used to perform load balance inbound requests to CTG for z/OS.
  - Sysplex distributor uses a cluster IP address to provide enhanced WLM across the LPARs in a sysplex.

  TCP/IP port sharing provides a simple way of workload balancing HTTP requests across multiple regions within an LPAR.

- CICS requests (ECI or EPI) can be workload managed using CICS scalability technologies. Workload management functions that are applicable when CICS Transaction Gateway resides in zSeries include:

  - External CICS Interface (EXCI) allows basic workload balancing by including a simple CICS availability check in the CTG DFHXCURM module.
  - CICS multi-region operation (MRO) is a widely used technique that is a central part of CICS scalability.
  - CICS distributed program link (DPL) requests from the CTG can be routed to a CICS program that resides in any CICS region in a sysplex.

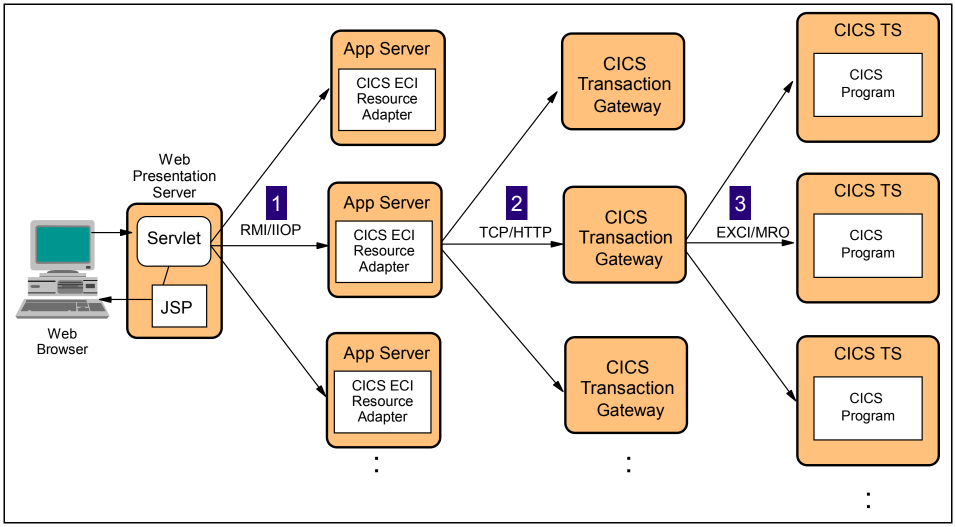


Figure 12-4 Scalability options

12.4.4 Security

The J2EE Connector Architecture security contract extends the J2EE security model to provide secure connections to EIS. To create a connection to an EIS, there must be some form of signing on to the EIS, to authenticate the connection requester. Re-authentication can also take place if supported by the EIS. This occurs when the security context is changed after a connection is made. (For example, connection pooling could cause a re-authentication when the connection is redistributed.)

The application component has the following two choices related to EIS sign-on:

- Container managed sign-on: The deployer sets up the resource principal and EIS sign-on information. For example, the deployer sets the user name and password for establishing a connection to an EIS instance.
- Component managed sign-on: Application code in the component performs the sign-on to an EIS by explicitly specifying the security information for a resource principal.

If you choose component managed sign-on, you need to specify user name and password at an instance of ConnectionSpec.

WebSphere V5.0 supports container managed sign-on with the use of a user ID and password credential (Option A in the J2EE Connector Architecture Specification) and component managed sign-on (Option C in the J2EE Connector Architecture Specification).

WebSphere V5.0 for z/OS can also flow an authenticated user credential through the CICS Transaction Gateway into CICS server when using container managed sign-on.

Signing on to the enterprise tier

Authentication can be performed against the Resource Access Control Facility (RACF®) using user ID and password authentication in the CTG for z/OS, by setting the variable AUTH_USERID_PASSWORD=YES in the ctgenvvar script.

If the CTG runs on a distributed platform, it is possible to use the ESIRequest to verify user IDs and passwords with the destination CICS region. On all platforms, it is also possible to use SSL client certificates to authenticate the CTG.

SSL encryption support

The client application must specify a CTG network protocol when it connects to the gateway daemon. There are basically two types of connections:

- TCP/IP sockets (TCP)
- HTTP sessions (HTTP)

For each of these, there is a secure version using SSL, namely SSL and HTTPS.

> **Note:** The local protocol can be used if the CICS Transaction Gateway and WebSphere Application Server are on the same machine. This bypasses the Gateway daemon and is also a performance optimizer.

CICS security

CICS uses the z/OS System Authorization Facility (SAF) to route authorization requests to an external security manager (ESM) to perform all its security checks. Any suitable ESM could be used, but because the IBM Resource Access Control Facility (RACF) product is the most commonly used ESM, we refer to RACF when discussing CICS external security.

For more details on CICS security, refer to the following IBM publications:

- *CICS RACF Security Guide*, SC33-1701
- *Securing Web Access to CICS*, SG24-5756

12.4.5 Standards compliance

Resource adapters that are compliant with the J2EE Connector Specification 1.0 are portable across J2EE 1.3 application servers. This makes it easier for vendors to provide resource adapters that support multiple J2EE-compliant application servers.

Using the standard Common Client Interface defined by J2EE Connector Specification 1.0 also simplifies application integration with diverse EISs. This common interface makes it easy to plug third-party or home-grown resource adapters into your applications.

12.4.6 Transactionality

In a non-managed environment, the Java application is responsible for managing transactions through the local transaction interface (provided that the resource adapter supports this). By using the managed environment, the programmer does not even need to think about managing the transaction, because the transaction manager provides this *quality of service*.

A *resource manager* consists of the resource adapter and underlying EIS. It may participate in transactions that are externally controlled by a transaction manager. A *transaction manager* controls and coordinates transactions across multiple resource managers.

A resource manager has three options for supporting transactions:

- No support: The resource manager does not support transactions.
- Local transactions: These are transactions that are managed internally by the resource manager. The coordination of such transactions involves no external transaction manager.
- Global transactions: There are multiple resource managers involved, and an external transaction manager must be used to coordinate the transaction using two-phase commit.

In a managed environment, there are two options to decide where to control the scope and the behavior of global transactions:

- Component managed transaction demarcation: The transaction is explicitly managed by the application component accessing the enterprise tier.
- Container managed transaction demarcation: The transaction is implicitly managed by the container of an enterprise bean accessing the enterprise tier.

The *local transaction optimization* forces the use of one-phase commit in the situation when two-phase commit is not needed for a global transaction. This is when only one resource manager was referenced, so two-phase commit is an unnecessary overhead.

The CICS ECI resource adapter implements the LocalTransaction interface, and supports local transactions. If you use an application server that supports last-resource optimization, such as IBM WebSphere Application Server Enterprise V5.0, the resource adapter can participate in a global transaction provided that it is the only local transaction resource in the global transaction. This is called “Last Participant Support” in WebSphere Enterprise.

12.5 Best practices

Some best practices for J2EE Connector application developers are:

- Use J2EE Connectors in a managed environment.

  From an application developer’s perspective, the greatest benefit of utilizing J2EE Connectors is the Quality of Services (QoS) provided by the system contracts. Choosing the managed environment simplifies the development of scalable, secure, and transactional resource adapters for a wide range of EIS.

- Minimize the resource adapter-specific calls.

  Do not use the resource adapter-specific calls directly if the function is provided by CCI. Encapsulate resource adapter-specific CCI classes—such as ConnectionSpec, InteractionSpec, or the other resource adapter-specific calls—to make the client application more independent of the resource adapters.

- Cache the connection factory to minimize JNDI lookups.

  Looking up the naming context and connection factory from JNDI for each interaction can be expensive. You can improve performance by caching the connection factory.

13

Using Java Message Service

This chapter discusses using Java Message Service (JMS) in an intra-enterprise integration scenario. We are using the JMS support provided with IBM WebSphere Application Server base V5.0 and IBM WebSphere MQ V5.3 in this scenario.

This chapter describes the following:

- Using JMS in the context of our ITSO Electronics business scenario.
- Design guidelines for using JMS and WebSphere MQ.
- Development guidelines for using JMS and WebSphere MQ.
- Quality of Service capabilities for JMS and WebSphere MQ.
- Best practices for JMS.

This chapter is based on the Java Message Service scenario described in the redbook *Patterns: Self-Service Application Solutions Using WebSphere Application Server V5*, SG24-6591.

13.1 Business scenario

In this scenario, ITSO Electronics has a legacy system running some of its core business processes that is accessible using existing messaging middleware infrastructure. The intention is not to replace or re-engineer this system, but to allow more modern and flexible J2EE applications to reuse the important business processes that this core system hosts. The goal is to leverage legacy systems in a way that enables ITSO Electronics to be responsive to their customer's needs.

The integration functionality to be implemented is explained in 6.2.1, “Stage I: Internal ordering on demand” on page 116. The internal retail ordering system needs to be integrated with the internal wholesale system to update inventory as replenishment orders are placed. For the scenario covered in this chapter, the legacy wholesale system is accessible on the internal network using existing IBM WebSphere MQ infrastructure.

If possible, it is preferable to use off-the-shelf products to provide connectivity between the J2EE application server and the wholesale system because they are usually faster to implement, and have wider support and lower costs.

> **Note:** We didn't implement this solution in the ITSO Electronics sample application provided with this redbook. For a sample JMS and WebSphere MQ implementation, see the redbook *Patterns: Self-Service Application Solutions Using WebSphere Application Server V5*, SG24-6591.

13.2 Design guidelines

Figure 13-1 shows the Runtime pattern and Product mapping for the Message Connection variation of the Direct Connection application pattern within the business domain of an organization, using the Java Message Service.

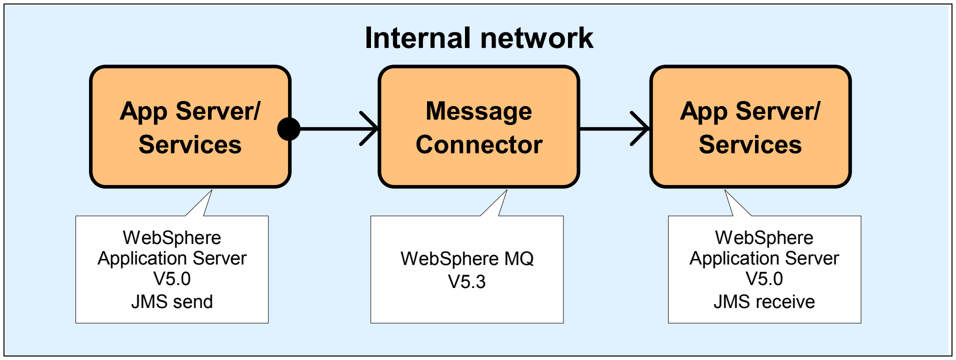


Figure 13-1 Direct Connection::Message Connection: JMS Product mapping

This product mapping uses WebSphere MQ as the transport mechanism for JMS messages. The product mapping uses a WebSphere MQ queue manager on each server to transport the messages. The source application uses JMS to place messages on a local queue. WebSphere MQ is then responsible for ensured delivery of this message to the proper destination, in our case, the WebSphere MQ queue manager on the target application server.

The following sections outline design guidelines for application integration using JMS. We start with an introduction to JMS, then discuss JMS messaging styles, messaging patterns, and provider considerations in the IBM WebSphere Application Server V5.0 JMS environment.

13.2.1 Java Message Service

The Java Message Service (JMS) API enables a Java programmer to access message-oriented middleware such as WebSphere MQ from the Java programming model. JMS has two messaging styles:

- Point-to-point model using queues
- Publish/subscribe model using topics

Communications are asynchronous, so clients can receive messages without making a request and send messages without waiting for a reply. Communications are loosely coupled. The sender and receiver do not have to be active or aware of each other, as the messaging system handles the delivery of messages.

JMS is only a specification. Each enterprise messaging system vendor must provide classes that implement the specification for their specific messaging system.

As shown in Figure 13-2, the JMS architecture has the following components:

- An administration tool for creating JMS connection factories and destinations (or *administered objects*) in the JNDI namespace.
- A JMS provider that is a messaging system implementing the JMS interfaces. J2EE 1.3 application servers, such as WebSphere V5.0, must include a JMS provider.
- JMS clients produce and consume messages. They use JNDI to look up connection factories and destinations so they can connect to the JMS provider to send and receive messages.

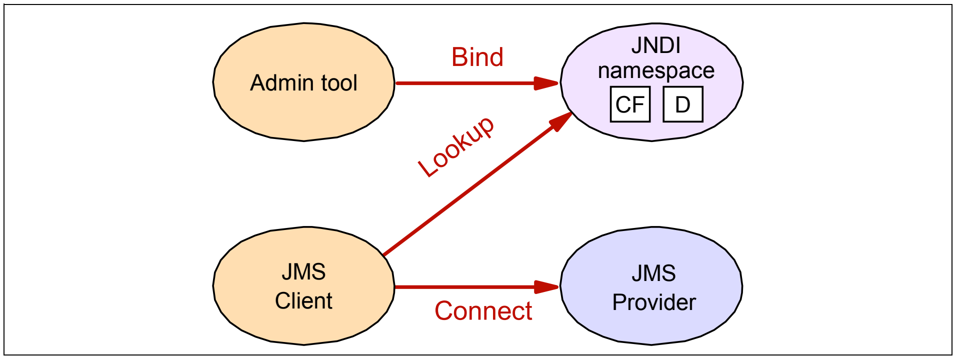


Figure 13-2 JMS components

13.2.2 Design considerations

A number of factors affect the design of systems using JMS. In this section we discuss some of the issues you should consider when building an application.

