

WebSphere and .Net Interoperability Using Web Services

Examples and guidance for building
interoperable Web services

Roadmap to Web services
specifications

Using Service-Oriented
patterns



Peter Swithinbank
Francesca Gigante
Hedley Proctor
Mahendra Rathore
William Widjaja



International Technical Support Organization

WebSphere and .Net Interoperability Using Web Services

June 2005

Archived

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

Archived

First Edition (June 2005)

This edition applies to WebSphere Studio Application Developer V5.1.2 running on Microsoft Windows XP Pro, WebSphere Application Server V5.1.1 with DB/2 8.1 running on Microsoft Server 2003, Microsoft.Net Framework 1.1, and Microsoft IIS V6.0 running on Microsoft Server 2003.

© Copyright International Business Machines Corporation 2005. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
The team that wrote this redbook	xiii
Become a published author	xv
Comments welcome	xv
Chapter 1. Introduction	1
1.1 Background of this book	2
1.1.1 The scenario	2
1.1.2 Use of Web services	3
1.1.3 Other approaches to interoperability	3
1.1.4 WS-I	4
1.1.5 Audience	5
1.1.6 Terminology	6
Part 1. Introduction to Web services	9
Chapter 2. SOAP primer	11
2.1 What is SOAP?	12
2.2 SOAP components	12
2.3 What is in a SOAP message?	14
2.3.1 Headers	14
2.3.2 Body	16
2.3.3 Fault	16
2.4 Message styles	18
2.4.1 RPC-Style	18
2.4.2 Document-Style	19
2.4.3 Document/Wrapped	20
2.5 SOAP interaction styles	21
2.5.1 Request-response	21
2.5.2 One-way	22
2.6 SOAP implementations over Http:	22
2.6.1 Microsoft .Net SOAP request over Http	22
2.6.2 IBM WebSphere Application Server SOAP request over Http:	23
2.7 Summary: Salient interoperability features of SOAP	24
Chapter 3. WSDL primer	27

3.1	Structure of WSDL definitions	28
3.2	Examples of WSDL definitions	30
3.2.1	Document/Literal Style	30
3.2.2	RPC/Literal Style	33
3.3	Future considerations	35
3.4	Summary: salient interoperability features of WSDL	36
Chapter 4. Web services primer		39
4.1	Web services concepts	40
4.1.1	What is a Web service?	41
4.1.2	Web services technologies	43
4.1.3	Web service properties	43
4.2	Web services and component architectures	45
4.2.1	Choosing between Web services and software components	46
4.3	Service-Oriented Architecture	50
4.3.1	Components of a Service-Oriented Architecture	51
4.3.2	Services and Web services	54
4.4	Web services and the Enterprise Service Bus	58
4.4.1	Transparency	60
4.4.2	Interoperability	61
4.4.3	Unified service discovery and addressing	61
4.4.4	Coexistence	62
4.4.5	Single point of control	62
4.4.6	Security	63
4.4.7	Robustness	63
4.4.8	Scalability	64
4.4.9	Problem determination	65
4.4.10	Conclusions: Web services, the ESB and service buses	65
4.5	Summary	66
Part 2. Web services interoperability		67
Chapter 5. Business scenarios		69
5.1	Business scenarios overview	70
5.2	Mergers and Acquisitions	71
5.2.1	Business goals	71
5.2.2	Solution context	72
5.2.3	Current IT infrastructure	74
5.2.4	Technical constraints	76
5.2.5	Solution level design	77
5.2.6	Technical approach	78
5.2.7	Target IT infrastructure	81
5.3	External claims assessor management	83
5.3.1	Business goals	83

5.3.2	Solution context	84
5.3.3	Current IT infrastructure	84
5.3.4	Technical constraints	85
5.3.5	Solution level design	85
5.3.6	Technical approach	88
5.3.7	Target IT infrastructure	90
5.4	Summary	92
Chapter 6. Interoperability patterns		93
6.1	The Patterns for e-business layered asset model	94
6.2	SOA approach and Patterns for e-business	95
6.2.1	Business::Self-Service pattern	96
6.2.2	Extended Enterprise business pattern	98
6.2.3	Discussion of patterns and Web services	98
6.3	Applying Interoperability patterns	102
6.3.1	Mergers and Acquisitions scenario	103
6.4	Summary	111
6.5	Where to find more information	112
Chapter 7. Web services roadmap		113
7.1	Introduction	114
7.2	List of Web services specifications	114
7.3	Summary of the Web services architecture stack	120
7.3.1	Foundations	121
7.3.2	Messaging	124
7.3.3	Security	129
7.3.4	Transacted	132
7.3.5	Meta-data	135
7.3.6	Resources	137
7.3.7	Composition	138
7.3.8	Management	140
7.3.9	Provisioning	143
7.3.10	WS-I	143
7.4	Summary	143
Chapter 8. Web service specifications		145
8.1	Web service Interoperability Organization (WS-I)	146
8.2	WS-I Basic Profile 1.0	146
8.2.1	Basic Profile 1.0 for WebSphere	148
8.2.2	Basic Profile 1.0 for Microsoft .Net	149
8.2.3	Summary	155
8.3	Interoperability standards: addressing	155
8.3.1	Insurance example	155
8.3.2	Summary	158

8.4 Security	158
8.4.1 Why do we need more security specifications?	158
8.4.2 WS-Security 2004	161
8.4.3 WS-I Security Profile	170
8.4.4 Summary	176
8.5 WS-Coordination	177
8.6 WS-Transactions	179
8.6.1 WS-Transaction in a WebSphere environment	181
8.6.2 WS transaction in a Microsoft .Net environment	182
8.7 Reliable messaging	182
8.7.1 What is WS-ReliableMessaging?	183
8.7.2 The three legged handshake protocol	183
8.7.3 WS-ReliableMessaging Protocol	184
8.7.4 Reliable messaging requirements	186
8.8 SOAP/JMS and SOAP/MQ	188
8.8.1 Interoperability of SOAP/JMS and SOAP/MQ	189
Chapter 9. Web services in Microsoft .Net and WebSphere	191
9.1 Microsoft .Net architecture	192
9.1.1 Microsoft .Net Web service application architecture	194
9.1.2 Developing software using Microsoft Visual Studio .Net 2003	196
9.1.3 Microsoft secure Web services implementation	200
9.2 WebSphere Java 2 Enterprise Edition architecture	201
9.2.1 Java 2 Enterprise Edition Web service architecture	204
9.2.2 Developing J2EE applications using WebSphere Studio Application Developer	206
9.2.3 IBM secure Web services implementation	208
9.2.4 Summary	212
Chapter 10. Deploying Web services	215
10.1 Overview	216
10.1.1 Web services publishing	216
10.2 WebSphere Web services deployment model	217
10.2.1 Web Services Gateway	217
10.2.2 IBM UDDI registry	220
10.2.3 Deployment architecture	220
10.3 Microsoft .Net Web service deployment model	222
10.3.1 Microsoft UDDI registry	222
10.3.2 Deployment architecture	223
10.4 Summary	226
Part 3. Claims scenario	227
Chapter 11. Designing the scenarios	229

11.1 Mergers and Acquisitions scenario	230
11.1.1 Use cases overview	230
11.1.2 Actors	231
11.1.3 Use case 001: Register claim	232
11.1.4 Realizing the use case	234
11.2 External Claims Assessors scenario	240
11.2.1 Use cases overview	240
11.2.2 Actors	240
11.2.3 Use case 002: Manage external claim assessors	241
11.2.4 Realizing the use case	244
11.3 Claim applications: table schema	245
11.4 XML schema data types as common denominator	246
11.4.1 Data type mapping	246
11.4.2 SOAP message for registerClaim()	247
11.4.3 SOAP message for findCustomer()	248
11.4.4 SOAP exception for findCustomer()	249
11.5 Summary	250
Chapter 12. Building the claims scenario	251
12.1 Building the scenario for WebSphere	252
12.1.1 Problem definition	252
12.1.2 Solution	252
12.1.3 Import Enterprise JavaBeans	252
12.1.4 Test imported Enterprise JavaBeans	256
12.1.5 Create a Web service from Enterprise JavaBeans	262
12.1.6 Test the created Web service	269
12.1.7 Deploy the created Web service	271
12.2 Building the scenario for Windows Server 2003	276
12.2.1 Prerequisites to run the Web service application	276
12.2.2 Create the Web Service	276
12.2.3 Import the existing classes	279
12.2.4 Build the Web service	282
12.2.5 Microsoft Internet Information Services (IIS)	283
12.2.6 Create Microsoft .Net Test Client	285
12.2.7 Summary	292
12.3 Building the Web services clients	293
12.3.1 Web service client for the WebSphere Web service	294
12.3.2 Web service client for the Microsoft .Net Web service	300
12.3.3 Microsoft .Net	303
12.3.4 Differences between the two Web services and conclusions	303
Chapter 13. Web service interoperability implementation guidance	311
13.1 Web service interoperability guidance	312

13.2	WebSphere client	312
13.3	WebSphere Web service	314
13.4	Microsoft .Net client	315
13.5	Summary	315
Part 4.	Appendixes	317
	Appendix A. Installation and setup	319
	Installation planning for the WebSphere environment	320
	WebSphere Application Server V5.1.1.1 requirements	320
	Installing WebSphere Application Server 5.1.1.1	321
	Installation of Application Developer 5.1.2	323
	Installation planning for the Microsoft .Net environment	325
	Appendix B. Additional material	327
	Locating the Web material	327
	Using the Web material	328
	System requirements for downloading and running the Web material	328
	How to use the Web material	328
	Related publications	329
	IBM Redbooks	329
	Online resources	329
	How to get IBM Redbooks	333
	Help from IBM	333
	Abbreviations and acronyms	335
	Index	339

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:

@server®

@server®

Redbooks (logo) ™

alphaWorks®

developerWorks®

ibm.com®

ClearCase®

Cloudscape™

CICS®

DB2®

IBM®

IMS™

Lotus®

MQSeries®

Notes®

PAL®

Rational Unified Process®

Rational®

Redbooks™

Redbooks (logo)™

RUP®

S/390®

Tivoli®

WebSphere®

XDE™

The following terms are trademarks of other companies:

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

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

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

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

Screen shot(s) reprinted by permission from Microsoft Corporation.

Permission to copy and display the "Business Process Execution Language for Web Services Specification, version 1.1 dated May 5, 2003" (hereafter "the BPEL4WS Specification"), in any medium without fee or royalty is hereby granted, provided that you include the following on ALL copies of the BPEL4WS Specification, or portions thereof, that you make a link to the BPEL4WS Specification at these locations:

<http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp>

<http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html/bpel1-1.asp>

<http://ifr.sap.com/bpel4ws/>

<http://www.siebel.com/bpel>

Preface

In this IBM Redbook, we use Web services to integrate the insurance processes of two merged companies. One insurance company is using WebSphere Application Server to host its applications, the other is using Microsoft .Net.

In this typical scenario, the IT director has decided to realize the increased revenue opportunity brought about by the merger of the two companies by presenting its customers and agents with a single customer view. The plan is to reuse both companies' existing insurance applications by converting them to Web services and integrating them with common business processes.

Our goal is to explore the following question:

How do we use the Web service capabilities in IBM and Microsoft®'s flagship WebSphere and Microsoft .Net products to integrate applications running in the two environments?

To give you a broad understanding of the capabilities and direction of Web services, we survey the specifications that have been published and the progress that is being made to ensure that implementations of the specifications work together. We also include a practical guide to using the Web services capabilities of WebSphere Studio Application Developer and Microsoft Visual Studio .Net 2003 by building the scenario.

The contents of the book are as follows.

- ▶ Part 1, "Introduction to Web services" on page 9 is an introduction to Web services.
 - Chapter 2, "SOAP primer" on page 11 describes what is inside a SOAP message.
 - Chapter 3, "WSDL primer" on page 27 describes how WSDL definitions are structured, and how they are used to generate SOAP messages.
 - Chapter 4, "Web services primer" on page 39 introduces the concepts of Web services and the relationship of Web services to Service-Oriented Architecture (SOA) and the Enterprise Service Bus (ESB).
- ▶ Part 2, "Web services interoperability" on page 67 covers the scenario requirements, solution architecture and surveys current and future Web services technology.
 - Chapter 5, "Business scenarios" on page 69 describes the business scenario what will be used throughout the book to define requirements,

and to provide the context for the examples of using Web services to build a solution using services hosted on WebSphere Application Server and Microsoft .Net.

- Chapter 6, “Interoperability patterns” on page 93 discusses mapping Patterns for e-business to a Service-Oriented Architecture implemented using Web services and identifies the patterns that are used to implement the business scenario.
- Chapter 7, “Web services roadmap” on page 113 summarizes most of the Web service specifications that have been published.
- Chapter 8, “Web service specifications” on page 145 looks at the work of the WS-I organization, explains the published WS-I profiles and looks in more detail at the security, transaction and reliable messaging specifications which are among the more important Web service specifications to build a robust Web services based infrastructure.
- Chapter 9, “Web services in Microsoft .Net and WebSphere” on page 191 describes how Web services are implemented in the Java 2 Enterprise Edition architecture of WebSphere Application Server 5.1 and Microsoft .Net architecture in Microsoft Server 2003.
- Chapter 10, “Deploying Web services” on page 215 discusses Web service deployment using UDDI in WebSphere Application Server and Microsoft Server 2003.
- ▶ Part 3, “Claims scenario” on page 227 continues the scenario development by taking the Runtime pattern from Chapter 6, “Interoperability patterns” on page 93 and mapping it to WebSphere and Microsoft .Net and then describes how to build the solution and some of the lessons we learned.
 - Chapter 11, “Designing the scenarios” on page 229 returns to the business scenario and describes the use cases that are implemented in the examples.
 - Chapter 12, “Building the claims scenario” on page 251 implements the first “Register Claim” scenario, showing a simple solution composed of services provided in both a Microsoft .Net and a Java 2 Enterprise Edition environment.
 - Chapter 13, “Web service interoperability implementation guidance” on page 311 looks at some differences we found when building Web services with WebSphere Studio Application Developer and Microsoft .Net Studio 2003, and what to do about them.
- ▶ Appendix A, “Installation and setup” on page 319 provides some hints and tips and instructions for building the scenario for yourself from the materials that are provided with the redbook.

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.



Mahendra, Peter, Francesca, William and Hedley

Peter Swithinbank is a new project leader at the International Technical Support Organization, Raleigh Center. Before joining the ITSO, he worked in IBM System House developing Business Scenarios for IBM Software Group. He has 26 years of experience in IBM. He holds a degree in Geography from Cambridge and a Diploma in Software Engineering from Oxford University. His areas of expertise include scenario driven development, Web services, messaging and adapters.

Francesca Gigante is an IT Architect working for IBM Global Services in Rome, Italy. She has eight years of experience in IBM, starting early with Java™ technology and moving later to J2EE and WebSphere® technology. She has a wide experience in building EAI architectures and delivering e-business customer projects. She also worked for Nokia as a technical architect in the mobile services area. She holds a degree in Electronic and TLC Engineering from Bari Polytechnic University, Italy.

Hedley Proctor is a software engineer at IBM Hursley, England. He studied physics at Oxford University and philosophy at Durham University before joining IBM in September 2002. He worked on versions 5 and 5.1 of the WebSphere

SDK for Web Services, specializing in the Eclipse plug-ins, samples and interoperability. Two of his tutorials on secure Web services interoperability are available on developerWorks®.

Mahendra Rathore is a Senior Project Manager and manages an e-marketplace development project at the Oilpalmworld Sendirian Berhad in Kuala Lumpur, Malaysia. His experience includes over six years of managing IT projects involving e-business and J2EE solutions and over eleven years of consulting in the IT industry. He has been serving as a Board Member at the Project Management Institute (PMI) Singapore Chapter since 2003. He is a Certified Information System Auditor (CISA). Mahendra holds a Master's degree in Computer Applications and Bachelor Degree in Electronics from the University of Jodhpur, India and currently lives in Singapore.

William Widjaja graduated from Florida Institute of Technology, Melbourne, Florida with a BS and MS in Computer Science and an MBA. He has worked as a consultant for over 20 years in financial companies such as JP Morgan, CS First Boston, American Express Bank and the insurance company Guardian Life Insurance of America. He is a Sun Certified Java Programmer, Solaris 8 System and Network Administrator, Certified Oracle 8 DBA and Microsoft Certified Professional.

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

Alan Hui of IBM System House for advocating a redbook on Web services standards and interoperability.

IBM System House for funding a number of projects in IBM, including this redbook, to improve the integration of software products and help to realize on demand computing.

Bill Moore and Martin Keen at the International Technical Support Organization, Raleigh Center for organizing the project and helping with the process of producing Redbooks™.

Chris Ferris, an IBM Senior Technical Staff Member and IBM representative to the WS-I organization.

Hyen-Vui Chung, IBM WebSphere Application Server development leader for Security.

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbook@us.ibm.com

- ▶ Mail your comments to:

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

Archived



Introduction

In this introduction, we discuss the general background for this redbook and the details thereof.

Archived

1.1 Background of this book

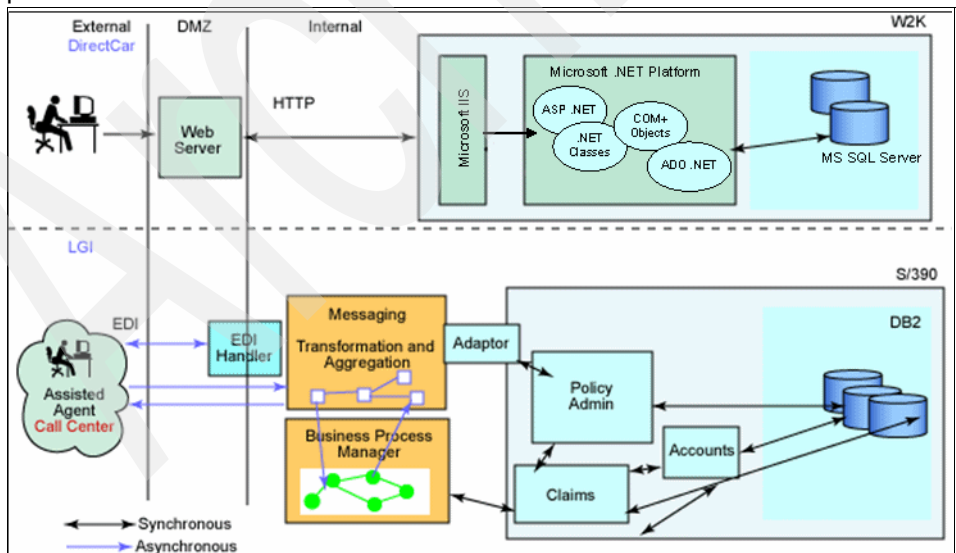
This book builds on work done by the IBM System House Business Scenarios team based in IBM's development laboratories in Hursley, England. Using typical business scenarios validated by IBM's customers, the System House scenario teams design and build solutions with the help of IBM's product development teams. The experience that is gained using this "outside in" design process is used to improve the integration of IBM's products. The experience is also used to publish Redbooks and developerWorks articles about how to build solutions. See:

<http://www-136.ibm.com/developerworks/scenarios/>

for more details.

1.1.1 The scenario

One of the companies is using WebSphere, CICS and WebSphere MQSeries to deliver insurance products through insurance agents and a call center. The other has developed a Microsoft .Net solution to provide a Web-only channel to sell car insurance directly to its customers. The merger of the two companies will increase revenues by providing multiple channels to its combined customer base over which it can sell its both sets of insurance products. The insurance company is also outsourcing its auto claims assessment process (loss adjustment) and using Web services to automate business processes shared with its business partners.



Merger of two insurance companies

The original scenario was published on developerWorks as “Merging disparate IT systems: Build a single integrated view for users quickly and with minimal disruption” and is available at:

<http://www-106.ibm.com/developerworks/ibm/library/i-merge.html>

The version of the scenario we use is described in Chapter 5, “Business scenarios” on page 69.

A major consideration in architecting the solution has been to realize the benefits of the merger quickly by keeping changes to existing systems and development environments as few as possible. The insurance applications developed and hosted in each insurance company need to be integrated into the Web, insurance agent and call center channels using a common service bus. Also, to keep the entry costs for claims assessors doing business with the insurance company as low as possible, the claims assessors are not tied to using a particular software platform by the insurance company.

1.1.2 Use of Web services

The company requires stable standards for interoperation of its development and runtime environments. It is running WebSphere Application Server and Microsoft Server 2003. It also needs to offer its business partners stable development and runtime interfaces to integrate their business processes with the insurance company’s automated claims assessor process, using whatever development and runtime environment suits each partner best.

In this redbook, we investigate how well Web services meet these interoperability requirements by building Web service interfaces to the insurance applications using WebSphere Studio Application Developer and Microsoft .Net Studio and providing standard Web service interfaces for business partners.

1.1.3 Other approaches to interoperability

Our approach to using Web services is only one of a number of ways to achieve interoperability between Microsoft .Net and Java 2 Enterprise Edition software systems. Microsoft and IBM have both published books on the Web that explore interoperability between Microsoft .Net and Java 2 Enterprise Edition, from a wider perspective than Web services.

The redbook *WebSphere and Microsoft .Net Coexistence*, SG24-7027 at <http://www.redbooks.ibm.com/abstracts/sg247027.html?Open> analyzes different coexistence scenarios, proposes various solution architectures and provides practical examples. Microsoft’s Patterns and Practices series includes *Application Interoperability: Microsoft .Net and J2EE* (found at <http://download.microsoft.com/download/7/2/6/7269f183-639a-4e99-bd84-cc>

[3e6515af86/PnP_J2EE_Interop_V1.pdf](#)). In the book, the authors compare Microsoft .Net and J2EE architectures, analyze interoperability between different layers of the two architectures and provide examples of making the layers interoperate. It is interesting to read the two accounts of Microsoft .Net and Java 2 Enterprise Edition side by side, and to contrast the solutions offered in the two books.

The method followed by the authors of the IBM and Microsoft books on Java and Microsoft .Net is to look at interoperability from a number of different technical and architectural perspectives. These books are essential reading to gain a good understanding of different ways to make Java 2 Enterprise Edition and Microsoft .Net work together.

Our objective is different. It is to look in detail at how to use Web services to implement the integration in a particular scenario. Where Web service capabilities are missing, we have not tried to implement an alternative solution that does not use Web services.

Limitations

Perhaps the most important missing capability is Web services security. We intend to demonstrate connecting the merged insurance company *securely* across the Internet with its external claims assessors using Web services. The solution depends upon WebSphere and Microsoft .Net implementing WS-Security 2004, and the WS-I organization approving the Basic Security Profile 1.0. Our expectation is that this will all happen in 2005 and we hope to return to this redbook then and build the external claims assessor part of the scenario which links the insurance company to its outsourced claims assessor.

1.1.4 WS-I

We have based our choice of Web service standards on the work of the WS-I organization. Part of this redbook is an account of WS-I and addresses why we think WS-I is so important to get Microsoft .Net and WebSphere (and other Web service vendors) to interoperate in a stable and economic fashion.

The standards profiles of the WS-I organization have only just started to appear in the last year. The practical scope of this redbook is limited to examining the interoperability of Web services software components defined by the WS-I using version 5.1 of the WebSphere platform software and the Windows® Server 2003 version of the Microsoft .Net platform.

1.1.5 Audience

There are many papers and books written about Web services for the IT professional who is principally concerned with creating a Web services infrastructure or building new Web service tools or runtimes. We wanted to write a book for IT consultants, architects, programmers and administrators who expect to build solutions using “out of the box” products from software vendors without having to employ specialists to build their own Web service infrastructures.

We wanted to re-use existing software components hosted on WebSphere Application Server and Microsoft .Net and integrate them into new solutions using a Service-Oriented Architecture. Our expectation is that Web services are reaching a level of maturity that will enable us to integrate existing software components into solutions using the tools that are provided by IBM and Microsoft. We want to know to what extent this is possible today without having to work around interoperability problems. We want to know what missing capabilities are being standardized and when the capabilities will be implemented and interoperable between Microsoft .Net and WebSphere Application Server.

Consultants, architects, programmers and administrators will find some chapters of this book of greater relevance.

- ▶ Consultants

Consultants need to advise their clients about interoperability in a Service-Oriented Architecture. They need to be able to separate fact from fiction: what can and cannot be done today in a heterogeneous environment, and what it is reasonable to expect from the next releases of WebSphere Application Server and Microsoft .Net.

Consultants will be interested in Chapter 5, “Business scenarios” on page 69, dealing with how the scenario is mapped to a Web services based solution. They will be interested in Chapter 6, “Interoperability patterns” on page 93, and in the roadmap for Web service standards in Chapter 7, “Web services roadmap” on page 113.

- ▶ Solution Architect¹

A Solution Architect plays the pivotal role in translating business requirements into the definition and scope of an IT project and is responsible for delivering value back to the business from its investment in IT.

The Solution Architect, in conjunction with the Application Architect, is responsible for making a decision about adopting a Web services based

¹ Also called the Systems Analyst in the Rational® Unified Process® (RUP®)

strategy for integrating the two insurance companies and automating the claims process with business partners.

The Solution Architect will be interested in the same chapters as the consultants and also in Chapter 8, “Web service specifications” on page 145.

- ▶ Application Architects²

An Application Architect is responsible for the major technical decisions that drive and constrain the development of the software system.

In addition to the chapters previously mentioned, the Application Architect will be interested in how Web services are implemented by IBM and Microsoft; this is detailed in Chapter 9, “Web services in Microsoft .Net and WebSphere” on page 191. Deployment of the solution is discussed in Chapter 10, “Deploying Web services” on page 215.

- ▶ Application Programmers

The Application Programmer will be interested in building and deploying the solution as detailed in Chapter 11, “Designing the scenarios” on page 229 and Chapter 12, “Building the claims scenario” on page 251. They should also look at 8.2, “WS-I Basic Profile 1.0” on page 146 to understand potential interoperability problems when developing Web services.

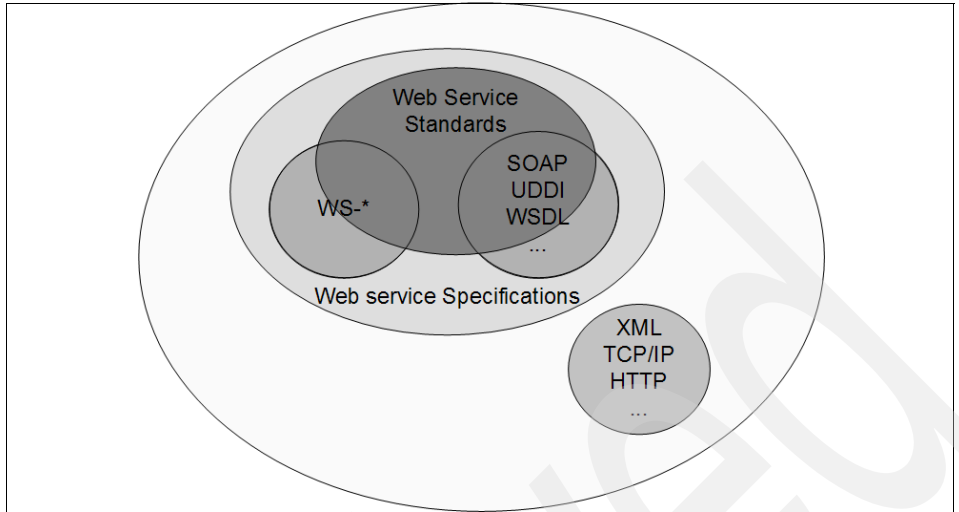
- ▶ System Administrators deploying a solution

The Systems Administrator will be interested in the pattern used to implement the scenario and UDDI and deployment, concepts behind Web services and the key standards WS-I references.

1.1.6 Terminology

We should clarify the use of the phrases “Web services standard” and “Web services specification” in this book. Web services are evolving rapidly and there are many Web service specifications and standards. It is confusing that the two terms are often used interchangeably. Unfortunately, the unqualified term “standard” is used rather loosely to refer to specifications in various states of standardization and can give the reader a spurious sense that the specification has been finally approved.

² Also known as the Software Architect in RUP



Web service specifications and standards Venn diagram

In this redbook, we will try to maintain the use of “Web services specification” when referring to any WS-* (pronounced WS splat) specification, and other specifications that are specifically about Web services, such as WSDL, SOAP and UDDI.

We will try to only use the term *Web service standard* when we are talking specifically in the context of standards-related activities, such as when using the expression “a proposed W3C Web service standard.” We hope this will avoid a misleading impression that all Web service specifications are finalized and ready to be used in solutions.

Archived



Part 1

Introduction to Web services

In this part, we sketch out the concepts of Web services, WSDL and SOAP. If you want to review more details, there are readily accessible online texts which will prove useful:

1. The IBM Redbook *WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook*, SG24-6891, available at:

<http://www.redbooks.ibm.com/abstracts/sg246891.html?Open>

This redbook provides a thorough description of Web services and discusses the WebSphere Application Server 5.1 implementation.

2. For a Microsoft perspective, visit their Web services Web page at:

<http://msdn.microsoft.com/webservices/understanding/default.aspx>

Archived

SOAP primer

SOAP is defined in Simple Object Access Protocol (SOAP) 1.1, W3C note 8, May 2000 (found at <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>) as follows:

“SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC.”

This is the version of SOAP referred to in WS-I basic Profile 1.1.

SOAP V1.2 (currently a recommendation of the W3C) shortens the definition to:

“SOAP Version 1.2 provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment.”

This is found at <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>

As witticists have it, SOAP is no longer simple, and it is not about objects! So what is SOAP?

2.1 What is SOAP?

SOAP was first created by Microsoft, UserLand, and DevelopMentor in 1998. The rule was: no new network technologies; use what is there and keep it simple so that it works. SOAP is seen as a way of overcoming the obstacles in making objects work together, though they may be written in different languages, running on different platforms, and communicating with different protocols over the Internet. It also enables programs to communicate through firewalls that are being administered to permit browser traffic.

SOAP rides on the wave of success of XML and the Internet. XML is the lingua franca for programs to exchange typed and structured information. Http: is the connectionless protocol to exchange information simply over the Internet. SOAP helps to clean up distributed application architecture by defining remote application access in terms of technology-neutral interfaces expressed in XML. SOAP is not yet another distributed component technology such as Java-RMI, DCOM or IIOP.

In summary, there are four factors that explain the popularity of SOAP:

- ▶ Technology-neutral
- ▶ Internet-friendly
- ▶ Simple to use to create compatible implementations on different platforms
- ▶ SOAP solutions are built on top of existing IT infrastructures

2.2 SOAP components

To enable an existing Internet-enabled infrastructure to communicate using SOAP, every node that sends or receives SOAP messages needs to have a SOAP component. These SOAP components put SOAP messages onto the Internet and take them off using the ubiquitous http protocol.

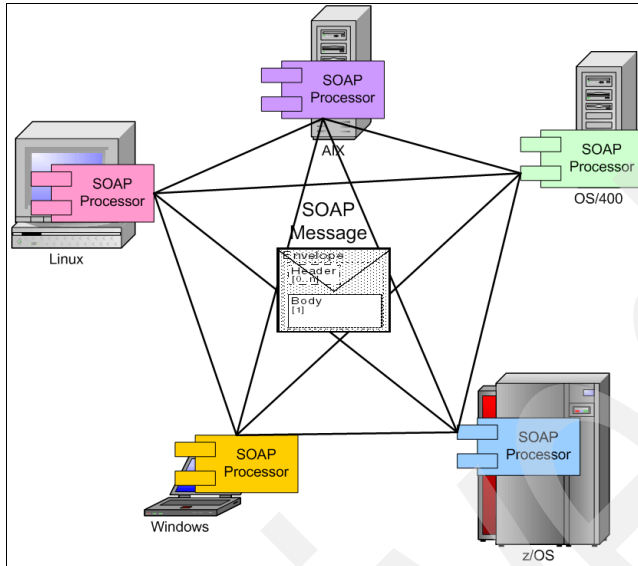


Figure 2-1 One SOAP processor per environment

The messages contain the type information necessary for the SOAP component to code and decode requests for any host software environment. With the type necessary to interface with host environments captured in the SOAP message in a technology-independent format, it is only necessary to write one SOAP processor for each software environment on the server (a very practical proposition) and not to write adapters for every combination of software environment in both the client and server.

In computer science jargon, it roughly reduces the n -squared size of a problem to n . What does this mean? As shown in Figure 2-1, each of the five types of nodes (say different operating systems or languages) has only one SOAP processor. If each node communicated in its native data types, then each partner would have to have a special partner SOAP processor to understand SOAP messages from each of its partners (four each, plus one to handle SOAP messages to itself). Each of the five SOAP processors would need to be written differently for the five nodes, so one would end up with 25 (n -squared) SOAP processors rather than just 5 (n).

The three most important innovations in SOAP have proven to be:

1. Using XML to type information being sent from one program to another; this enables programming language interoperability and is the natural transport for XML documents.

2. Isolating a separate application access layer (the SOAP component) to manage application routing.
3. Defining an extensible architecture to build application access and routing mechanisms. The architecture is implemented by the SOAP component which is independent of the network protocol and the software environment.
This SOAP component encapsulates application interoperability issues and is not concerned with the complications of different network transports and different software environments.

SOAP is complemented by two other technologies which we will look at in later chapters:

- ▶ WSDL is the second element of Web services and has proven to be just as important as SOAP. We will look at WSDL in the next chapter.
- ▶ The third element, UDDI, has not yet had such a large influence on software architecture; this reflects that we are only now moving on from the early adoption of Web services to their use in production applications. We will look at UDDI in Chapter 10, “Deploying Web services” on page 215.