JMS point-to-point model

As shown in Figure 13-3, point-to-point messaging involves working with queues of messages. One or more clients might send messages to a queue, but a message is taken out by only one client. Messages remain in the queue until they are removed, so the availability of the receiver client does not affect the ability to deliver a message. In a point-to-point system, a client can be a sender (message producer), a receiver (message consumer), or both. In JMS, point-to-point types are prefixed with “Queue.”

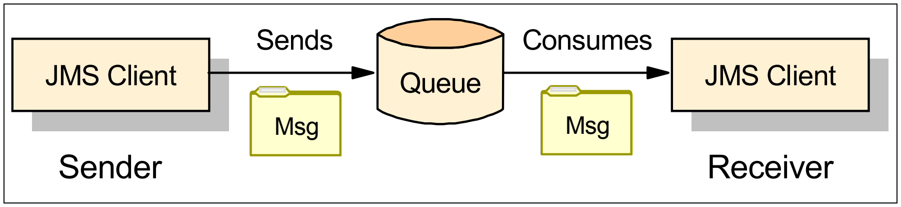


Figure 13-3 JMS point-to-point model

In point-to-point messaging, there are generally three messaging patterns:

- Message producer or send-and-forget
- Message consumer
- Request/reply

We look at these messaging patterns in more detail in “Synchronous versus asynchronous design considerations” on page 284.

JMS publish/subscribe model

In contrast to the point-to-point model of communication, the publish/subscribe model, shown in Figure 13-4, enables the delivery of a message to multiple recipients. A sending client publishes the message to a topic to which multiple clients can be subscribed. A *durable* subscription stores messages when the consumer is not connected, whereas a *non-durable* subscription discards messages when the consumer is not connected. In a publish/subscribe system, a client can be a publisher (message producer), a subscriber (message consumer), or both. In JMS, pub/sub types are prefixed with “Topic.”

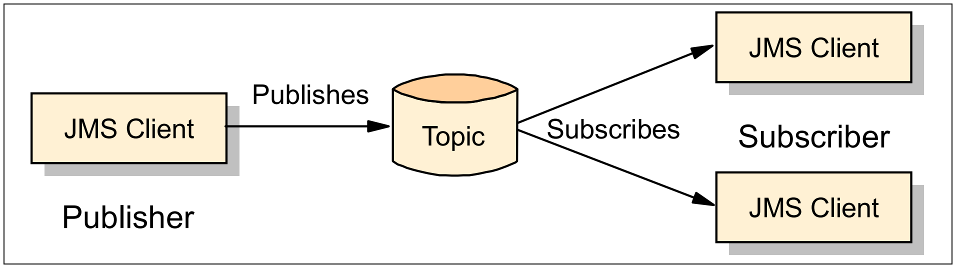


Figure 13-4 Publish/subscribe model

We look at the pub/sub pattern in more detail in “Synchronous versus asynchronous design considerations” on page 284.

JMS messages

Another design choice is the JMS message type to use. JMS provides the following message types:

- BytesMessage
- StreamMessage
- ObjectMessage
- MapMessage
- TextMessage

Each message type contains specific interfaces pertaining to its content and allows specific operations on the messages.

JMS messages are composed of the following parts:

- Header: Contains information to identify and route messages.
- Properties: Custom values that can optionally be added to messages. Properties can be:
  - Application-specific: Properties used by JMS applications.
  - Standard: JMS properties.
  - Provider-specific: Properties that are specific to a messaging provider.
- Body: The message data.

A couple of message properties are also important to look at:

- Delivery mode: When delivery must be assured by the business requirements, persistent messages are needed. But when this is not needed, performance can be gained by the use of non-persistent messages.
- Message expiration: When using non-persistent messages, message expiration can be used to discard messages that have remained on a queue or topic for longer than required. This prevents unprocessed messages from building up over time.

Synchronous versus asynchronous design considerations

In the Web application environment, choosing an asynchronous or synchronous approach to JMS communication will significantly affect the design of the application. The effects could ripple as far as the user interface interaction (or user experience) or they could be felt only in the low-level design and behavior of the underlying application. In this section we look at both the user interaction differences and the system design considerations.

For the purpose of discussion, let's consider an example Web application that provides Web banking and needs to connect to an enterprise application that is hosting the bank account data.

First, it is important to go over some basic Web application principles. The Web is a *stateless* environment; typically a request is received and the reply sent back immediately within the same client session. A Web server is not normally able to initiate a connection to a Web client out of the blue. Information about the requesting client is retained while the request is being serviced and not lost until a reply is sent back. The Web is a typical request/reply model. Most Web applications are built using this model and this style of user interaction, where the user can expect a reply back from the server that will be the result of making a request.

Using our Web banking example, let's assume a Web request requires information from the enterprise application about the bank balance. The JMS interaction between the Web application and the enterprise application can be achieved using:

- Request/reply pattern
- Send-and-forget pattern
- Message consumer pattern
- Publish/subscribe pattern

Request/reply pattern

Using this approach, we fit the standard Web model by providing a complete round trip for the client request that results in a reply. Users do not have to visit another results page to see the results of their request.

As shown in Figure 13-5, the Web application sends a request message, then waits for a reply. The response message needs to be linked to the request message using the request message ID as the correlation ID of the response message.

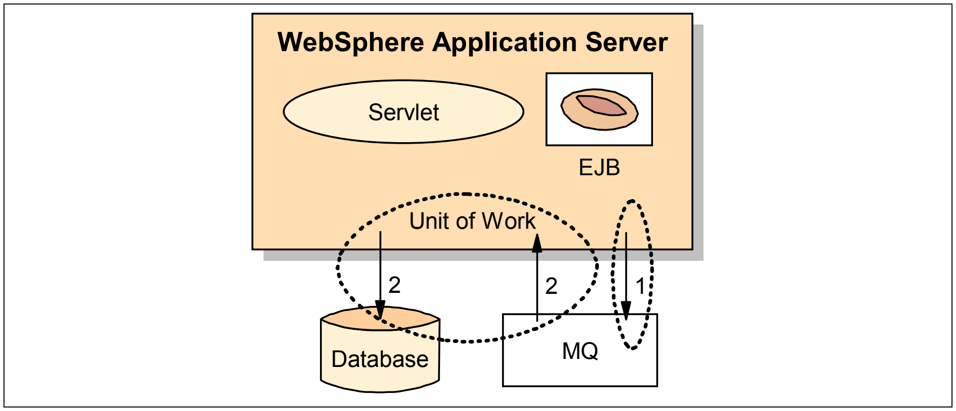


Figure 13-5 Request/reply pattern

The overriding factor in a request/reply pattern is the time delay before a reply gets back. You should remember that request/reply is a synchronous communication over an asynchronous transport. For request/reply, two queues are needed: one for the sender to send messages and one for receiving the responses back. The request/reply consists of two units of work.

- Putting the message on a queue.
- Receiving the response, and for example, inserting the message in a database.

These actions can never be one unit of work because the real put of the message only takes place after the commit. No message will be sent without a commit, and when no request message is sent, no reply will arrive!

An example of a request/reply scenario is getting your account balance. A message is first sent with the account ID, then the application waits until a response message is sent back with the balance of the account, with the results logged to an application database.

Request/reply design considerations include:

- Applications should not be designed without appropriate timeout or re-try capability. Non-persistent messages and message expiration can help with management of reply messages not received within the timeout window.
- If the message producer is implemented as a session EJB, then in the request/reply JMS model, the EJB must wait (or block) until the enterprise application has replied before it can continue processing. Blocking in an EJB is not generally recommended because it restricts the EJB container's ability to effectively manage its resources. Care must be taken to ensure that the EJB is not waiting indefinitely and that there is a timeout in place.

Send-and-forget pattern

In send-and-forget (or fire and forget), shown in Figure 13-6, the Web application will initiate the request to the enterprise application using a JMS destination, but it will not wait for the outcome. This design has important repercussions on the user interaction. A message consumer pattern could be used for receiving the reply from the enterprise application. The user must at some point go to a result page to see their bank balance when it has been retrieved from the enterprise application.

From an implementation point of view, the blocking EJB dilemma is avoided. However, a new page is required to allow the customer to come back to check their last balance request from the local database. This design alleviates the need for a blocking EJB, but the user experience is drastically different from the request/reply model.

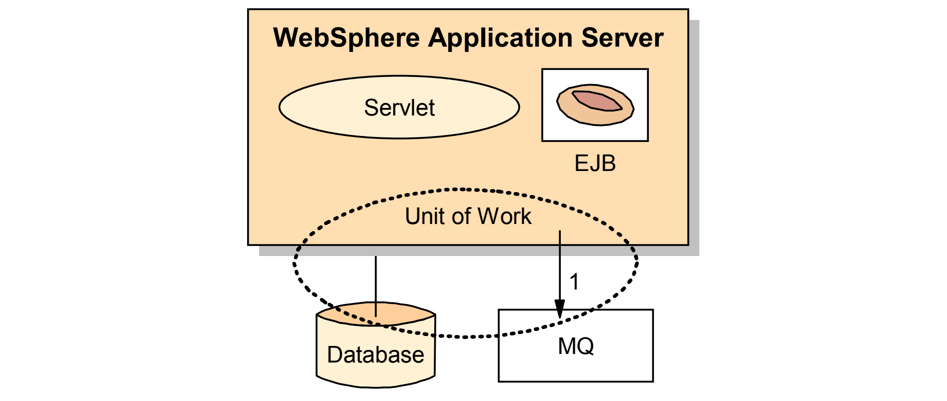


Figure 13-6 Send-and-forget pattern

Send-and-forget design considerations include:

- Non-persistent messages can be processed much quicker by the JMS provider because they do not incur any disk I/O (for persistence). The decision to use persistent or non-persistent messages will generally be governed by the business requirements.

  In the case of getting a balance, no funds transfer occurs; as such, a lost message has little impact and may not warrant persistent messages.

Message consumer pattern

Message consumers can be implemented by message-driven beans that are invoked by the container when a message arrives on a destination. When a message arrives, the EJB container passes the message to an instance of a user-developed message-driven bean.

This pattern can be used by a catalogue application receiving updates for changes in the online catalogue. In this scenario, a message-driven bean receives an incoming message and updates a database, as shown in Figure 13-7. The pattern is typically useful in a business-to-business situation where no user interaction is needed.

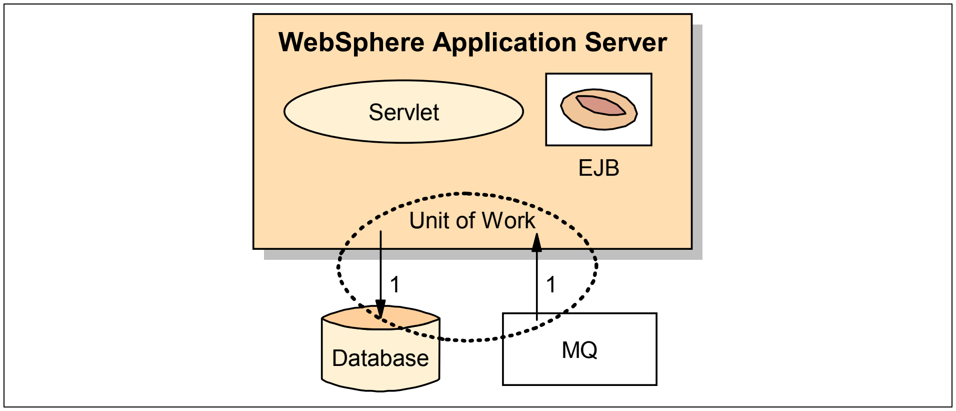


Figure 13-7 Message consumer

Message consumer design considerations include:

- Use message-driven beans only for handling the message. Move the business logic to another bean, which will be invoked by the message-driven bean. This way it is also possible to call the business logic out of another channel, like a servlet, which has been activated by a user.
- Message-driven beans can’t throw exceptions to the user, so exceptions have to be logged in an error report.

Publish/subscribe pattern

Using this approach, we can provide the user with an immediate reply without them having to explicitly go to a separate Web page to see the results. However, this can only be achieved if a local copy of data is used.

The source application will register interest in information from the target application upon startup. Periodically, the target application will publish information to the subscribers (source application). The message consumer pattern can be implemented at the subscriber site to receive the publications of the target application. The source application will store this information in a local database and use it when a Web request is being serviced.

Using this approach, the source application can operate in its native modes (stateless and request/reply) and the user can see the results of their request within the same user transaction. However, the information may be slightly outdated.

Publish/subscribe design considerations include:

- Never cache a non-durable subscription, use durable subscriptions instead.

For further information about how IBM WebSphere MQ can be used in the pub/sub model, refer to the redbook *MQSeries Publish/Subscribe Applications*, SG24-6282.

Selecting a messaging pattern

It is not inherently incorrect to select any of the messaging patterns discussed, provided the selection is *implemented* correctly. The user's requirements and experience will dictate which decision is the correct one.

A request/reply JMS communication model is ideal in a Web environment. However, if EJBs are to be the implementers of the enterprise access, care needs to be taken during implementation to prevent blocking calls from EJBs. If the user is willing to accept a different user interaction model, then asynchronous fire-and-forget is also an acceptable option. The middle ground could be achieved using full publish and subscribe; however, the accuracy of the information may be at stake.

> **Note:** The request/reply blocking stateless EJB *must* be implemented such that appropriate timeout and re-try conditions are applied.
>
> The EJB 2.0 specification does point out that only one client will have access to an instance of a stateless EJB while it is servicing a client-invoked method. If, however, a blocking wait occurs for an indefinite period, the container may run short of available instances of the specific EJB to service other clients and thus slow down the overall performance of the application.

For further information, refer to:

http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/jmsj2ee.html

Where to implement message producers and consumers

There are a number of options for where to implement your JMS message producers and consumers in the J2EE application architecture. These options include:

- Producers

  If the Model-View-Controller (MVC) pattern is invoked, then the model is typically where the producer would be implemented. In J2EE application architecture, this is likely to be a session EJB.

  However, it is possible to implement the message producer almost anywhere. A simple JavaBean could also implement the message producer and fit in with the MVC pattern.

If the producer is participating in a transaction of some kind, then session EJBs may be a better implementation choice. Transaction creation and management is gained almost for free within EJBs, whereas it would have to be explicitly created and managed within other implementation choices such as JavaBeans.

Servlets can also be used as message producers. They offer a simpler programming model than EJBs. Servlets, however, are usually implementers of the controller aspect of the MVC pattern, and do not take advantage of the EJB container facilities.

- Consumers

  Just as there are for producers, there are a number of implementation choices for consumers. When consumers are used in request-reply scenarios, you then have the choice to implement this in a servlet or an EJB. Implementing the consumer as an EJB has the advantage of container transaction management and security management. But the disadvantage is that an EJB will be blocked until a response arrives. Some extra programming is needed to disregard a response when it takes too much time.

  Another option for consumers is a message-driven bean. The request and reply will be loosely coupled when using a message-driven bean, which makes it more complex. A message-driven bean is a good solution for subscription and message consumer patterns.

Embedded JMS provider versus WebSphere MQ

In line with the J2EE 1.3 specification, IBM WebSphere Application Server V5.0 has an embedded JMS provider, or messaging service, included in the application server. This internal JMS provider can be used for asynchronous JMS communications with other WebSphere applications. The internal messaging service cannot be used for messaging with other messaging systems, such as WebSphere MQ. If you need to communicate with other systems using WebSphere MQ, then you need to install WebSphere MQ as a JMS provider on WebSphere Application Server.

IBM WebSphere MQ client or server?

With most middleware-oriented software, there are client and server components. The client is usually a smaller piece of software that provides local access to the remote server. The server implements all of the functionality and the client provides a relatively light facade for accessing the server’s function.

The WebSphere MQ client provides access to all of the WebSphere MQ API and is typically used where there are limited hardware or system administration requirements. The WebSphere MQ client is also able to connect to multiple queue managers on different platforms.

The server provides richer administration functionality to take advantage of the full suite of MQ functionality, such as failover support or scalability configuration options.

During development, the client option is useful where machine resources are limited. Any complex changes to the WebSphere MQ configuration will, however, require the server version. The embedded WebSphere JMS provider available in the WebSphere Studio test environment can also be used during development.

13.3 Development guidelines

In this section we consider the steps necessary to add JMS connectivity to your application. To create the Java application:

1. Configure your JMS provider and destinations in your Integrated Development Environment (IDE). For development you can use the internal JMS provider that is included in IBM WebSphere Studio Application Developer.
2. Get an instance of JMS javax.jms.ConnectionFactory (usually through a JNDI lookup).
3. Get an instance of JMS Connection from the ConnectionFactory.
4. Start the JMS Connection.
5. Get a JMS Session from the Connection object.
6. Using the Session object, create either a producer (QueueReceiver) or consumer (QueueSender) on a specified destination (IBM WebSphere MQ queue).
7. Use this producer or consumer to access the Destination.
8. Close the message consumer, session, and connection. Closing the connection will close the session and the message producers and consumers associated with it.

Detailed development steps for a sample JMS scenario are in *Patterns: Self-Service Application Solutions Using WebSphere Application Server V5*, SG24-6591.

13.4 Quality of Service capabilities

In this section we discuss Quality of Service capabilities and considerations specific to JMS and WebSphere MQ.

13.4.1 Autonomic

This section briefly discusses some of the tools important for fault monitoring and isolation.

WebSphere MQ monitoring

There are two Windows Service snap-ins that use WebSphere MQ instrumentation to present the user with a GUI event and monitoring tool:

- WebSphere MQ Alert Monitor (Windows)

  The WebSphere MQ Alert Monitor is an error detection tool that identifies and records problems with WebSphere MQ on a local machine. It displays information about the current status of the local installation of a WebSphere MQ server.

- Performance Monitor (Windows)

  The Performance Monitor is a standard component of Windows. It enables you to select and display a variety of data about the performance of the Windows environment as tabular reports or graphs. You can use it to monitor the depth of messages on WebSphere MQ queues, and the rates of message arrival and removal.

WebSphere MQ restart and recovery

WebSphere MQ ensures that messages are not lost by maintaining records (logs) of the activities of the queue managers that handle the receipt, transmission, and delivery of messages. It uses these logs for three types of recovery:

- Restart recovery, when you stop WebSphere MQ in a planned way
- Crash recovery, when WebSphere MQ is stopped by an unexpected failure
- Media recovery, to restore damaged objects

In all cases, the recovery restores the queue manager to the state it was in when the queue manager stopped, except that any in-flight transactions are rolled back, removing from the queues any messages that were not committed at the time the queue manager stopped. Recovery restores all persistent messages; non-persistent messages are lost during the process.

Logging and tracing

It is often helpful to examine application server log and trace files when your application experiences JMS errors or problems. It is always important for applications to record their activity to a logging facility. When you write a log to the standard output file or standard error file, the application server will record it to the corresponding log files.

13.4.2 Availability

The approaches discussed in “IBM WebSphere MQ clustering” on page 294 also provide the node redundancy and failover capabilities needed to eliminate single-point-of-failure for production messaging systems.

13.4.3 Performance

Some issues that play a role in JMS messaging performance are:

- WebSphere MQ client versus bindings mode connection

  Using a bindings mode connection to a queue manager situated on the same machine as the application server will speed up communication. In addition to the performance gain, it also makes it possible to join a global transaction.

- Generic versus specific message structure

  Making the message structure more generic requires more translation and interpretation time at the sender and receiver ends. Making a message too specific will reduce flexibility for even small changes in the message structure.

- Message persistence

  Using persistent messages in MQ requires writing the messages to disk, which takes time and reduces performance.

- Request/reply scenario

  In a request/reply scenario, EJBs should only be used with appropriate request/reply timeouts and re-tries.

- Message-driven bean

  Minimize the time spent in a message-driven bean processing the message. Let the pool of message-driven beans depend on the number of messages that arrive at the queue.

- Optimization with connection

  Start the connection when appropriate, so consumers are ready to consume messages before the producers are started. Process messages concurrently using a server session pool and close the connection when you are finished.

IBM WebSphere MQ clustering

WebSphere MQ offers the ability to create clusters. MQ clusters provide a number of benefits that are silently utilized by JMS applications. Clusters offer:

- Simpler administration of logically related queue managers

  Clustering allows communication between queue managers to promote information about the queues they offer. Once in a cluster, queues on remote queue managers are visible to all queue managers if the queues are defined as cluster queues. The number of explicit definitions within IBM WebSphere MQ administration is reduced with the use of clusters.

- Workload and failover management

  Adding queue managers to clusters allows access to WebSphere MQ workload and failover features.

  As shown in Figure 13-8, QM3 is able to load balance across the queue named ReplyQ, since it is available on both QM1 and QM2. Similarly, if QM1 is disabled, all messages for ReplyQ are routed to QM2.

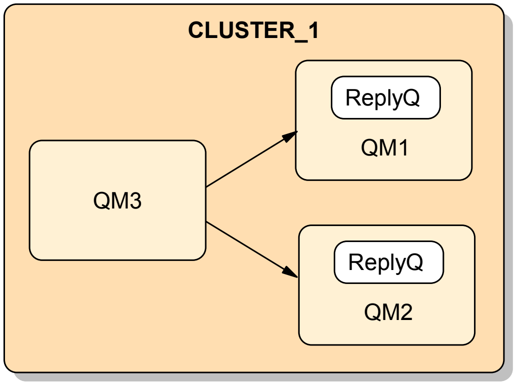


Figure 13-8 Cluster workload management

None of these features can be controlled through the JMS interfaces. However, MQ will automatically utilize the workload and failover under JMS.

These and other features of MQ offer significant benefits and demonstrate that IBM WebSphere MQ is a reliable, scalable, and mature JMS Provider.

13.4.4 Security

The JMS specification does not specify any features for controlling message integrity or authentication. It is expected that a JMS provider will provide these services. Security is considered to be a JMS provider-specific feature that is configured by an administrator rather than controlled via the JMS API by clients.

Messages arriving at a message-driven bean listener port have no client credentials associated with them. The messages are anonymous. However, some security can be provided if the JMS listener assumes the application server processes credentials when invoking the message-driven bean.

WebSphere MQ provides security for its administrative functions and for access to WebSphere MQ objects. This security relies on authentication being performed by the security system of the underlying operating system. Authorization of the users or groups to MQ resources is performed through the use of an access control list (ACL) maintained through the WebSphere MQ Object Authority Manager (OAM) command interface.

For more information see the InfoCenter article, *Asynchronous messaging - security considerations* at:

```
http://www.ibm.com/software/webservers/appserv/infocenter.html
```

13.4.5 Standards compliance

The JMS enterprise messaging API has achieved wide cross-industry support. It allows Java applications to leverage existing, enterprise-proven messaging systems, such as WebSphere MQ. It provides application designers and developers with standard messaging concepts and conventions that apply across a wide range of enterprise messaging systems.

13.4.6 Transactionality

JMS providers may support transactions, but only between the client and the messaging system, not to the target application. Once the client has sent the message to the messaging system, the client no longer has control of its propagation.

The WebSphere MQ JMS provider includes the JMS XA interfaces. These interfaces allow MQ JMS to participate in two-phase commit transactions that are coordinated by a transaction manager that complies with the Java Transaction API (JTA).

13.5 Best practices

Some best practices for JMS and WebSphere MQ are:

- Use message timeouts.

  Using message timeouts avoids large numbers of messages remaining on a queue, thus reducing performance overheads. It also allows the relevance of messages to be evaluated. For example, a message may not be relevant after a certain period of time because the information has been superseded.

- Use message selectors.

  Message selectors allow the underlying provider (through JMS) to browse messages before they are retrieved with little application code.

- Persistent versus non-persistent.

  Only use durable messages when necessary. This option needs to be explicitly set to non-persistent, as the default in JMS is persistent.

- Clusters.

  The use of IBM WebSphere MQ clusters allows simple administration, and automatic workload management and failover.

- Message producers.

  EJB message producers: In a request/reply scenario, it is important that the issue of blocking calls is dealt with correctly. Essentially, EJBs should only be used with appropriate request/reply timeouts and retries.

- Message consumers.

  Message-driven beans: Business logic should not be implemented in the message-driven bean. Implement the business logic in another component, such as a session bean, and use message-driven beans only for receiving the message. Never throw application exceptions in the onMessage method.

- Consider using XML-based messages for inter-application integration.

  XML is commonly used as a messaging structure that allows for a more portable inter-application integration model. Although it does add some overhead to the message payload size and requires XML parsers, it is quickly becoming a standard for inter-operability.

Part 4

Extended Enterprise scenarios

Part 4 provides detailed design, development, and runtime guidelines for inter-enterprise integration solutions. It teaches you by example using IBM WebSphere Application Server V5.0 with Web services.

Included in Part 4 are the following chapters:

14

Using inter-enterprise Web services

This chapter discusses using Web services in an inter-enterprise integration scenario. We are using Web services for J2EE as provided with IBM WebSphere Application Server base V5.0.2 in this scenario.

This chapter describes the following:

- Using inter-enterprise Web services in the context of our ITSO Electronics business scenario.
- The high-level designs applied to our inter-enterprise Web services scenarios, and examination of some asynchronous Web service approaches.
- Brief development guidelines for integrating Web service providers and requesters across enterprise boundaries.
- Additional Quality of Service capabilities, focusing on security capabilities that are critical in inter-enterprise integration scenarios.

14.1 Business scenario

As described in 6.3, “Inter-enterprise scenarios” on page 121, ITSO Electronics wishes to enable their external resellers to place orders to the wholesale group. Currently, the resellers fill out an order form and mail it to the wholesale organization. Difficulties arise when orders cannot be filled because of latency in the manual process and outdated inventory.

The Extended Enterprise business pattern applies to this situation. This pattern can be applied for inter-enterprise integration activities. It will enable ITSO Electronics to integrate their business processes with processes and information that exist at partner organizations.