2.3 What is in a SOAP message?

A SOAP message is an XML document. The outermost root element is the SOAP Envelope. These examples use the SOAP 1.1 specification to comply with WS-I basic profile 1.1

```
<?xml version='1.0'?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header> <!-- optional -->
    <!-- headers... -->
  </soap:Header>
  <soap:Body>
    <!-- payload or fault message -->
  </soap:Body>
</soap:Envelope>
```

Namespace defines SOAP version

Figure 2-2 SOAP envelope

2.3.1 Headers

A SOAP Header is optional. It is an extension mechanism to pass information in the SOAP message that is not part of the application payload. Inside a SOAP Header are header entries. Each header entry is a child of the SOAP Header.


```

<?xml version='1.0'?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://schemas.xmlsoap.org/soap/envelope/actor/next"
      env:mustUnderstand="1">
      <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:dateAndTime>
    </m:reservation>
  </env:Header>
  <env:Body>
    <!-- payload or fault message -->
  </env:Body>
</env:Envelope>

```

Figure 2-3 SOAP Header

Header entries:

- ▶ Must be qualified by a namespace and a local name to uniquely identify the header.
- ▶ May include a SOAP encodingStyle attribute, although use of XML schemas to type data is now preferred. This is not allowed in SOAP 1.2 except where indicated in the specification.
- ▶ May include the attribute mustUnderstand="1" or "0" (SOAP 1.1) or the logical values true or false (SOAP 1.2) to indicate if the recipient must interpret the header or not. The default is "0".
- ▶ May include the attribute actor, termed role in SOAP 1.2. role=URI. The role indicates which SOAP node should process the header.

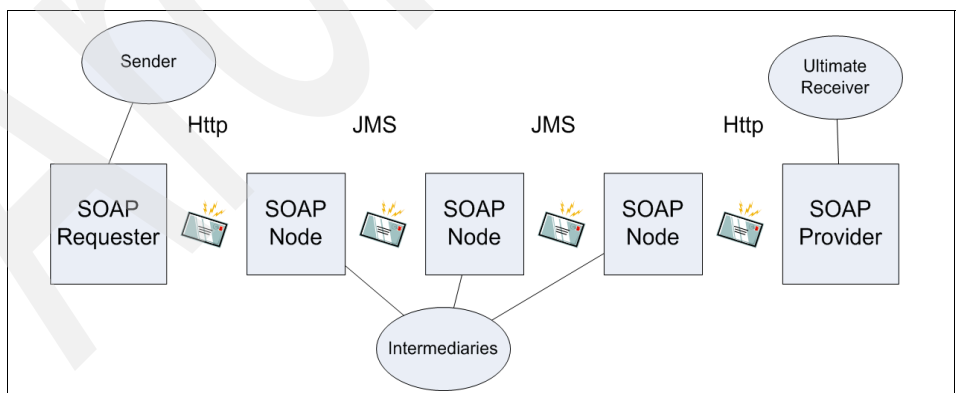


Figure 2-4 SOAP intermediaries

In SOAP 1.2, there are three standardized URIs defined for the role: none, next and ultimateReceiver. Alternatively, a specific URI may be specified. The default role is next. SOAP 1.2 introduced none and ultimateReceiver.

SOAP 1.2 adds an additional header entry attribute: relay. The default behavior is to remove header entries, unless the processing requires it to be reinserted. The relay attribute is used to specify that unprocessed headers are relayed rather than removed.

2.3.2 Body

The SOAP Body carries the application “payload” or fault messages. Typical uses of the Body are to provide parameters for RPC calls, exchange XML documents and report fault messages.

The Body is semantically equivalent to the Header. The only difference is that the Body is targeted at the final recipient, whereas the header is also processed by intermediates. From the final recipient’s perspective, the Header and Body are equivalent. It is up to the application designer how to use them.

The rules for the Body are:

1. A Body entry is an immediate child of the Body
2. A Body entry is identified by a its a fully qualified name: that is, a namespace and a local identifier. However, immediate children of the SOAP Body element are optionally namespace qualified.

```
<?xml version='1.0'?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope">
<!-- Header -->
<env:Body>
<p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
<p:departure>
  <p:departing>New York</p:departing>
</p:departure>
</p:itinerary>
</env:Body>
</env:Envelope>
```

Body entry local id and namespace

using local id, not explicitly providing namespace

Figure 2-5 SOAP Body

2.3.3 Fault

The SOAP Fault is a Body entry with the local Id of Fault that can only appear once. In SOAP 1.2, it must be the only element of the SOAP Body. It has the following sub-elements:

► Faultcode, called Code in SOAP 1.2

The Faultcode must be present, and must be a fully qualified name. In practice, one of the predefined SOAP faultcodes from the Envelope namespace is normally used:

- VersionMismatch
- MustUnderstand
- Client
- Server

► Faultstring, called Reason in SOAP 1.2

The Faultstring must be present, and provides a textual reason for the Fault. In SOAP 1.2, provision is made for multiple language versions of the Reason.

► Faultactor

Principally used by nodes other than the ultimate receiver to provide information (in the form of a URI) about where a Fault occurred.

► Detail

Provides application specific details about a fault incurred when processing the Body of a SOAP message. Errors in processing the Header must be reported in the Header entries. Absence of the detail element in a Fault message absolutely distinguishes errors caused by processing the Header or the Body.

```
<?xml version='1.0'?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope">
<!-- Header -->
<env:Body>
  <env:Fault>
    <env:Faultcode>Server</env:Faultcode>
    <env:Faultstring>Processing error</env:Faultstring>
    <env:Detail>
      <e:myFaultDetails xmlns:e="http://travelcompany.example.org/faults">
        <e:message>Name does not match card number</e:message>
        <e:errorCode>999</e:errorCode>
      </e:myFaultDetails>
    </env:Detail>
  </env:Fault>
</env:Body>
</env:Envelope>
```

Figure 2-6 SOAP fault due to error in Body

2.4 Message styles

There are two main SOAP message styles, RPC and Document. The intent of RPC-Style is to model a remote procedure call as an XML document, whereas Document-Style is intended for the transmission of XML documents. RPC comes in two flavors, RPC/Encoded and RPC/Literal. With RPC/Encoded the RPC request is encoded using special SOAP-encoded XML tags. The usual Document-Style is Document/Literal, and like RPC/Literal leaves the interpretation of the XML tags to the use of standard XML mechanisms, such as XML schemas. There is a difference between the RPC/Literal and Document/Literal style that is explained below.

2.4.1 RPC-Style

Back in 1998 SOAP was mainly seen as an RPC mechanism over http:. It had its own typing mechanism (SOAP-encoding) because XML schema hadn't been agreed. This style of using SOAP is called RPC/Encoded.

There is also a RPC/Literal style that doesn't use an encoding scheme to indicate the type of XML elements in the SOAP message. The client and server must agree on types "out of band."

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<soap:Body>
  <m:GetLastTradePrice xmlns:m="http://itso.ra1.ibm.com/SG24-6395/">
    <symbol type="string">IBM</symbol>
  </m:GetLastTradePrice>
</soap:Body>
</soap:Envelope>
```

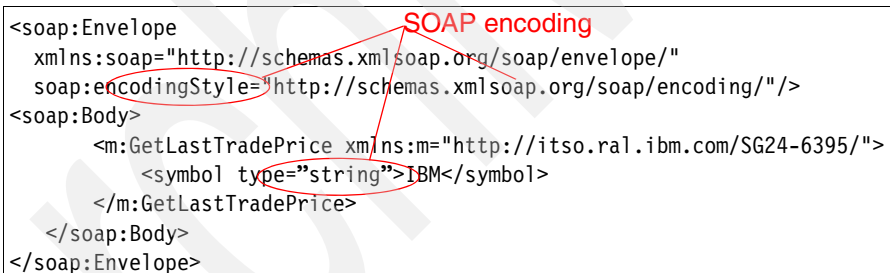


Figure 2-7 RPC/Encoded Style - SOAP 1.1

RPC/Encoded SOAP has given rise to a number of interoperability problems, and WS-I prefer the use XML schemas to provide type information. In fact Figure 2-7 is not a valid WS-I basic profile 1.1 SOAP message because it breaks rule 1006:

R1006 An ENVELOPE MUST NOT contain soap:encodingStyle attributes on any element that is a child of soap:Body.

Figure 2-8 is a valid RPC/Literal SOAP message.

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
<soap:Body>
  <m:GetLastTradePrice xmlns:m="http://itso.ral.ibm.com/SG24-6395">
    <symbol>IBM</symbol>
  </m:GetLastTradePrice>
</soap:Body>
</soap:Envelope>
```

Figure 2-8 RPC/Literal Style - SOAP 1.1

2.4.2 Document-Style

The attraction of the RPC-Style was that it:

- ▶ Provided a ready-made type system, something that was hard to do before XML schemas were standardized and before the adoption of WSDL and UDDI provided a means to exchange type information about SOAP messages.
- ▶ Was easy for application programmers to use before more sophisticated development tools were developed.
- ▶ Was necessary to have an RPC object access protocol to be considered as an serious alternative to DCOM and CORBA.
- ▶ RPC-encoded messages are self-describing. This was important before WSDL was invented.

The Document-style of SOAP was there from the beginning - in order to exchange more complex XML documents. Unlike RPC, the Document-style makes no assumptions about the interaction style and is the format to use for EDI type exchanges using loosely coupled transports such as messaging or Email. Unlike RPC/encoded messages, Document-style messages are not self describing. The sender and receiver need to share the WSDL definition. Document-Style has grown to be the preferred way to use SOAP for a number of reasons:

- ▶ Typing is possible using XML namespaces; why have two ways of encoding data?
- ▶ RPC using Document-Style is now straightforward, using tools to generate the client and server stubs from the WSDL file.
- ▶ De-coupling the message from the means of interaction: Document-Style makes no presumptions about how the message is to be delivered
- ▶ When using SOAP to implement Web services, the sweet spot is the coarse grained business service, which in all likelihood could have of the order of

hundreds or thousands of attributes, and the interface will be changing regularly.

RPC-Style is suitable for interactions between low-level objects with small and stable method signatures.

- ▶ Document-Style is more appropriate than RPC-Style for asynchronous messaging.

Interactions between coarse grained services, frequently deployed over different networks, can only achieve high availability by being designed for loosely coupled asynchronous behavior.

- ▶ Document-Style processing is also significantly more efficient than RPC-Style as payload increases.

See *Frank Cohen, Discover SOAP encoding's impact on Web service performance*, found at:

<http://www-128.ibm.com/developerworks/webservices/library/ws-soapenc>

On the wire, Document/Literal style SOAP need look no different to RPC/Literal. Example 11-1 on page 247 shows a Document/Literal SOAP message generated by WebSphere Studio Application Developer.

Document/Literal style is generally preferred over RPC/Literal because all the type information to interpret the SOAP message is defined in the WSDL for a Document literal message. In the case of RPC/Literal one needs to know the rules for constructing the RPC message as well as the WSDL to fully understand the SOAP message

2.4.3 Document/Wrapped

There is one other variety of SOAP message style, called Document/Wrapped that is effectively a Document/Literal with a single outmost complex type containing the rest of the XML document. It has its advocates, as it combines RPC-style in carrying the RPC operation name unambiguously in the outermost XML element, and Document-style in being a fully typed XML document. However, you will only infrequently see Document/Wrapped style as an option in tooling. WS-I refer to Document/Literal as the preferred style for interoperability.

The WS-I committee preferred to leave the mapping of the operation name open in the case of Document/Literal rather than specify a syntax convention, such as that of Document/Wrapped,

"In the document-literal case, since a wrapper with the operation name is not present, the message signatures must be correctly designed so that they meet this [operation name is used as a wrapper for the part accessors] requirement"¹

2.5 SOAP interaction styles

SOAP messages are fundamentally one-way transmissions. They are combined to form interaction patterns such as request-response. The modeling of interaction patterns is only evident in WSDL and in mapping to a SOAP transport. There is nothing in the SOAP message, such as a means of correlating replies with requests, that indicates the interaction style. A new specification, WS-Addressing, introduces headers to assist managing more sophisticated interaction styles.

WS-I limits its attention to only one-way and request-response interaction styles because of ambiguities in the WSDL for other interaction style:

R2303 A DESCRIPTION MUST NOT use Solicit-Response and Notification type operations in a wsdl:portType definition.

2.5.1 Request-response

Perhaps because SOAP is most frequently married to the Http: request-response model as its transport, and because of SOAPs origins in RPC-Style distributed computing, SOAP is most often associated with request-response interactions.

When using Http: as a transport SOAP request-response must be implemented by using an Http: Post.

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0'?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >
<env:Header>... </env:Header>
<env:Body>
  <m:chargeReservation xmlns:m="http://travelcompany.example.org/">
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
      <m:code>FT35ZBQ</m:code>
    </m:reservation>
  </env:Body>
</env:Envelope>
```

Figure 2-9 SOAP Request implemented as and Http: POST

and its corresponding response message:

¹ R2710, WS-I Basic Profile Version 1.0, found at, <http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html>

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0'?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >
  <env:Header>... </env:Header>
  <env:Body>... </env:Body>
</env:Envelope>
```

Figure 2-10 *Http: SOAP response message*

2.5.2 One-way

With the one-way interaction model, WS-I stipulates that there must be no SOAP reply. When using a request-response protocol, such `Http:`, then no SOAP message must be flowed back by in the response. The `Http:` protocol should still be observed, flowing an `Http:` response code back in the reply.

2.6 SOAP implementations over `Http:`

Implementations of SOAP request-response over `Http:` should conform to `Http:` as well as SOAP specifications. For example, `Http:` response codes must be returned from SOAP operations. More controversial is the question of how to identify the service to be invoked. Good `Http:` practice is for application writers to uniquely identify the Web service in the URI supplied to `POST`. Considerations of discovery, packaging and performance have led to using the URI as a starting point from which to identify the Web service method to invoke, rather than fully specifying it.

2.6.1 Microsoft .Net SOAP request over `Http:`

Microsoft uses an additional SOAP header, `SOAPACTION` to express “the intent of the message”. In this example, taken from Aaron Skinnard, “*How ASP.NET Web Services Work*”, found at:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/howwebmeth.asp>

The dispatching of the correct method is as follows. The `.asmx` handler introspects the `math` class for a Web-method with a `SOAPACTION` value of `Add`. If there is no `SOAPACTION` attribute defined for the method it uses the namespace of the class. With no namespace specified for the class either, it

assumes `http://tempuri.org` which matches the SOAPACTION header in the SOAP message

```
POST /math/math.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPACTION: "http://tempuri.org/Add"

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Add xmlns="http://tempuri.org/">
      <x>33</x>
      <y>66</y>
    </Add>
  </soap:Body>
</soap:Envelope>
```

Figure 2-11 Dispatching the Web service method “Add” in Microsoft .Net

2.6.2 IBM WebSphere Application Server SOAP request over Http:

For WebSphere Application Server (see Figure 2-12), the Http: POST URI points at the servlet in the Web project that has been deployed to handle this Web service request.

```
POST /ItsoClaimRouterWeb/services/LGIClaimRegistration HTTP/1.0
Host: localhost:9081
Content-Type: text/xml; charset=utf-8
Content-Length: 419
SOAPACTION: ""

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:q0="http://ejb.claim.examples.itso"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <q0:findCustomer>
      <q0:customerID>abc</q0:customerID>
      <q0:policyID>123</q0:policyID>
    </q0:findCustomer>
  </SOAP-ENV:Body>
```

Figure 2-12 Dispatching the Web service method “findCustomer”

The deployment descriptors (web.xml and webservices.xml) point the request to the service endpoint - in this example the LGIClaimRegistration EJB. The SOAP Body outermost entry name is mapped to the corresponding method in the EJB.

2.7 Summary: Salient interoperability features of SOAP

In summary, what are the features of the SOAP specification that have made it a good protocol for application interoperability?

- ▶ It is based on XML. XML is widely accepted and XML parsers exist on virtually all software platforms
- ▶ It uses XML schemas to provide type information rather than a programming language specific type system
- ▶ It restricts itself to the presentation layer (layer 7) of the ISO communication stack which makes it relatively easy to implement:
 - The specification is relatively small
 - The basic SOAP protocol doesn't get bound up in network and platform specifics.
- ▶ It supports different interaction styles (principally request-response, one-way) and both RPC and messaging (or document) types of interface.

It combines in one architecture the interaction styles predominantly used by distributed components within an application (RPC) on the one hand, and by applications integrated together within a solution (Messaging), on the other. SOAP has provided a common protocol for two communities of programmers who are finding themselves increasingly working together driven by the business potential of the Internet.

- ▶ SOAP has been almost universally implemented on Http: making SOAP/Http: available on most platforms. This sounds like saying SOAP is successful because it is successful! And so it is: the "network effect" has favored SOAP/http. Initially, the simple marriage of SOAP to the Internet, via Http, was crucial in SOAP/http adoption by vendors - because it worked.
- ▶ Does it work well enough for adoption by enterprises? The evidence (Figure 7-1 on page 115) points to the technology breaking through, at least on the intranet. There is a widespread belief that it will be good enough for Internet and intranet applications. But we can't yet say that SOAP is used widely in production. Deployment on the intranet leads over deployment on the Internet by a wide margin reflecting probably two things
 - Trialing in a more controlled environment which puts a company's value less at risk
 - Need for more secure and robust qualities of service on the Internet

- ▶ Which leads to the last important characteristic: SOAP is extensible. It has changed since it was first conceived around 1997, and it needs to continue to change to meet the needs of solution integration.

Archived

Archived

WSDL primer

The Web Service Description Language (WSDL) was created shortly after SOAP to describe Web services. WSDL describes:

- ▶ The information carried in a Web service request
- ▶ How the Web service is realized by a particular protocol, such as SOAP over Http:
- ▶ The location of the service

The structure of this information is illustrated in Figure 3-1 on page 28. The discussion that follows is not a complete description of the specification, but is sufficient to understand how WSDL and SOAP are related, and to be able to interpret WSDL documents produced by WebSphere Studio Application Developer or Microsoft Visual Studio .Net 2003.

A full definition of WSDL 1.1 is available from W3C at:

<http://www.w3.org/TR/2001/NOTE-wsd1-20010315>

3.1 Structure of WSDL definitions

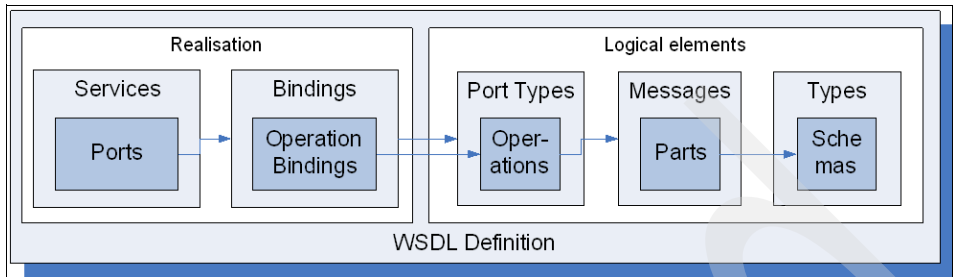


Figure 3-1 Structure of a WSDL document

1. WSDL definition

A WSDL definition is an XML document, or a number of documents that can be imported to compose a complete WSDL definition. The WSDL definition has a target namespace to identify all its elements.

2. Logical elements

The logical elements describe the information in the Web service. They are:

a. Data types used in messages.

Types are described using schema definitions. SOAP encoding is an alternative, in which case a type section is unnecessary. The WSDL standard allows the use of alternative type grammars to XML schema.

The schema definition can be imported rather than provided inline in the WSDL document.

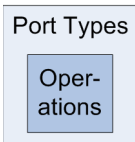
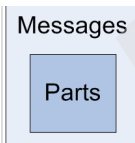
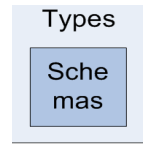
b. Messages exchanged in requests and replies.

Each message has a name by which to refer to it, and one or more message parts. The parts are just a way of assembling the message from different elements or types. You can have anything between one part and a single complex type, or many parts and simpler elements.

A part has a name by which to refer to it, and either the element name or type it refers to in a type definition.

c. portTypes (called Interfaces in WSDL 1.2) are collections of operations. portTypes and operations each have names.

Each operation can have three message types, an input, output and fault message type which can also be named. For each message type, the operation refers to one of the message definitions given previously.



3. Realization

The realization of a logical Web service comes in two parts:

a. Binding to a specific protocol

Extra protocol-specific information is added to the Web service definition, such as whether the WSDL is to be mapped to a SOAP message using the Document or RPC-Styles.

Common to all protocol bindings are:

- i. Naming the binding
- ii. Associating it with a portType
- iii. Specifying the operations supported by the binding (by referring to the appropriate portType)
- iv. Providing optional binding specific Input, Output and Fault attributes for each operation

b. Addressing a specific Service instance (a Service endpoint)

Service endpoints are called Ports.

What follows is a discussion of the specific binding extensions for the SOAP binding. The SOAP binding is not limited to Http: but that is the only transport we will consider in this section because it is the only transport referred to in the WS-I 1.1 profile.

a. SOAP binding

A SOAP binding specifies the style of the SOAP message, document or RPC, as well as the type of transport defined for the binding as a whole.

Example 3-1 The SOAP RPC binding

```
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
```

Each operation in the portType that is defined in the binding is qualified by providing a SOAPACTION attribute to assist in routing the operation, and an override for the style of the operation (RPC or Document).

For the input and output parts of the operation, the binding provides more attributes specifically for the SOAP Body, for the SOAP Header and for Headerfaults.

The additional attributes of the Input and Output parts have identical attributes. The fault part is simpler than the Input and Output parts so we will just concentrate on those, particularly on the Body. Let us see how the binding specifies the appearance of the Body in a SOAP message.

How the Body appears will depend on whether the message style is Document or RPC. If the style is RPC then each part of the message is

Bindings

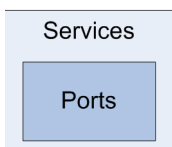
Operation
Bindings

treated as a separate parameter, or the return value. The message is “wrapped” inside an additional element that is created with the name of the operation and takes its namespace from the binding’s Body attribute. If the style is Document then the message parts are created directly as the SOAP Body element.

The Body attribute also defines whether the SOAP message elements are literal or encoded, and, if encoded, what the encoding style is. There is also the capability to filter out some of the message parts. Not all of the parts defined for a message need actually appear in the generated SOAP message.

The header and headerfault attributes enable headers and header faults to be defined.

The fault part of the operation defines how a fault message should be encoded.



b. Services

The service section of the WSDL definition collects together all the port definitions. As we have seen in the specific SOAP/Http: binding we are considering, the transport is defined to be "http://schemas.xmlsoap.org/soap/http".

The port definition couples a specific binding identified by name with a SOAP-specific address in the form `<soap:address location="uri"/>`. There is an example of a SOAP Service definition in Example 3-2 on page 32.

3.2 Examples of WSDL definitions

We will use the example that is described later in this redbook to illustrate the relationship between a WSDL definition and generated SOAP messages. The WSDL was generated by WebSphere Studio Application Developer. For a comparison, we will look at the same application using first the Document/Literal style of WSDL and then compare some aspects with the RPC/Literal style.

3.2.1 Document/Literal Style

Figure 3-2 on page 31 shows the schematic view of a WSDL definition from WebSphere Studio Application Developer laid out to match the model in Figure 3-1 on page 28.

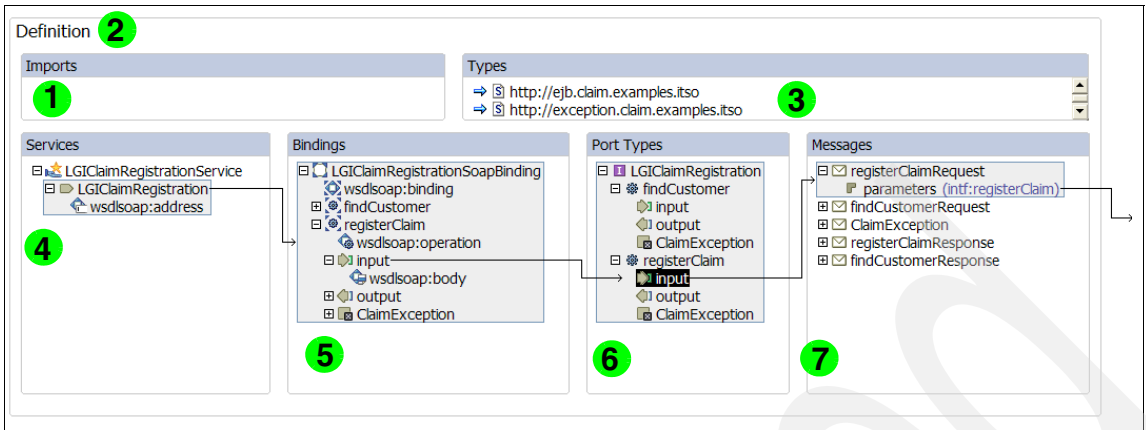


Figure 3-2 Document/Literal style WSDL

We will walk through the WSDL, examining the RegisterClaim request that is part of the example we will be building. At this time, you can follow the walk-through by referring to Figure 3-2.

1. At the top of Figure 3-2, you can see there are no import files.
2. By clicking the **Definitions** heading, you will see the namespaces in a pop-up. Figure 3-3 shows the namespaces that were defined and used in the RegisterClaim example.
3. The Types elements show the namespaces that are defined in the Types section of the WSDL file.

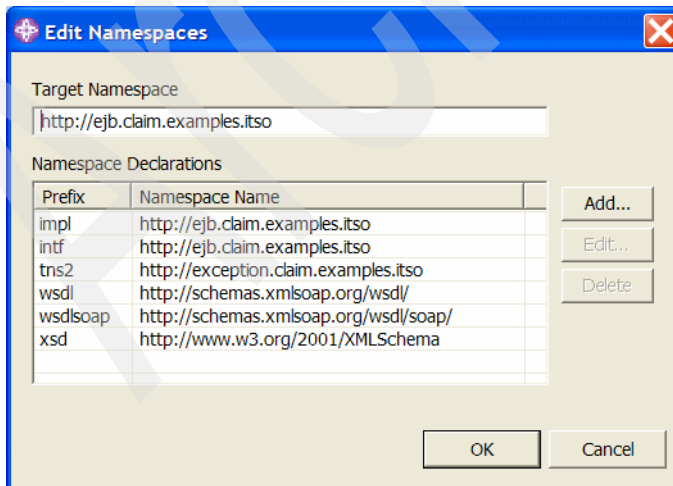


Figure 3-3 Namespaces

4. The Services section points to the binding and names the SOAP port. In Example 3-2, the local IDs, such as `wSDL:`, match the namespace declarations in Figure 3-3 on page 31.

Example 3-2 Service definition

```
<wSDL:service name="LGIClaimRegistrationService">
  <wSDL:port binding="intf:LGIClaimRegistrationSoapBinding"
    name="LGIClaimRegistration">
    <wSDLsoap:address
      location="http://localhost:9080/ItsoclaimRouterWeb/services/LGI
      ClaimRegistration"/>
    </wSDL:port>
  </wSDL:service>
```

5. In Example 3-3, we have picked out the RegisterClaim operation input part to look at the Binding section. The Binding is named, and points at the `intf:LGIClaimRegistration portType`.

The Document-Style and SOAP transport are specified. The operation name `registerClaim` will match the operation name in the `intf:LGIClaimRegistration portType`. The input name also matches the input name in the `portType`. A SOAPACTION string is provided for the operation.

Example 3-3 SOAP binding

```
<wSDL:binding name="LGIClaimRegistrationSoapBinding"
  type="intf:LGIClaimRegistration">
  <wSDLsoap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  ...
  <wSDL:operation name="registerClaim">
    <wSDLsoap:operation SOAPACTION=""/>
    <wSDL:input name="registerClaimRequest">
      <wSDLsoap:body use="literal"/>
    </wSDL:input>
  ...
  </wSDL:operation>
</wSDL:binding>
```

6. The `portType` is straightforward. The operations `findCustomer` and `registerClaim` included in the `portType` are displayed in Figure 3-2 on page 31.
7. The `registerClaimRequest` message defines the part *parameters* which points to the `intf:registerClaim` element illustrated in Figure 3-4 on page 33.

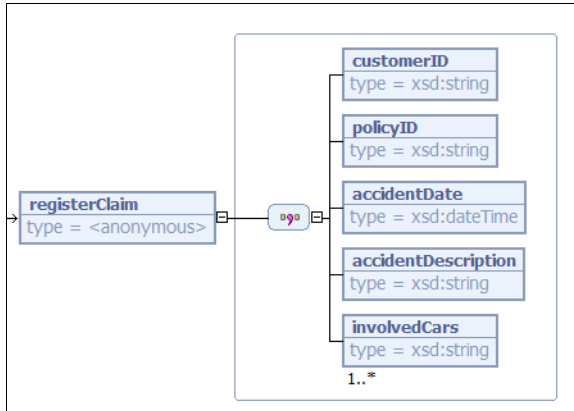


Figure 3-4 Types

8. Figure 3-5 shows the SOAP message which is generated by this WSDL.

```

<?xml version="1.0" encoding="UTF-8" ?><SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:q0="http://ejb.claim.examples.itso"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body>
  <q0:registerClaim>
    <q0:customerID>123</q0:customerID>
    <q0:policyID>abc</q0:policyID>
    <q0:accidentDate>2004-10-20T08:03:34.985Z</q0:accidentDate>
    <q0:accidentDescription>Hit stationary object</q0:accidentDescription>
    <q0:involvedCars>Alvis</q0:involvedCars>
  </q0:registerClaim>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Figure 3-5 SOAP message generated from Document/Literal WSDL

3.2.2 RPC/Literal Style

The SOAP message generated by RPC/Literal is shown in Figure 3-6 on page 34.

```

<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" No target namespace
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body>
<ns0:registerClaim xmlns:ns0="http://ejb.claim.examples.itso" Outer element is the binding operation name
namespace from binding-operation
<customerID>123</customerID>
<policyID>weta</policyID>
<accidentDate>2004-10-20T08:19:12.924Z</accidentDate>
<accidentDescription>Hit Stationary object</accidentDescription>
<involvedCars>
<item>Alvis</item>
</involvedCars>
</ns0:registerClaim>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Figure 3-6 SOAP message generated from RPC/Literal WSDL

The outline of the WSDL in Figure 3-7 is only a little different from Figure 3-2 on page 31.

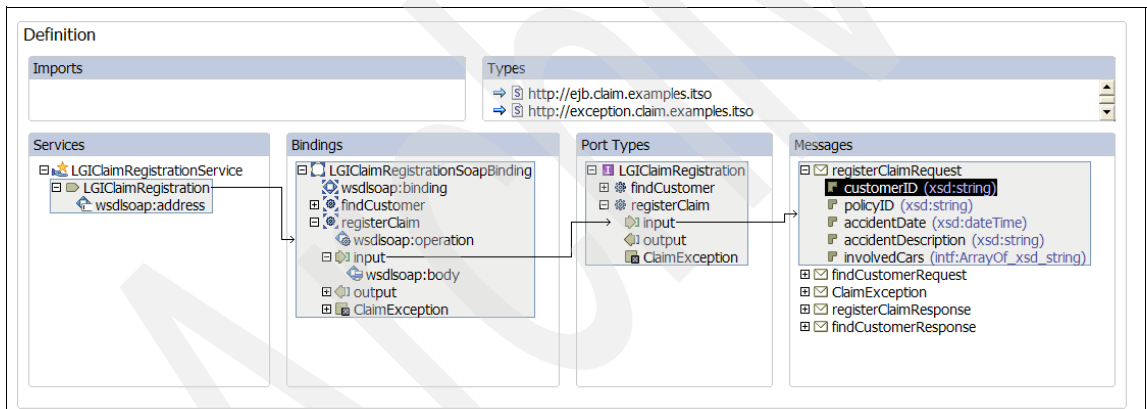


Figure 3-7 RPC-Style WSDL

Rather than stepping through the WSDL again in detail, let us just look at the three salient differences:

1. The most obvious difference is in the message structure (see Example 3-4 on page 35). This is a result of the registerClaimRequest message being constructed from multiple parts containing simple types in the RPC case. In the previous example, a single complex type was referenced to construct the message.

Example 3-4 Message definitions

From the RPC-Style WSDL

```
<wsdl:message name="registerClaimRequest">
  <wsdl:part name="customerID" type="xsd:string"/>
  <wsdl:part name="policyID" type="xsd:string"/>
  <wsdl:part name="accidentDate" type="xsd:dateTime"/>
  <wsdl:part name="accidentDescription" type="xsd:string"/>
  <wsdl:part name="involvedCars" type="intf:ArrayOf_xsd_string"/>
</wsdl:message>
```

From the Document-Style WSDL

```
<wsdl:message name="registerClaimRequest">
  <wsdl:part element="intf:registerClaim" name="parameters"/>
</wsdl:message>
```

2. In the portType, the RPC-Style includes the parameter order for each operation, that is:

```
<wsdl:operation name="registerClaim" parameterOrder="customerID policyID
accidentDate accidentDescription involvedCars">
```

3. In the binding, the binding style is now RPC rather than Document.

Example 3-5 Binding style differences

From the RPC-Style WSDL

```
<wsdl:binding name="LGIClaimRegistrationSoapBinding" ...
<wsdlsoap:binding style="rpc" .../>
```

From the Document-Style WSDL

```
<wsdl:binding name="LGIClaimRegistrationSoapBinding" ...
<wsdlsoap:binding style="document" .../>
```

3.3 Future considerations

WS-I basic profile 1.1 references WSDL 1.1. But WSDL will change: WSDL 2.0 is in the process of being drafted. It completed its last call working draft on October 4th, 2004. The next standardization step for the specification is as a proposed candidate standard to W3C.

WSDL is no longer seen as a complete description of a SOAP message. Other WS-* meta-data specifications have been published which complement WSDL, particularly WS-Policy and WS-meta-dataExchange. These are discussed further in Chapter 7, “Web services roadmap” on page 113.