When an external reseller needs to notify the wholesale department that a part must be ordered, the Message variation of the Exposed Direct Connection application pattern can be applied. This Application pattern is suitable because the external reseller will be notifying the wholesale department that a part must be ordered, but the reseller does not require any response as part of this process.

When the external reseller needs to know the expected delivery date for a part on order, the Call variation of the Exposed Direct Connection application pattern can be applied. In this instance, the external reseller requires a response from the wholesale department, advising them of the expected delivery date for the part in question.

The Web service can be applied to both processes. Web services used with the HTTP transport allow integration between business partners via the Internet. Web services also have the advantages of providing heterogeneous platform support and loose coupling between the two systems.

This chapter extends, for inter-enterprise use, the intra-enterprise Web services scenarios from previous chapters, specifically:

- Chapter 8, “Using RPC style Web services” on page 147
- Chapter 9, “Using document style Web services” on page 183
- Chapter 10, “Using the Web Services Gateway” on page 215

14.2 Design guidelines

This section presents the Runtime patterns and Product mappings we used to demonstrate the Exposed Direct Connection application pattern using:

- Web services
- Web Services Gateway

Web services solution overview

Figure 14-1 shows the Runtime pattern and Product mapping we used to demonstrate the Exposed Direct Connection application pattern, using Web services technology across enterprise boundaries. The Partner A Secure Zone uses the same nodes and products as the source application for the intra-enterprise Direct Connection pattern in Figure 8-3 on page 153 (and Figure 9-1 on page 185). The path connector from Partner A Secure Zone includes firewalls, DMZ, and the Internet. Partner B's infrastructure is unspecified.

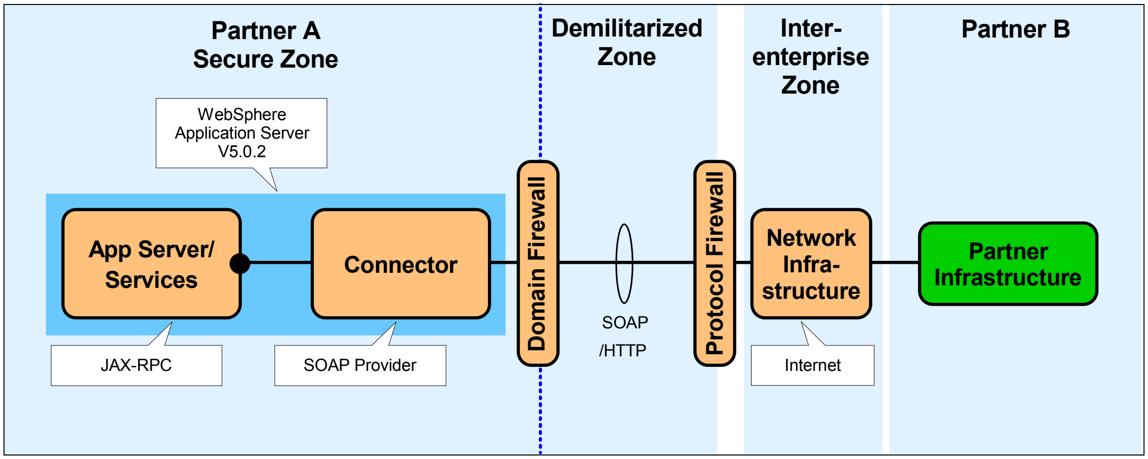


Figure 14-1 Exposed Direct Connection::Message Connection: Web services Product mapping

Web Services Gateway solution overview

Figure 14-2 shows another Runtime pattern and Product mapping used in this scenario. It is based on IBM WebSphere Application Server V5.0.2 and the Direct Connection application pattern. It also includes the Web Services Gateway packaged with IBM WebSphere Application Server Network Deployment V5.0.2.

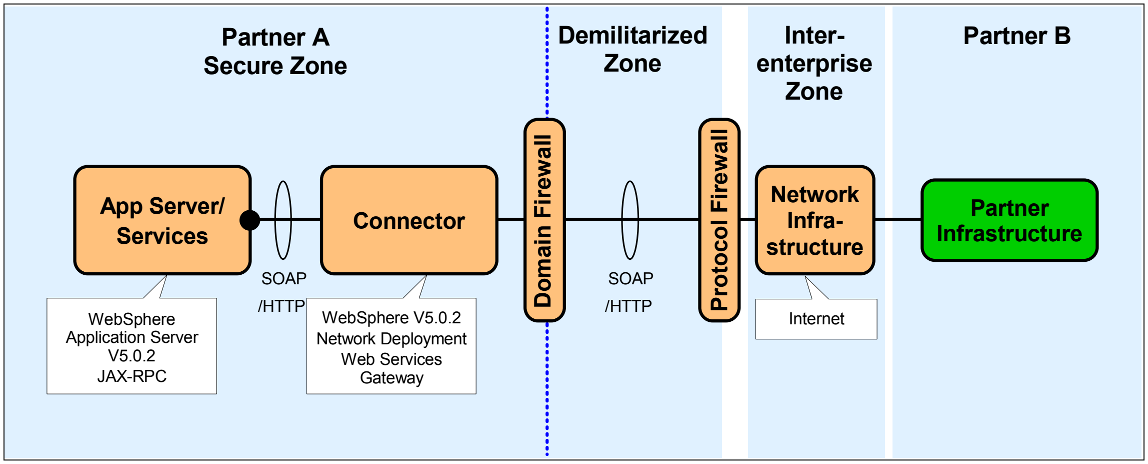


Figure 14-2 Exposed Direct Connection::Message Connection: Web Services Gateway Product mapping

The Partner A Secure Zone uses the same nodes and products as the source application for the intra-enterprise Direct Connection pattern in Figure 10-3 on page 220. As in Figure 14-1, the path connector from Partner A Secure Zone includes firewalls, DMZ, and the Internet. Partner B's infrastructure is unspecified.

14.2.1 Design considerations

A number of factors affect the design of a Web service. In this section we discuss some of the factors we considered when using Web services in our inter-enterprise sample application.

This section extends, for inter-enterprise use, the intra-enterprise Web services design considerations from the following sections:

- 8.4.1, "Design considerations" on page 153 for RPC style Web services
- 9.3.1, "Design considerations" on page 186 for document style Web services

We recommend reading the intra-enterprise Web services design considerations first.

Asynchronous Web services

One risk for a source application communicating via the Internet with a target application in another enterprise is the source application can be blocked because of problems in the Internet communication channel. You can minimize this risk by decoupling both applications as much as possible. One possibility is to use asynchronous messaging style communications, where the source and target applications don't need to be aware of each other or active at the same.

In our business scenario we describe two use cases, Update Inventory and Get Delivery Date. The Update Inventory use case does not require a response from the target application, so it can be implemented with a loosely coupled one-way request, as described in Chapter 8, “Using RPC style Web services” on page 147.

The Get Delivery Date use case is more complex because an in-parameter and a return value are needed. On first view this use case is a typical example of the Call variation of the Exposed Direct Connection application pattern, which has a more natural fit with a synchronous communication channel. If we assume that asynchronous communication between the source and target application is required in order to reduce coupling between partner applications, we can decompose this interaction into a one-way request and a separate one-way response.

In this scenario the target application could provide a RequestDeliveryDate service and the source application could provide a ResponseDeliveryDate service, which is used to communicate the response back to the source application.

Advantages of asynchronous communication

In asynchronous communications using messaging systems, the source application sends the message to message-oriented middleware. The source application sends the message and control returns immediately back to the source application. Now the message-oriented middleware is responsible for delivering the message to the target application. Therefore, the source application is decoupled from the target application, and the communication channel, during the interaction.

Disadvantages of asynchronous communication

If the source application needs a result from the target application in a later stage of the business process, then the source application must be known to the target application. This effort must take place either during the request or as an initial effort before the first interaction. The first solution results in a message overhead and the second requires a more complex infrastructure. Furthermore, the response has to be synchronized with the corresponding request.

Asynchronous Web services approaches

In this section we describe approaches for implementing asynchronous communication with Web services between the source and target application. Common requirements of the approaches discussed include:

- Both partner applications can determine the end point URL for each other.

- The request interaction doesn’t require an immediate response, except for a request acknowledgement.

The first two approaches have to be implemented during the application development phase of the source and target applications. The last approach uses an asynchronous transport protocol that is transparent to the application developer.

In the approaches presented we are focusing on the topic of decoupling the request channel from the source to the target application. The channel from the target application to the source can be de-synchronized with the same techniques if you reverse the roles in the figures.

A difficulty arises when a significant part of the transport channel between the source and target applications requires synchronized communications. These links will not have the same relaxed flexibility offered by asynchronous communications. A way to solve the problem is to introduce an asynchronous component into the channel, which is configured to present a synchronous interface to the synchronous connection. This “de-synchronized” link effectively acts as a “proxy” or “buffer” between the asynchronous and synchronous communications.

One-way invocation

In this approach we assume that the transport protocol used between the enterprises is HTTP. Therefore, the communication is in principle synchronous, because HTTP is a synchronous protocol. We de-synchronize the communication at the application level within the Partner A enterprise, which is initiating the communication.

With our Update Inventory use case, the Web service does not require an output message so the request is defined as only having input parameters and the Web service invocation is one-way and non-blocking. The critical path is marked with a dotted ellipse in Figure 14-3 on page 305.

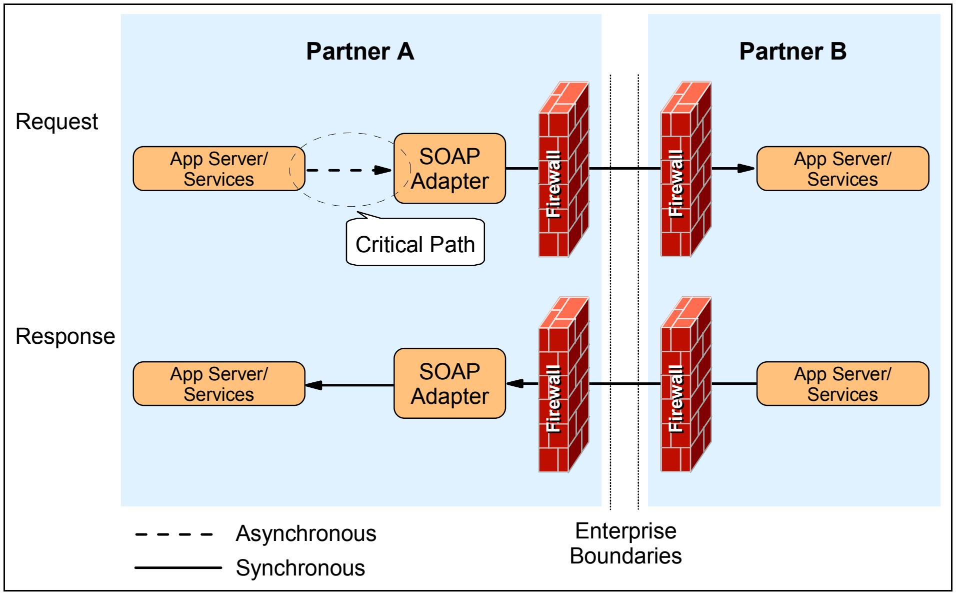


Figure 14-3 Using one-way Web service invocations

Advantages of one-way invocation include:

- The source application is decoupled from the behavior of the channel between the enterprises.
- The source application is decoupled from the behavior of the target application in the partner enterprise.
- The source application is decoupled from any intermediary (such as a gateway) which is used as an exit point from the enterprise.

Some disadvantages of one-way invocation are:

- Services provided by target applications must only implement one-way transmission style.
- The delivery of the request message is not reliable.

Using a de-synchronization node

Similar to the first approach, we assume that a synchronous transport protocol such as HTTP is used. In this approach, we try to give the source application an interface that is more service oriented. The service that the target application is offering to the source application should be asynchronous without any additional effort for the source application developer. Therefore, the critical path for de-synchronizing the source and target application is within the target enterprise, as shown in Figure 14-4 on page 306.

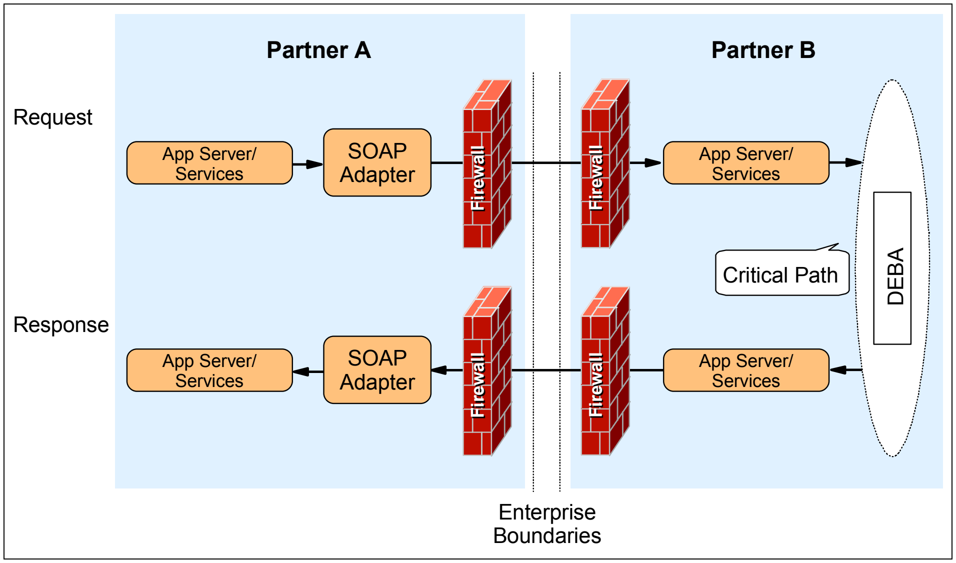


Figure 14-4 Using the Distributed Event-Based Architecture for de-synchronization

One approach is to open a separate thread in the target application to execute the request. Meanwhile, the request channel can be closed by sending back an empty response. However, such an approach may impact application server performance and scalability and is usually not recommended.

The Distributed Event-Based Architecture (DEBA) framework, which is available from IBM alphaWorks®, provides another approach. The Distributed Event-Based Architecture for Web services is basically an implementation of the Observer pattern. From a high-level view, DEBA is simply implementing the components of this pattern within the Web services context.

DEBA helps to introduce a multi-port communication in a Web services world simply by using a framework. You can find out more about the full power of this framework at:

`http://www.alphaworks.ibm.com/tech/DEBA4WS?Open&ca=daw-flnt-022702`

For a brief introduction, a good starting point is:

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

Each service requester is an observer that is interested in a state or result of the service provider. The service provider is the subject of the observation. In standard Web services scenarios today, the requester connects to the provider and receives a response immediately. Following this tactic means that the observer has frequently to ask if there is any new information on the subject that the observer is interested in. For our scenario, the Observer pattern is a good solution. The observer (requester) attaches itself to the subject (provider) via a Web service invocation, and sends its own URL for the update response. For the update response, the subject initiates a separate Web service invocation to the observer.

Note that the requester and provider roles are temporary in the Observer pattern; they only indicate which component initiates the communication. This application of the Observer pattern shows how Web services can provide highly decentralized and distributed solutions, even when using SOAP to establish a direct connection between just two partners.

If you reduce the number of observers to one you can implement the Message variation of the Exposed Direct Connection application pattern. Keep in mind that this doesn't necessarily mean that the interaction from requester to provider is asynchronous at the implementation level. This depends on the communication protocol you are using between requester and provider. If you are using HTTP, then the communication is synchronous, even when you don't expect any return value.

Advantages of using de-synchronization include:

- The asynchronous mechanism is transparent to the source application.
- A receive acknowledge comes from the Partner B enterprise which is hosting the target application.

Some disadvantages of using de-synchronization include:

- The source application is not decoupled from the behavior of the transport channel, such as the Internet.
- Application development effort is required in the target application in the Partner B enterprise.

Using an asynchronous transport protocol

The architectural overview diagram for this approach is similar to the diagram for the one-way invocation in Figure 14-3 on page 305. In fact it is the same figure, but the transport protocol used is different.

The main disadvantage of previous approaches is that the asynchronous mechanism is not transparent for the source application or the target application at the implementation level. These approaches are not flexible for future migration to new transport technologies. For example, when applications switch to a new asynchronous transport protocol (such as from HTTP to JMS), then the asynchronous mechanism will be implemented twice.

Therefore, we want to briefly discuss locating the asynchronous mechanism in an asynchronous transport protocol, like Java Message Service (JMS).

In Figure 14-3 on page 305, we de-synchronized the communication from the source application to the target application. The channel back from the target application doesn't need to be the same channel or even the same transport protocol. It is also conceivable that the response is communicated to a different application and not to the initial source application.

In both cases, we simply use JMS to delegate the task of delivering the message to the target application to the message-oriented middleware. Using this approach give us the following advantages:

- The asynchronous mechanism is transparent to the application level.
- It is possible to switch to other transport protocols without affecting the application.
- Using message-oriented middleware has the additional value of a reliable message exchange.

The price for this approach is more complex infrastructure.

One approach can be IBM WebSphere MQ with the MQ SupportPac™, MS81: WebSphere MQ internet pass-thru. Provided both partners are using WebSphere MQ, this SupportPac can be used to implement messaging solutions between remote applications across the Internet. For details, see:

http://www.ibm.com/software/integration/support/supportpacs/individual/ms81.html

The WS-ReliableMessaging draft standard is being developed to address the issue of interoperability between different reliable transport infrastructures. For further information on WS-ReliableMessaging, refer to:

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

IBM Web Services Gateway considerations

Chapter 10, "Using the Web Services Gateway" on page 215 describes use of the Web Services Gateway in intra-enterprise scenarios. The gateway operates by adding a layer of abstraction that separates deployment from invocation. While this is important in an intranet environment, it is even more important in an Extended Enterprise environment because of the diversity of applications, environments, and users who will be interacting with the exposed service.

In inter-enterprise scenarios, the Web Services Gateway can be used to:

- Secure your Web services using the gateway access control mechanisms

- Act as an reverse proxy providing indirect access to your internal Web services
- Provide a common access point for partners needing access to your infrastructure
- Provide protocol transformation, so a HTTP/SOAP client can access a JMS/SOAP service, for example

As shown in the Web Services Gateway product mapping in Figure 14-2 on page 302, you may also want to access external Web services from inside your enterprise. In this case, the gateway:

- Provides a single point for controlling access to external services.
- Hides changes to the external Web services from your internal client applications.

The Product mapping in Figure 14-2 on page 302 places the gateway on the secure internal network behind the DMZ. The advantage of this configuration is the gateway can provide access services using protocols that are not as firewall-friendly as HTTP. For example, the gateway can enable Web service clients to access EJB, JMS, and JavaBean applications.

The gateway could also be located in the DMZ. All the firewalls in our example are configured for network address translation, so SOAP over HTTP can be used to pass through the firewalls. This may not work if the gateway is using RMI to access an EJB on the secure network, for example.

The performance overhead of introducing an additional node, such as the gateway, should also be considered. There may be performance impacts due to:

- Extending the path length.
- Converting into an internal data representation.

14.3 Development guidelines

To test the Extended Enterprise Web services patterns presented in this chapter, we simply introduced the firewalls and a simulated Internet between the source and target application scenarios from:

- Chapter 8, “Using RPC style Web services” on page 147
- Chapter 9, “Using document style Web services” on page 183
- Chapter 10, “Using the Web Services Gateway” on page 215

Our target application Web services and our source application Web service clients were unchanged. We just modified the end point DNS entries where needed and opened the required firewall ports. You could also update the end-point SOAP address location in the client WSDL, then redeploy the client application to WebSphere, selecting the Deploy WebServices option (see Figure 8-14 on page 172).

When using the Web Services Gateway between the source and target applications, that gateway service can be redeployed to use an external Web service provider with no change required to the source application. See 10.4.2, “Deploying the Web Services Gateway service” on page 224

See Appendix A, “Scenarios lab environment” on page 329 for a description of our lab environment.

14.4 Quality of Service capabilities

In this section we discuss Quality of Service capabilities and considerations specific to inter-enterprise Web services. For further discussion on Quality of Services for Web services in general, see 8.6, “Quality of Service capabilities” on page 177. For document style Web services see 9.6, “Quality of Service capabilities” on page 209. For the Web Services Gateway see 10.5, “Quality of Service capabilities” on page 233.

One of the main challenges when developing applications in an inter-enterprise environment is the uncertainty about the connection between the enterprises and the partner enterprise infrastructure.

14.4.1 Security

Security is one of the crucial QoS aspects which needs to be addressed when an enterprise plans to expose their internal applications to partner organizations. For the whole context and its consequences, see:

- *Security in a Web Services World: A Proposed Architecture and Roadmap*, an IBM white paper available at:

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

- The OASIS Web Services Security (WSS) Technical Committee home page at:

  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

To address the security risks in an inter-enterprise scenario with Web services, it is critical to understand the how the Web services security context is defined. Web services defines an end-to-end security context as shown in Figure 14-5 on page 311. Furthermore, it is necessary to recognize that this not necessarily congruent with the security context of the underlying transport protocol.

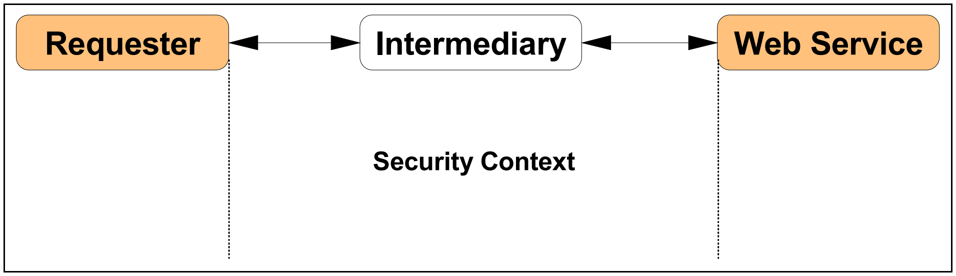


Figure 14-5 End-to-end Web services security context

A good example of this is the HTTP transport protocol, commonly used in Web services scenarios. As shown in Figure 14-6, HTTP itself defines a security context from point to point. In reality, an HTTP request passes through several intermediaries (for example, routers and Web server in a DMZ) before the request reaches the endpoint.

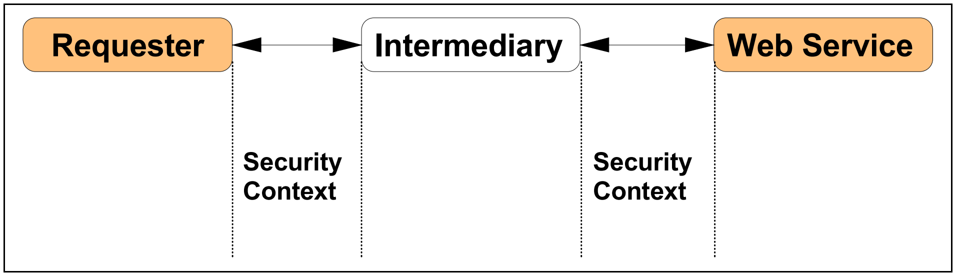


Figure 14-6 Point-to-point security context in a Web services scenario

Consequently, each intermediary that is working on the application level with the SOAP message is a potential security risk. This is true even when you use HTTPS instead of HTTP.

We have to apply security at a higher level because securing the message at the transport protocol level is not sufficient. As described in 8.6.4, “Security” on page 179, WS-Security describes how to secure the message using XML encryption and XML digital signature. WS-Security defines how to integrate these specifications into a SOAP message.

Non-repudiation is also more significant in an inter-enterprise scenario than in an intra-enterprise scenario. This requirement can be fulfilled with a combination of sender authentication and message signature.

Using firewalls

In the case of inter-enterprise communication, data is often exchanged over a public network such as the Internet. The exposed interface for communications is visible to everyone on the Internet. Communication channels and data must be secure because there are unknown Internet users with unknown intentions. In the simplest scenario a firewall as entry point to the enterprise infrastructure may be sufficient.

But the higher the value of the services an enterprise offers, the more the channels must be secured. In such cases it may be necessary to implement a demilitarized zone (DMZ) with several security levels. Figure 14-7 shows a possible scenario.

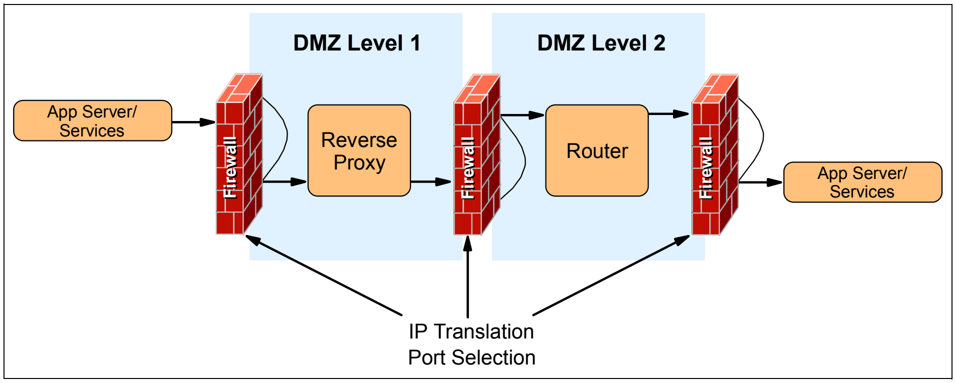


Figure 14-7 Securing the enterprise infrastructure with a two-level DMZ

In this scenario we introduce multiple levels to secure the enterprise infrastructure. Each level is protected by a separate firewall which performs network address translation and only allows traffic through explicitly enabled ports. In the first level we introduce a reverse proxy to hide the internal network structure. Using a router in the second level decouples the intranet from the Internet. The router may also introduce several subnets, which gives you additional security because the requests have to be routed to cross the subnet boundaries. This means you have better control because you can decide which request has to go to a specific port and which ports are allowed.

It is clear that infrastructure increasing security on the transportation level also has its drawbacks. The most important disadvantage is the increased path length from the source to the target application. Therefore, you have to take into account the increased execution time of requests. A good load balancing approach is also crucial for high traffic servers.