3.4 Summary: salient interoperability features of WSDL

In summary, what are the features of WSDL that have made it a good language for specifying interoperable application interfaces? Some of these features mirror those of SOAP:

1. It is based on XML. XML is widely accepted and XML parsers exist on virtually all software platforms.
2. It uses XML schemas to provide type information rather than a programming language specific type system (for the purists, WSDL is not tied to use XML schemas as a typing system, but the flexibility given to WSDL to use other typing systems has not been a factor in improving interoperability).

The switch from using SOAP-encoding to using standard XML schemas (XSDs) for type definition was ratified in the WS-I Basic profile¹. It improves interoperability by removing some ambiguities in SOAP encoding; also, by reusing XML schema that are also used elsewhere for type definition, it leverages existing tooling support for XSD, and developers have been able to reuse existing XSDs.

3. WSDL is an interface definition language, something it has in common with its predecessors, such as IDL.

The innovation that distinguishes it from IDL is that in addition to defining logical interfaces such as:

- Definitions (Types, Imports)
- Messages (Parts)
- portTypes (Operations)

it also defines the physical interfaces:

- Bindings (Protocol, Operation bindings)
- Services (Ports).

This has resulted in standardization of the physical format (“wire format”) of SOAP messages, including both the way to map WSDL to a SOAP message, and the way to map SOAP to the chosen transport.

So not only are abstract interfaces interchangeable between tools, but the tools map WSDL to SOAP sufficiently closely so that the SOAP messages, if not physically identical, have the same meaning. Also, the use of the underlying Http: transport is sufficiently similar that (increasingly) SOAP clients and servers really do interoperate without prior testing and debugging.

Interoperability works fairly well for SOAP/http: which has been thoroughly studied and refined as part of the WS-I profile. It gets rather more theoretical

¹ See “*The argument against SOAP encoding*”, Tim Ewald, MSDN Oct 2002, found at, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap/html/argsoape.asp>

than practical for other protocols that have been added but not yet effectively standardized. This leads to the next point: has the extensibility of WSDL made it more or less a good basis for defining interoperable interfaces?

4. WSDL is extensible.

In the last point, we saw that the extensibility of WSDL results in vendors defining extensions that do not always interoperate. But the extensions are architecturally compatible. We mean by this that WSDL has been used to define both standard and proprietary service interfaces in a common way. Vendors are able to leverage much of the same tooling and runtime to implement the extensions in addition to the standardized protocols. From a user's perspective, whether working with standard or extended capabilities, the services framework doesn't change.

From an interoperability perspective, having this extensibility capability is a benefit over the alternative of freezing a specification and, at some point in the future, making a quantum leap to an new standard. As we have seen, SOAP and WSDL were not born perfectly formed. Extensibility is important in modifying the standards to accommodate the needs of interoperability as the scope of the standard is changed.

In general terms, the arguments in favor of extensibility of a specification are:

- The extensions will be easier to standardize in the future than if new features are specified in a wholly new way.
- Having both standards and extensions coexisting in a compatible specification is an effective way of tackling the problem that business requirements tend to change faster than standards.

5. The scope of WSDL is stable.

Specifications need to be small enough to be practically stable for implementation and revision within a reasonable timescale.

There are numerous additional Web service meta-data standards being proposed to address security, addressing, introspection, and policy, to name but a few. Many of these additional specifications result in additional XML structures in a SOAP message. The meta-data to generate the additional structure could just as well have been specified in WSDL whereas in fact the new SOAP structures derive from a different meta-data specification, such as WS-Policy.

By choosing to create new specifications for these meta-data extensions rather than adding to the WSDL specification, the focus of the WSDL specification has been to solve today's interoperability problems, rather than to introduce new capabilities.

Archived

Web services primer

In this chapter, we look at Web service concepts and then compare Web services with software components, Service-Oriented Architecture and the Enterprise Service Bus.

4.1 Web services concepts

Web services are self-contained applications that are published, located, described and invoked over the Internet or intranet.

Web services perform business functions, ranging from a simple query to complex business process interactions. A Web service can be built from the ground up as a new application or an existing legacy system can be re-engineered to make it Web service enabled.

There are three main participants in the generic Web service pattern.

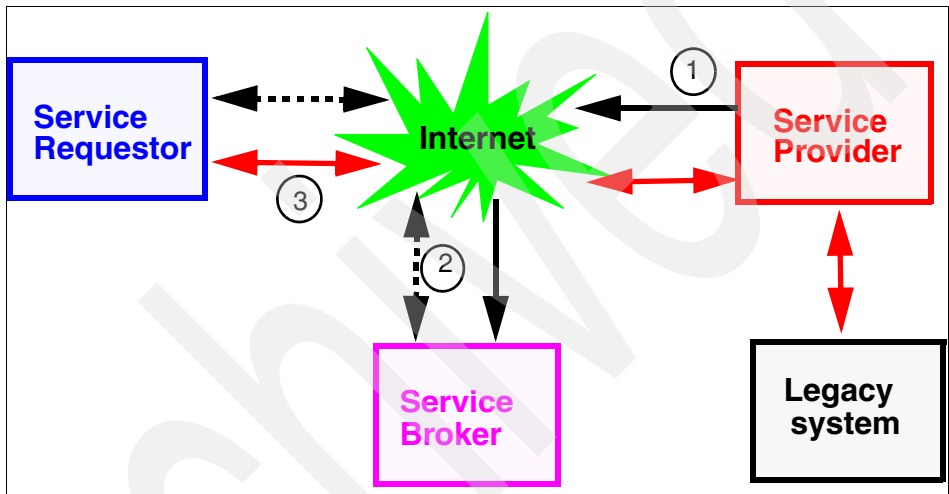


Figure 4-1 Web services roles and operations

1. The service provider creates and hosts a Web service and possibly publishes its interface and access information to the service registry, such as to a UDDI registry.

The service provider decides which services to expose to the Internet, to extranets or to intranets, manages the access to the service, and provides information about the service to the service broker.

2. The service broker (also known as the service registry) is responsible for making the Web service interface and implementation access information available to any potential service requestor. Typically, a broker, such as a UDDI registry, contains information to identify the service and points back to the service provider for obtaining details about the service interface.

The implementers of a broker have to decide about the scope of the broker. Public brokers are available all over the Internet, while private brokers are only accessible to a limited audience, for example, users of a company-wide

intranet. Furthermore, the width and breadth of the offered information has to be decided. Some brokers will specialize in breadth of listings. Others will offer high levels of trust in the listed services. Some will cover a broad landscape of services, and others will focus within a given industry. Brokers also exist that simply catalog other brokers. Depending on the business model, a broker may attempt to maximize look-up requests, number of listings, or accuracy of the listings.

3. The service requestor (the Web service client) locates entries in the broker registry using various find operations and then connects to the service provider in order to invoke one of its Web services.

One important issue for users of services is the degree to which services are statically chosen by designers compared to those dynamically chosen at runtime. Even if most initial usage is largely static, any dynamic choice opens up the issues of how to choose the best service provider and how to assess quality of service. Another issue is how the user of services can assess the risk of exposure to failures of service suppliers.

Web services are sometimes created as brand new software components, but more frequently are composed of existing (“legacy”) systems. The Web service provider may expose the Web service interface on a gateway server and connect the gateway to the legacy systems using an enterprise service bus.

4.1.1 What is a Web service?

Let’s turn to the W3C Web services Architecture Working Group. In <http://www.w3c.org/TR/ws-arch/#what is>, they define a Web service succinctly as:

“... a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

What does this definition mean? Let’s examine the definition in more detail.

- ▶ “[A Web service is] a software system ...”

Any software system can be a Web service. Web services are technology-, language- and platform-independent. The software system could be a WebSphere Enterprise JavaBean, a Microsoft .Net class, a CICS® transaction, a WebSphere MQSeries Workflow; virtually any existing software system can be turned into a Web service.

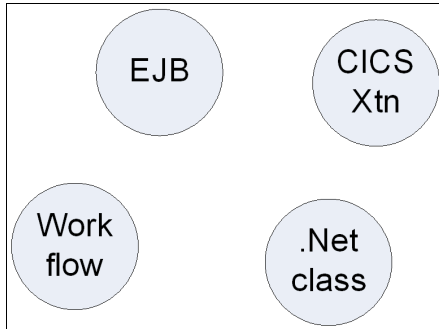


Figure 4-2 Software systems

- ▶ *“... designed to support interoperable machine to machine interaction over a network.” ...*

A Web service is designed to be used over a network and is invoked using a well-defined network protocol. A Web service needs to be a coarse-grained software component which implements a self-contained service.

- ▶ *“It has an interface described in a machine-processable format (specifically WSDL)”*

A Web service is intended to be used in a distributed application that could well be provided by a third party. They must be able to rely solely on information that is published in the UDDI and as WSDL to build their client for the service.

- ▶ *“Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”*

The WSDL binding defines the Internet protocols used to invoke the Web service, usually a SOAP binding specifying `Http` as the transport protocol. Both parts of the interface are illustrated in Figure 4-3 on page 43.

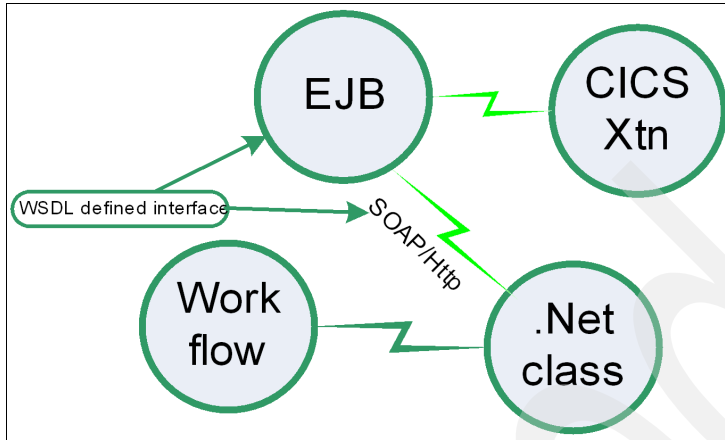


Figure 4-3 Software systems as Web services

4.1.2 Web services technologies

The following are the core technologies used for Web services.

- ▶ XML (eXtensible Markup Language) is the markup language that underlies most of the specifications used for Web services. XML is a generic language that can be used to describe any kind of content in a structured way, separated from its presentation to a specific device.
- ▶ SOAP (Simple Object Access Protocol) is an XML-based network, transport-, programming language and platform-neutral protocol that allows a client to call a remote service.
- ▶ WSDL (Web services description language) is an XML-based interface and implementation description language. The service provider uses a WSDL document in order to specify the operations a Web service provides, as well as the parameters and data types of these operations. A WSDL document also contains the service access information.
- ▶ UDDI (universal description, discovery, and integration) is both a client-side API and a SOAP-based server implementation that can be used to store and retrieve information about service providers and Web services.

4.1.3 Web service properties

Web services have properties which fit them well for use in an on demand environment:

▶ **Web services are self-contained.**

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

▶ **Web services are self-describing.**

WSDL contains all the interface information needed to build both the client and the server side of a Web service. A service provider can publish all the information that developers need to understand how to build and deploy Web service clients using the UDDI repository and meta-data definitions, such as WSDL.

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

This technology uses established lightweight Internet standards such as HTTP. It leverages the existing infrastructure. It is firewall-friendly.

▶ **Web services are language-independent and interoperable.**

Client and server can be implemented in different environments. Existing code does not have to be changed in order to be Web service enabled. In this publication, however, we assume that Java is the implementation language for both the client and the server side of the Web service.

▶ **Web services are inherently open and standard-based.**

XML and HTTP are the major technical foundation for Web services. A large part of the Web service technology has been built using open-source projects. Therefore, vendor independence is a realistic goal this time.

▶ **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. Web services can be chained together to perform higher-level business functions. This shortens development time and enables best-of-breed implementations.

▶ **Web services are built on proven technology.**

The concepts behind Web services have been proven in earlier distributed computing models. Web services architects have also learned from the problems of complexity of earlier distributed computing models, and have been built on top of proven and ubiquitous standards such as XML, Http: and the Internet.

► **Web services are built up from a simple foundation.**

The basic Web service foundation has no bells and whistles; security, reliability, state management and other extensions are additional to the foundation. The layering helps to achieve interoperability in practical stages.

► **Web services enjoy widespread support.**

All the major software vendors endorse Web services. Many users of Web services have become members of Web service bodies and the Web services Interoperability council (WS-I). Gartner, Inc. reports that 40-50% of the Web services audience uses SOAP and 80% of Gartner, Inc.'s Application Integration and Web service conference attendees.

► **Web services are loosely coupled.**

Traditionally, application design has depended on tight interconnections at both ends. This is true both of development and production. Web services require a simpler level of coordination that allows a more flexible re-configuration for an integration of the services in question.

► **Web services provide programmatic access.**

The approach provides no graphical user interface; it operates at the code level. Service consumers have to know the interfaces to Web services but do not have to know the implementation details of services.

► **Web services provide the ability to wrap existing applications.**

Already existing stand-alone applications can easily be integrated into the Service-Oriented Architecture by implementing a Web service as an interface.

4.2 Web services and component architectures

The WSDL binding is one of the key ways a Web service differs from a software component. A software component *can* be a Web service, the difference being that whereas a Web service is invoked using the binding information in its WSDL definition, the means of invoking a software component is implicit. A software component may exploit local linkage mechanisms, or it could be distributed over protocols such as IIOP or COM+ that tie it to a specific software system. The means of invoking a Web service can be by various - it is defined in its WSDL definition.

For a Web service, both the interaction protocol and the service interface are specified in the WSDL definition. Taken together these definitions allow Web services to be composed into a solution without regard to how they were built or in what environment they are running.

A deployed Web service is available to clients who are authorized to use it and support the same protocol as specified in the binding.

By comparison, systems viewed as software components have to have a shared knowledge of a mechanism for components to interact with one another. The knowledge of the mechanism enabling requester and provider to communicate is outside the scope of the software component definition.

Interaction between components is managed either by their component containers, or, where no container is provided, by custom designed bridges or connectors between the components, as shown in Figure 4-4.

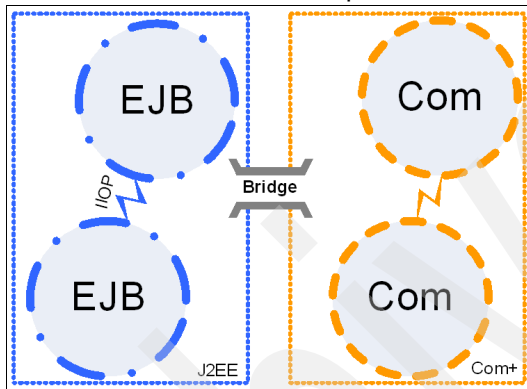


Figure 4-4 Software systems as components

4.2.1 Choosing between Web services and software components

The choice between composing an application from Web services, and building an application from components comes down to weighing factors such as:

- ▶ Heterogeneity of the operating environments.
- ▶ Desired openness of the interaction protocols.
- ▶ Coupling of the components.
- ▶ What qualities of service required, such as security, performance and robustness.
- ▶ Ease of use: what are the strengths or weaknesses of the tooling for composing a solution made up of software components compared with the tooling for Web service?

As long as an application is only going to be composed of components running in a common container, such as Java 2 Enterprise Edition or Microsoft .Net, there will be less value from using Web services. However, when interactions need to

take place between different software environments, it becomes a lot harder to use software components and Web services should be considered.

Web services are also likely to be preferable if the software component is to be used in an unmanaged environment such as the Internet, or where the costs of management are high, such as between two organizations connected by an extranet. In these environments any coupling of technologies between partners creates additional costs that grow as coupling tends to increase with time. Web services helps to reduce this cost by defining interfaces between services that can be implemented in different technologies and reduces the coupling between partners. The key concept here is one of "Governance." *Governance* is the term given to an organization's sphere of control. Web service architectures decouple a solution from the technologies used to implement it. A Web service architecture gives the partners in a solution to freedom to choose their own technologies and software suppliers.

In an unmanaged environment, Web services have a very clear advantage over distributed solutions built from proprietary software components. But even in a managed environment, Web services may offer a practical and lower cost alternative to building a bridge, adapter or broker to link different software components.

As a case study to illustrate the possible benefits offered by Web services in a managed environment, let's examine the steps involved in building a bridge between a Java 2 Enterprise Edition system and an Enterprise Information System (EIS). As an example of a software component architecture, we will make use of the Java Connector Architecture (JCA) and describe the process of connecting WebSphere to the EIS and compare it to using a Web services approach.

Building a JCA connector between software environments

The JCA standard specifies the way to build a connector between a Java 2 Enterprise Edition system and an typical EIS such as CICS or SAP. A connector implemented to the JCA specification will connect any compliant Java 2 Enterprise Edition server to a specific target EIS such as CICS.

Rather than build their own JCA connectors enterprises are likely to purchase toolkits based on a pre-packaged JCA connector, such as the IBM WebSphere Business Integration Adapter for mySAP.com, or the CICS gateway, rather than implement their own JCA components.

Figure 4-5 shows the steps involved in integrating two EIS with a Java 2 Enterprise Edition server. The adapter vendor:

1. Builds two JCA connectors, one for each EIS.

2. Selects and builds EIS specific means of connecting to the target systems.
3. Makes available different configuration choices about how to connect to an EIS. The enterprise learns how to install and manage the connections
4. Creates tooling to integrate the connectors with the development environments used to build solutions for the Java 2 Enterprise Edition server.

There is no standard way to do this. The interface the developer sees depends upon:

- a. How the EIS interfaces are described in the EIS,
- b. How the EIS presents the interfaces to the development tools
- c. How the provider of the Integrated Development Environments (IDEs) for the Java 2 Enterprise Edition system plugs in new tooling to the IDE.

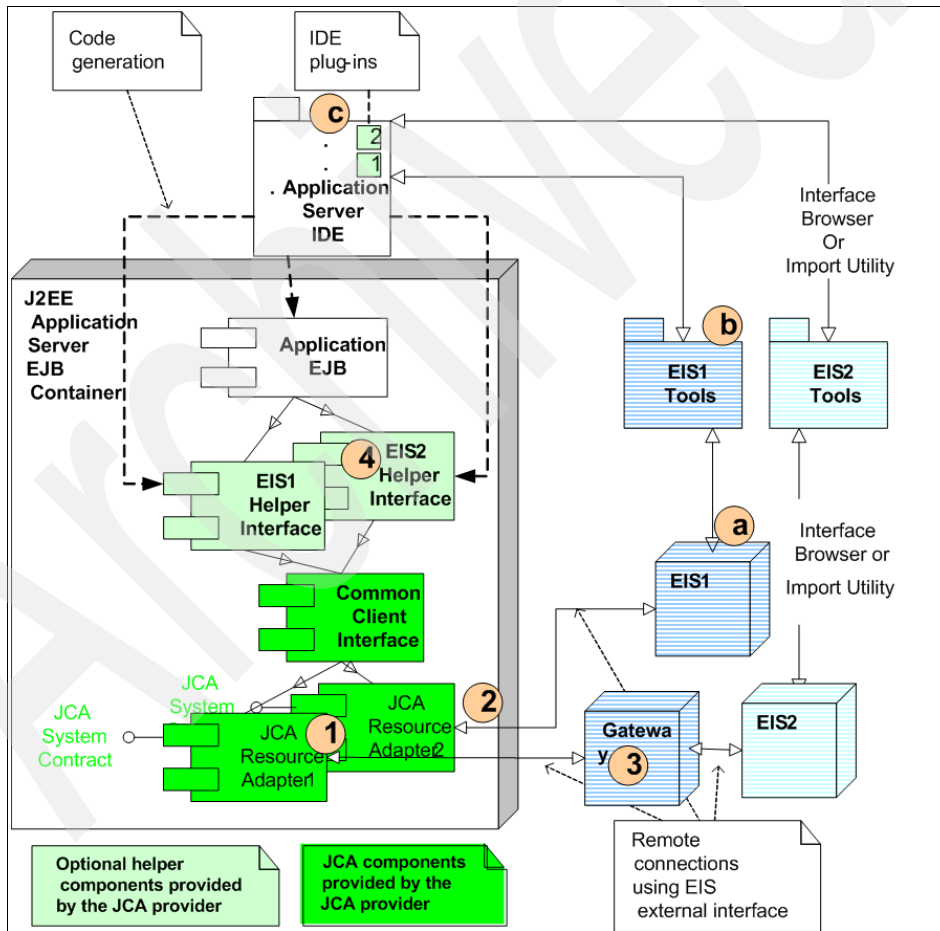


Figure 4-5 Integrating two EIS using JCA

With the multiplicity of environments to support, particularly in step 4, software vendors have typically opted to build a separate adapter server. Rather than having to interface different applications servers to different EISs, it becomes a question of porting just the interface to the adapter server. This is the architecture adopted for the first version of the IBM WebSphere Business Integration Adapters.

Connecting software environments using Web services

Now let's see how Web services promise to simplify the problem of connecting two complex software environments.

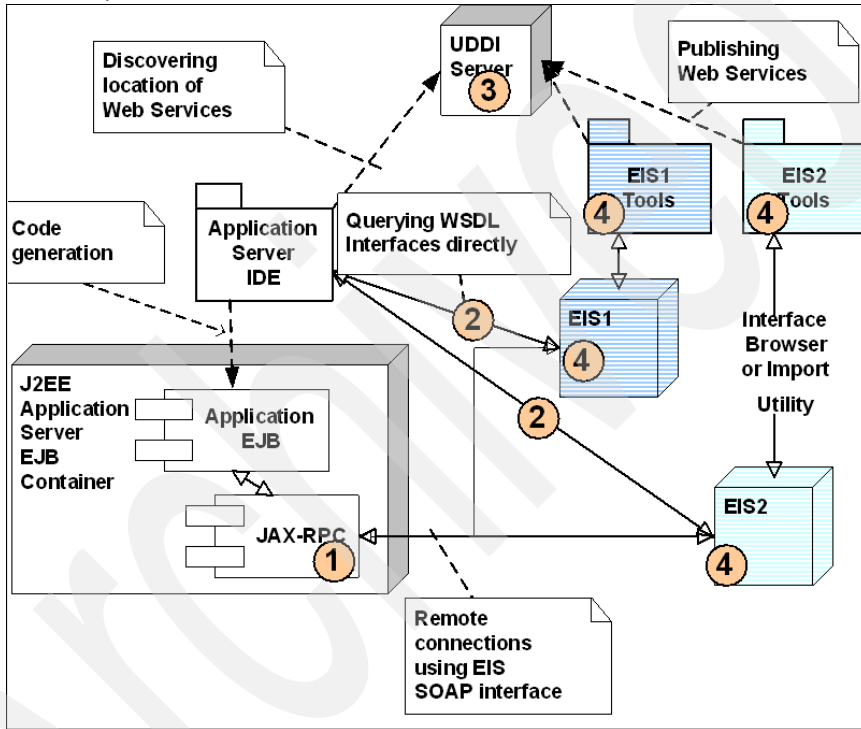


Figure 4-6 Integrating two EIS using Web services

1. SOAP provides the interoperability of the runtime. There are no additional gateway or adapter servers to deploy that need custom configuration by specialists.
2. WSDL provides the standard way to describe the EIS binding and operational interfaces of a Web service. Skills and tooling used to build solutions composed of services on one EIS should be transferable to building solutions using services of another EIS.

3. UDDI and WSIL (see “WS-Inspection” on page 136) are less well established than SOAP and WSDL but they do provide a solution to deploying service descriptions in a standardized way. The descriptions can be accessed using tools from any vendor to deploy and manage connectors.

In contrast, the location and type of meta-data necessary to build parts of a component based solution are scattered in different locations depending on the vendor. The runtime configuration information will be similarly vendor-specific.

4. Most importantly, vendors such as SAP, PeopleSoft, Intentiona and others are investing in Web services. These are currently offered as an alternative to their Enterprise Application Integration (EIA) solutions.

4.3 Service-Oriented Architecture

Most enterprises are facing the challenges of cutting costs and maximizing the utilization of existing technology, while continuously serving customers better, being more competitive, and being more responsive to the business's strategic priorities.

Heterogeneity and change are two underlying themes behind all of these pressures. Organizations contain a range of different systems, applications, and architectures of different ages and technologies. Integrating products from multiple vendors and across different platforms is almost always a nightmare. But we also cannot afford to take a single-vendor approach to IT, because application suites and the supporting infrastructure from a single vendor can be so inflexible.

Change is the second theme underlying the questions that today's IT executives face. Globalization and e-business are accelerating the pace of change. Globalization leads to fierce competition, which leads to shortening product cycles, as companies look to gain advantage over their competition. Customer needs and requirements change more quickly, driven by competitive offerings and wealth of product information available over the Internet. In response the cycle of competitive improvements in products and services further accelerates. Improvements in technology continue to accelerate, feeding the increased pace of changing customer requirements. Business must rapidly adapt to survive, let alone to succeed in today's dynamic competitive environment, and the IT infrastructure must enable businesses' ability to adapt.

As a result, business organizations are evolving from the vertical, isolated business divisions of the 1980's and earlier, to the horizontal business-process-focused structures of the 1980s and 1990s, towards the new ecosystem business paradigm. Business services now need to be

componentized and distributed. There is a focus on the extended supply chain, enabling customer and partner access to business services.

How do we make our IT environment more flexible and responsive to the ever changing business requirements? How can we make those heterogeneous systems and applications communicate as seamlessly as possible? How can we achieve the business objective without bankrupting the enterprise? The IT answers/enablers have been evolving in parallel with this evolution of business, as shown in Figure 4-7. Currently, many IT executives and professionals believe that now we are getting very close to providing a satisfactory answer with Service-Oriented Architecture.



Figure 4-7 The evolution of architecture

In order to alleviate the problems of heterogeneity, interoperability and ever changing requirements, such an architecture should provide a platform for building application services with the following characteristics:

- ▶ Loosely coupled
- ▶ Location transparent
- ▶ Protocol independent

Based on such a Service-Oriented Architecture, a service consumer does not even have to worry about a particular service it is communicating with because the underlying infrastructure, or service “bus,” will make an appropriate choice on behalf of the consumer. The infrastructure hides as many technicalities as possible from a requestor. Particularly technical specificities from different implementation technologies such as J2EE or Microsoft .Net should not affect the SOA users. We should also be able to reconsider and substitute a “better” service implementation if one is available, and with better quality of service characteristics.

4.3.1 Components of a Service-Oriented Architecture

Service-Oriented Architecture presents an approach for building distributed systems that deliver application functionality as services to either end-user applications or other services. It is comprised of elements that can be

categorized into functional and quality of service. Figure 4-8 shows the architectural stack and the elements that might be observed in a Service-Oriented Architecture.

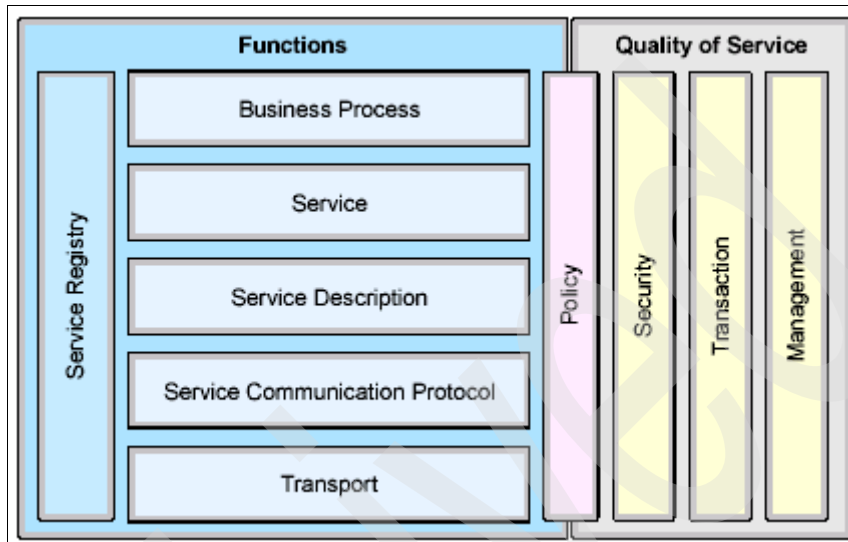


Figure 4-8 Elements of Service-Oriented Architecture

The architectural stack is divided into two halves, with the left half addressing the functional aspects of the architecture and the right half addressing the quality of service aspects. These elements are described in detail as follows:

Note: Service-Oriented Architecture stacks can be a contentious issue, with several different stacks being put forward by various proponents. Our stack is not being positioned as the services stack. It is just presented as a useful framework for structuring the SOA discussion in the rest of the redbook.

- ▶ Functional aspects include:
 - Transport is the mechanism used to move service requests from the service consumer to the service provider, and service responses from the service provider to the service consumer.
 - Service Communication Protocol is an agreed mechanism that the service provider and the service consumer use to communicate what is being requested and what is being returned.
 - Service Description is an agreed schema for describing what the service is, how it should be invoked, and what data is required to invoke the service successfully.

- Service describes an actual service that is made available for use.
- Business Process is a collection of services, invoked in a particular sequence with a particular set of rules, to meet a business requirement. Note that a business process could be considered a service in its own right, which leads to the idea that business processes may be composed of services of different granularities.
- The Service Registry is a repository of service and data descriptions, which may be used by service providers to publish their services, and service consumers to discover or find available services. The service registry may provide other functions to services that require a centralized repository.
- ▶ Quality of service aspects include:
 - Policy is a set of conditions or rules under which a service provider makes the service available to consumers. There are aspects of policy, which are functional, and aspects, which relate to quality of service; therefore we have the policy function in both functional and quality of service areas.
 - Security is the set of rules that might be applied to the identification, authorization, and access control of service consumers invoking services.
 - Transaction is the set of attributes that might be applied to a group of services to deliver a consistent result. For example, if a group of three services are to be used to complete a business function, all must complete or none must complete.
 - Management is the set of attributes that might be applied to managing the services provided or consumed.

SOA collaborations

Figure 4-9 on page 54 shows the collaborations between the core elements of the SOA.

- ▶ All elements use XML including XML namespaces and XML schemas.
- ▶ The service requestor and provider communicate with each other.
- ▶ WSDL is one alternative to make service interfaces and implementations available in the UDDI registry.
- ▶ WSDL includes the workflow description (business process execution language for Web services, BPEL4WS)
- ▶ WSDL is the base for SOAP server deployment and SOAP client generation.

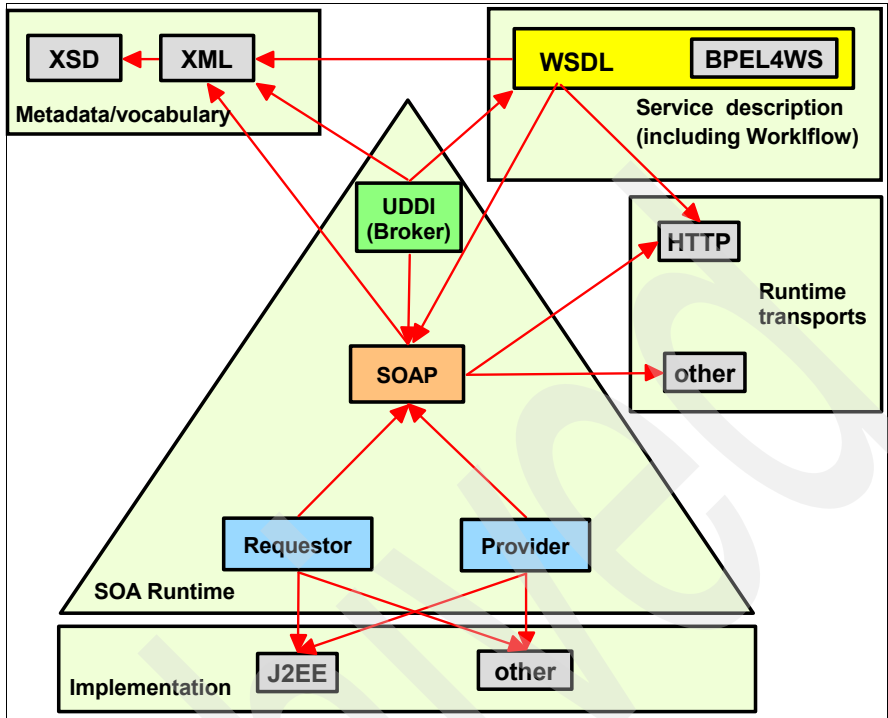


Figure 4-9 Main building blocks in an SOA approach based on Web services

4.3.2 Services and Web services

What is the relationship of SOA to Web services?

SOA is an abstraction of Web services. SOA has adopted Web service concepts such as WSDL, but SOA does not tie down the realization of a service to using the SOAP binding or using Internet protocols. Figure 4-10 on page 55 shows the idea of Service-Oriented Architecture bringing together different software technologies under a unifying approach.

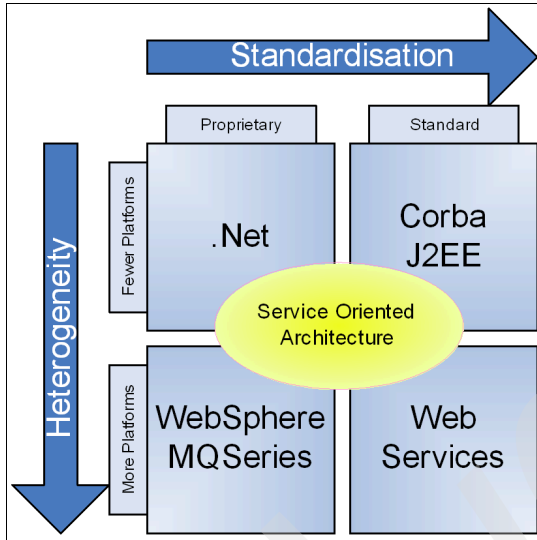


Figure 4-10 Implementations of services in a Service-Oriented Architecture

The most commonly agreed aspects of the definition of a service are that services

- ▶ Are defined by explicit, vendor neutral, implementation-independent interfaces.
- ▶ Are loosely coupled and invoked through communication protocols that stress location transparency and interoperability.
- ▶ Encapsulate reusable business function.