In addition to securing the communication channel and message exchange, it is important to:

- Avoid the risk of malformed messages from outside, for example, using the Web services engine to validate messages before invoking the application.

- Avoid denial-attacks, for example, using dedicated Web services servers.
- Increase the control of Web services interactions, for example, using separate ports in the firewalls.

15

Using WebSphere Data Interchange

This chapter discusses using WebSphere Data Interchange in an inter-enterprise integration scenario. We are using WebSphere Data Interchange V3.2 and iSoft Peer-to-Peer Agent V3.1.2 in this scenario.

This chapter describes the following:

- Using WebSphere Data Interchange in the context of our ITSO Electronics business scenario.
- Design guidelines for using WebSphere Data Interchange.
- Development guidelines for using WebSphere Data Interchange.

This chapter is based on the following IBM Redpapers:

- *WebSphere Data Interchange Installation and Configuration*, REDP3600
- *Implementation of iSoft and Integration with an EAI solution*, REDP3625

15.1 Business scenario

As described in our inter-enterprise Web services scenario in 14.1, “Business scenario” on page 300, ITSO Electronics wishes to enable their external resellers to place orders to the wholesale group. Currently, the resellers fill out an order form and mail it to the wholesale organization. Difficulties arise when orders cannot be filled because of latency in the manual process and outdated inventory.

In this scenario we examine an Electronic Data Interchange (EDI) solution. The ITSO Electronics wholesale organization needs to integrate with diverse external resellers. Each reseller may have their own internal representation of the data required in the order process. The EDI solution needs to translate internal order data of each partner into an agreed EDI-compatible format. It also needs to be capable of exchanging documents between partners over the Internet in a secure and reliable way.

Note: We didn’t implement this solution in the ITSO Electronics sample application provided with this redbook. For sample implementations using WebSphere Data Interchange, see the redpapers listed at the start of this chapter.

15.2 Design guidelines

Figure 15-1 shows the Runtime pattern and Product mapping for the Message Connection variation of the Exposed Direct Connection application pattern across enterprise boundaries, using WebSphere Data Interchange for Multiplatforms.

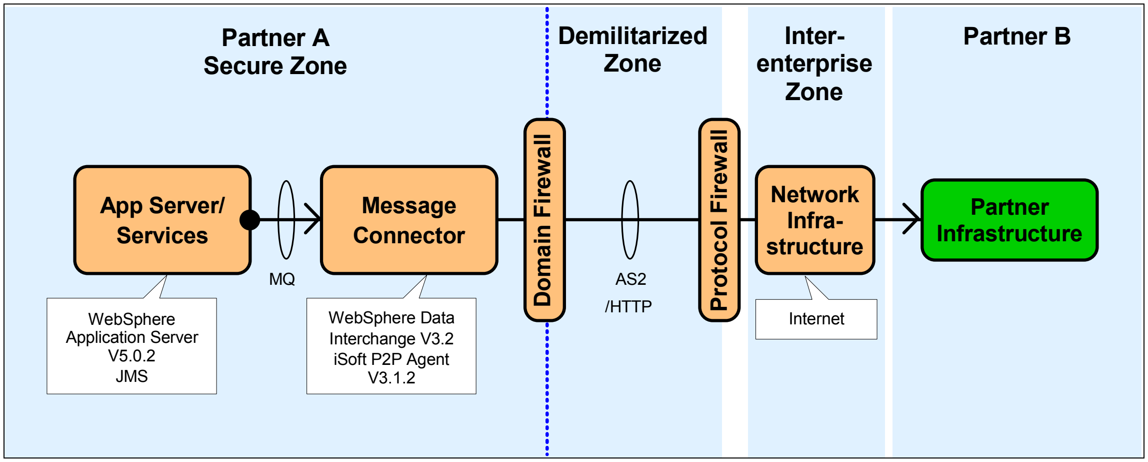


Figure 15-1 Exposed Direct Connection::Message Connection: WebSphere Data Interchange Product mapping

This product mapping uses IBM WebSphere MQ V5.3.1 as the transport mechanism between WebSphere Application Server, WebSphere Data Interchange, and iSoft Peer-to-Peer Agent.

WebSphere Data Interchange V3.2 with CSD1 is used to adapt each type of message/document to partner requirements. The product mapping uses iSoft Peer-to-Peer Agent V3.1.2 to adapt MQ messages and documents to the AS2 EDI protocol for secure and reliable transport of messages and documents with business partners via the Internet.

15.2.1 Electronic Data Interchange

Electronic Data Interchange (EDI) is widely accepted by companies all over the world as the way to electronically exchange business data. Business documents such as purchase orders, invoices, shipping notices, and price catalogs are exchanged between companies in a structured and computer-processable format.

Organizations are recognizing the value of many years of investments in EDI. Rather than replacing existing solutions, they are extending and evolving their EDI transactions. These existing EDI solutions are considered an integral part of a multi-modal B2B gateway or hub alongside XML, Web solutions, and portals. By integrating B2B and EDI technologies, event-driven or process-driven integration models can be supported using the existing EDI solution.

Advantages of EDI

The market is driving every business to act smarter and quicker and to be more visible. Much of this can be achieved using EDI. Even better, EDI can give companies a better knowledge of their markets, because it opens up possibilities to collect and analyze information from the EDI transactions they are generating.

Among the most visible benefits of adopting EDI are:

- Reduction of data entry errors
- Reduced cycle time
- Minimization of paper use
- Improved relationships with your business partners
- Information in electronic form is more easily shared throughout the organization
- Improved inventory management

EDI transmission

EDI is a *concept*. It does not define any techniques or point to any specified product or service. An EDI transmission can basically be divided into two logical parts: the message itself and the communication.

Message standards

Since the idea of EDI is to have a standardized message, a number of different standards have been developed and established over the years. The most commonly used message standards are:

- ANSI ASC X12 - US standard
- EDIFACT - standard recommended by the United Nations, used mainly in Europe
- UNTDI - UK retail standard
- ODETTE - European automotive industry
- Others, such as HIPAA, VICS, VDA, UCS, and so forth

Communication

Transportation of the EDI file over a network can be done in many ways. Any network and any protocol can be used as long as it fits the needs. Three types of communication are:

- Value Added Networks (VAN) communication

  Using a value added network for the transmission of files is traditionally seen as the most secure way of communication. Apart from doing pure communication, a VAN also provides value adds such as built-in security, restart and recovery facilities, archive capability, 24x7 availability, and notification of message arrivals.

- EDI over the Internet

  The initiative to move toward securely transmitted EDI messages over the Internet is known as EDI INT. Presently there are two main EDI INT initiatives, known as applicability statements AS1 and AS2, which describe how current Internet standards can be used to achieve VAN functionality.

  - AS1 uses MIME (Multipurpose Internet Mail Extensions) and SMTP (Simple Mail Transfer Protocol).
  - AS2 uses MIME and HTTP (Hypertext Transfer Protocol) for process-to-process real-time EDI.

  The Internet solutions are often considered much cheaper than traditional VANs, but Internet solutions often leave it to the user to add functionality to achieve adequate security, reliability, and other features that are included in a VAN.

  IBM Business Exchange Services - Internet transfer is an example of Internet communication.

- Message queuing

  Message queueing (MQ) connects commercial systems in today's business. It provides assured, once-only delivery of data in any format.

  IBM WebSphere MQ is an example of this.

While the use of EDI technology is widespread, technology changes and evolution have resulted in the use of many types of B2B communication infrastructures. Besides the traditional VAN-based EDI communication, AS1 and AS2 Internet protocols are still tied more or less to traditional EDI communications. More recently, Web services-based technologies also became available for use in the B2B area. While this technology is still maturing, it is clear that a flexible B2B solution should handle multiple communication techniques.

The Internet is widely perceived as being much less expensive than a VAN, but this is not necessarily the case. VANs generally provide valuable services, such as TPA management, service-level administration, security, and store-and-forward capability. The Internet requires you to manage these elements yourself, which means the total costs are not always lower than a VAN.

Elements of an EDI solution

As well as obvious components of an EDI solution, such as application programs and systems, VANs, and trading partners, a complete and flexible solution should include the following important elements:

- Translators

  A universal problem in integration of applications is the conversion of shared data from one format to another. Common data fields—such as names, addresses, and numbers—often have different formats across disparate systems. The traditional approach to EDI implementation is to place the function that converts application data to the EDI standard directly into the business application. This approach is less effective because a separate program is required for each transaction as well as for each trading partner. In addition, it is difficult to keep up with new versions of standards because programs must be modified every time a trading partner adopts a newer standard or version of the standard.

  This approach has changed with the introduction of third-party translation software, also known as mappers. The translator is responsible for mapping application data to the specific EDI format and vice versa. This translation software is implemented in either a centralized engine or in an adapter. It must handle primary EDI standards as well as different and evolving versions of each standard.

- Batch enveloper/de-enveloper

  Typically, because VAN charges are based on each transaction sent, enterprises have been driven to find ways to reduce the number of transactions and to compress more information into each. Consequently, EDI messages are sent in large batches, which can then be grouped from, or split out to, several divisions or areas of an enterprise.

  Enveloping batch messages involves placing the EDI standard header and trailer around transactions in preparation for sending. When the envelope is complete, the package can then be sent to a trading partner through a VAN. Similarly, batch transactions must be de-enveloped when they are received from the VAN.

- Message router

  Once the EDI message is de-enveloped, it can be divided into function groups. Each function group may relate to a different division or area of the business. A mechanism is needed to sort messages destined for different groups and deliver them to the appropriate target applications. This means there is a requirement to fan in and fan out messages. Message transformation may also be required to get the message into the correct format for the end applications.

- Trading Partner Agreements (TPA)

  A TPA is an agreement related to the exchange of information in electronic transactions. The term includes a particular agreed-upon standard for business documents, as well as communications and business protocols, the service-level agreement, and more. TPAs can also be extended to include business events. For example, if an event occurs in one organization that might affect processes in a second organization, the TPA can specify that the second organization be alerted to the event.

15.2.2 WebSphere Data Interchange

IBM WebSphere Data Interchange for Multiplatforms fulfills the core role of EDI Broker in the IBM EDI solution, shown in Figure 15-2.

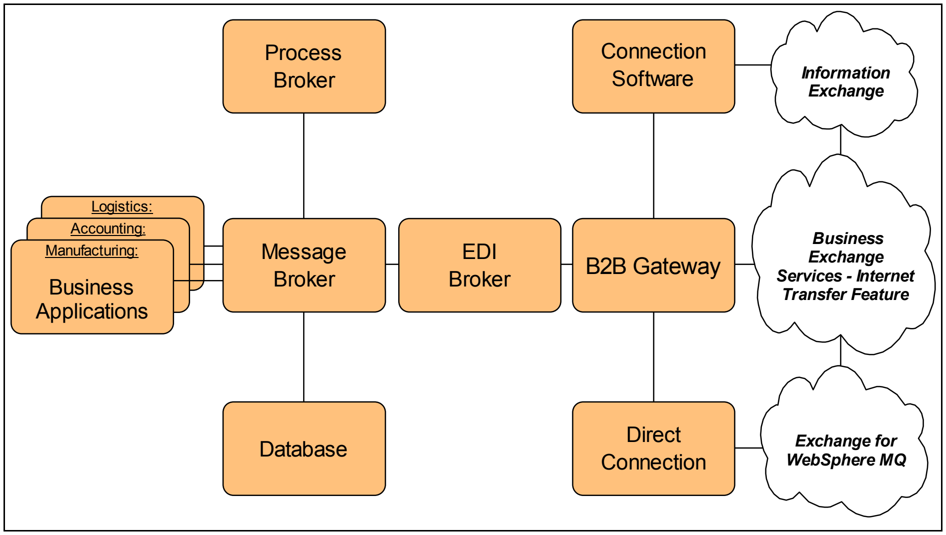


Figure 15-2 The IBM EDI solution

In the context of a typical enterprise integration architecture, WebSphere Data Interchange performs the specialist EDI validation, transformation, and exchange functions, and propagates the resulting transformed information either internally or externally. Internal propagation of transformed information may be via a message broker, a process broker, direct to the business applications, or any combination of those depending on the needs of the enterprise. External propagation of transformed information or receipt of information may be via a specialized dedicated VAN gateway, an Internet B2B gateway, directly to a trading partner, or any combination of those interfaces depending on the nature of the trading relationship between the enterprise and its trading partner.

See also 5.2.5, “WebSphere Data Interchange” on page 106 in Chapter 5, “Node types and Product descriptions”.

15.2.3 The iSoft Peer-to-Peer Agent

iSoft’s Peer-to-Peer Agent (P2PAgent) allows you to exchange documents between trading partners over the Internet in a secure and reliable way. The P2PAgent program can accept data from internal applications in a number of ways.

A traditional way of passing data to the agent is by delivering files to a given directory. The agent can filter through these files using a number of selection criteria to determine what to do with a given file. Also, when a file has been sent, you can choose to rename the file or to delete it. Simply preserving the file may result in the file being sent multiple times. The file system can also be used to store received files. You can configure the agent in such a way that the original file name (as it was named by the sender) is preserved, or that the file name is generated. This last option can avoid files being accidentally overwritten.

A more recent addition to the product is support for WebSphere MQ. The P2PAgent can retrieve messages from an inbox queue. It considers each message as a separate entity that should be sent to the correct destination trading partner. Also, when the agent receives documents from trading partners, it can store the received document as a single message in a queue. By default, such a message will be prefixed with an MQRFH2 header that contains meta-data information, such as the trading partner that had sent the document and the target trading partner ID. The MQRFH2 header is constructed in such a way that this information is also available to JMS clients in the form of message properties.

Further internal data delivery mechanisms include support for SMTP and HTTP. A received document or a received receipt can be delivered as an e-mail to a configured e-mail address via an SMTP server. HTTP communication for sending documents is used, for example, in a multi-machine setup of iSoft’s P2PAgent. But if your internal applications can hand over EDI documents via HTTP, then they can hook into the P2PAgent directly.

Since the P2PAgent program is an AS2 client program, it is no surprise that the agent supports HTTP and HTTPS for sending and receiving documents via the Internet. The agent is also an AS1 client, which means that it needs to support SMTP for sending and receiving documents as an e-mail attachment.

Figure 15-3 summarizes the support for the different techniques to move data to and from the agent.

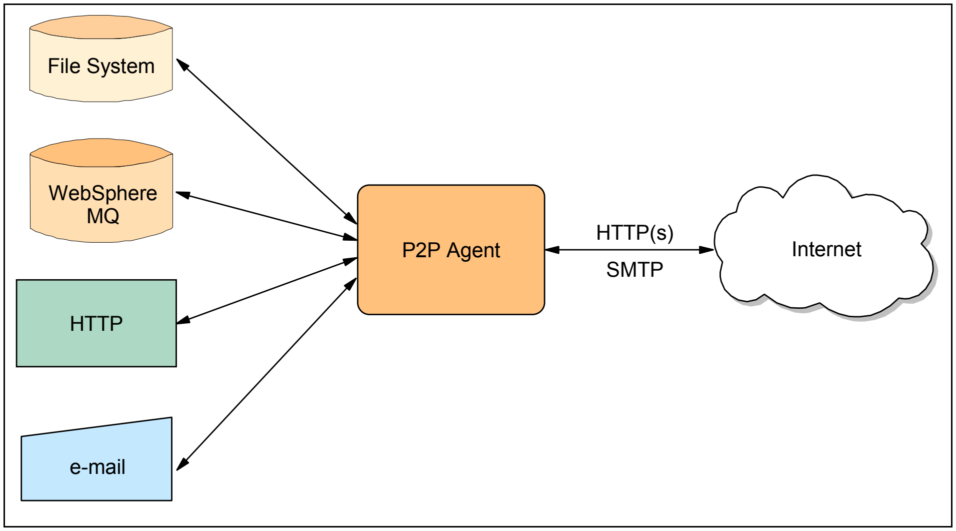


Figure 15-3 Inbound and outbound communication options

15.2.4 Integrating iSoft with WebSphere Data Interchange

Figure 15-4 shows the source application can pass outbound data to the EDI translation engine and WebSphere Data Interchange via a queue, using JMS (or the standard MQ API) with WebSphere MQ V5.3.1. Based on the setup of WebSphere Data Interchange, the translation engine then generates an EDI document in a queue, where it is picked up by the iSoft P2PAgent.

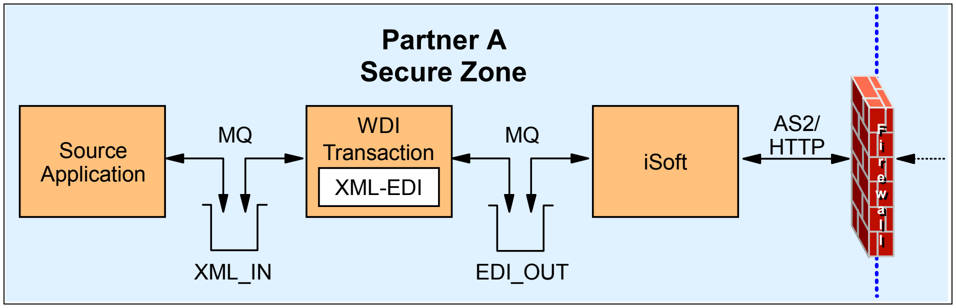


Figure 15-4 Integrating iSoft with WebSphere Data Interchange

For inbound communication, the iSoft P2PAgent drops the EDI document in a queue, where the translation engine retrieves it. The translation engine then produces a new document in an internal format and stores it as a message in a queue. The source application retrieves the inbound message using JMS.

15.3 Development guidelines

When installing and configuring a WebSphere Data Interchange environment, we need to make a distinction between a development environment and a runtime environment. In the development environment, users will create maps and profiles. The runtime environment is the environment where the translation and distribution of EDI messages will occur. The runtime platform may be different from the development environment platform.

15.3.1 Development environment

The development environment consists of the WebSphere Data Interchange Client on a number of Windows machines and a DB2 database accessed via ODBC, as shown in Figure 15-5. The database itself can be local or remote. It can even be on a non-Windows platform, such as AIX.

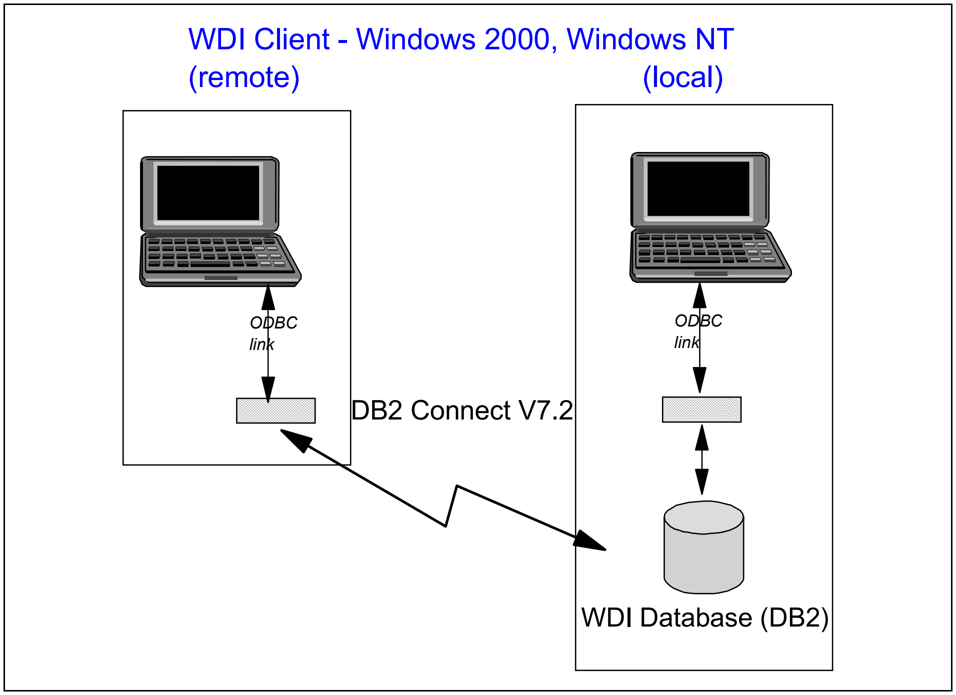


Figure 15-5 WebSphere Data Interchange development environment

Development machines that need access to the WebSphere Data Interchange database will need the DB2 Connect™ product to be able to connect to the database.

The WebSphere Data Interchange Client environment is also used to configure several aspects of the runtime environment that are not directly related to building maps. Settings such as partner profiles, queue profiles, and locations and names of files are all set via the WebSphere Data Interchange Client. This functionality of the client might have an impact on an installation and how database access and security is configured.

Given the possibilities for using the WebSphere Data Interchange Client, the installation and configuration of a WebSphere Data Interchange development environment will involve configuration work at the database level and possibly at the WebSphere MQ level.

15.3.2 Runtime environment

The runtime environment for a WebSphere Data Interchange solution can be quite broad, with many complex interactions between WebSphere Data Interchange itself and other applications, as shown in Figure 15-6 on page 326. But in general, WebSphere Data Interchange operates in two modes:

- Batch mode
- Launching the WDIAdapter program

In batch mode, an automation product launches the ediservr program. This program receives instructions from a command file that contains so-called PERFORM and other commands. The WebSphere Data Interchange Server will then execute these commands, which will usually result in sending, receiving, and translating a number of EDI documents that are ready to process. One reason for doing this in a batch mode is to save costs. Combining several EDI documents into a single EDI transmission document is often cheaper than sending each individual document separately.

When EDI documents are being sent over the Internet, there is no reason to delay transmission. Communication costs for the Internet are normally not measured on a transaction basis. Therefore, a company that uses an Internet connection for EDI communication wants to send EDI documents as soon as they become available. In this situation, the second method of using WebSphere Data Interchange is launching the WDIAdapter program. This program is designed to be started by WebSphere MQ’s trigger monitor. As shown in Figure 15-6, when a message arrives in queue from enterprise applications, the trigger monitor will start the WDIAdapter program, which will translate it according to the definitions in the WebSphere Data Interchange database. The translated message is then written on another queue for transmission by an Internet gateway product, such as iSoft or CrossWorldsTPI, or by WebSphere MQ itself.

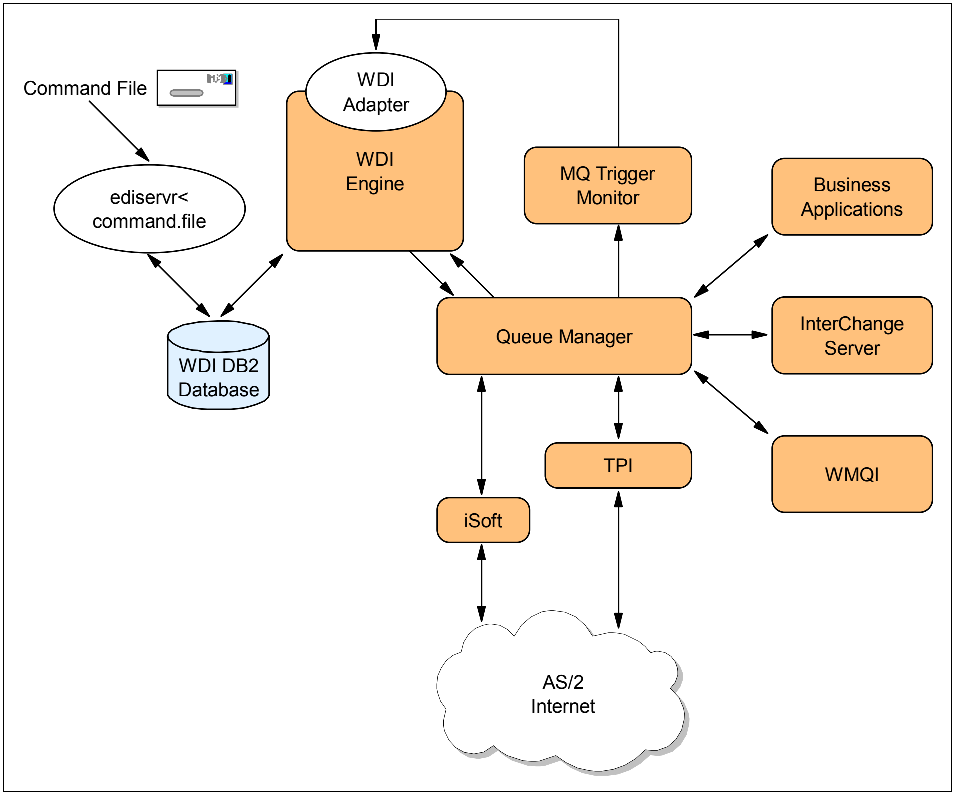


Figure 15-6 WebSphere Data Interchange runtime environment

Given the possible scenarios in a runtime environment, the configuration and installation of the WebSphere Data Interchange Server will usually include database tasks as well as WebSphere MQ tasks.

Part 5

Appendixes

A

Scenarios lab environment

In this appendix we describe the lab setup we used when deploying our Direct Connection Web services scenarios.

We then explain how to set up the ITSO Electronics sample in the IBM WebSphere Studio Application Developer development/test environment.

Lab setup

Figure A-1 shows the Windows lab environment we used when deploying the Direct Connection Web services scenarios described in Part 3, “Application Integration scenarios” on page 213 and in Part 4, “Extended Enterprise scenarios” on page 297.