Broadening the concept of a Web service to a service

Figure 4-11 on page 56 shows different services being used in a SOA.

1. Within a managed environment supporting different interaction protocols such as IIOP and WebSphere MQSeries, SOA can provide more qualities of service than available today than Web service using SOAP bound to Http.
2. Sometimes the deployment of a SOAP binding is not feasible, and a different implementation of SOA offers the prospect of realizing some of the benefits of Web services without incurring the costs of modifying existing legacy applications to work in a SOAP environment.

An SOA also offers different interaction protocols, such as publish-subscribe or event based paradigms that are not yet fully specified as Web services.

3. By declaring the behavior of interfaces in WSDL, and separating the service binding from the service operational interface, applications can be composed

from services using the same interface definitions, regardless of the binding of a service to a particular interaction protocol such as SOAP/http.

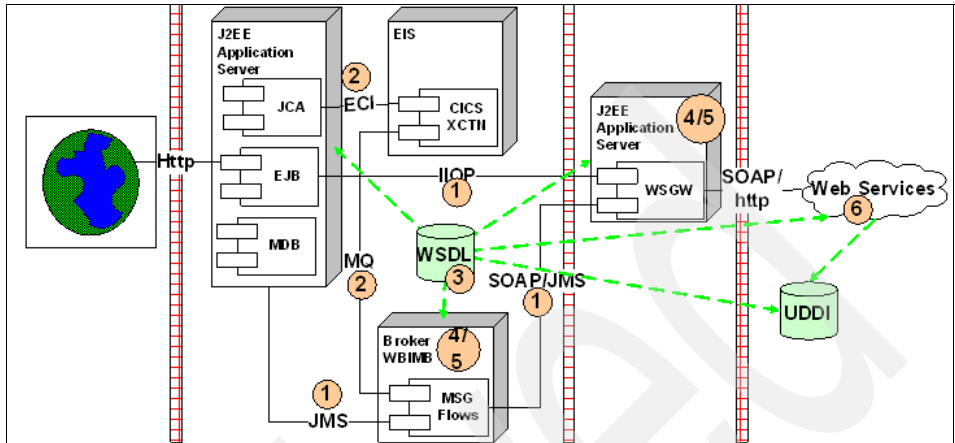


Figure 4-11 Mixing different SOA implementations

4. Web services are tied to a distributed point-to-point architecture, with no central point of control. SOAs can be mapped to different topologies, including hub-and-spoke and bus topologies that can manage mapping of service requesters to service providers.
5. SOAs address a wider scope of requirements than Web services, providing a framework for solving other important interoperability issues such as transforming or adapting existing interfaces and providing management and autonomic capabilities.
6. Web services are used in an SOA, particularly to connect to services outside the local governance zone - to different departments in the enterprise, to suppliers, partners or customers.

Why focus particularly on Web services?

What special position do Web services play in an SOA? Figure 4-12 on page 57 is a simplification of Figure 4-11 showing Web services being used across the Internet, and also sharing the same infrastructure as the intra-enterprise service bus.

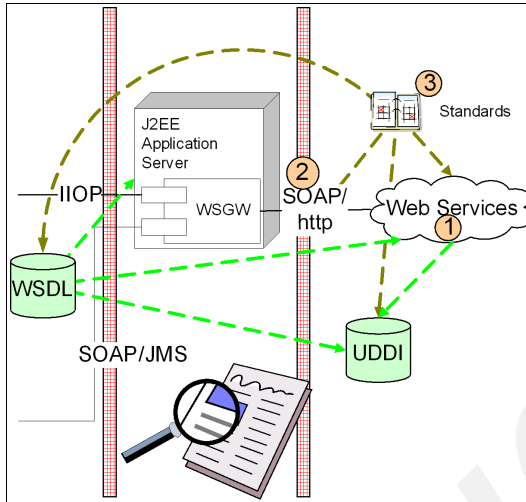


Figure 4-12 Use of Web services in SOA

The three points the diagram makes are:

1. Suppliers have universally adopted the Web services paradigm for SOA. Other implementations of an SOA, while perhaps more capable, are proprietary, so Web services will often be the service technology of choice across the Internet.

“SOA is finally entering the enterprise mainstream. Several factors are enabling this change ... The unanimous vendor acceptance of Web services standards, especially the basic Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL) specifications. Unlike CORBA and DCE, Web services standards have no naysayers among vendors.”

Note: The above quote is taken from Roy Schulte’s *Predicts 2003: SOA is Changing Software*. The PDF is currently available at:

<http://www.gartner.com/resources/111900/111987/111987.pdf>

2. SOAP/http has become a universal way of doing SOA on the Internet due to the following factors.
 - Universal availability of the http protocol on the Internet and in different vendors’ enterprise software platforms
 - Ability of SOAP to cross enterprise firewalls
 - Expectation that SOAP/http based Web services will interoperate “out of the box” between different vendors’ environments

- Business value of deploying services for use over the Internet
 - Simplicity and low cost in deploying Web services to the Internet
3. Web service specifications are being standardized and proving to be interoperable. The formation of the WS-I council to coordinate the implementation of Web service standards, and its success in defining the Basic Profile, augurs well for increasing the adoption of Web service implementations to deliver a practical, interoperating SOA that satisfies the needs of a large part of the SOA market.

From a standards perspective, the advantages of limiting the standardization of SOA to Web services comes down to the practical consideration of making implementations from different suppliers work together - the old adage of “K.I.S.S” (Keep It Simple Stupid). Simpler specifications have a better chance of being implemented consistently, and being interoperable, especially if the specifications are based on protocols that already have widely accepted currency.

On the other hand Web service specifications and implementations are advancing in scope and complexity, and promise to meet many of the needs of a broader SOA, as illustrated in the Web services specifications stack in Chapter 7, “Web services roadmap” on page 113. It will take some time for these specifications to be standardized, and then to be implemented in an interoperable way. So for some time yet proprietary SOA alternatives will coexist with Web services, and enterprises need to be able to use both Web services and other implementations of SOAs where they are most appropriate.

The infrastructure concept that is has come to the fore to manage different types of SOA implementation is called an Enterprise Service Bus (ESB).

4.4 Web services and the Enterprise Service Bus

The concept of an ESB was first described by Roy Schulte in the same paper as cited above.

“ESB is a new architecture that exploits Web services, messaging middleware, intelligent routing, and transformation”.

Roy Schulte’s description of an ESB identifies some of the main capabilities that ESBs provide, and by inference some of the requirements that lie behind ESBs such as,

- ▶ Unifying Web services with existing Message Oriented Middleware (MOM) infrastructures

- ▶ Overcoming the problems of managing the deployment and routing of Web services and integrating with other types of services
- ▶ Improving the reuse and manageability of services by providing a transformation service to mediate between different service requestor and service provider interfaces

Note: The redbook *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346, published in July 2004, is a comprehensive discussion of the ESB concept.

The Enterprise Service Bus is emerging as a service-oriented infrastructure component that makes large-scale implementation of the SOA principles manageable in a heterogeneous world. The relationships between Web services, SOA and ESB are shown in Figure 4-13.

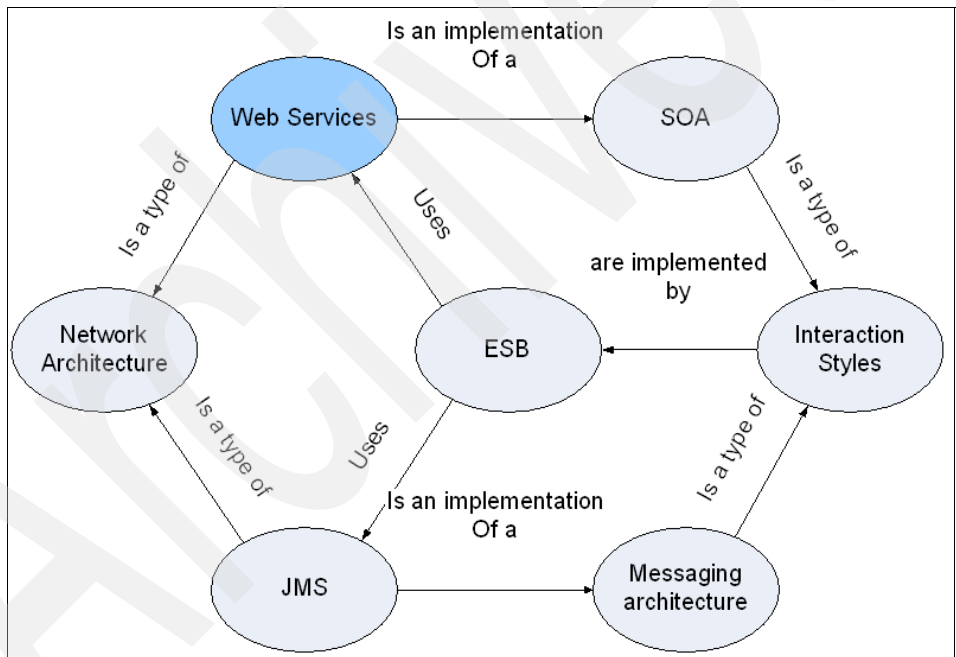


Figure 4-13 Relationships between Web services, SOA and ESBs

An ESB should provide a rich implementation of SOA as well as integrating existing network architectures without diluting their capabilities. The goals, or best of bred characteristics, of an ESB are listed below.

- ▶ Transparency
- ▶ Interoperability

- ▶ Service discovery and addressing
- ▶ Co-existence
- ▶ Unified administration and management
- ▶ Security
- ▶ Robustness
- ▶ Scalability
- ▶ Problem determination

Note: This list is compiled from a number of sources, in particular *“Identifying best-of-breed characteristics in Enterprise Services Buses (ESBs)”*, Steve Craggs, June 2003, found at:

http://www.integrationconsortium.org/docs/member%20docs/BestofBreed_ESBs.pdf

Let’s examine how each of these ESB characteristics relate to Web services implemented to be compliant with WS-I basic profile 1.1, i.e using SOAP/http 1.1.

4.4.1 Transparency

At the interface between an ESB and a Web service provider or Web service requestor is an ESB-Web service touch point. The touch point isolates the requestor from the provider. The requestor is connected to an outward facing Web service endpoint on the ESB and the provider is connected to the ESB. The same is true in reverse - the ESB isolates a requestor on the ESB from an external provider. Changes to the location the Web service does not affect its requesters - the changes are managed at the ESB-Web service touch point.

For the designers of ESB implementations the challenge is to isolate the impact of changes to the touchpoints and to provide administrative means to configure touch points without requiring administration, redeployment or reprogramming of the endpoints.

Web services comparison

If one contrasts the ESB connection model with Web services the difference is in the level of isolation achieved. In the Web services model the service provider’s interface is declared in its WSDL definition so that the requestor can design the right kind of connection to the provider. But once a simple connection has been built the requestor code will have been coupled to the provider. Changes to the provider will require changes to all its requesters. To avoid this coupling more sophisticated dynamic clients can be built. But fundamentally it is more manageable to resolve the de-coupling issue at the provider rather than at the requestor end of the connection. (For a discussion of different relationship styles in a SOA see section 3.2.1 “Coupling and de-coupling of aspects of service invocations”, in *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346.)

4.4.2 Interoperability

An ESB provides a framework and tools to solve interoperability problems. The interoperability problem may be that two Web service implementations do not support the same levels of specifications; or it may be a Web service requester needs to use a service provided as an EJB, a SAP IDOC, a CICS transaction or a Microsoft .Net class rather than as a Web service. An ESB provides bridge between different protocols, as well as runtime support and tooling for mediating different data formats, different qualities of service, and different application protocols.

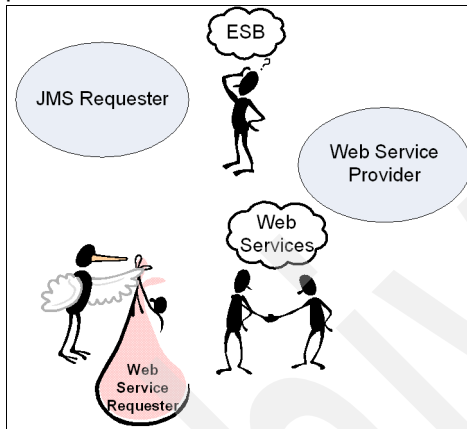


Figure 4-14 Interoperability: Use ESB or Web services?

Web services comparison

A typical Web service scenario involves creating a new service requester to work with an existing service provider (see Figure 4-14). Web services specifications are not focused on making *existing* requesters and providers interoperate, but on enabling the creation of providers that will work with many different requesters as long as they are all built to the same provider interface. The Web service goal is that “anonymous” services, deployed over the Internet, will work with clients however they are built, as long as they implement the interfaces described in the Web service’s WSDL definition.

ESBs, on the other hand, also have the objective of enabling existing services which have different interfaces interoperate, and to manage changes to service interfaces.

4.4.3 Unified service discovery and addressing

The discovery and addressing of services by a client of the ESB should follow the same rules regardless of where the service is hosted and how it is addressed by the ESB on behalf of the client. The ESB manages which interfaces are made

visible to service requesters and publishes these interfaces as WSDL to isolate clients from the how and the where of hosting a service.

The ESB also manages the flow of a message to and from the Web service endpoint addressed by the service requester and the actual service provider. The request message could go through a number of nodes and processes. The response message may retrace its path through the ESB, or take a different path through the ESB back to the requester.

Web services comparison

As well as using Web services to publish the touch point between a requester and an ESB, an ESB can use Web service specifications to architect how to identify intermediate processing nodes (in SOAP 1.2) and also how to pass addressing information in a SOAP message so as to be able to return to the same processing node in a return path, or return to the same endpoint transparently to the Web service client. (See 8.3, “Interoperability standards: addressing” on page 155).

4.4.4 Coexistence

An ESB supports applications using different styles of integration.

- ▶ SOA - not limited to Web services, but also supporting non-SOAP bindings
- ▶ Message-driven architectures in which applications send messages through the ESB to receiving applications. The ESB may physically incorporate existing MOM channels of communication without change
- ▶ Event-driven architectures in which applications generate and consume messages independent of one another

An ESB enables Web services to interact with applications using these other styles of communication where it makes sense to do so.

Web services comparison

There are Web service specifications for event and notification styles of interaction emerging (see “WS-Notification” and “WS-Eventing” on page 127).

4.4.5 Single point of control

An ESB aims to bring different SOAs under a single administrative umbrella so that all the moving parts involved in integrating a solution comprising Web services, some MOM components, an adapter to an EIS, some Microsoft .Net classes and some enterprise services using EJBs are configured within one administrative framework. The ESB administrator is responsible for creating the

ESB topology, defining connection points and deciding services are deployed on each node and which ones are published outside the ESB.

Web services comparison

Web services are composed of autonomous clients and servers with point-to-point connections. The administrative model is essentially distributed. Services are published to centralized UDDI registries. The public registries have very little control. But private UDDI servers can be built to manage the deployment of Web services within a governance domain.

4.4.6 Security

ESBs aim to provide a managed and trusted environment for the execution of services. In collaboration with third party security servers, the ESB Administrator needs to provide:

- ▶ Bus boundaries where security and access is managed by Reverse Proxy Security Servers (RPSS), and tools to assist in (for example) filtering out hackers, and limiting the impact of denial of service attacks
- ▶ Web Trust Association (WTA) through the management of trust, security context and access onto and from the bus. This shares out the development and runtime overhead of security management between services - reducing the security overhead of calling multiple services in the same security domain.
- ▶ Provision of security services, such as logging, to assist in detection and analysis of security breaches

Web services comparison

Web service security specifications provide the means for securing exchanges between Web service requester and provider (see “Security” on page 158).

An ESB also needs to map Web services security to the access, privacy and authentication mechanisms of other services that are sharing the bus.

4.4.7 Robustness

There are a two distinct aspects of robustness to consider in an SOA:

- ▶ Transactionality of the interaction model
- ▶ Service Availability

Transactionality

An ESB uses a variety of different transactional implementations such as the Java Transaction API (JTA) in J2EE to do one, two and distributed two-phase

commits, JMS or WebSphere MQSeries to provide the distributed three units of work model, and execution of Business Process Execution Language (BPEL) to provide business agreement models.

Web services comparison

Web services provide transactional interface specifications that are described in WS-Coordination and “WS-Transactions” on page 179 and “Reliable messaging” on page 182. An ESB can implement the Web service transaction specifications using all the transactional implementations it supports - and in so doing provide transactional interoperability between environments supported by the ESB as well as extending transactionality to Web services outside the ESB.

Service availability

Part of an ESB implementation is to integrate Web services with its implementation of clusters of servers, alternative network paths and provision for hot failover.

Web services comparison

Web services standards describe interfaces and rely upon the capabilities of the underlying platforms and networks for availability of services and redundant network paths to the services. There are no Web services specifications that are relevant to service availability.

4.4.8 Scalability

Scalability has an administrative as well as a performance dimension. Administratively an ESB provides a centralized, or federated, means managing services. From the performance perspective an ESB makes use of underlying platforms to provide load-balancing transparently to Web service clients.

Web services comparison

From one perspective, Web services have highly scalable distributed administration, in the sense that no more effort is required to deploy the thousandth Web service than the first one. However, from another perspective unless there is a centralized deployment, management of Web services is problematic. Something like an ESB, or a cluster of centrally administered application servers is essential to managing Web services.

From a throughput perspective Web services can be scaled using the Tcp/ip based approaches used to scale up Web sites, or using scalability capabilities built into the application server. There are not Web service specifications that are relevant to scalability, except perhaps WS-Addressing which can be used with load balancing to provide server affinity - see 8.3, “Interoperability standards: addressing” on page 155.

4.4.9 Problem determination

The ESB, as an infrastructure for SOA, should provide problem determination tools, such as monitoring the availability and performance of services, tracing Web service requests, and logging and correlating their execution on different nodes.

Web services comparison

The Web services specification that help with problem determination is the proposal for WS-Manageability (see “Management” on page 166).

4.4.10 Conclusions: Web services, the ESB and service buses

The comparison between Web services and the ESB has illustrated these are different but related animals. Web services specifications have the principle objective of enabling the development of services without regard for the software platform, and with the minimum of contact between the service provider and service requester beyond what is published along with the service in meta-data. The ESB has, as we have seen, broader objectives, in particular to decouple the addressing of clients from servers and to provide control over the service bus itself.

This is the same conclusion as reached by the authors of *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346, who have examined the capabilities of service buses in depth in their section 4.3 “A capability model for the Enterprise Service Bus”. They also make the point that basic SOAP/Http and WSDL are not an ESB.

The <WS-I Basic Profile 1.1 compliant> service bus

Nonetheless the ability to seamlessly plug together service requesters and service providers that conform to the WS-I basic profile 1.1 is a useful defining characteristic of a software bus. When Web services that comply with a set of specifications, such as the WS-I basic profile 1.1, are simply used together in a solution without any additional form of mediation between the connections, the services can be thought of as forming a type of service bus. We will use this type of bus in the design for the scenarios in this book. But it is worth repeating the limitations of this simple kind of service bus

1. The configuration of the bus can't easily be changed. Depending on how clients and servers are built it is usually not possible to move or change a service without impacting its clients
2. The bus has no defined topology; it is built from point-to-point connections mapped onto addresses in the underlying network. There is no bus level network model.

In the Patterns for e-business (P4eb) method a new set of SOA patterns are defined using the concept of an ESB. In Chapter 6, “Interoperability patterns” on page 93 we have taken the liberty of introducing a simpler type of service bus: the “<WS-I basic profile 1.1 compliant>” service bus.

The value of the concept is that it simplifies the use of the Patterns for e-business (P4eb) to design solutions like the business scenarios used in this book. Rather than show each service requestor and service provider linked in a pairwise connection, all the service requesters and providers are connected to a common bus.

From an architectural perspective, the value in showing Web service requesters and providers in a pairwise relationship is to identify that some specific adaptation or mediation needs to be associated with each particular connection. In the case of a WS-I Basic Profile 1.1 compliant service bus the only piece of information that is specific to each connection is the address of the provider, and that we have chosen to characterize as part of the definition of this simple bus.

4.5 Summary

This concludes our review of the basic technology that comprise Web services, and the discussion of what Web services are and how they relate to Service-Oriented Architecture and the Enterprise Service Bus.

In the next chapter of the book, we will introduce the scenario we will be using to demonstrate using Web services to integrate the Microsoft .Net and WebSphere software environments.



Part 2

Web services interoperability

In this part of the redbook, we start by describing the business scenario and the problem that is going to be solved using Web services. Next, we use e-business patterns to analyze the problem from an IT perspective and propose a solution architecture. Then, continuing to use the e-business patterns method, we map the solution to a Web services implementation.

The remaining chapters in this part of the book survey the current and future Web service specifications, the work of the WS-I (Web services interoperability) organization, and Web service implementations in WebSphere and Microsoft .Net.

Archived

Business scenarios

In this chapter, we introduce two scenarios. These scenarios are representative of actual customers' business requirements in terms of interoperability between WebSphere and Microsoft .Net platforms and the adoption of a Service-Oriented Architecture (SOA) based on Web services technology.

Each business scenario is fully examined and the high-level design approach to the target solution is achieved through the following stages:

- ▶ Business goals identification and requirements definition
- ▶ Current IT infrastructures assessment and technical constraints identification
- ▶ Solution context details
- ▶ Proposed IT infrastructure

5.1 Business scenarios overview

Business scenarios describe how customers use IT solutions to accomplish their business goals. They are based on a thorough knowledge of customers' business goals and are refined to help the consultant, solution and application architect during requirements validation phase. The identification of use cases facilitates communication among customers, analysts, developers and testers. Scenarios clarify an evolving agreement between the sponsors of a solution and the development teams.

As a reference point for scenario selection, we use those provided by the IBM System House Business Scenarios team. This team works with customers and IBM customer-facing teams (marketing, industry solutions, solution builders) to identify solutions that many customers need and prioritizes the solutions in terms of their importance to the development and marketing teams in IBM. The Business Scenarios team then takes on the role of the customer in designing solutions for the scenarios using a mixture of products from IBM and other vendors. The objective is to create a process to assist IBM development teams in improving the design of their products working together as an integrated software platform. By designing solutions for the business scenarios, the development teams identify integration gaps that need to be addressed to make the experience of implementing these solutions easier and to make the solution more effective in helping customers to achieve their business needs.

Each System House Business Scenario includes a description of the business context, the business requirements, the interactions between the users of a system and an understanding of business events, objects and transactions within that environment. The scenarios are validated by IBM customers in the business sector to ensure that they represent areas of active investment, that they identify the business problems they are tackling, and that the solution architectures are representative of what IBM's customers are building.

According to the main purpose of this book, we selected and customized two scenarios requiring integration over different IT infrastructures:

- ▶ **Mergers and Acquisitions:** this scenario represents a merger between two insurance companies. One is a typical property and casualty insurance company providing insurance through agents using CICS and WebSphere MQSeries products; the other is an example of a dot.com insurance company working entirely through the Web using a Microsoft .Net platform. We look at the integration of the claims process across the merged companies.
- ▶ **External Claims Assessor Management:** this scenario extends the first scenario to automate a common outsourcing operation to external claims assessors.

5.2 Mergers and Acquisitions

This scenario is related to the merger of two companies that are representative of many companies in the insurance industry¹. A short company profile is provided in order to introduce the reader to the business goals driving the required solution.

Lord General Insurance (LGI), a property and casualty insurance company, is a large enterprise with more than five million policy holders, looking to boost its auto-insurance business and requiring a quick entry to the e-business direct insurance market. LGI has a large IT infrastructure based on S/390® and CICS.

LGI has acquired DirectCarInsure.com (DCI), a modern dot.com auto insurance company that sells insurance through the Internet and has fewer than one million policy holders. It has an e-business focused infrastructure based on Microsoft .Net.

5.2.1 Business goals

The major business goal for a merger or acquisition in financial services companies is the profitability and core business value improvement as well as a market share increase; this target can be achieved by providing value-added customer services, which means:

- ▶ Adding new channels to market for the merged company's products
- ▶ Broadening the products that can be offered down existing channels
- ▶ Reducing services costs
- ▶ Providing customers with a better service experience by exploiting new channels and more responsive internal processes

Different segments within the financial services sector have different goals with respect to integrating merged and acquired companies within their business that affect the IT solutions they adopt.

In the banking field, it is often the case that acquired companies are fully absorbed into the merged company. This means that elimination of duplicate IT capabilities is a common cost reduction goal. Typically, this implies selecting common IT suppliers and amalgamating IT departments and probably reductions in staffing.

In the insurance segment, if the opportunity arises to sell the acquisition to return value to the shareholders, or to finance another acquisition, it is often taken. The impact on IT policies during a merger is that there is less emphasis on merging

¹ These particular companies were invented for this scenario.

IT infrastructures (indeed, frequently the reverse to make divestment easier) and more emphasis on producing speedy returns and not disrupting existing IT organizations.

Dealing with this specific scenario, our target is to achieve the following LGI goals:

- ▶ Improve access to the market, extending products range and making them available to all customers
- ▶ Provide a direct customer channel
- ▶ Satisfy service level agreements as defined in customer requirements, for example, performance and security guidelines
- ▶ Improve the company's profitability by reducing overall administration costs, with an immediate focus on claims administration for existing products
- ▶ Improve the ability to monitor and manage business processes across both LGI and DirectCarlInsure.com
- ▶ Gain a complete view of the total business process and related information
- ▶ Achieve all of the previous goals as fast as possible in incremental steps

5.2.2 Solution context

The solution context related to the merger scenario involves all LGI and DirectCarlInsure.com business processes and systems which are, in part, out of the scope of our more limited aims for this redbook. In this section, we describe all the areas impacted by the new solution and detail which is the one we want to take as an example for the achievement of an interoperable final working solution.

Figure 5-1 on page 73 considers all systems currently involved in the LGI and DirectCarlInsure.com business process; the main areas which will be impacted by the merging solution are the policy administration and claims handling.

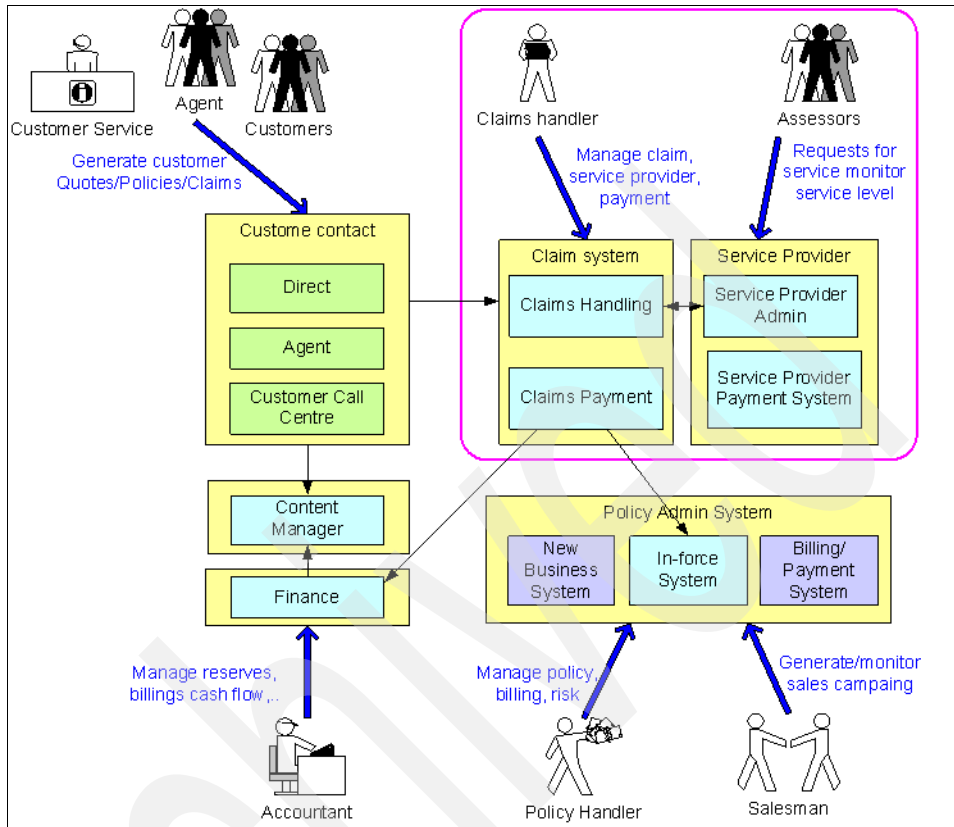


Figure 5-1 Mergers and acquisitions scenario business context

Regarding the policy administration area, the convergence of the two companies can be achieved by means of a single joint Web application which must be able to access both back-end systems and return the best insurance quote.

The claims system, instead, refers to the combination of business processes and IT systems tasks executed to handle a claim raised by a client. As part of the merger of LGI and DCI, a consolidation of the existing individual claims systems is essential. During the evaluation of the two claims systems, the following two requirements were highlighted to merge the systems: improve claims administration, and reduce costs.

- ▶ Re-engineer the business by combining the two claims departments and creating a single claims process
- ▶ Improve the claims process management

Both the policy and claim system implementation are impacted by interoperability issues; in the policy area, the key requirement is maintaining responsiveness and availability when dealing with high quote volumes. The solution requires broadcasting quotes and aggregating responses within a target response time so as not to lose Web-based clients. The claims area is focused on managing a complex workflow automation in a heterogeneous systems environment. The goals are to reduce costs and improve customer responsiveness by improving business processes.

We decided to examine the claims system for a number of reasons:

1. Availability and performance of the policy system is more business-critical than the claims system because the policy system is winning new business. It is harder to win new business than it is to retain existing customers.

If possible, it is beneficial to gain experience with a new technology, such as Web services, before exposing it to the harsh environment of the Internet in an application that is going to expose any shortcomings directly to the insurance companies' customers. This reflects the observed take-up of Web services in the industry today. Most Web service applications are within the intranet.

2. The second scenario (External Claims Assessor Management) is an *extension* of the claims scenario, in which we investigate secure interoperable Web services between LGI/DCI and its outsourced claims assessors.
3. We believe the decisions regarding the architecture, design and development tasks needed to build interoperable Web services can be reused in the policy administration area

5.2.3 Current IT infrastructure

Before proceeding with the new automated claim assessment solution design, we need to fully understand existing technologies and staff roles that form the claims process.

In this section we describe the architecture of the environment that supports the existing claims system. We also describe the interactions, which are divided between users and components, and between the components and any other remaining components. Both LGI and DCI current IT infrastructures are shown in Figure 5-2 on page 75.

The claims system for DCI in is based around a three-tiered net-centric architecture that lets clients register claims online and receive updates on the status of their claims through e-mail or traditional mail. The IT infrastructure that supports the claims processing consists of a cluster of application servers that handle both the transformation and collation of data provided by the client, and

sends this data in the form of requests to an off-the-shelf claims application in the back-end system. Replies from the back-end applications are sent to the application servers, where they are presented dynamically to the client as Web pages. Other processing is performed manually or in the back-end system.

LGI has a message based hub-and-spoke infrastructure with all client applications sending requests to a central broker that handles the transformation and routing to the back-end applications or workflow systems. All replies from the applications are sent to the message hub for transformation and routing to the client application. A customer registers a claim by contacting a call center or their insurance agent where the claim agent collects the required information and uses EDI or a dedicated client to input the information to LGI. As with DCI, all the claims processing in LGI is done manually, or through dedicated client applications accessible by the claims handler and claims supervisor. LGI provides channels for business partners into the message broker.

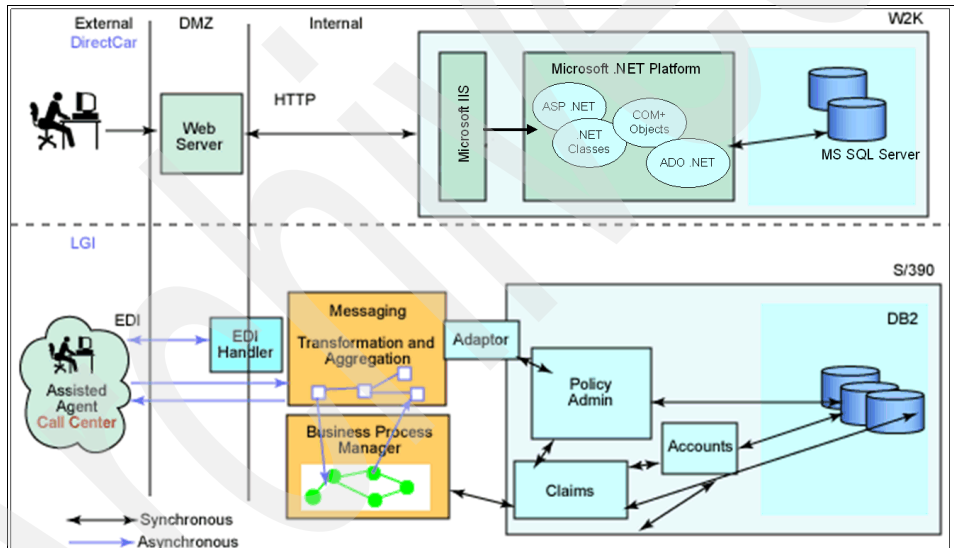


Figure 5-2 LGI and DCI current IT infrastructure

From a workflow point of view, the existing claims system for LGI and DCI can be divided into four different business processes:

Register claim

In the case of LGI the customer contacts the call center or their insurance agent, who records their accident details. The call center agent uses a dedicated client application to enter the claims information directly into the claims database and completes the registration process by giving the client a claim number for reference.

The insurance agent completes the registration forms using an EDI package which then passes all the claims onto LGI to insert into the claims database and receives a claim number back to handle subsequent queries from the customer.

In the case of DCI, the client logs on to the DCI Web site and is able to register claims online. The claim reference is presented to the client online once the required processing is completed.

Validate claim

The raised claim is authenticated to confirm that the client's policy is valid and not expired, that the driver is insured and paid up on the policy, and that the details provided are accurate.

Investigate claim

The claim is investigated by requesting and acquiring the police and medical reports from the authorities. At this stage, the assessment company is contacted to perform an assessment of the damage and to make recommendations on how to proceed.

Judge claim

Based on all the information provided, the claims handler or claims supervisor makes a decision on whether the car is to be fixed or replaced, or the claim rejected.

In the following sections we will focus only on claim registration and validation processes because they are considered to provide a full subset of significant interoperability alternatives.

5.2.4 Technical constraints

The following policy directives have been set for the merger solution being implemented by LGI; each of them is also detailed with the consequent impacts on the solution building process and design.

- ▶ Total Cost reduction
 - No spending for change on current IT infrastructure
 - Reuse of the current investments and applications
- ▶ Use open standards-based technologies (J2EE, BPEL, UML2, Web services)
- ▶ Look for opportunities to build common infrastructures that can be used for multiple solutions
- ▶ Short delivery term: solution must be totally implemented within one year
- ▶ Merge disparate IT system

- Build a connectivity infrastructure for the application and integration layer as well as for the transport and network layer on top of the existing IT infrastructure.
 - Difficulty in monitoring and managing different distributed solution components
 - Possible delays in problem determination due to cross product, solution or organization boundaries
- ▶ Merge disparate long running processes
- Monitoring and rationalization of single process steps
 - Reducing overlapping and optimizing interactions

5.2.5 Solution level design

The business vision, as shown in Figure 5-3, is to create a one-company-for-all-channels view that hides the customer from the complexity of the LGI and DCI back-end application. The main requirement of this merged claims system is a common front end for the merged company.

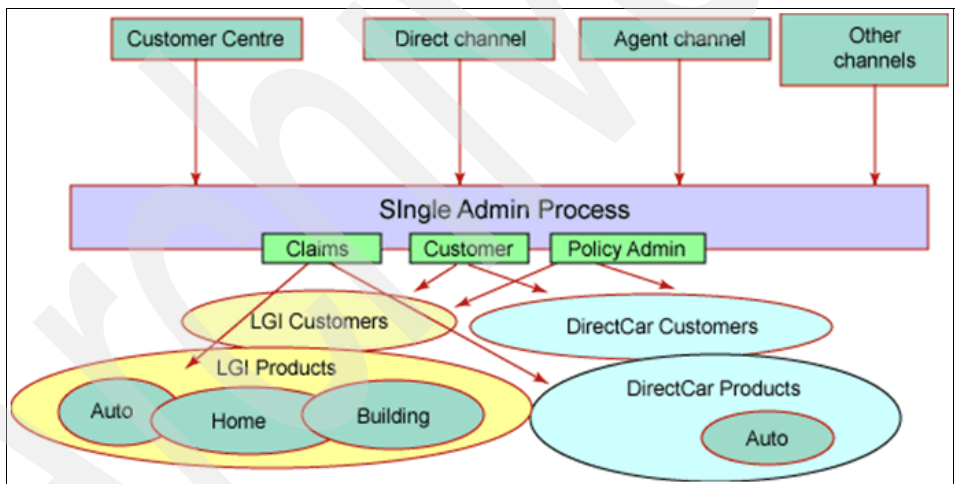


Figure 5-3 Business vision of merged companies process

Workflow automation

The improvements and implementation required to create the merged claims system are described below:

Register claim

The register claim process is automated by creating an online common Web interface for the joint customer base. Clients are able to raise claims online and

get their claim references. The common front end authenticates the users against the federated secured user directory, and provides a customized user interface for the internal users.

Validate claim

The validation of claims is automated by developing business process flows that represent the validation process and deploying them in a workflow engine.

Investigate claim

The investigation process is automated with integrated access to the assessment companies and other business partners. Using workflow technologies, the claims handler and supervisors have access to the different claims and are able to make decisions or correct errors where required.

Judge claim

This process is automated and linked to the other processes and services for access to any data that might be needed to help make a judgment.

5.2.6 Technical approach

In this section we describe technical directions needed to correctly address the solution's high level architecture in an interoperable environment. Solution design and interoperability patterns will be detailed on the following chapter.

As part of merging the claims systems, a consolidation of the IT infrastructure to leverage the best of both systems is crucial, and where possible, reuse of technologies and skills are advised. One of the biggest challenges in the merger is that the DCI and LGI back-end systems must be kept separate. Merging the security policies of the two companies also influences some of the changes to be made to the business processes.

The technical solution focuses on the achievement of a new application integration layer being able to provide a new integrated business process which includes all existing infrastructures of the two companies.

Enterprise application integration (EAI)

The required solution must be able to integrate the two merged insurance companies without replacing actual back-end systems. We need to make use of their existing functions. A typical Enterprise Application Integration approach to the solution is needed. We need to try to interface existing functions to a single application infrastructure layer so the users of the back-end systems are unaware that the services they are using different applications hosted on different systems.

Business process management (BPM)

The requirements we are focusing on require a solution based on a business process management engine. This engine could be both WebSphere WebSphere MQ Workflow and WebSphere Business Integration Server Foundation.

The choice of one specific engine is not strictly required. Although both products have the similar functionality, each has its own strengths. For example, if part of a business process interacts frequently with Java applications, Enterprise JavaBeans (EJBs) or Web services, then we can model this part as a Process Choreographer subprocess in an WebSphere MQ Workflow business process. Process choreographer has the advantage of

- ▶ Better build-time support in WebSphere Studio Application Developer Integration Edition for modeling EJB and Web service interactions
- ▶ Transactional support for EJB invocations
- ▶ Transactional support for Web-service invocations using EJB or JMS bindings.
- ▶ Being able to model a sequence of transactions as a non-interruptible processes to reduce the number of interactions

If we model Process Choreographer processes as sub-processes in an WebSphere MQ Workflow master process we can monitor the entire flow using WebSphere MQ Workflow monitoring tools. The Process Choreographer subprocesses are treated as activities in the WebSphere MQ Workflow.

Where we need to reuse our existing WebSphere MQ Workflow processes in Process Choreographer, we can *model* them as subprocesses inside new Process Choreographer processes, but continue to execute them in WebSphere MQ Workflow. That way, we can avoid migrating WebSphere MQ Workflow processes to the Process Choreographer.

In the solution proposed for the current scenario, each Microsoft .Net Web service can be considered a task of the Process Choreographer process. The process acts as a Web service client and the task is implemented by the service provider external to the system context.

The interoperability requirement is that Process Choreographer is able to import a Microsoft .Net produced WSDL file and see the service as one of the available tasks. The interoperability requirement can be considered similar for any Microsoft .Net Web service which is consumed in a WebSphere environment.

Service-Oriented Architecture (SOA)

What the proposed solution needs to achieve is that both existing back-end environments are able to provide their existing applications as Web services and,

if possible, it would be useful if these services were invocable in the same way regardless of the underlying technology that is going to provide them. Our main purpose matches the goals of Service-Oriented Architecture (SOA).

SOA is an emerging architecture which is trying to realize the integration of heterogeneous systems as part of an on demand architecture. The architecture proposes a model in which the whole system is made of disparate nodes, each of them providing its own services to the other ones. Services provided by each single node are required to be loosely coupled, locationally transparent and protocol independent; the accomplishment of these requirements allow the services to be consumed by the other nodes in a disparate IT infrastructure.

Web services (WS)

Web services technology is to be used for this solution as the integration layer. In our scenario, Web services are the best choice for implementing a Service-Oriented Architecture.

This choice is driven by two factors:

- ▶ Web services are a practical implementation of an SOA
- ▶ Web service technology is strongly focused on achieving interoperability, particularly between Microsoft and IBM environments.
 - Web services are developed through open specification processes
 - IBM, Microsoft and other vendors are running joint workshops to make sure that the resulting specifications are practical and interoperable
 - The WS-I organization is running a specification and testing process to assist vendors achieve interoperability

Service-Oriented Architecture stack layers and Web services architecture are detailed in the following chapters.

Web services security (WS-Security)

Our plan is to start with a simple register claim use case that gives reader an understanding of interoperability problems. Then our plan is to introduce security by investigating use cases involving external claims assessors.

For this reason, we suppose that the claims use case does not require security features. It is justified by the business and technical context: after the merger, both LGI and DCI back-end system can be considered part of the same company, sharing the same intranet. Disclosure of information is allowed without any restriction inside the intranet itself.

The second scenario detailed later in this chapter has the purpose of adding more complicated service requirements to the solution and requires the adoption of WS-Security.

Interoperability and WS-I

Even though Web services stack elements are based on standards specifications, some of these specifications have not been finalized. Others, even if already published in a final version, have unfortunately been implemented in different ways. For that reason, we need to focus on any differences in the WebSphere and Microsoft .Net implementations of current Web services specifications.

In the following chapters we give a detailed description of the interoperability issues between the two platforms providing some recommendations about the correct application of design patterns and the implementation of services.

5.2.7 Target IT infrastructure

In this section we describe the high-level architecture of the environment that is proposed to support the new merged claims system. We also describe the interactions between users and components.

The solution is built assuming the complete reuse of the existing legacy infrastructure and claims applications from LGI and DCI. The existing DCI Web layer is replaced with a brand new Web layer not reusing the Web interface from DCI. This choice is driven by the following reasons:

- ▶ Business and strategic reasons:
 - LGI strategy in acquiring DCI was much more related to the acquisition of the market share than the acquisition of the Microsoft .Net technology.
 - The merger of the two companies required a change of the look and feel of the associated Web site.
- ▶ Technical reasons:
 - Current IT hardware infrastructure is not sized for the new expected workload: DCI has less than one million policies while LGI has more than five million
 - More Web functions are in plan to be developed
 - The existing controller on the server side needs to be changed in order to communicate with both DCI and LGI, and needs to follow new business rules.

The approach we decided to follow for building the solution is to focus on providing an interface between the presentation layer and the two existing back

ends. For the purposes of the redbook, we won't actually build the presentation layer and its associated Web pages.

The interface between the presentation layer and the back-end systems is realized via Web services using the same component to invoke both service providers. Since one service provider is based on WebSphere and the other on Microsoft .Net platforms, being able to achieve this goal means we have successfully addressed the development of an interoperable solution between WebSphere and Microsoft .Net.

Architecture overview diagram

Figure 5-4 shows the proposed architecture overview diagram. The figure does not aim to address hardware platforms or software configuration which is the purpose of the following chapter. We want to focus on:

- ▶ Components we intend to reuse
- ▶ Components we intent to not reuse
- ▶ New required components
- ▶ The proposed interaction between users and components

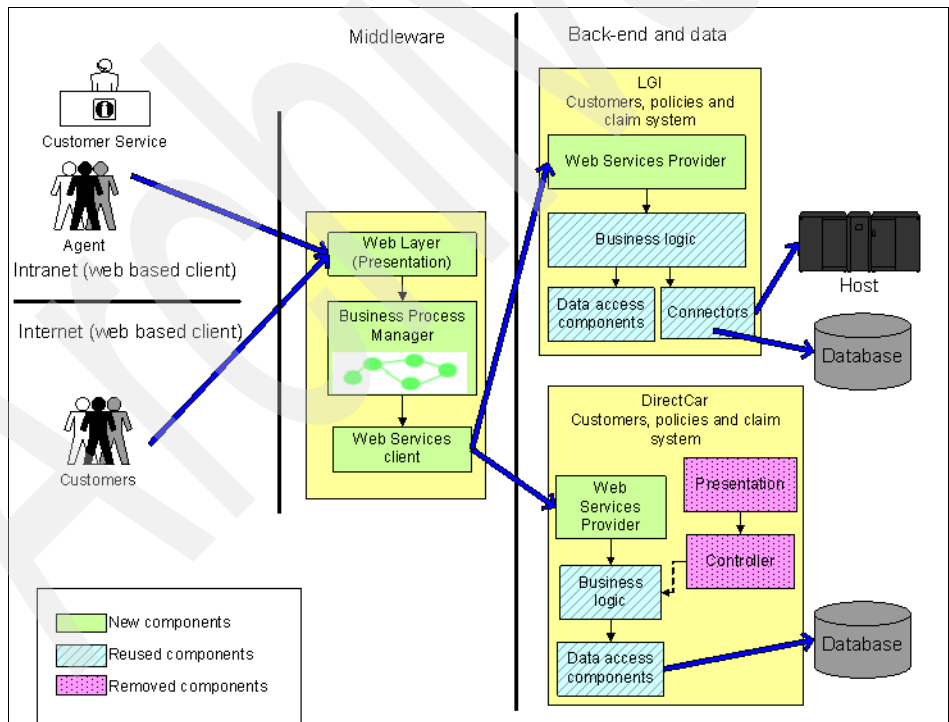


Figure 5-4 Merged claim process architecture overview diagram

As shown in the figure the solution architecture is based on the following decisions:

- ▶ Creation of a new Web layer composed by a presentation logic layer, a business process manager layer and a Web services client layer
- ▶ Creation of two Web services provider layers on both back-end systems exposing business logic services
- ▶ Complete reuse of both back-end components such as business logic, data access and connectors
- ▶ Removal of the DCI Web layer (presentation and controller)

5.3 External claims assessor management²

This scenario is an extension of the previous one providing further interoperability requirements and use cases. It supposes that LGI has recently completed the claims and policy processes related to the previous scenario; however, LGI still feel their view of the new processes is fragmented and there are still parts of the flow which are outside of their control and cannot be monitored from one single point.

5.3.1 Business goals

The business goal of this scenario to improve company profitability by optimizing the cost and duration of the claim process workflow. In analyzing the system LGI has identified a particular problem in the assessment activity of the Claims Process. Costly delays have been identified in the selection and follow-up of external claims assessors. LGI would like to automate the selection of claims assessors and get more visibility of their progress and efficiency.

The following business goals were identified in the current scenario:

- ▶ Monitor and manage the entire claims process including activities performed by external claims assessors
- ▶ Reduce administration costs by minimizing manual activities involved in managing claims assessment
- ▶ Increase customer satisfaction by reducing administrative delays on claims queries
- ▶ Identify and resolve business processing delays quickly

² As mentioned in "Limitations" on page 4 we intend to use the Claims Assessor scenario to demonstrate secure interoperable Web services when the WS-I security profile 1.1 is approved and implemented by WebSphere and Microsoft .Net. which we expect to be during 2005

5.3.2 Solution context

We assume that the merged claims system in the first scenario already exists and is fully operational. We now suppose that LGI wants to reduce the costs of assessing claims by automating the process of getting loss adjustments from external assessors for both LGI and DCI.

LGI cannot change the implementation of its assessor automation system without disrupting its operations and existing assessors. To encourage seamless integration, LGI provides its assessors with details of a number of alternative interfaces to the assessor automation system to which the assessors must conform, including a browser interface for assessors who choose not to automate their own processes. LGI is not concerned with how the external assessors implement their back-end code as long as they conform to the given interfaces to LGI's own automated assessor system.

The external claims assessor company we are going to consider has strong Microsoft skills and wants to use a Microsoft .Net system-based implementation to interact with the existing LGI infrastructure.

The required solution must automate all the “happy path” tasks involving LGI and external assessors. These tasks have been identified and detailed in term of specific single requirements:

- ▶ List retrieval of candidate based on clients' territory and car brand
- ▶ Check of assessors availability among all candidate assessors
- ▶ Assessor selection starting from candidate availability list based on business rules reflecting the cost, reliability and quality of the assessments made by the assessor
- ▶ Request of assessment to the selected assessor
- ▶ Receiving of assessment report

In the following sections we give an architecture overview of the full proposed automated process focusing only on those tasks regarding communication about LGI and external assessors because these are the one affected by interoperability requirements.

5.3.3 Current IT infrastructure

As we assume that this scenario is an extension of the previous one, the starting IT infrastructure for this scenario corresponds to the target IT infrastructure of the first scenario. As shown in Figure 5-5 on page 85, the current claims process is a single common process able to handle the administration of auto claims both for

LGI and DCI, the two merged companies. External assessors are only managed by means of manual activities that is using phone, fax or e-mail.

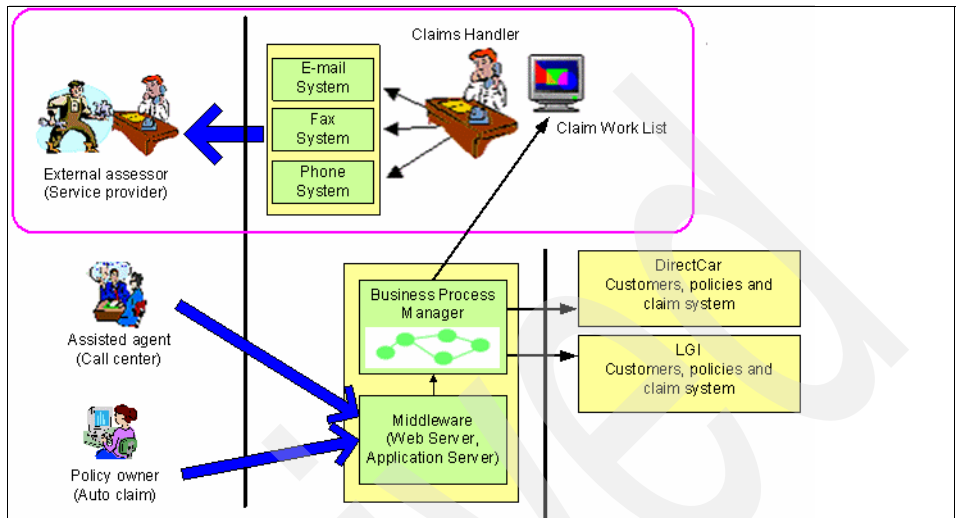


Figure 5-5 Current IT infrastructure of the external claims assessor management system

5.3.4 Technical constraints

The following list summarize technical constraint established by LGI:

- ▶ Minimize impact of new solution on existing applications and processes
 - Maintain existing channels
 - Reuse of existing applications including BPM system
 - New applications must use the corporate LDAP directory for staffing roles and definitions
- ▶ Demonstrate development productivity improvements
- ▶ Use open standards

5.3.5 Solution level design

This technical solution addresses the extension of the current claims business process to include a full automated workflow management system to handle auto claim assessments by external independent assessors.

The Assessor System will be responsible for identifying and selecting an assessor based on availability and a match to the needs of the claimants and their damaged vehicles. The system will:

- ▶ Manage the communications between the Claims Assessor(s) and the company both through the initial selection process and the stages of the actual assessment using an external gateway.
- ▶ Hold details of the available claims assessors and their communication details
- ▶ Link to the main claims process and allow authorized claims personnel to monitor the progress of a claim assessment
- ▶ Hold details of the state of a claim assessment through its life cycle
- ▶ Allow event points to be established which can feed event information (including state) to the main claims process for display on a Business Monitor/Dashboard
- ▶ Provide a rules engine which can be updated by an authorized Claims Administrator with business rules which determine the type of Claims Assessor for a particular claim

Workflow automation

The workflow part of this scenario is tightly connected to the previous scenario business process because we are now exploiting a specific activity in the investigation task, when an external service provider is delegated to perform the assessment.

Figure 5-6 on page 87 shows all activities composing the required workflow. We give a short description of each activities focusing only those regarding the interactions with External Claim Assessors.

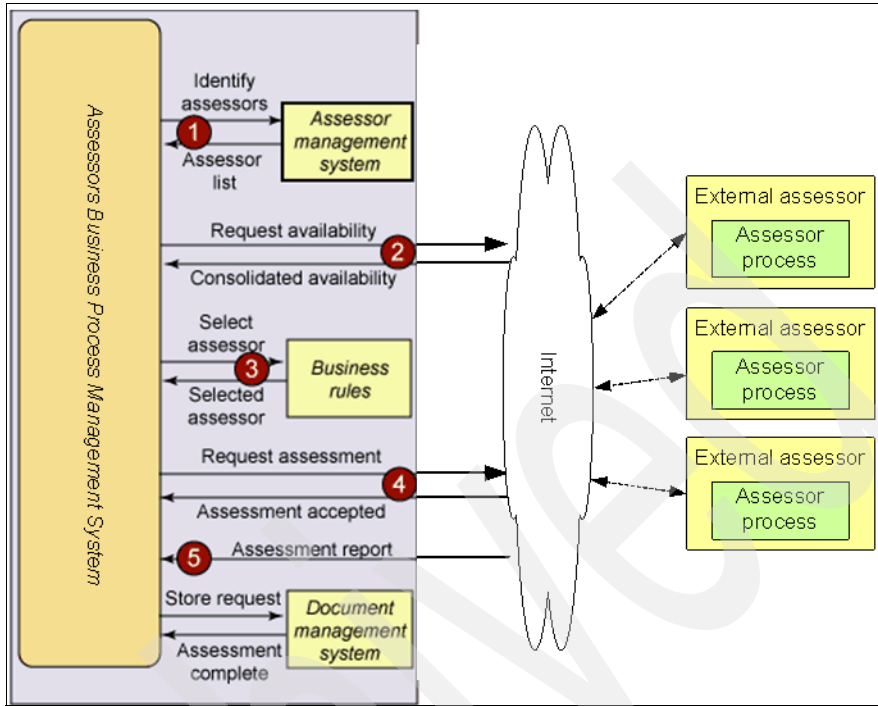


Figure 5-6 External claims assessor management workflow automation

Activity 1 - Identify assessors

This activity will identify suitable assessors from the existing assessor management system using the claimants vehicle type and postal code exacted from the claim record identified by the claim id. A list of assessors matching the criteria will be returned back to the assessor process manager. If no match is found then an exception message will be generated and displayed on the claims handler's work-list.

Activity 2 - Request Availability

Send an assessment request with the list of assessors to the distribution system. The distribution system will handle all replies, and consolidate the list of assessors returned. The assessors are given a date and time to reply. This time will trigger the next activity.

Activity 3 - Select assessor

If no assessors have been returned then trigger an exception message for viewing and action by the claims handler. If more than one assessor is selected then apply the business rules to identify the most suitable assessor. The criteria

for selection are based on response time, priority of assessment, ranking of assessor and availability to perform assessment.

Activity 4 - Request assessment

The system send the full claim details to the selected external assessor.

Activity 5 - Receive Assessment report

When the assessment report is returned, store the report reference in the claims document management system, and return control to the main claims system with a response that the assessor activity is complete. If the report is not received within the date and time agreed, an exception should be generated and again displayed on the claim handler's work-list.

At any point in the process, the claims handler should be able to select the claim from their work-list and display the status of the assessment.

5.3.6 Technical approach

The technical approach to the external claims assessor scenario is carried out supposing that all issues detailed in the previous scenario, such as BPM, EAI, SOA, ESB and Web services remain still valid. The current scenario, in fact, requires the adoption of additional technologies and components in order to meet the integration requirements. These technologies have to address:

- ▶ Quality of service, especially in terms of security
- ▶ Web services publishing over the Internet and being used by disparate external consumers

Web service security

This scenario can be considered the most suitable to address interoperability issues related to Web services security for the following reasons:

- ▶ The scenario uses Web service in the insecure Internet environment
- ▶ We are exchanging contractual information and it is important that all parties can sign up to the exchanged information being legally valid. This will involve us in issues of data integrity and non-repudiation of delivery of material.

A deep analysis of the messages supposed to be exchanged between LGI and external assessor, reveals that not all of them need the same kinds of security. Detailed functional and business requirements analysis identifies the following security requirements for the different interactions involved in dealing with the external claims assessors:

- ▶ **Client and server authentication based** security for the first message, which is the availability request. The assessors need to trust who is

requesting their availability while LGI needs to trust who is claiming availability. Encryption of exchanged information is not needed because their content is of no relevance.

- ▶ **Server authentication based security and digital signature** for the second message, which is the assessment acceptance. Server authentication is required, because we suppose that, from a client side point of view, it is needed to provide the assessors with an authenticated assessment request and, from a server side point of view, it is required that the assessment acceptance is signed because the message must be delivered as without modification.
- ▶ **Encryption based security and digital signature** for the third message, which is the assessment report; cryptography is required because the content of the message is confidential and must be protected during the network route from the assessor IT system to the LGI IT system, while digital signature is required because the assessment report has the validity of an official document.

Web services dynamic invocation and UDDI

When Web services are consumed via the Internet by multiple clients, as in the External Claim Assessors scenario, it is recommended to publish the Web service in an UDDI registry. UDDI is equivalent to the yellow pages of a telephone systems providing, for a specific service, both the location and the interface specification.

UDDI registry provides several advantages to both service producer and consumers:

- ▶ The location and interface of a service can be updated without the need to contact all known consumers to send them new specifications.
- ▶ Consumers can access to the service using dynamic invocation at running time which is a valid alternative to the static invocation made at development time. Dynamic invocation has the advantage of not requiring a client update if the service interface or location is has changed in a simple way, such as a change to the location of the service.

Web Services Gateway

The adoption of a Web Services Gateway must be considered mandatory in the selected scenario; this component is in fact the most suitable in an inter-enterprise environment providing a number of advantages:

- ▶ Central access point for all services crossing the enterprise boundary
The gateway provides a single, well-known point to for internal service consumers to access external service providers, and vice versa

- ▶ De-coupling the deployment of Web services from clients
The gateway isolates any changes in the deployment of services 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 consumers are allowed to access services. Typically the gateway is deployed in the DMZ with only the necessary ports open to the Internet to protect the servers in the intranet that are hosting the Web services from attack.
- ▶ Protocol conversion between Web service requesters and providers
Access to the services of applications that use protocols other than HTTP is planned for the near future. Therefore, access to the Web services has to be open for different protocols.

Enterprise Service Bus

The Enterprise Service Bus is an emerging model in the evolution of Service-Oriented Architecture. Its implementation means the joining of:

- ▶ A Service-Oriented Architecture
- ▶ A Message-driven architecture
- ▶ An Event-driven Architecture

In the full claims scenario, an ESB could provide easier administration over the deployment and management of services inside LGI and DCI, and mediating between different Web service interfaces for applications implemented with different interfaces in LGI and DCI. In the claims assessor scenario, the Web services Gateway provides a piece of ESB capability by de-coupling the deployment of a service from its external endpoint advertised to claims assessors.

Interoperability

Among all technologies required for the implementation of the proposed target solution, the following list continues the ones investigated in following of this redbook with the purpose to achieve to an interoperable solution:

- ▶ Web services security
- ▶ UDDI
- ▶ Web Services Gateway

5.3.7 Target IT infrastructure

The proposed solution IT infrastructure is based on the existing merged LGI/DCI solution. The existing middleware components are reused to define a new

business process for the workflow automation of the External Claim Assessors management. The major extension in this scenario is the communication layer between the LGI IT system and the external assessors IT system is via Web services.

The extension has:

- ▶ Different quality of service requirements
- ▶ The need to communicate with more than one unknown remote system
- ▶ Use of Internet channel which is intrinsically insecure, devoid of guaranties of message delivery and devoid of monitoring systems.

The achievement of the interoperability between the LGI WebSphere based IT infrastructure and an external assessors Microsoft .Net IT system is, in this case, more complicate to address; the reason is that security standards implementation is required and this increases the risk of failure in case of consistent differences among different implementations.

Architecture overview diagram

The architecture overview diagram of the proposed solution is shown in Figure 5-7 on page 92. As explained for the merger solution architecture, hardware platforms or software configuration are detailed in the following chapter. The figure shows all required components, the most part of them already existing.

The only one new required component is the Web Services Gateway which is required as de-coupling platform from the internal LGI infrastructure and the external assessors systems available through the Internet channel.

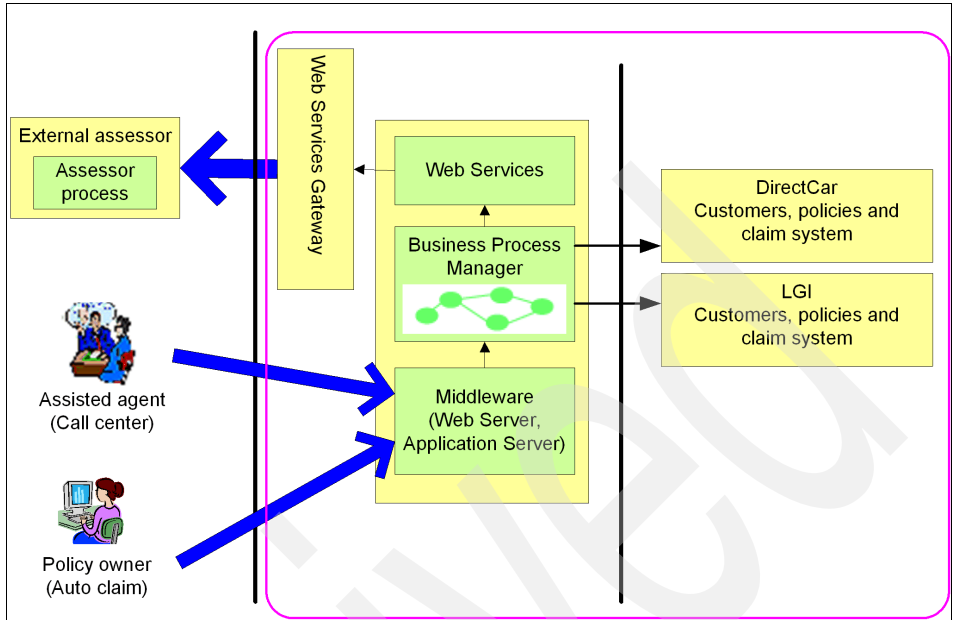


Figure 5-7 External claims assessor management system architecture overview diagram

5.4 Summary

This chapter should be used by consultant and solution architects who need to address customer proposals for solutions requiring system integration between a WebSphere-based application architecture and a Microsoft .Net based application environment.

Two different scenarios have been described in terms of solution architecture; the approach was to start from the analysis of business goals, current technical environment and technical constraint, then to pass through the analysis of available technologies, and finish with the proposed architecture overview diagram.

The first scenario is simpler in terms of the technologies used, the second one is more complex because it requires security, dynamic publishing and a gateway. This difference is useful for readers who start inspecting basic interoperability issues, thus learning to understand the most complex ones.

Interoperability patterns

In this chapter, we discuss how to refine the IBM Patterns for e-business to use Web services as an implementation of Service-Oriented Architecture (SOA).

The chapter includes the following topics:

- ▶ A brief introduction to the Patterns for e-business layered asset model
- ▶ Patterns for Service-Oriented Architecture and Web services
- ▶ Discussion of patterns and Web services, including the impact on patterns
- ▶ Identification, selection and application of patterns for a given business scenario with interoperability in consideration
- ▶ Any variation in standard implementation due to interoperability
- ▶ Discussion about emerging patterns for Web services to further improve interoperability

6.1 The Patterns for e-business layered asset model

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

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

These assets and their relationships to each other are shown in Figure 6-1 on page 95.

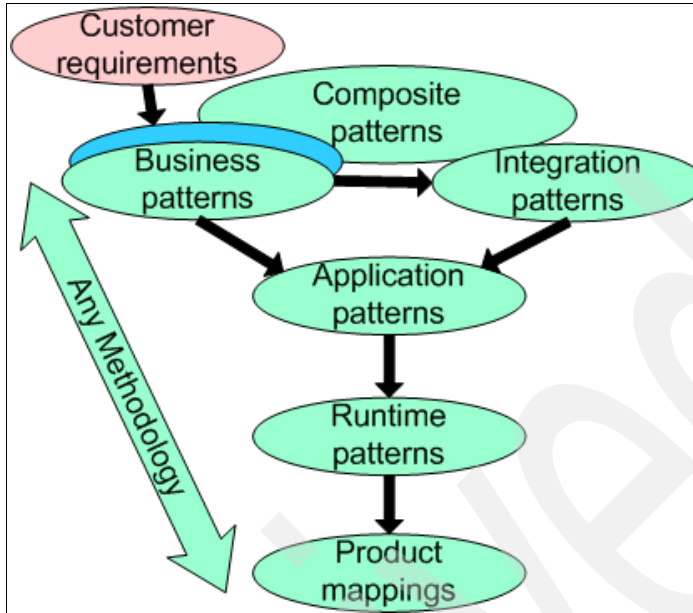


Figure 6-1 The Patterns for e-business layered asset model

Patterns for e-business Web site

The Patterns for e-business 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/>

6.2 SOA approach and Patterns for e-business

With a Service-Oriented Architecture approach, the Patterns for e-business method uses service integration bus Runtime patterns to connect to application components. This enables a business to be agile and respond quickly and efficiently to changes in the market and its customers' requirements, as well as to stay competitive.

When used with the SOA approach, the focus is on creating and integrating loosely coupled services to build the solutions required by a business. The service-oriented paradigm leverages the notion of services as discrete building blocks of business functionality, which are composed together to satisfy business requirements. Focusing on building services for reuse from existing applications

can be a more effective way to build solutions than building discrete point-to-point connections between applications.

The existing Business, Integration, Application and Runtime patterns are consistent with the SOA approach. The business problem will drive the identification of the appropriate patterns involved in a potential solution. Runtime patterns that involve two or more middleware nodes connecting logical application tiers will have additional communication options (and hence product mappings) between tiers with the use of services in a SOA.

In December 2004, the Patterns for e-business Web site, found at <http://www-106.ibm.com/developerworks/patterns/web-soa.html> was revised to accommodate SOA. The SOA concept simplifies designing runtime architectures. Using the concept of a service bus, multiple point-to-point connections between applications realized through connectors are replaced by single connections between applications realized as services. This can greatly simplify mapping Application patterns to the runtime architecture of the solution.

Service buses are not a “one size fits all” solution. In 4.4, “Web services and the Enterprise Service Bus” on page 58, we summarize the concept of an ESB. The discussion draws attention to how the qualities of service offered by Web services differ from those offered by an ESB. From a Patterns for e-business perspective, this notion of choosing between different types of service bus is modeled as defining the “X-Type” of service bus (see Figure 6-7 on page 102).