The scenarios are:

1. Application Integration using RPC and document style Web services:
   - Chapter 8, “Using RPC style Web services” on page 147
   - Chapter 9, “Using document style Web services” on page 183
2. Application Integration using Web services and the Web Services Gateway:
   - Chapter 10, “Using the Web Services Gateway” on page 215
3. Extended Enterprise using Web services and the Web Services Gateway:
   - Chapter 14, “Using inter-enterprise Web services” on page 299
4. Application Integration using the Web Services Gateway to access a CICS application:
   - Chapter 11, “Using the Web Services Gateway with J2EE Connectors” on page 237
5. Application Integration using Web services to access a .NET application:
   - 9.5, “Integration with .NET-based Web services” on page 205

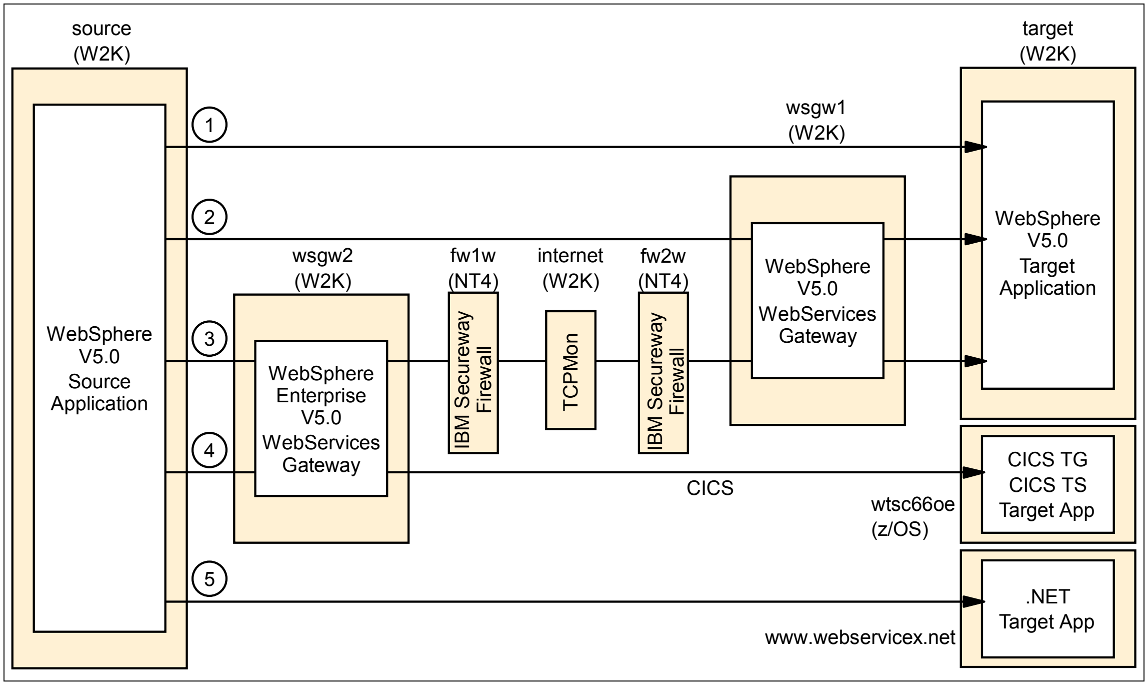


Figure A-1 Lab environment

Sample application setup

To install the ITSO Electronics sample application in the IBM WebSphere Studio Application Developer V5.1 for Windows workspace:

1. Download the ITSO Electronics sample. See Appendix B, “Additional material” on page 335 for details.
2. Extract the ITSO Electronics sample to the required location, for example `C:\` on Windows
3. Stop WebSphere Application Server if it is running locally.
4. Start Application Developer using the `-data` option to specify the ITSO Electronics workspace folder. For example, on Windows:

   ```
   wsappdev -data C:\workspace
   ```

5. Import the ITSO Electronics projects into your WebSphere Studio workspace:
   a. Select **File** → **Import** from the Studio main menu.
   b. In the Import window, select **Existing Project into Workspace** as the import source and click **Next**.

c. In the next window, click **Browse**, then navigate to the **Build** project folder. For example, on Windows:

```
C:\workspace\Build
```

Click **OK**, then click **Finish** to import the project.

d. Repeat steps a to c for each of the remaining ITSO Electronics projects:

- ITSOSourceAppWeb
- ITSOSourceApp
- ITSOTargetAppEJB
- ITSOTargetAppWeb
- ITSOTargetApp

6. Set up the service endpoint DNS names.

Our development/test environment consisted of two machines, one machine running the ITSOSourceApp and ITSOTargetApp in the WebSphere Studio test environment, and a second machine running the Web Services Gateway on WebSphere Application Server.

You can use the sample application without modifying the WSDL endpoint addresses if you add the entries shown in Example A-1 to your system **`hosts`** file on both machines (<WINDIR>\system32\drivers\etc\hosts on Windows). Just substitute **`10.10.10.10`** with the IP address of your WebSphere Studio machine, and **`10.10.10.11`** with the IP address of your Web Services Gateway machine.

Example: A-1 Sample application hosts file entries

```
...
10.10.10.11 wsgw1.itso.ral.ibm.com wsgw1
10.10.10.11 wsgw2.itso.ral.ibm.com wsgw2
10.10.10.10 target.itso.ral.ibm.com target
...
```

7. Deploy the following services on your Web Services Gateway(s):

– InventoryDoc, as described in Chapter 10, “Using the Web Services Gateway” on page 215.

– InventoryCics, as described in Chapter 11, “Using the Web Services Gateway with J2EE Connectors” on page 237.

8. Generate the Web service classes and deployment descriptors.

You can do this as described in the scenarios chapters, Chapter 8, “Using RPC style Web services” on page 147 through to Chapter 11, “Using the Web Services Gateway with J2EE Connectors” on page 237.

Alternatively, you can use the **`wsdeployall.cmd`** command script in the Build project as follows:

a. Select, then right-click the **Build/wsdeployall.cmd** script, and select **Open With** → **Text Editor** from the pop-up menu.

b. Set the **WAS_HOME** and **STUDIO_WSPACE** variables for your environment.

c. Save your changes and close the editor.

d. Right-click the **wsdeployall.cmd** file and select **Open With** → **System Editor** to launch the script.

> **Tip:** To skip deployment of any of the Web service servers or clients, comment out the `call` statement(s) in wsdeployall.cmd for the service. Use the WSDL file name immediately before each call statement to work out which service is which. (The Inventory and InventoryDoc services have two call statements; one to emit the Web service server and one for the client.)

9. Generate the EJB deployment code:

   a. Select, then right-click the **ITSOTargetAppEJB** project, and select **Generate** → **Deploy and RMIC Code** from the pop-up menu.

   b. In the Generate Deploy and RMIC Code window, click **Select all** to select all the EJBs, then click **Finish**.

10. Start the applications:

    a. Right-click **ITSOTargetApp** and select **Run on Server...**.

    b. In the Server selection window, set the Server type to WebSphere version 5.0 Test Environment and click **Finish**.

    c. Right-click **ITSOSourceAppWeb** and select **Run on Server...**.

    The ITSO Electronics welcome page should appear, as shown in Figure 8-2 on page 150.

In Chapter 11, "Using the Web Services Gateway with J2EE Connectors" on page 237, we implement the ITSOConnectorApp for deployment on the Web Services Gateway. To install this application in the IBM WebSphere Studio Application Developer Integration Edition V5.0 for Windows workspace:

1. Start WebSphere Studio Integration Edition using the `-data` option to specify the workspace folder. For example, on Windows:

   ```
   wsappdevie -data C:\workspaceIE
   ```

2. If needed, import the CICS ECI resource adapter, as described in "Import the resource adapter into the workspace" on page 242.

3. Import the ITSOConnectorApp.ear file into your WebSphere Studio Integration Edition workspace:

a. Select **File** → **Import** from the Studio main menu.

b. In the Import window, select **EAR file** as the import source and click **Next**.

c. In the next window, click **Browse**, then navigate to the **ITSOConnectorApp.ear** file. For example, on Windows:

```
C:\workspace\Build\ITSOConnectorApp.ear
```

Click **OK**, then click **Finish** to import the EAR file.

B

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described in this appendix.

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/SG246933

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246933.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
sg246933.zip	Zipped ITSO Electronics sample

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	20 MB
Operating System:	Windows, AIX, Linux
Processor:	500 MHz Pentium, pSeries®
Memory:	384 MB RAM minimum

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.

Abbreviations and acronyms

B2B	Business-to-business
API	Application Programming Interface
BPM	Business Process Management
CCI	Common Client Interface
CICS	Customer Information Control System
CICS TG	CICS Transaction Gateway
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
DEBA	Distributed Event-Based Architecture
DMZ	Demilitarized zone
DNS	Domain Name System
DOM	Document Object Model
EA	Enterprise Architecture
EAI	Enterprise Application Integration
EAR	Enterprise Archive
ECI	External Call Interface
EDI	Electronic Data Interchange
EIS	Enterprise Information System
EJB	Enterprise JavaBean
EPI	External Presentation Interface
ERP	Enterprise Resource Planning
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	J2EE Connector
J2EE	Java 2 Platform, Enterprise Edition
JAR	Java archive
JDBC	Java database connectivity
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
JSR	Java Specification Requests
JTA	Java Transaction API
JVM	Java Virtual Machine
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MQAI	WebSphere MQ Administration Interface
MQSC	WebSphere MQ Commands
MVC	Model-View-Controller
OAM	Object Authority Manager
OLTP	online transaction processing
ORB	Object Request Broker
PDA	Personal Digital Assistant
PKI	Public-Key Infrastructure
QoS	Quality of Service
RACF	Resource Access Control Facility

RAR	Resource Adapter Archive
RMI	Remote Method Invocation
SAX	Simple API for XML
SOA	Service oriented architecture
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TPA	Trading Partner Agreement
UDDI	Universal Description Discovery and Integration
URL	Uniform Resource Locator
VAN	Value Added Networks
WAR	Web Archive
WAS	WebSphere Application Server
WLM	Workload Management
WSDL	Web Services Description Language
WS-I	Web Services Interoperability Organization
WSIF	Web Services Invocation Framework
XML	Extensible Markup Language
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 about ordering these publications, see “How to get IBM Redbooks” on page 341. Note that some of the documents referenced here may be available in softcopy only.

- *WebSphere Data Interchange Installation and Configuration*, REDP3600
- *Implementation of iSoft and Integration with an EAI solution*, REDP3625
- *Revealed! Architecting Web Access to CICS*, SG24-5466
- *Securing Web Access to CICS*, SG24-5756
- *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133
- *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series*, SG24-6195
- *B2B solutions using WebSphere Business Connection*, SG24-6197
- *IBM WebSphere V5.0 Applications: Ensuring High Performance and Scalability*, SG24-6198
- *MQSeries Publish/Subscribe Applications*, SG24-6282
- *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401
- *MQSeries Programming Patterns*, SG24-6506
- *Using Web Services for Business Integration*, SG24-6583 (to be released late in 2003)
- *Patterns: Self-Service Application Solutions Using WebSphere Application Server V5*, SG24-6591
- *EJB 2.0 Development with WebSphere Studio Application Developer*, SG24-6819
- *Patterns for the Edge of Network*, SG24-6822
- *WebSphere Version 5 Web Services Handbook*, SG24-6891

Other publications

These publications are also relevant as further information sources:

- Jonathan Adams, Srinivas Koushik, Guru Vasudeva, George Galambos, *Patterns for e-business: A Strategy for Reuse*, IBM Press, 2001, ISBN 1-931182-02-7
- *CICS Application Programming Guide*, SC33-1687
- *CICS RACF Security Guide*, SC33-1701
-

Sun ONE article *Riddle Me This: Is Your XML Data Safe?* by Brett Mendel:

`http://sunonedev.sun.com/building/tech_articles/xmldata.html`

-

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 aphaWorks

  `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 Business Integration Adapters

  `http://www.ibm.com/websphere/integration/wbiadapters`
- IBM WebSphere Data Interchange

  `http://www.ibm.com/software/integration/wdi/`
- 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
- Apache Web Services Project

  http://ws.apache.org/
- Apache XML Project

  http://xml.apache.org/
- ebXML

  http://www.ebxml.org/
- Java Community Process

  http://www.jcp.org/
- OASIS Web Services Security (WSS) Technical Committee

  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
- Sun ONE for Developers

  http://sunonedev.sun.com/
- Sun Java 2 Platform, Enterprise Edition

  http://java.sun.com/j2ee
- Sun Java Technology Products and APIs

  http://java.sun.com/products/
- WebserviceX.NET

  http://www.webservicex.net/
- Web Services Interoperability Organization

  http://www.ws-i.org/
- World Wide Web Consortium (W3C)

  http://www.w3.org/

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

Index

Redbooks

Patterns: Direct Connections for Intra- and Inter-enterprise

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Patterns: Direct Connections for Intra- and Inter-enterprise

Select an intra- or inter-enterprise integration approach

Explore J2EE, Web services and EDI solutions

Learn by example with sample scenarios

The Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying Web applications. This IBM Redbook focuses on point-to-point application integration using the Process-focused Application Integration::Direct Connection application pattern for intra-enterprise, and the Extended Enterprise::Exposed Direct Connection application pattern for inter-enterprise.

Part 1 guides you through the process of selecting an Application and Runtime pattern. Next, the platform-specific Product mappings are identified based upon the selected Runtime pattern.

Part 2 presents guidelines on applying the Patterns approach to a sample business scenario and on selecting application integration technologies.

Part 3 provides detailed design, development, and runtime guidelines for intra-enterprise integration solutions. It teaches you by example using IBM WebSphere Application Server V5.0 with Web services, J2EE Connectors, and JMS.

Part 4 provides detailed design, development, and runtime guidelines for inter-enterprise integration solutions. It teaches you by example using IBM WebSphere Application Server V5.0 with Web services.

SG24-6933-00 ISBN 0738453099