One obvious candidate for an “X-type” service bus is a WS-I compliant Web services bus. We have used this kind of service bus to implement this scenario.

6.2.1 Business::Self-Service pattern

The Business::Self-Service pattern captures the direct interactions between users and an enterprise, which range from simple information access to complex updates involving core enterprise systems data. This fits in nicely with a Service-Oriented Architecture, which consists of service consumer and service providers. Users such as customers, business partners, stockholders and employees are service consumers, while the enterprise is the service provider.

The Self-Service::Directly Integrated Single Channel Application pattern, for example, provides a user access channel to presentation logic running in the presentation tier. The presentation tier can request or consume services provided on the application tier. The application tier, in turn, can consume services provided on the back-end or enterprise tier, as shown in Figure 6-2 on page 97. 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).

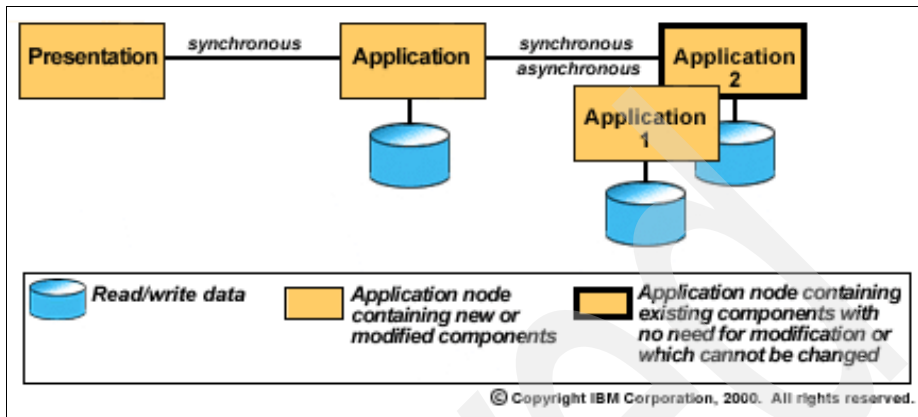


Figure 6-2 Self-Service::Directly Integrated Single Channel application pattern

The Self-Service::Decomposition application pattern handles a slightly more complicated situation, where data resides in two or more separate and dissimilar databases. The user request would actually require data from multiple, disparate back-end systems. The request is broken down into multiple requests (decomposing the request) which are sent to the different back-end databases, then the information sent back from the requests is gathered and put into the form of a response (recompose), as shown in Figure 6-3.

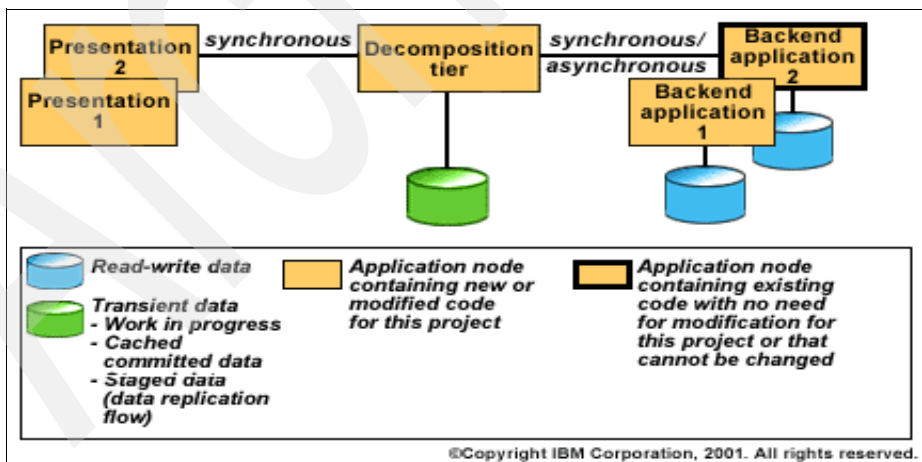


Figure 6-3 Self-Service::Decomposition application pattern

6.2.2 Extended Enterprise business pattern

The Extended Enterprise business pattern, which is also known as the Business-to-Business pattern 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 using a user interface by business partners across organizational boundaries.

The Extended Enterprise::Exposed Direct Connection pattern is the simplest pattern that allows a pair of applications to communicate with each other between organizational zones.

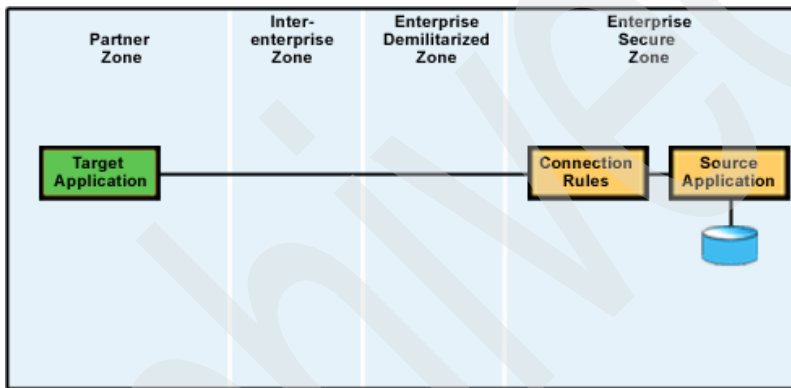


Figure 6-4 Extended Enterprise::Exposed Direct connection Application pattern

The different zones are most commonly thought of as being within different enterprises, as in Figure 6-4, or between the Internet and intranet. But the zones could be between different departments or companies in the same enterprise which have a degree of autonomous management of IT services in the different entities.

This pattern is an obvious candidate for the External Claims Assessor extension to the scenario. For the register claims scenario, the external interaction is via a Web browser interface rather with a partner application program and the Self-Service::Direct connection patterns are a better choice.

6.2.3 Discussion of patterns and Web services

It is no surprise that the introduction of a Service-Oriented Architecture has no effect on the Business and Integration pattern layer, except to suggest that the scale of horizontal integration could be more ambitious. SOA is more likely to

affect the lower level Runtime patterns, and will provide a model to allow enterprises to exploit Web services to implement Business and Integration patterns.

Business patterns

Web services technology can be used to implement all four types of Business patterns.

- ▶ **Self-Service pattern:** organizations can publish core business functions as Web services that can be consumed by customers and business partners.
- ▶ **Information Aggregation pattern:** content aggregates can access Web services provided by content providers. They can also use Web services to make content available to consumers.
- ▶ **Collaboration pattern:** individuals, programs and organizations can collaborate with one another by accessing standardized Web services deployed at their business partners' Web sites.
- ▶ **Extended Enterprise pattern:** Web services technologies can be used to simplify the process of integrating systems and business processes across the value chain.

Integration patterns

Web services technologies have a major impact on how Integration patterns are implemented.

- ▶ **Access Integration pattern:** Web services technologies allow us to integrate various back-end services to provide a seamless front end that can be accessed from multiple access channels.
- ▶ **Application Integration pattern:** Web services technologies simplify the application integration process by enabling loose coupling between the partner entities.

Application patterns

The Application pattern layer itself is unaffected by the introduction of a new technology. Nevertheless, these Application patterns, using lower level Runtime patterns, will allow enterprises to exploit Web services underneath some of these patterns, as shown in Figure 6-5 on page 100.

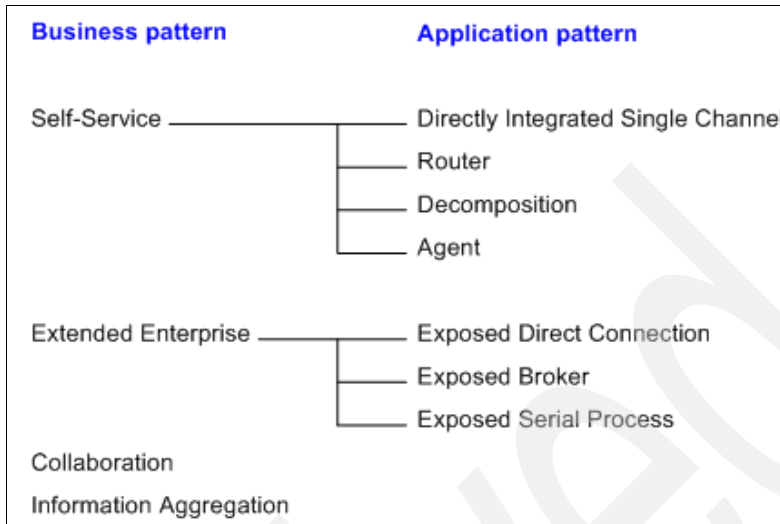


Figure 6-5 Application patterns whose runtimes could exploit Web services

Runtime and SOA patterns

Whenever a Runtime pattern shows two or more middleware nodes enabling two applications to communicate, the introduction of SOA allows an additional option for connecting these applications through a common bus rather than connecting the applications pairwise.

Between selecting an Application pattern and choosing a Runtime pattern, there is now an additional step for choosing whether to use a traditional or an SOA Runtime pattern. On the Patterns for e-business Web site, every Runtime pattern is being updated with an SOA variant. The new patterns have an [SOA] qualifier before the Runtime pattern name.

Let's take the Application Integration::Direct Connection application pattern as an example.

► Direct Connection

The Direct Connection pattern is appropriate when the applications share the same protocol and no adapter is needed, for instance when connecting two EJBs using RPC over IIOP.

There are two variations of the Direct Connection pattern:

– Direct connection single adapter

This pattern is appropriate for simple point-to-point integration where there is no requirement for reuse. The adapter is coupled to both applications and therefore specific to this connection.

- Direct connection federated adapters

The federated adapters pattern is suitable for point-to-point integration where there is a reuse requirement for the connections. Each adapter is effectively a half adapter. Two adapters are needed to connect two applications together. One side of the adapter implements a specific application connector; on the other side, all the adapters share a connector to a common protocol.

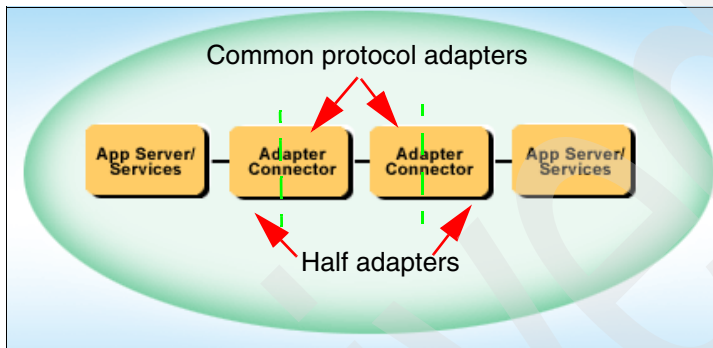


Figure 6-6 Direct connection: federated adapter

A new application is integrated into any of the existing applications by writing a half adapter to map between the new application and the shared common adapter protocol; the other half adapter will already be implemented for the existing application.

The [SOA] Runtime pattern variations on these are as follows.

- ▶ [SOA] Direct Connection

This is appropriate when the application is realized as a “native” service and can connect directly to the service bus. What is “native” depends on the implementation of the service bus. If the service bus is a WS-I compliant Web services bus then the application service needs to be WS-I compliant to connect directly to the bus.

- [SOA] Direct Connection single adapter

This variation is *inapplicable*: applications are always connected using a common service protocol.

- [SOA] Direct Connection federated adapter

The [SOA] Direct connection federated adapters pattern is appropriate when the application needs an adapter to connect it to the service bus. For the examples in this book, we have used this variation to wrap our legacy claims application in a WS-I compliant Web service.

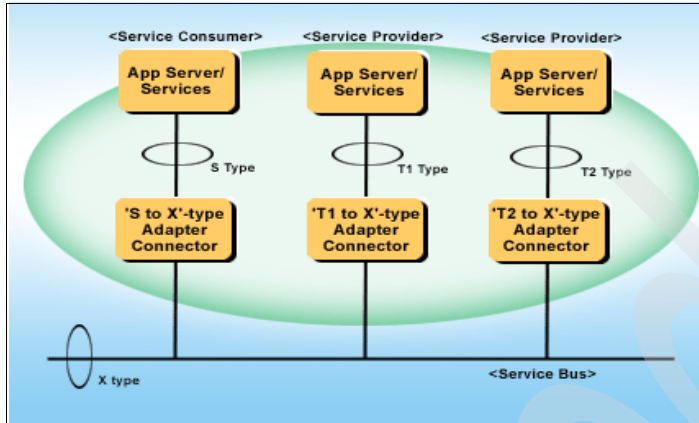


Figure 6-7 [SOA]Direct connection federated adapters

The SOA approach should simplify the runtime topology of a solution. Whether it does depends on a bus really being the seamless medium for connectivity shown in the architecture diagrams and having the appropriate qualities of service for the supported services. Guidance on the qualities of service to look for in a service bus can be found in the Patterns books about Web services that are beginning to appear, such as:

- ▶ *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346
- ▶ *Patterns: Service Oriented Architecture and Web Services*, SG24-6303

In the case of the scenarios in this redbook, the main quality of service we are looking for in the service bus is interoperability by requiring compliance with the WS-I basic profile 1.1.

6.3 Applying Interoperability patterns

In previous sections, we have described the Patterns for e-business layered assets model and patterns relevant to Web services. Now, in this section we will identify, select, apply and investigate various patterns for the given business scenarios, as described in Chapter 5, "Business scenarios" on page 69.

There is one simple scenario and one complex scenario we will implement that will demonstrate using Web services to integrate a solution using WebSphere and Microsoft .Net. components.

1. Merger and Acquisition scenario: in this scenario a large general insurance company, Lord General Insurance (LGI), has acquired a typical modern

dot.com auto insurance company, DirectCarInsure.com (DCI). LGI has a large legacy IT infrastructure based on S/390 and CICS whereas DCI has an e-business focused infrastructure based on Microsoft .Net.

2. External Claims Assessor Management: this scenario extends the first scenario to external claims assessors.

6.3.1 Mergers and Acquisitions scenario

As discussed in 5.2.6, “Technical approach” on page 78, Web services are a well-suited technology for this business scenario.

This scenario involves exposure of back-end business applications between different organizations, so we have taken a leveraging services approach (bottom-up, from legacy and packaged application).

Figure 6-8 shows high-level collaboration diagram for direct Internet access to the merged insurance company. It shows that a user will interact with the common front-end component and the common front-end component will consume various services that are deployed on both the Organizations, LGI and DCI.

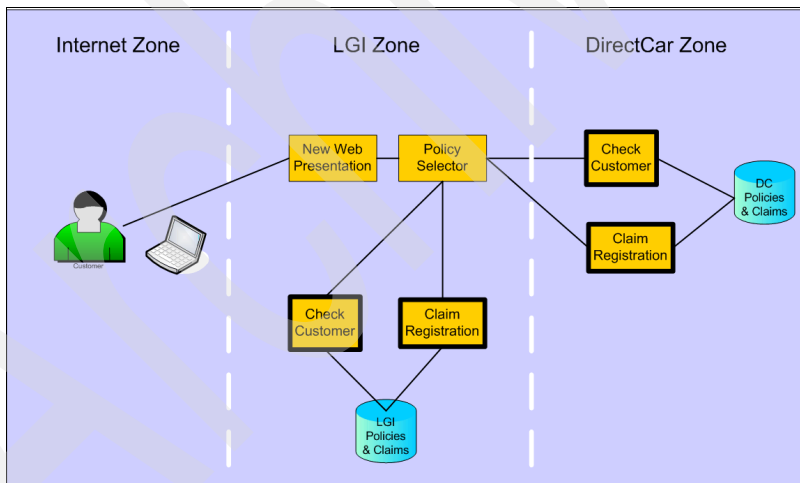


Figure 6-8 Merger and Acquisition: high-level [PI] collaboration diagram

Now we will select patterns for the scenario and map the patterns down to runtime components using the following steps.

1. Select Business patterns and Integration patterns.
2. Select Application patterns.
3. Apply Runtime patterns.

4. Apply product mappings.

Select Business patterns and Integration patterns

Business patterns identify the interaction between users, businesses, and data. In the claims registration scenario, we have users submitting their claim online, from which we can infer the use of the Self-Service pattern. Composition with an Application Integration pattern is also required to integrate the LGI and DCI businesses. We applied these patterns to the Merger and Acquisition scenario, as illustrated in Figure 6-9.

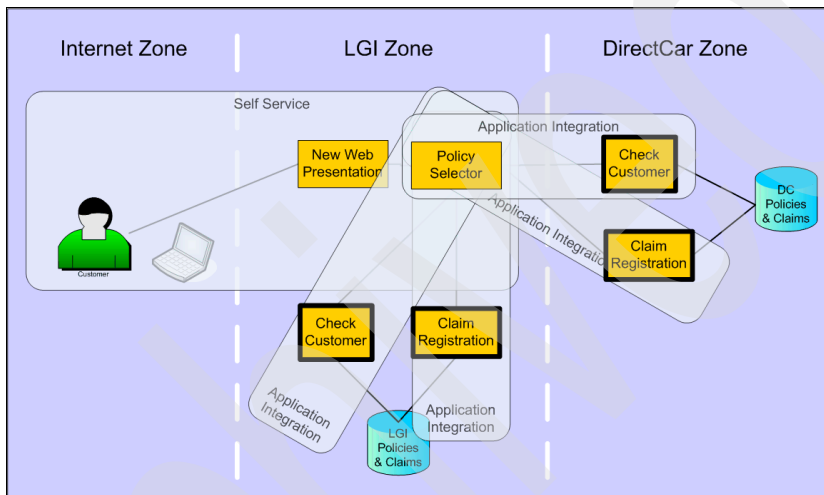


Figure 6-9 Business and Application Integration patterns

Select Application patterns

A Business pattern can be implemented using any of the Application patterns, related to the corresponding Business pattern. Selection of different Application patterns for the Business pattern provides solution flexibility so that the applied Business pattern can address the specific needs of the business process. The relevant Application patterns are as follows.

Application::Stand-Alone Single Channel

The Application::Stand-Alone Single Channel pattern is good for connecting a Web delivery channel to a single back-end system. However, integration with the rest of the enterprise is not automated in the interests of time-to-market and minimizing application complexity.

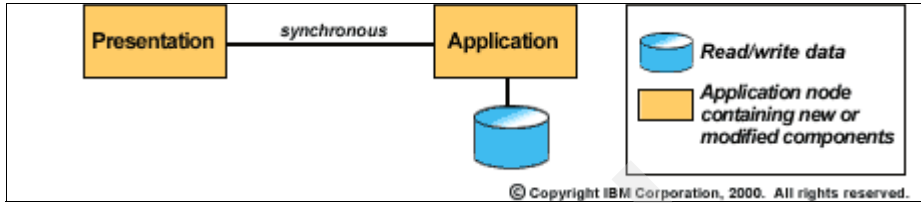


Figure 6-10 Stand-Alone Single Channel

In the case of claims registration, although time-to-market is of the essence, we must integrate with the rest of the claims applications in both LGI and DCI.

Application::Directly-Integrated Single Channel

The Directly Integrated Single Channel pattern shown in Figure 6-2 on page 97 provides a structure for applications that need one or more point-to-point connections with back-end applications but only need to focus on one delivery channel. This Application pattern can also be used to implement any one of the delivery channels.

This pattern is a closer fit to the claims registration scenario. The policy selector application is responsible for selecting either the LGI or DCI claims application to register the claim from the Web user.

Each of the connections between the new policy selector application and the back-end applications uses a Direct Connection Application Integration pattern as illustrated in Figure 6-11.

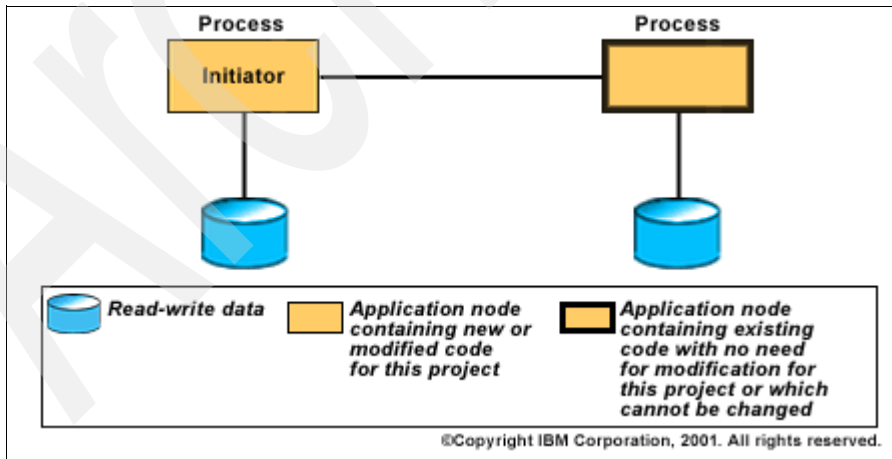


Figure 6-11 Direct-Connection Application Integration

In the full claims registration scenario, there are multiple rather than a single presentation channels; there is an agent channel and a call center channel in addition to the Web channel.

One consequence of using the Direct Integrated Single Channel pattern is that there will be no direct synchronization between different user channels other than as a result of using common back-end claims and policy applications. Historically, this has not been regarded as too big a problem in the insurance industry. With integration between channels to enable Web and call center channels to be used together cooperatively so as to improve the quality of customer service, this pattern is not going to fit the needs of newer applications.

Application::Router

The pattern for providing applications with multiple presentation delivery channels is the Application::Router pattern. The router becomes responsible for the details of different delivery channels and for session management to couple multiple delivery channels together to a single user.

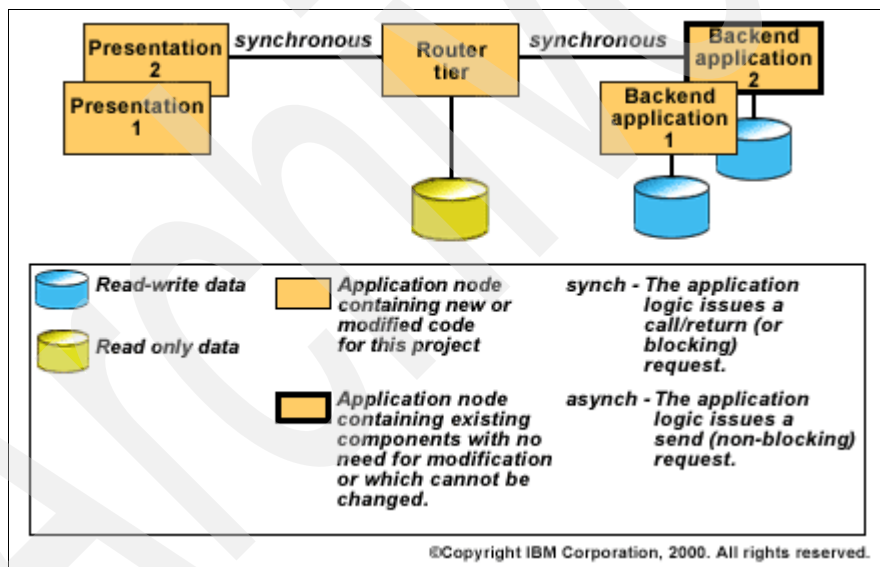


Figure 6-12 Router pattern

As in the Application::Directly Integrated Single Channel pattern, the router remains responsible for selecting the back-end application to handle the claims registration.

There is a weakness in the preceding patterns from the perspective of integrating the entire claims management process together so as to be able to modify and monitor the process. There is no explicit connection between all the steps in

registering a claim and then between registering the claims and the rest of the claims handling process.

Application::Decomposition

The Decomposition pattern addresses this requirement. The Application::Decomposition pattern shown in Figure 6-3 on page 97 extends the hub-and-spoke architecture provided by the Application::Router pattern. It decomposes a single, compound request from a client into several, simpler requests and intelligently routes them to multiple back-end applications. Typically, the responses from these multiple back-end applications are recomposed into a single response and sent back to the client.

The decomposition tier would typically be implemented by a workflow process, in our case by the claims handling workflow. Claims registration is the first sub-process in the workflow.

Our scenario is a variation of the Application::Decomposition pattern. The user submitting the claim initiates a claim workflow, receiving an acknowledgement containing a claim ID. The claimant is not involved in the process until contacted again.

To meet our business requirements, a further elaboration to the pattern will be needed (to enable the claimant to rejoin the claims process to query or discuss the status of their claim), either getting information about the status of a claim directly from the Web or through a call center, or routing a query for manual investigation by the claims handler.

Selected pattern

For the purposes of this redbook, and after exploring the interoperability between Microsoft .Net and WebSphere, we chose to limit ourselves to the Application::Directly Integrated Single Channel pattern combined with the Application Integration::Direct Connection pattern. They are sufficient to build the interoperability examples. The policy selector application will be responsible for controlling the interaction steps involved in registering a claim in a sequence of direct connections with the back-end LC and DCI systems. The mapping of the Application patterns is show in Figure 6-13 on page 108.

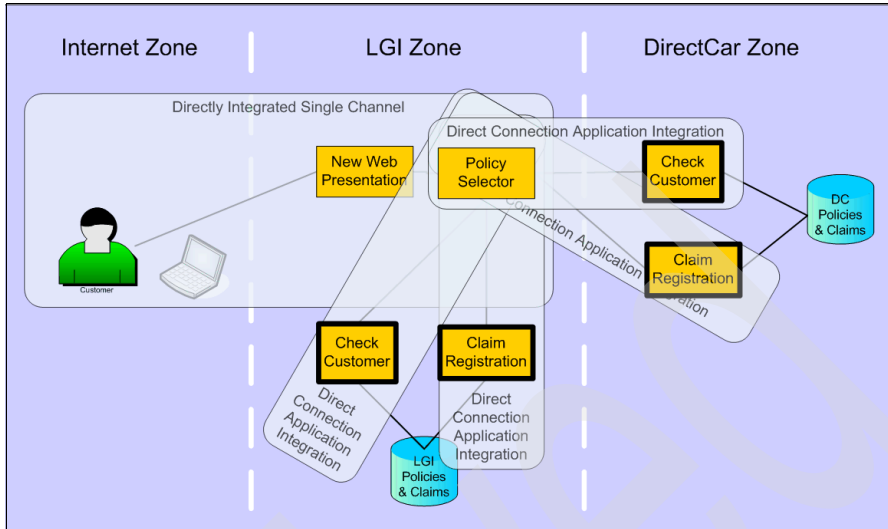


Figure 6-13 Selected Application patterns

Note: The System House team that developed the scenario selected the Application::Decomposition pattern because it addresses many more of the business requirements of the scenario.

Apply Runtime patterns

The next step is to choose Runtime patterns that most closely match the requirements of the application and use the quality of service and policy requirements. Each of the two Application patterns leads a choice from one or more underpinning Runtime patterns.

Figure 6-14 on page 109 shows the Runtime pattern we selected for the directly integrated single channel. This variation has a simple Web server redirector (also know as a reverse proxy server) to load balance and act as the first line of defense against hacker attacks by isolating the application server from the Web.

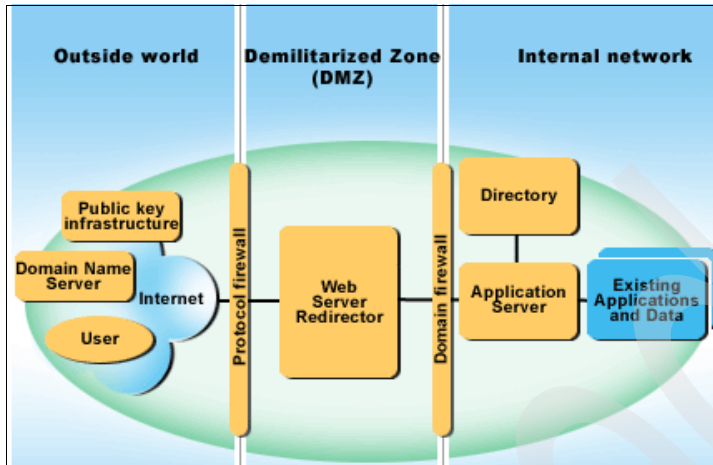


Figure 6-14 Directly Integrated Single Channel application pattern: Runtime Pattern: Variation 1

For the Application Integration::Direct Connection pattern, we selected the [SOA] Direct connection federated adapters Runtime pattern (see Figure 6-6 on page 101). The Service bus is a WS-I Basic Profile 1.1 compliant service bus. We considered using the Extended Enterprise [SOA] Exposed Direct Connection pattern between DCI and LGI and effectively having two service buses connected between the organizations. The decision to use an Extended Enterprise pattern would be based on the degree of autonomy in the administration of the two merged companies. Purely for the purposes of demonstrating interoperability of Web service for this redbook, we opted for a single bus linking the two organizations. We collapsed LGI and DCI into a single zone to emphasize that service buses should be contained within a zone. The pattern for the Claims Assessor extension to demonstrate the use of WS-Security will require the use of an Extended Enterprise pattern and will demonstrate connecting two service buses inside and outside the enterprise.

Another alternative is to use the [SOA] Broker variation pattern with federated adapters. The Broker pattern had a lot to recommend it in this scenario. Rather than having the policy selector application sequentially query LGI and DCI to match the claimant against a policy, there are benefits in having the policy selector broadcast the claimant's information over an Enterprise Service Bus and let LGI and DCI try to match the details in parallel.

The Broker pattern would ideally use the more capable Enterprise Service Bus to broadcast the claimant's request and the legacy applications would use specialized Enterprise Service Bus adapters implementing a publish-subscribe protocol to attach to the legacy applications to the ESB. A halfway house would be to have the broker use a WS-I compliant protocol across the service bus, and

make the broker responsible for distributing requests and selecting the preferred response.

With only two policy systems to concern us, we opted for the design which has the policy selector application querying LGI and DCI sequentially to seek a match to the claimant's policy. If the endpoints being considered in the choice were more numerous, or if they were continually changing, or if there were a clear performance requirement to process the request selection in parallel, then there would be a strong case for using a broker. In fact, in the Claims Assessor extension to this scenario, we do consider using a broker. So, in the interest of simplicity for the Claims registration scenario, we have opted to use the Runtime::Direct Connection pattern.

The result of mapping the Runtime patterns is shown in the Figure 6-15.

The Runtime::Directly Integrated Single Channel pattern uses a Web server redirector containing a Web server in the DMZ and an application server plug-in in the intranet, to provide more security by keeping application server in the internal zone. The Web server re-director is used to direct requests to the application server and keeps the application server secured in the internal zone. The Runtime::[SOA] DirectConnection pattern provides interaction between the components within the enterprise using a service bus.

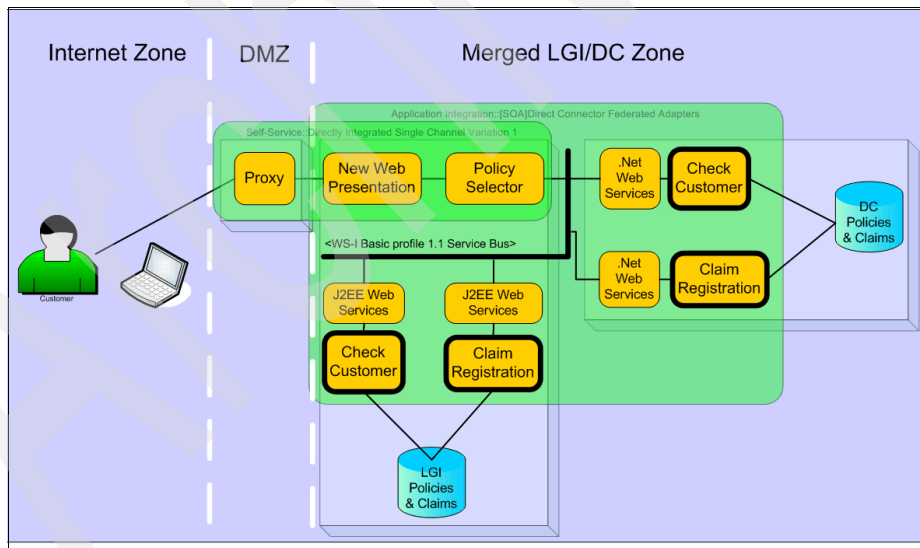


Figure 6-15 Application of Runtime patterns

Apply product mappings

In the product mapping, we identify, select and map the logical nodes defined in the Runtime pattern with the proven and tested products, which implement the runtime solution design on a selected platform with the chosen qualities of service and conforming to strategic IT policies concerning suppliers and technologies. The product mapping identifies the platform, software product name and version numbers of the products as well.

In the current scenario, we have two different platforms. LGI uses the WebSphere platform whereas DCI is based on the Microsoft .Net platform.

For LGI, we have used WebSphere Application Server V5.1.1.1 on Windows 2003 to host all the services and applications, and for DCI we have used IIS 6.0 on Windows 2003 to host all the services, as shown in Figure 6-16.

It is at this stage in the refinement process that we can finally make the decision to use WebSphere Studio Application Developer and Microsoft Visual Studio .Net 2003 to generate the WS-I compliant Web services adapters.

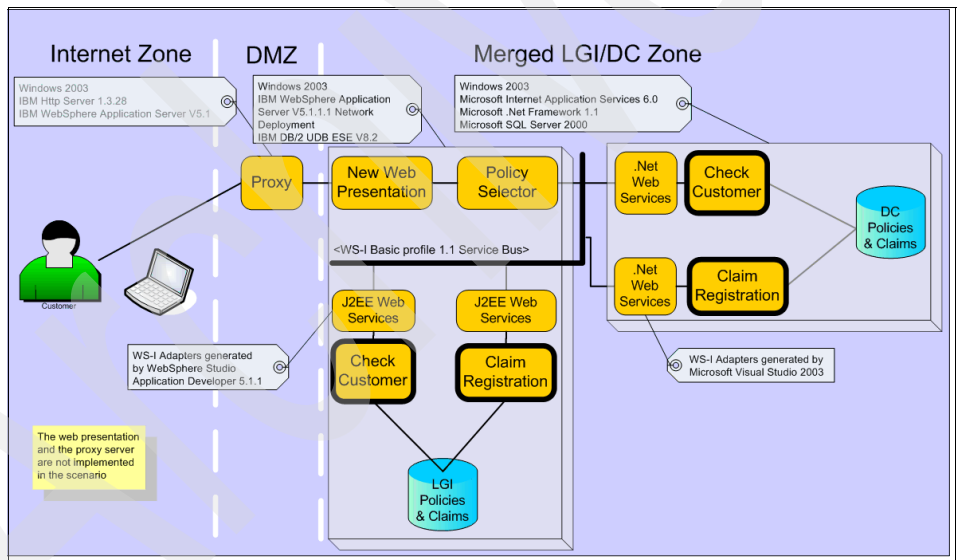


Figure 6-16 Merger and Acquisition: Product mappings

6.4 Summary

In this chapter, we discussed how to combine the existing Business, Integration, Application and Runtime patterns with an SOA approach. We then used the

e-business patterns approach to select a product mapping for the claims registration process in the Mergers and Acquisitions scenario.

6.5 Where to find more information

The following Web site provides a collection of IBM resources on the topic Patterns for e-business.

<http://www-106.ibm.com/developerworks/patterns/>

Archived

Web services roadmap

This chapter summarizes the various existing and emerging specifications of Web services. It then groups the services according to the Web services stack layers. It provides a quick reference of different specifications to a developer in one location.

7.1 Introduction

Web services are independent of platforms, applications and languages so that users of a service need not know about the computers and software that make the service available. Historically, Web services have worked smoothly only when the producer and the consumer of a Web service were created using tools from the same vendor such as Microsoft's .NET or IBM's WebSphere Studio Application Developer. A Microsoft-created service may not provide functional and quality service to an IBM-created consumer, or vice versa, because of the different ways each company goes about implementing the Web services specification. The differences do not exist because vendors do not agree on Web service specifications. They do, but many specifications are not finalized, or the specifications are written ambiguously, or the specifications provide options for making the Web service flexible. Flexibility and ambiguity in specifications sometimes result in inoperable Web service among different vendors.

Four initiatives are improving the practical interoperability of Web service specifications in practice.

1. Specifications are revised and tightened up where ambiguities have been found.
2. Vendors are running Web service workshops where interested parties get together and test the interoperability of specific specifications and combinations of specifications using scenarios.
3. The WS-I (Web service Interoperability) organization is publishing profiles to restrict the use of specifications to ways that are known to interoperate, and are publishing scenarios and tests for conformance.
4. WebSphere Studio Application Developer now includes WS-I conformance checking so that new Web services are automatically checked for conformance with WS-I.

This chapter describes the progress that is being made on Web service specifications and provides a short description of each of the Web service specifications.

7.2 List of Web services specifications¹

There are many Web service specifications in different states from proposed in principle, to standards that are approved and into revisions. Table 7-2 on page 116 lists the status of most Web service specifications. A different view of

¹ There is also a well-organized Web services roadmap at:

<http://www.w3c.or.kr/~hollabit/roadmap/ws-specs/index.html>

progress in adopting Web service is shown in Gartner, Inc.'s "hype cycles." Figure 7-1 shows a view of Web services adoption in business in 2004.

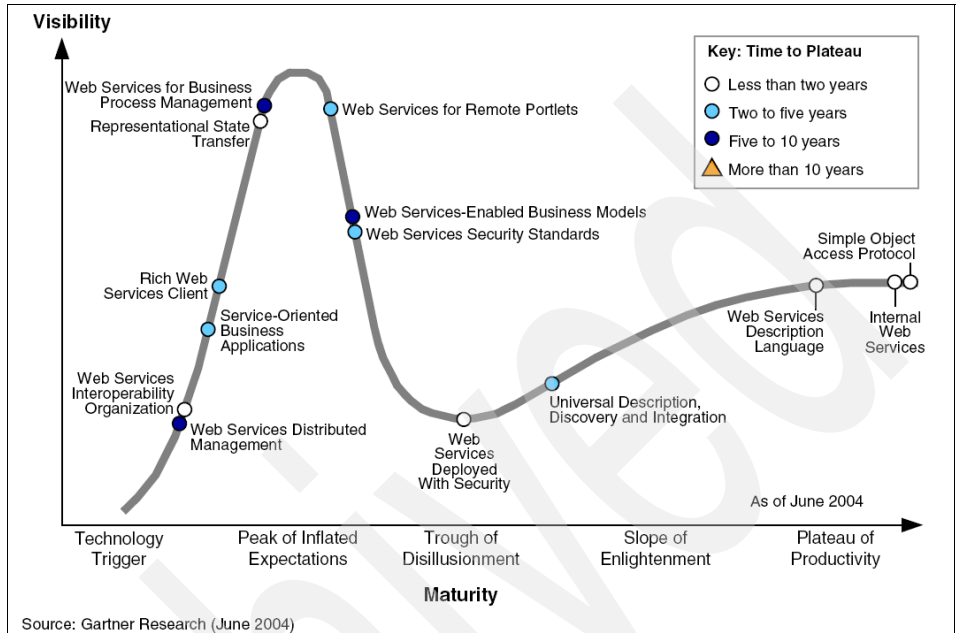


Figure 7-1 Hype Cycle for Web services, 2004

From W.Andrews, D. Smith, C. Abrams, R. Wagner, R. Valdes, C. Haight, M. Govekar, Gartner Strategic Analysis Report, 9 June 2004. Reprinted with permission from Gartner, Inc.

Notable is the appearance of the WS-I organization in the Technology trigger segment, with a very rapid (two year) expectation of adoption. Table 7-1 shows how selected Web service technologies have progressed up the Hype cycle curve between 2003 and 2004.

	Technology Triggers	Peak of Inflated Expectation	Trough of Disillusionment	Slope of Enlightenment	Plateau of Productivity
Remote Portlets	→				
Business Security	→				
UDDI		→			
WSDL			→		
Internal WS				→	
SOAP					→

Table 7-1 Change in Web service adoption 2003 - 2004 (Based on information supplied by Gartner, Inc.)

Table 7-2 gives a summary of the existing specifications and their sponsors and status as of 1 November 2004.

Table 7-2 Summary of Web service and related specifications

Specifications Title	Publisher	Status	
BPEL4WS 1.1	IBM, BEA, MSFT	2nd draft 5 May 2003	http://www-128.ibm.com/developerworks/library/ws-bpel/
JSR 101 1.1	JCP	Maint Rel Oct 14 2003I	http://jcp.org/en/jsr/detail?id=101
JSR109 1.0	JCP	Final Sept 21 2002	http://jcp.org/en/jsr/detail?id=109
SOAP 1.1	W3C	Note 8 may 2000	http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
SOAP 1.2	W3C	Recommend 24 June 2003	http://www.w3.org/TR/soap12-part1/
UDDI 2.0	OASIS	Standard	http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2
UDDI 3.0.1	OASIS	Tech Committee Specification	http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3
WS-Addressing	W3C	W3C Input 10 Aug 2004	http://www-106.ibm.com/developerworks/library/specification/ws-add/
WS-AtomicTransactions	IBM, BEA, MSFT	Specification Sept 2003	http://www-106.ibm.com/developerworks/library/ws-atomtran/
WS-Attachments	IETF(IBM and MSFT)	Internal Draft 17 June 2002	http://www-106.ibm.com/developerworks/webservices/library/ws-attach.html

Specifications Title	Publisher	Status	
WS-Notification WS-BaseNotification WS-BrokeredNotification WS-Topics	IBM, Sonic Software, TIBCO Software, AKAMAI Technologies, SAP AG, Computer Associates International, Fujitsu, Laboratories of Europe, Globus, Hewlett-Packard	Initial Draft Specification 3 May 2004	http://www-106.ibm.com/developerworks/library/specification/ws-notification/
WS-BusinessActivity	IBM, BEA, MSFT	Initial Review Specification Jan 2004	http://www-106.ibm.com/developerworks/webservices/library/ws-busact/
WS-CAF WS-CTX WX-CF WS-TXM	Fujitsu, IONA, Oracle, Sun, Arjuna Technologies	OASIS Committee Draft Specification July 8 2003	http://developers.sun.com/techtopics/webservices/wscaf/
WS-Coordination 1.0	IBM, BEA, MSFT	Initial Draft Specification Sept 16 2003	http://www-106.ibm.com/developerworks/library/ws-coor/
WS-Eventing	IBM, BEA, Computer Associates, MSFT, SUNW, TIBCO Software	Public draft release Specification Aug 2004	http://www-106.ibm.com/developerworks/webservices/library/specification/ws-eventing/
WS-Experience Language (WSXL) 2.0	IBM	Specification 10 April 2002	http://www-106.ibm.com/developerworks/library/ws-wsxl/
WS-Federation Language WS-Federation:Active Requestor Profile WS-Federation:Passive Requestor Profile	IBM, BEA, MSFT, RSA, VeriSign	Initial Draft Specification 8 July 2003	See White paper - <i>Federation of Identities in a Web service world</i> , IBM & Microsoft http://www-106.ibm.com/developerworks/library/ws-fedworld/
WS-I Attachments Profile 1.0	WS-I	Final Material 25 Aug 2004	http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html

Specifications Title	Publisher	Status	
WS-I Basic Profile 1.0	WS-I	Final Material April 16 2004	http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html
WS-I Basic Profile 1.1	WS-I	Final Material 24 Aug 2004	http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html
WS-I Simple SOAP Binding Profile 1.0	WS-I	Final Material 24 Aug 2004	http://www.ws-i.org/Profiles/SimpleSoapBindingProfile-1.0-2004-08-24.html
WS-Inspection 1.0	IBM,MSFT	Initial Draft Specification November 2001	http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html
WS-Manageability 1.0 WS-Manageability - Concepts WS-Manageability-Representation	IBM,Talking Blocks, Computer Associates	OASIS Submission 10 Sept 2003	http://www-106.ibm.com/developerworks/webservices/library/ws-manage/
WS-MetadataExchange	IBM, BEA, MSFT, SAP AG, CA, Sun, webMethods	Initial Working Draft Specification Sept 2004	http://www-106.ibm.com/developerworks/library/specification/ws-mex/
WS-Notification WS-BaseNotification WS-BrokeredNotification WS-Topics	IBM, Sonic Software, TIBCO Software, AKAMAI Technologies, SAP AG, Computer Associates International, Fujitsu, Laboratories of Europe, Globus, Hewlett-Packard	Initial Draft Specification 3 May 2004	http://www-106.ibm.com/developerworks/library/specification/ws-notification/

Specifications Title	Publisher	Status	
WS-Policy WS-PolicyAssertions WS-PolicyAttachments WS-PolicyFramework WS-SecureConversation WS-SecurityPolicy	IBM, BEA, Computer Associates, Layer 7 Technologies, MSFT, Netegrity, Oblix, OpenNetwork Technologies, Ping Identity Corp, Reactivity, RSA Security, VeriSign, Westbridge Technology	Initial Draft Specification May - Sept 2004	http://www-106.ibm.com/developerworks/library/specification/ws-polfram/
WS-Provisioning	OASIS	Initial call for participation Oct 2001 Draft 0.7 Oct 17 2003	http://www-106.ibm.com/developerworks/library/ws-provis/
WS-Reliability	OASIS	Committee Draft 24 Aug 2004	http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrc
WS-ReliableMessaging	IBM, BEA, MSFT, TIBCO	Review Draft 4 march 2004	http://www-106.ibm.com/developerworks/webservices/library/ws-rm/
WS-Resource WS-ResourceLifetime WS-ResourceProperties WS-BaseFaults WS-ServiceGroup	IBM, Globus Alliance, Hewlett Packard	Specification	http://www-106.ibm.com/developerworks/library/ws-resource/
WSRP 1.0	OASIS	OASIS Approved August 2003	http://www-106.ibm.com/developerworks/webservices/library/ws-wsrp/ http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf
WS-Security 1.0 (WS-Security 2004)	OASIS	OASIS Approved March 2004	http://www-106.ibm.com/developerworks/webservices/library/ws-secure/ http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

Specifications Title	Publisher	Status	
WS-Security Kerberos Binding	IBM, MSFT	Initial public draft Dec 2003	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-security-kerberos.asp
WS-Transaction 1.0	IBM, BEA, MSFT	Draft review specification 9 Aug 2002	http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/
WS-Trust 1.1	IBM, BEA, Computer Associates, Layer 7 Technologies, MSFT, Netegrity Oblix, OpenNetwork Technologies, Ping Identity Corp, Reactivity, RSA Security, VeriSign, Westbridge Technology	Initial draft specification May 2004	http://www-106.ibm.com/developerworks/library/specification/ws-trust/
WSDL (Web Services Description Language) 1.1	W3C (IBM and MSFT)	Note 15 Mar 2001	http://www.w3.org/TR/2001/NOTE-wsd1-20010315
WSDL (Web Services Description Language) 2.0	W3C (IBM and MSFT)	Working Draft 3 Aug 2004	http://www.w3.org/TR/2004/WD-wsd120-20040803/

7.3 Summary of the Web services architecture stack

Figure 7-2 shows a classification of the WS-Specifications of which IBM is a co-contributor. It omits the specifications from Table 7-2 that IBM has not contributed to.

The major IBM and Microsoft sources for WS- specifications are to be found at the developerWorks topic “*Web services standards*”:

<http://www-128.ibm.com/developerworks/views/webservices/standards.jsp>

and the MSDN topic, “*Web services specifications*”:

<http://msdn.microsoft.com/webservices/understanding/specs/default.aspx>

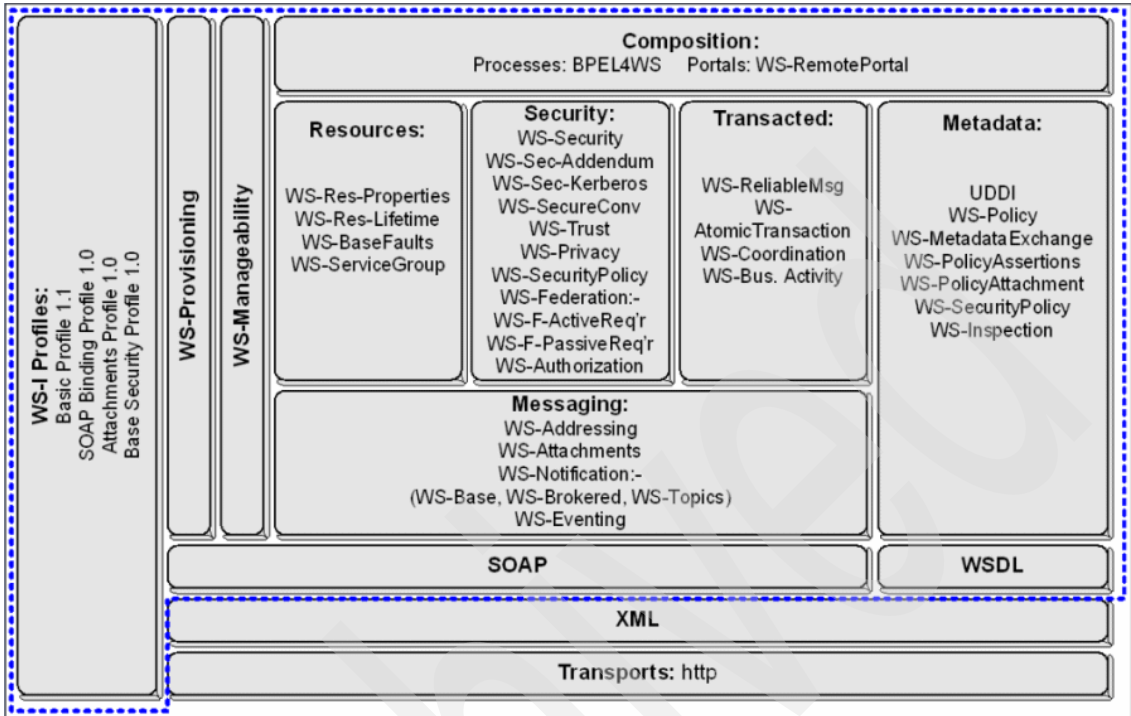


Figure 7-2 IBM Web services architecture stack

The dotted boundary encloses Web services specifications.

7.3.1 Foundations

The foundations of Web services architecture are the transport protocols and XML and XML schemas.

Transport protocols

Http/Https version 1.1 is the synchronous application-level protocol that is most commonly used with SOAP.

Java Messaging Service (JMS) is a reliable asynchronous alternative to Http:, and because of the widespread use of WebSphere MQSeries in large enterprises, is a viable choice for interoperable Web services. Its use is discussed further in 8.8, "SOAP/JMS and SOAP/MQ" on page 188. SOAP/JMS is not included in the Web service stack although it has been implemented by a number of vendors; there is as yet no WS-* specification for SOAP/JMS.

SOAP-over-UDP has been proposed by Microsoft, Lexmark, BEA and Ricoh. Since IBM is not a joint proposer it, too, has been omitted from the stack.

For a pointer to the levels of http transport protocols, see the references in the *WS-I Basic Profile 1.1*, found at:

<http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html#references>

XML

XML 1.0 is the current version referenced by the WS-I Basic profile 1.1. XML 1.1 is a recommendation of W3C.

There are two parts to the XML Schema Definitions (XSDs): structures and datatypes. Particularly important for interoperability are the datatypes that are supported by Web services. These datatypes, and their usage, enable the mapping (or serialization) of hardware and language specific datatypes to a SOAP message and vice versa.

There are two major issues to consider in serialization: the equivalence of language-specific type mappings and any ambiguity in vendors choosing different type mappings; this we found to be the case with the way arrays were described in XML by WebSphere and Microsoft .Net. In the tables of type mappings, Table 7-3 to Table 7-5, not all the types defined in the SOAP encoding schema are clearly defined by Java or Microsoft .Net mappings.

There are also differences in usages of nillable and minOccurs in the specification of arrays; these are dealt with by WS-I, but sometimes only by a “recommended” rather than a “mandatory” usage definition.

These differences are not insurmountable, but one has to understand the implications when using development tools that may be optimized to the vendor’s own type mapping defaults; also, generated code may need some fine-tuning to work as expected.

The data type mapping specifications for Java are documented in JSR 101 1.0 and 1.1. The column for Microsoft .Net mappings is taken from *Interoperability Fundamentals, MSDN December 2003*:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/jdni_ch03.asp

There are some “?”s in the tables, where the Microsoft .Net type was inferred where it couldn’t be found in the Microsoft .Net documentation.

Table 7-3 JSR 101 1.0 and Microsoft .Net mappings for built-in XML data types

Simple Type	Java Type	Microsoft .Net Type
xsd:string	java.lang.String	String
xsd:integer	java.math.BigInteger	Int64(?)
xsd:int	int	Int32
xsd:long	long	Int64
xsd:short	short	Int16
xsd:decimal	java.math.BigDecimal	Decimal
xsd:float	float	Single
xsd:double	double	Double
xsd:boolean	boolean	Boolean
xsd:byte	byte	SByte
xsd:QName	javax.xml.namespace.QName	String (?)
xsd:dateTime	java.util.Calendar	DateTime
xsd:base64Binary	byte[]	Byte(Array)
xsd:hexBinary	byte[]	Byte(Array)

JSR 101 also defines the rules for mapping arrays and complex structures, and provides a tables of all the other types defined in the XML schema that are derived in Java from the basic types. Examples includes types such as xsd:gYear and xsd:unsignedLong.

In addition, the following rules are observed. If the XML type is defined to be nillable, then the XML type is mapped to the Java wrapper class for the primitive type. So, for example, int becomes java.lang.Integer.

Table 7-4 Additional SR 101 1.1 and Microsoft .Net mappings for built in XML data types

Simple Type	Java Type	Microsoft .Net Type
xsd:date	java.util.Calendar	DateTime (?)
xsd:time	java.util.Calendar	DateTime (?)
anyURI	java.net.URI (J2SE 1.4 only) java.lang.String	System.Uri
anySimpleType	java.lang.String	String (?)

Table 7-5 Additional Microsoft .Net and derived JSR 101 1.1 mappings

Simple Type	Java Type	Microsoft .Net Type
xsd:negativeInteger	java.math.BigInteger	System.Decimal
xsd:nonNegativeInteger	java.math.BigInteger	System.Decimal
xsd:nonPositiveInteger	java.math.BigInteger	System.Decimal
xsd:unsignedInt	(?)	UInt32
xsd:positiveInteger	java.math.BigInteger	System.Decimal (?)
xsd:unsignedLong	java.math.BigInteger	UInt32 or UInt64 (?)

Microsoft .Net uses the `System.Xml.Serialization.XmlSerializer` class to map between XML types and the Common Language Runtime (CLR).

SOAP

SOAP 1.1 is the current specification. See Chapter 2, "SOAP primer" on page 11. SOAP 1.2 is a W3C recommendation.

7.3.2 Messaging

Messages carry information to and from a Web service. The messages specify which operations to carry out in a Web service and provide data for the operations.

In our stack, the messaging layer specifications also include definition of the interaction model (point-to-point, publish-subscribe, broadcast, request-response, one-way, asynchronous) and how attachments are transferred (binary or formatted, inline or out-of-band by reference).

Currently, WS-I have restricted their consideration of interaction styles to (see Figure 7-3)

1. One-way
2. Request-response

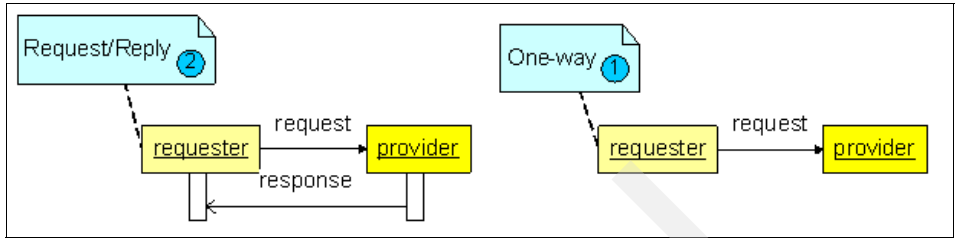


Figure 7-3 WS-I basic interaction models

WS-I have not yet dealt with more complex styles of interaction familiar to MOM architects such as in Figure 7-4.

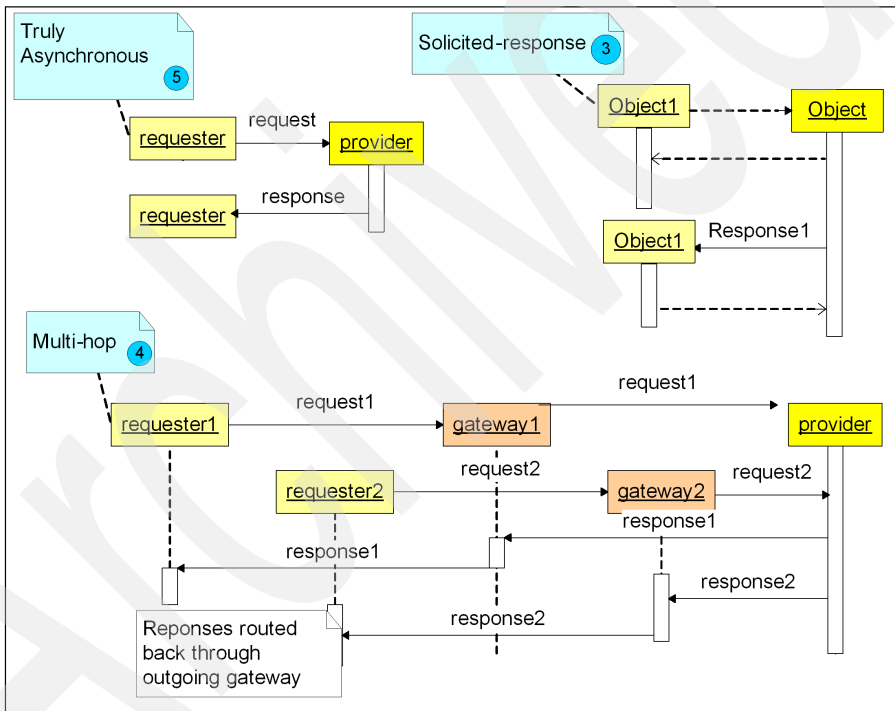


Figure 7-4 Examples of more complex point-to-point interaction styles

3. Callback (A message is solicited by the invoker, either as a separate callback)
4. Multi-hop (such as through a gateway or an ESB)

In the multi-hop example in Figure 7-4, the requests to a single provider are routed through different gateways (by an load balancing scheme perhaps). In this example, the gateway needs to be able to insert return routing

information so that the service provider returns the response through the same gateway that fielded the request, and the response ends up at the original requester.

5. Truly asynchronous response

Also in Figure 7-4, the truly asynchronous model does not return a response to the request (and so can be implemented without a continuous connection between the requester and provider) until it is ready to do so.

The requester lifeline may be interrupted, and the requester would have to be restarted to respond to the request. Additional routing information is needed so the response can be correlated to the original requester.

(6,7 and 8 are not illustrated by a figure)

6. Treating the response and fault message differently

7. Interactions with affinity, where the client addresses subsequent messages to a service provider that is allocated dynamically on an earlier interaction

8. WS-I has also not tackled publish-subscribe (notification or event based interaction styles)

WS-Addressing

The WS-Addressing standard proposed in 2003 introduces two new SOAP concepts, Endpoint References (EPR) and Message Information Headers (MI) to architect more complex interaction styles. The specification is defined in a way that allows dynamically generated location references to be opaque to the requester. WS-Addressing could be used to implement the interaction styles in Figure 7-4. WS-Addressing is described in more detail in 8.3, “Interoperability standards: addressing” on page 155.

Note: For an interesting article on the subtleties of implementing WS-Addressing fully, see *The hidden impact of WS-Addressing on SOAP*, Doug Davis, in IBM developerWorks, July 2004 at:

<http://www-106.ibm.com/developerworks/webservices/library/ws-address.html>

WS-Notification

The WS-Notification standards proposals are based on a topic model of publish-subscribe, and include the WS-Topic specification to describe the topic space, WS-Base to describe interactions between producers and consumers, and WS-Brokered which enables a publish-subscribe interaction style on behalf of endpoints that do not themselves publish or subscribe messages.

WS-Eventing

The WS-Eventing standards proposal involves just producers and consumers. A subscription sent by a consumer to a producer includes a filter expression to determine whether a particular event is propagated to a particular consumer.

WS-Eventing is inherently a point-to-point topology and very suitable for use in applications such as device control, whereas WS-Notification has a concept of a topic space which enables centralized access management to topics and is suited for distributing business information.

WS-Attachments

Although WS-I have finalized their SOAP attachments profile using existing facilities in SOAP and standard MIME mechanisms to carry and reference attachments, it looks unlikely to be adopted. For this reason, we haven't worked through any practical examples of using attachments in this redbook.

Attachments have presented SOAP architects with a number of problems:

1. The application inline binary attachment data within the SOAP envelope using base64binary or hexBinary datatypes.

Example 7-1 Inline SOAP Binary

```
<SOAP:Envelope ... >
...
<Picture>FF0AB013</Picture>
...
</SOAP:Envelope>
```

This results in up to a four-fold expansion of the binary data and a processing overhead as the binary data has to be converted to and from a "character" representation to conform to XML's UTF-8 or UTF-16 encoding specifications.

Nonetheless, this is an extremely attractive way to exchange binary data as you are well assured of interoperability and it is entirely compatible with the rest of the SOAP architecture.

2. Placing the binary attachment out of line, as a part of a larger structure outside the SOAP envelope, as in a Multipart message (MIME), leads to two major difficulties:
 - a. How to address the binary parts? Schemes based on using identifying tokens (SOAP with Attachments (SwA), see Example 7-2) and on offsets, DIME (Direct Internet Messaging Encapsulation) have been proposed.

```
<SOAP:Envelope ... >
...
<Picture>cid:peter@images.itso.ibm.com</Picture>
...
</SOAP:Envelope>
--MIME_boundary
...
Content-ID: <peter@images.itso.ibm.com>
FF0AB013
--MIME_boundary
```

For an introduction to DIME and how it includes attachments, start with “Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation”, by Jeannine Hall Gailey, found at <http://msdn.microsoft.com/msdnmag/issues/02/12/DIME/default.aspx>

- b. How to apply a uniform processing model efficiently to the SOAP message and all its binary parts to deal with issues such as security.

What is needed is an attachment with the appearance that it is inline (in the SOAP envelope), so that existing SOAP processes handle the binary data just like anything else that appears in the SOAP envelope, but with the efficiency of attaching the binary outside the SOAP attachment so that it can be transmitted as raw binary rather than nibbled in some way.

The latest favored solution is Message Transmission Optimization Mechanism (MTOM). The specification is undergoing rapid development with W3C and will probably reach final recommendation by 2005. MTOM makes use of an XML mechanism, XML Binary Optimized Package (XOP) to serialize the SOAP envelope. From the point of view of software accessing the XML in the SOAP message, the attachment appears as a base64Binary type. From the transmission perspective, the raw binaries are placed into attachments. The XML layers manage the translation between the wire format and the XML infoset.

To get the value of this solution, the XML layers must understand XOP encoding, rather than passing the transmission format to the application. This requires changes in the XML layers of the SOAP stack, but no changes to higher levels of the stack. So for example, when calculating a digital signature to sign a SOAP message, the security layer would calculate its hash on the base64Binary representation of the binary data passed to the application by the XML parser. It would not need code to go and locate the binary data in the transmission format of the SOAP message. So the MTOM solution is seen as providing SOAP processors with the appearance of an inline attachment, and yet the SOAP

message contains the attachment outside the SOAP envelope, avoiding the overhead of encoding the binary data.

7.3.3 Security

There are a number of Web service security specifications that provide mechanisms not only to share information securely, but also to manage a security infrastructure.

1. The joint IBM and Microsoft white paper, *Security in a Web Services World: A Proposed Architecture and Roadmap*, found at <http://www-128.ibm.com/developerworks/webservices/library/ws-secmap/> is a good place to go to read more about Web service security specifications.
2. Section 8.4, “Security” on page 158 in this redbook is a detailed explanation of the WS-Security 2004 specification

This section gives a brief description of the content of all the WS-* security specifications.

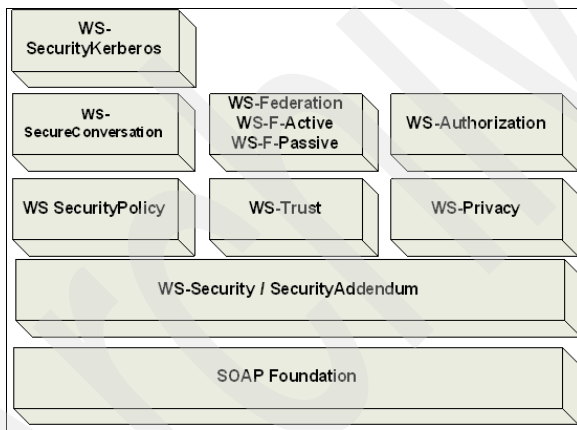


Figure 7-5 Web service security specifications

WS-Security

Web services security specification 1.0 (WS Security 2004) addresses three main security mechanisms to exchange information securely:

1. Sending security tokens as part of a message - enables authentication
2. Message integrity - digital signature
3. Message confidentiality - encryption

WS-SecurityAddendum

The SecurityAddendum corrects and clarifies the original WS-Security specification.

WS-Trust

WS-Trust Language is a specification for establishing that the security tokens exchanged in WS Security can be trusted. A simple example of the use of WS-Trust would be to use it to require that a requester provides a security token that proves that the request could only have been initiated by the requester. One way for the requester to demonstrate this would be by using a digital signature generated with a unique private key, the authenticity of which can be verified by a third party certification agency.

WS-SecurityPolicy

WS-SecurityPolicy uses the WS-Policy generalized specification language for describing the properties required of Web services as policies. An example of using WS-SecurityPolicy would be to require “exactly one type of security token is acceptable,” and that token “must be a Kerberos token.” Another example would be to specify what set of encryption algorithms are acceptable. A further example of a security policy would be to make an assertion about how old a security token may be before it is invalidated.

WS-Privacy

WS-Privacy is a proposal for a specification for organizations creating, managing, and using Web services to state their privacy policies and to require that incoming requests make claims about the senders' adherence to these policies. There is no proposed standard as yet. WS-Privacy is closely related to Enterprise Privacy Authorization Language (EPAL 1.1) (<http://www.zurich.ibm.com/security/enterprise-privacy/epal/>) which is the basis for monitoring privacy in the use of Web services in emerging technology; for example, see *Declarative Privacy Monitoring for Tivoli Privacy Manager*, found at:

<http://www.alphaworks.ibm.com/tech/dpm>

WS-Federation

WS-Federation specifies how to manage and broker trust relationships in a federated environment. It enables a user's credentials to be established indirectly and shared in a number of well defined trust relationships. WS-Federation is used in implementations of federated identity management.

WS-SecureConversation

WS-SecureConversation defines how a security context is established and managed to enable multiple messages to be sent securely. This specification plugs a loophole in WS-Security which opens up when more than one message is exchanged. An eavesdropper could record and replay a secure message undetected. The second message would have the same credentials as the first and not be detected as a spoofed message. So, for example, a deposit into a bank account might be repeated. The WS-SecureConversation specification secures the session so that messages cannot be replayed.

WS-SecurityKerberos

WS-SecurityKerberos builds on WS-Security, WS-Trust and WS-SecureConversation to specify how Kerberos is used to secure Web services

WS-Authorization

WS-Authorization will describe how access policies for a Web service are to be specified and managed. No specification has been published for WS-Authorization. The concepts are likely to be similar to the OASIS *eXtensible Access Control Markup Language (XACML)*:

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

Summary

Figure 7-6 shows some of the relationships between the security specifications.

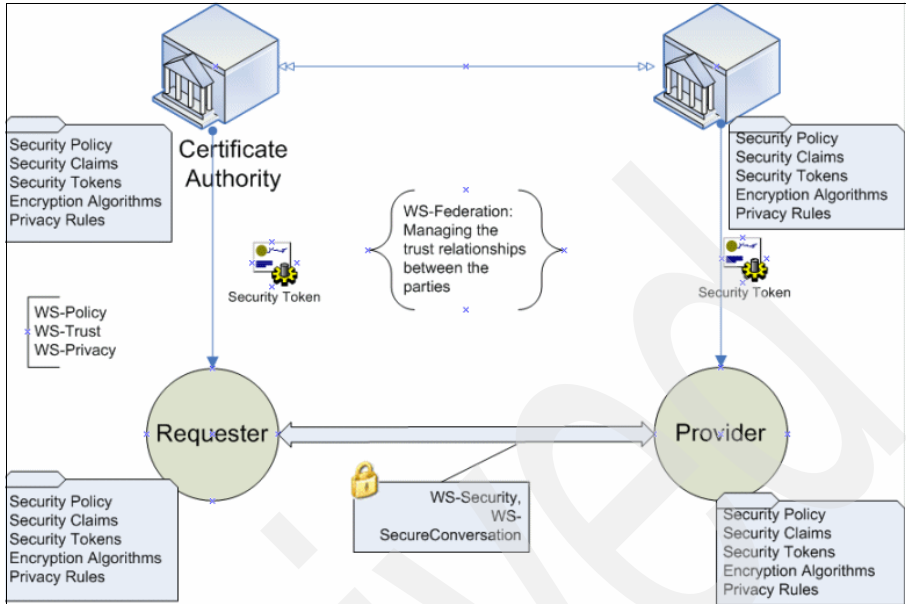


Figure 7-6 Usage of Web services security specifications

7.3.4 Transacted

There are Web services specifications for reliable messaging and coordinating and participating in transactions. These specifications enable transactional interactions between Web services running on different platforms. As with other Web services, the implementation is left for the platform vendor to decide. The specifications define the services and protocol flows needed to make Web services transactional.

The transactional specifications (but not reliable messaging) are based on the OASIS Business Transaction technical Committee specification *Business Transaction Protocol 1.0* announced in May 2002. The standard is available from http://www.oasis-open.org/committees/download.php/1184/2002-06-03.BTP_cttee_spec_1.0.pdf

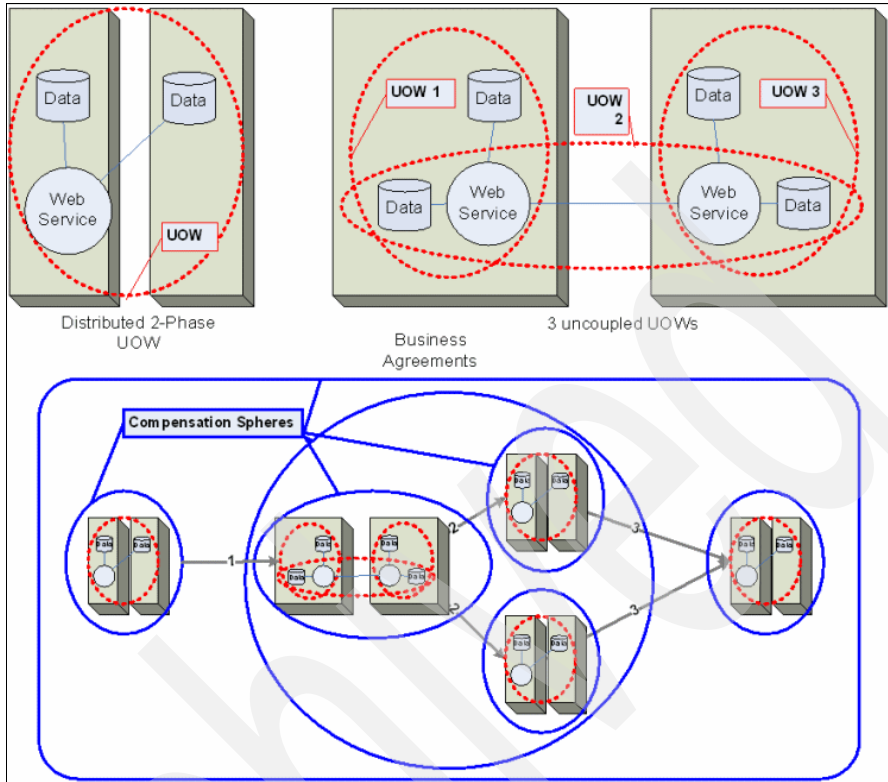


Figure 7-7 Transaction models

There are four Web service transactional specifications being proposed by IBM, BEA and Microsoft which enable the standard transactional usages shown in Figure 7-7:

- ▶ WS-AtomicTransaction
- ▶ WS-Business Agreement
- ▶ WS-Coordination
- ▶ WS-ReliableMessaging

There are alternative Web service transactional specifications being proposed to OASIS by Arjuna Technologies Limited, Fujitsu Software, IONA Technologies PLC, Oracle Corporation, and Sun Microsystems called the WS-Composite Application Framework (WS-CAF). There is also a reliable messaging proposal before OASIS, called WS-Reliability, edited by Fujitsu Software, Oracle Corporation, Sun Microsystems and Novell Inc. We will only look at the BEA, Microsoft and IBM proposals for transactions and reliable messaging.

WS-AtomicTransaction

WS-AT, or Atomic transactions are the familiar ACID transactions (Figure 7-8) found in CICS, J2EE and Microsoft Transaction Services (MTS).

Atomicity: A transaction is an atomic unit of processing; it is either performed in its entirety or not at all.	Isolation: A transaction should not make its updates visible to other transactions until it is committed. It should run as if no other transaction is running.
Consistency: A correct execution of the transaction must take the system from one consistent state to another.	Durability: Once a transaction commits, the committed changes must never be lost in the event of any failure.

Figure 7-8 ACID transaction properties

Section 8.6, “WS-Transactions” on page 179 has more details on WS-AtomicTransactions. A technical preview of WS_AT is available for WebSphere Application Server 5.0.2 at:

<http://www.alphaWorks.ibm.com/tech/wsat>

WS-BusinessAgreement

The Business Agreement protocol is a proposed specification for implementing long running transactions that do not behave in an ACID manner. Workflow and business processes are examples of long running transactions.

WS-Coordination

WS-Coordination defines the Web services protocols to enable multiple transaction coordinators to work together - for instance, to hand off coordination of resources on another platform to another coordinator, rather than having all resources managed by a single coordinator.

WS_Coordination describes a coordination service that includes an activation service to initiate a new coordinator, a registration service to register resources with the new coordinator, and a coordination protocol service to implement a variety of different transaction protocols.

Section 8.5, “WS-Coordination” on page 177 has more details about WS-Coordination.

WS-ReliableMessaging

WS-RM (Reliable Messaging) has been demonstrated by IBM and Microsoft in October 2003, and is available as an Emerging Technologies Toolkit (ETTK) from IBM alphaWorks®. WS-RM is a reliable messaging protocol, with similar messaging qualities of service supported by IBM WebSphere MQSeries. It differs

from WebSphere MQSeries in being a protocol specification that vendors are free to implement however they wish.

Section 8.7, “Reliable messaging” on page 182 has more details about WS-ReliableMessaging.

7.3.5 Meta-data

The meta-data specifications, sometimes called the description specifications, are concerned with providing a standard way to describe Web services to aid in the discovery of Web services, understanding their interfaces, and describing properties of the services.

WSDL

WSDL Version 1.1 is current and Version 2.0 is a W3C last call. Chapter 3, “WSDL primer” on page 27 gives an overview of WSDL.

UDDI

UDDI (Universal Description, Discovery and Integration) is a specification and a consortium. The specification is now into its third release, and the consortium has moved on from the dot.com hyperbole that saw UDDI replacing traditional supplier relationships with a cyberspace trading floor. The UDDI consortium has adopted an evolutionary approach that seeks to provide a means for users to control their trading relationships electronically. UDDI is described in Chapter 10, “Deploying Web services” on page 215.

WS-Policy

WS-Policy provides a way to express the requirements, capabilities and characteristics of a Web service. A policy is a collection of alternatives expressed as PolicyAssertions. WS-Policy provides a grammar for choosing between the policy alternatives by using quantifiers such as “Exactly One” of a collection of policy assertions. One example would be to specify Kerberos as the only acceptable provider of security tokens. WS-Policy doesn’t provide a means of expressing the priority of one policy over another.

WS-PolicyAssertions

This specifies what policy assertions may be made about a Web service. There are four types of policy assertion:

1. Text encoding
2. Natural language usage
3. Specification version
4. Message predicate - typically expressed using XPath

WS-PolicyAttachment

WS-PolicyAttachment specifies how a policy assertion is associated with a Web service. It provides for two alternative means of attachment: directly, termed intrinsic to a service, and indirectly or externally from the definition of a Web service. When all the policies that are associated with a Web service have been resolved, the resulting policy is termed the effective policy.

WS-Inspection

Web services Inspection Language (WSIL) pulls together references to different kinds of service descriptions for a Web service. The two goals of WSIL are to make the WSIL document easy for:

1. Clients to select only those service descriptions they will understand
2. Service providers to maintain by making it a collection of references, and not the descriptions themselves.

Figure 7-9, taken from Peter Brittenham, *An overview of the Web Services Inspection Language*, found at, <http://www-106.ibm.com/developerworks/webservices/library/ws-wsilo1> illustrates how WSIL documents relate to UDDI and other service descriptions.

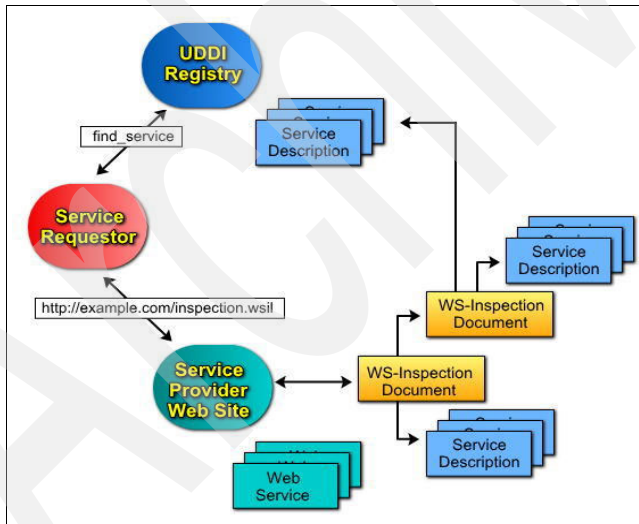


Figure 7-9 WSIL and UDDI

WSIL documents form a hierarchy of descriptions like HTML pages. By using naming conventions for the pages service providers, you can simplify the task of searching and browsing WSIL documents.²

The main historical difference between WSIL and UDDI has been that UDDI is centrally maintained, whereas WSIL is not. The central maintenance of UDDI has been perceived as one of its difficulties, as the entries in UDDI are thought to reflect a lack of control of their content and authenticity. There are estimates that two-thirds of UDDI entries are invalid, either dead links or badly formatted³.

By going to the Web site of the service provider themselves, the proponents of WSIL suggest authenticity and quality issues could be addressed by the providers themselves.

However, UDDI is evolving to meet these business needs. Version 3 of the UDDI specification includes the capability for providers to digitally sign entries as proof of their authenticity, and commentators see UDDI and WSIL as complementary rather than competing technologies.

WS-MetadataExchange

This specifies two request-response protocols to retrieve meta-data about a Web service. The two forms allow a request for a specific type of meta-data, for example WS-Policy information, or to retrieve all the meta-data about a Web service. This protocol is not intended to be a general purpose query mechanism.

7.3.6 Resources

The WS-Resources set of specifications were proposed by IBM, Globus, HP and Fujitsu in May 2004. They define how to specify relationships between Web services and stateful resources such as business entities like a business agreement, or an IT entity - for example, the availability of a processor in a GRID. A major motivation behind WS-Resources is the desire for an open GRID architecture based on Web services. Resources, such as processors, are a central concern of GRID computing. The WS-Resources specifications define how to use Web service messages to express life cycle operations (Create, Update, Delete), to address specific instances of resources using WS-Addressing, and to notify interested parties when resources change state, using WS-Notification.

² In "WSIL: Do we need another Web Services Specification?", by Tarak Modi, found at, <http://www.webservicesarchitect.com/content/articles/modi01.asp> Tarak Modi argues that WSIL could complement standard search engines like Google by providing a standard searchable format of Web service descriptions.

³ SalCentral, a WSDL search engine, found at, <http://www.salcentral.com/Search.aspx>

7.3.7 Composition

There are two Web service interfaces for composition of applications, process composition using BPEL4WS, and integration “on the glass” using portals, WS-RP.

BPEL4WS

The BPEL4WS 1.1 is a specification for describing protocols between business entities (called partners) using Web service messages.

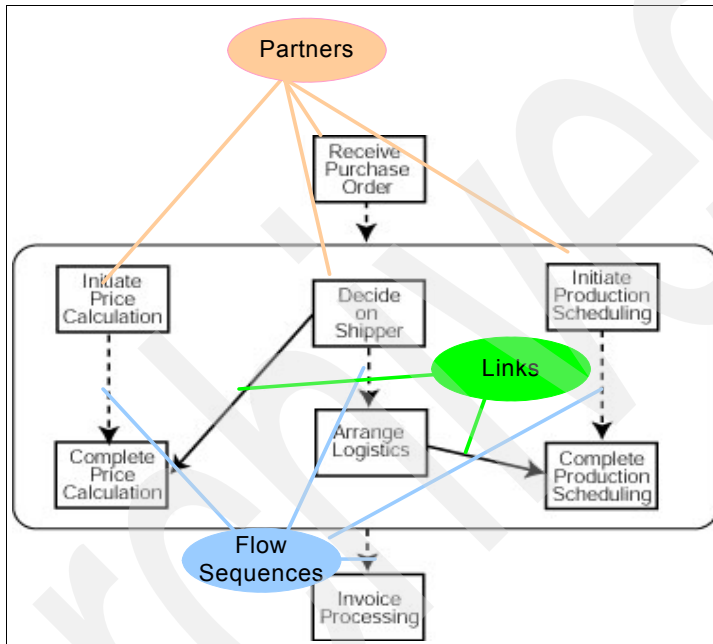


Figure 7-10 Business process example from BPEL4WS specification

The basic concepts of BPEL4WS can be applied in either of two ways:

1. As an abstract process that defines the public relationship between the partners. Only data that is relevant to the partner relationship is exposed as message properties. Operation of the private processes in each of the partners is opaque and is modelled as a non-deterministic choice between outcomes.
2. As an executable process. The logic and state of the process determine the nature and sequence of the interactions at each business partner. The interactions are ideally implemented as Web services. Practically speaking, however, parts of the implementation will use private platform dependent functionality. The platform dependent interfaces can be modelled as Web

services to maintain the integrity of the process definition and to connect the legacy application to the business process.

BPEL4WS includes the ability to deal with:

- ▶ Data-dependent behavior.
- ▶ Exceptional conditions and their recovery using compensations.
- ▶ Long running interactions involving nested units of work and requiring cross partner coordination.

BPEL4WS does not explicitly deal with:

- ▶ Data transformation or translation.
- ▶ Human workflow.
- ▶ Existing B2B concepts such as Trading Partner Agreements and protocols like Resultant.

WS-RP

WS-Remote Portlets 1.0 was approved by OASIS in August 2003.

Portal producers publish applications using a Web service interface that encapsulates an application and the user interactions as a portal. Portal consumers aggregate portals from different producers and present them to end users through a browser page. Figure 7-11 shows a weather portal and a human resources (HR) portal being combined into a single employee portal.

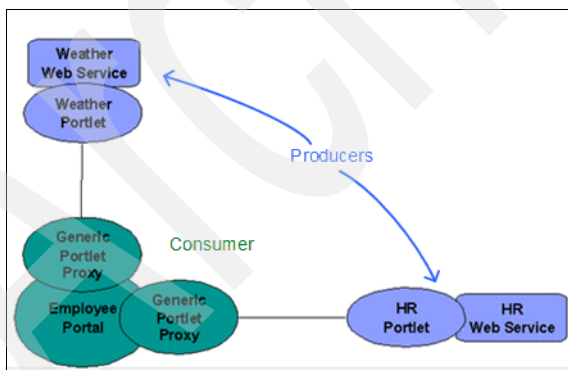


Figure 7-11 Combining remote portlets

The advantage of remote portlets is that specific portals do not have to be produced for each Web service. Remote portlets can be dynamically added to the user's environment.

The addition of Web services to remote portlets adds the capability of discovering portlets through UDDI, as well as all the other benefits of using Web services. WS-RP defines a set of attributes to define remote portlets to UDDI to understand the capabilities of the portlet.

WS-RP defines four Web service interfaces for portal producers to aggregate applications and present them as remote Web service portal interfaces to portal consumers.

1. Service Description
Obtains the producer's meta-data about a portlet
2. Markup
Returns the markup and processes users' interactions with the portlet
3. Registration
An optional interface to set up a particular relationship or session between a consumer and producer
4. Management
Another optional interface to get further meta-data, and to customize a portlet

7.3.8 Management

The WS-Management specifications were submitted to the OASIS Web services Distributed Management technical committee in 2002 by IBM, Talking Blocks and CA. The specification is sometimes known by its acronym, MUWS, Management Using Web Services. WS-Manageability Concepts defines manageability requirements and patterns. WS-Manageability Specification defines the model of a manageable Web service endpoint, and WS-Manageability Representation the XML representation.

WS-Manageability concepts

The manageability goals are:

1. Web service infrastructures to provide:
 - a. Standard metrics
 - b. Management operations including configuration control and lifecycle
 - c. Base set of management events
 - d. A standard way to access management capabilities of a Web service infrastructure
2. Web services to:
 - a. Expose Web service specific metrics, configuration, operations and events
 - b. Support discovery of its management capabilities

- c. Define a standard way to access the management capabilities of a Web service

There are two management patterns.

1. Consolidated Interfaces

Business functionality and manageability are offered by a Web service in a single description of the service. The drawback of this simple pattern is that all the management capabilities are exposed to a user who is only interested in the business capability.

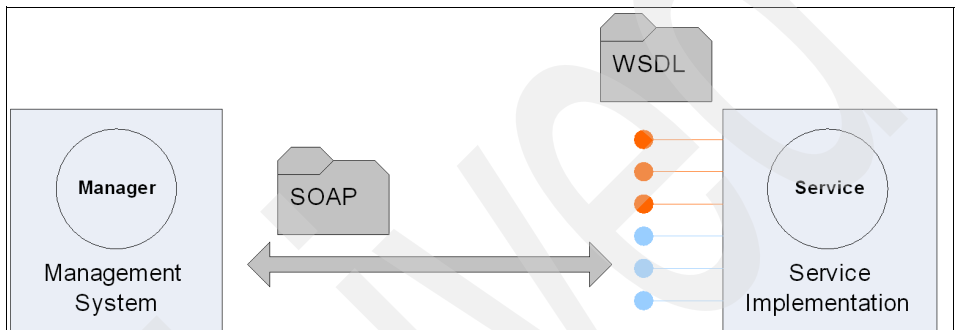


Figure 7-12 Consolidated Interfaces Management Pattern

2. Associated Interfaces

Manageability is offered separately to the business capability of a service. It is more complex to implement, deploy and use for the managers of the service, but it is much simpler for the developers and clients of the service's business capability. Two WSDL descriptions are needed for the business and manageability interfaces, as well as a referencing mechanism to associate the two service endpoints.

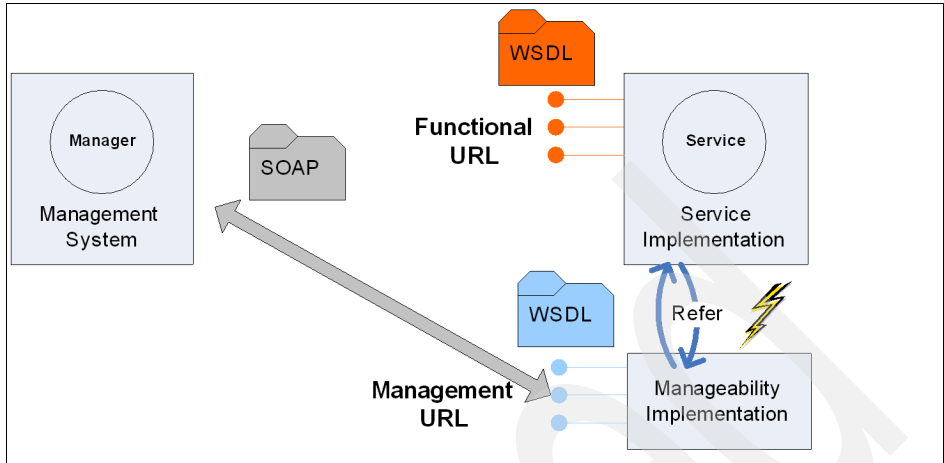


Figure 7-13 Associated Interfaces Management Pattern

WS-Manageability Specification

WS-Manageability Specification is a UML model of a manageable Web services endpoint. The model has the capabilities needed for implementations to meet the overall manageability goals stated earlier. In the model, a manageable endpoint has a number of management topics (Identification, State, Configuration, Metrics and Relationships) which are modelled as a number of different aspects (Properties, Operations and Events).

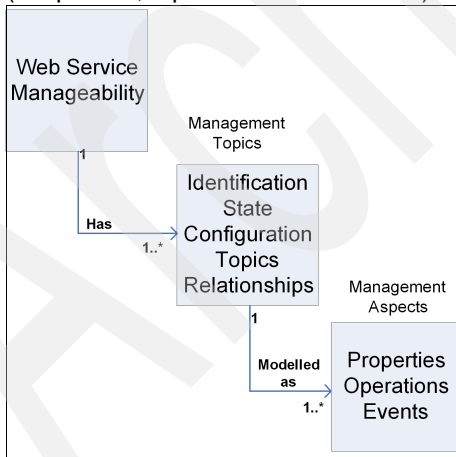


Figure 7-14 Management endpoint

The WS-Manageability representation

WS-manageability representation provides the XML and WSDL syntax for the WS-Manageability specification.

7.3.9 Provisioning

The WS-Provisioning standard has been proposed by IBM to OASIS. The specification defines protocols and operations to be used to manage computer resources. The resources, such as accounts, software executables and configuration files, are described using a provisioning markup language, such as Service Provisioning Markup Language (SPML) that is now an OASIS standard.

7.3.10 WS-I

WS-I basic profile attempts to integrate different referenced specifications from different layers of the standards stack to achieve greater interoperability among vendors of Web services.

WS-I Basic Profile 1.0

Version 1.0. provides guidelines of interoperability for XML, XML Schema, SOAP, WSDL, and UDDI.

WS-I Basic Profile 1.1

Version 1.1. WS-I Basic Profile 1.0 has been upgraded to WS-I Basic Profile 1.1 by relocating the SOAP Binding into a separate profile, Simple SOAP Binding 1.0 profile. WS-I Basic Profile 1.1 has also included corrections of the Basic Profile 1.0 Errata. It is now a final specification. The basic profile is discussed in depth in 8.2, “WS-I Basic Profile 1.0” on page 146.

WS-I Attachments Profile

Version 1.0 is a final specification. We have not discussed attachments in this document because it is an area that is still undergoing significant change.

WS-I Basic Security Profile

Version 1.0 is being drafted. There are details in 8.4.3, “WS-I Security Profile” on page 170.

7.4 Summary

We have listed the available and emerging standards for Web services in a table with their sponsors and current status. These will be changed as some standards will become specifications in the future and new standards will come out. We grouped some of the standards into Web service stack layers and briefly explained them.

Archived



Web service specifications

In this chapter, we discuss a few of the Web service standards in more detail, focusing on those that are finalized and those that are important for delivering a robust Web services implementation. We begin by introducing the WS-I organization and describing the profiles it has defined.

8.1 Web service Interoperability Organization (WS-I)

WS-I is an organization that integrates the specifications from different standards organizations and different vendor groups. It does not write the specifications, does not guarantee interoperability and does not relax standards. It provides advice, guidance, requirements, best practices, sample applications, use cases, usage scenarios, restrictions, education and tools for testing conformance ; see Figure 8-1.

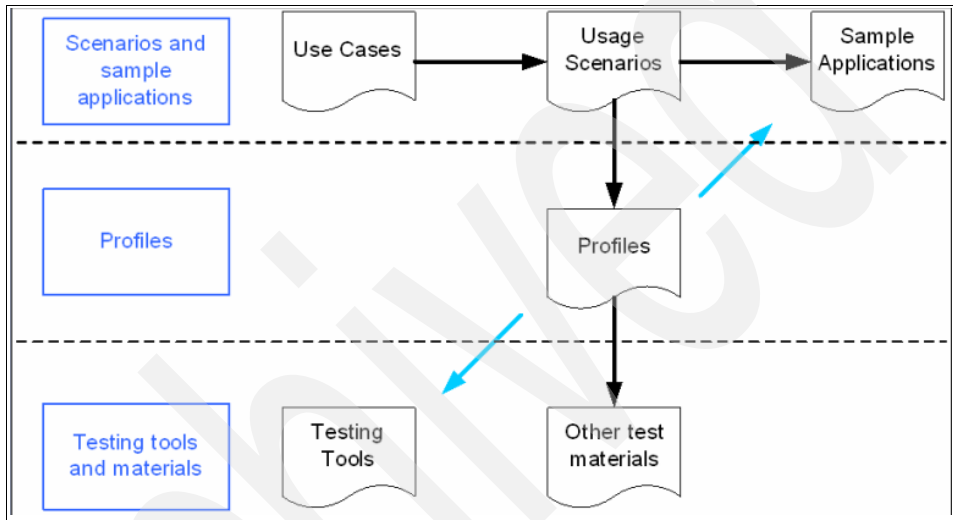


Figure 8-1 Relationship of WS-I deliverables

Its goal is to achieve Web service interoperability among different vendors, despite the fact that different vendors are free to implement Web service toolkits differently by extending their functionalities.

WS-I references a set of standards which are listed in Appendix I of the Basic Profile 1.1. It only addresses issues related to the set of standards. These standards allow extensions (Appendix II of Basic Profile 1.0) which may result in interoperability problems. WS-I does not limit the open-ended extensions of these standards, but other profiles may place restrictions on these extensions.

8.2 WS-I Basic Profile 1.0

The Profile is a set of named and versioned specifications which guide the implementation of the Web service so that it is more likely to be interoperable.

The Profile requires conformance to a set of requirements for specifying the XML tags in WSDL, SOAP, or UDDI to achieve interoperability.

The profile makes requirements statements about three kinds of artifacts: MESSAGE, DESCRIPTION and REGDATA.

- ▶ *MESSAGE* - protocol elements that are exchanged (usually over a network) to effect a Web service
- ▶ *DESCRIPTION* - descriptions of types, messages, interfaces and their concrete protocol and data format bindings, and the network access points associated with Web services (for instance, WSDL descriptions)
- ▶ *REGDATA* - registry elements that are involved in the registration and discovery of Web services (for instance, UDDI models)

In addition, it states measures of conformance of the service instances, service consumers and registries, by examining if the artifacts they produce and consume are conformant.

Service instance/consumer must produce only conformant artifacts and must be capable of consuming all conformant artifacts where multiple artifacts are possible. Service instances and consumers must comply with requirements for both sending and receiving Web service messages, as appropriate. That is, software that implements both a service instance and consumer must be conformant in both respects, or neither; it cannot be just a conformant service instance or a conformant consumer.

Service Instances must provide a WSDL 1.1 service description or a UDDI binding template, or both, to a consumer to be conformant. (The WSDL may be made available “out-of-band”). It is not necessary to register a Web service in a UDDI registry to be conformant.

Basic Profile 1.0 refers to the following standards (Appendix I of Basic Profile 1.0 specification) at the specified version levels:

Table 8-1 WS-I Basic Profile 1.0 base specifications

Specification	URL
Simple Object Access Protocol (SOAP) 1.1	http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
Extensible Markup Language (XML) 1.0 Second Edition	http://www.w3.org/TR/REC-xml
RFC2616: Hypertext Transfer Protocol - HTTP/1.1	http://www.ietf.org/rfc/rfc2616

Specification	URL
RFC2965:HTTP State Management Mechanism	http://www.ietf.org/rfc/rfc2965
WSDL 1.1	http://www.w3.org/TR/wsd1.html
XML Schema Part 1: Structures	http://www.w3.org/TR/xmlschema-1
XML Schema Part 2: Datatypes	http://www.w3.org/TR/xmlschema-2
UDDI Version 2.04 API Specification	http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm
UDDI Version 2.03 Data Structure Reference	http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm
UDDI Version 2 XML Schema	http://uddi.org/schema/uddi_v2.xsd
RFC2818: HTTP Over TLS	http://www.ietf.org/rfc/rfc2818
RFC2246: The TLS Protocol Version 1.0	http://www.ietf.org/rfc/rfc2246
SSL Protocol Version 3.0	http://wp.netscape.com/eng/ss13/draft302.txt
RFC2459: Internet X 509 Public Key infrastructure Certificate and CRL Profile	http://www.ietf.org/rfc/rfc2459

Note: The WS-I uses:

- ▶ *MUST* to indicate that something is required
- ▶ *SHOULD* to indicate that it is recommended
The implications of not implementing a recommendation must be weighed.
- ▶ *MAY* to indicate it is truly optional
Options must not compromise the interoperability of the basic implementation.

8.2.1 Basic Profile 1.0 for WebSphere

WebSphere Studio Application Developer 5.1.2 has an option for a developer to set the level of compliance to the WS-I Basic Profile 1.0 in either the workspace preference or the project properties. Use **Window** → **Preferences** → **Web services** or right-click **Project** → **Properties**.

In addition, we can also use the **Window** → **Preferences** → **Web services** → **Code Generation option** to configure WebSphere Studio

Application Studio 5.1.2 settings for different runtime environments such as SOAP runtime or IBM WebSphere runtime. The Axis runtime codes can be generated using command tools (`java2wsdl` and `wsdl2java`) outside of the WebSphere Studio Application Developer. The IBM WebSphere runtime environment conforms to the Basic Profile 1.0; SOAP and Axis runtime environments do not conform to Basic Profile 1.0.

There are three levels of reporting compliance to the WS-I Basic Profile 1.0 in the WebSphere Application Studio Developer 5.1.2:

- ▶ Requires compliance to Basic Profile 1.0
No error message will be generated, but the Web service proxy will not run if it does not conform to the Basic Profile 1.0.
- ▶ *(Default)* Suggests compliance to Basic Profile 1.0
Warnings will be displayed in the task list of the WebSphere Studio Application Developer 5.1.2, if there is non-conformance.
- ▶ Ignores Basic Profile 1.0
No error or warning will be displayed. WebSphere Application Studio Developer 5.1.2 is allowed to generate code that is not conformant to Basic Profile 1.0.

IBM WebSphere Studio Application Developer 5.1.2 also allows the option to create a document/literal style, rpc/literal style or rpc/encoded style WSDL description. However, Basic Profile 1.0 only allows the document/literal style and rpc/literal style of binding in the WSDL description. Microsoft Visual Studio .Net 2003 only allows the document/literal style of binding in the WSDL description.

Also refer to the Microsoft guidance on interoperability with WebSphere Studio Application Developer 5.1.2, found at:

<http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnbd/html/wsinteroprecsibm-final.asp>

8.2.2 Basic Profile 1.0 for Microsoft .Net

There are a number of WS-I requirements which may require our attention since there is the possibility of generating a Microsoft .Net Web service that does not conform to the Basic Profile 1.0, or requires clarification even though it conforms to the Basic Profile 1.0.

WS-I Basic Profile 1.0 uses the tag `R####` to accompany each requirement in the profile. R1000 means requirement 1000 in Basic Profile 1.0.

Refer to the specification of the Basic Profile 1.0 and the Errata for all the requirements to <http://www.ws-i.org>.

Annotation of conformance in the message

Basic Profile 1.0 does not require the inclusion of conformance claims in the WSDL description. Microsoft .Net Web service does not support including a conformance claim when generating a WSDL description. We can include a conformance claim in the WSDL for our Web services by saving the auto-generated WSDL and then manually edit the WSDL file.

The following tables (Table 8-2 on page 151 to Table 8-7 on page 154) list the WS-I rules that are described in *Building Interoperable Web services, WS-I Basic Profile 1.0, V1.0*, Microsoft 2003. Only those rules that Microsoft .Net Web services are either typically or potentially compliant with, or that are “unique” (meaning you will have to make program changes to comply with WS-I Basic Profile 1.0.), are listed.

Note: These tables only list WS-I “MUST or MUST NOT” rules. By definition, there is no compliance issue concerning WS-I “SHOULD” and “MAY” rules.

Because Microsoft .Net does not generate any `soapbind:fault` elements in the WSDL description of a Web service, there are a number of rules that Microsoft .Net complies with because the generation of more descriptive WSDL is left as a task for the programmer. Microsoft classifies .NET as compliant with these rules.

You may need to make code changes to adhere to typically compliant rules, and probably will need to make code changes to adhere to potentially compliant rules. The table omits the rules with which Microsoft states it is compliant.

There are further technical details in the Microsoft document that you will need to refer to.

WebSphere Studio Application Developer includes an automatic compliance checker, so we have not constructed the equivalent tables for IBM.

The entries in the table are shaded as follows:

- ▶ **Unique**
- ▶ Potentially Compliant
- ▶ Typically Compliant

Messaging

Table 8-2 Messaging requirements to take note of in Microsoft .Net

Number	Rule	Comments
R1000 R1001	Within the <soap:Fault>, we must only have <faultcode>, <faultstring>, <faultactor> and <detail> without any qualified namespace, because these child elements are local to <soap:Fault>. So, it is incorrect to specify <soap:faultcode>, <soap:faultstring>, <soap:faultactor> or <soap:detail> with the soap namespace.	Microsoft .Net runtime automatically wraps any exception that does not inherit from Microsoft .Net SoapException class in a Microsoft .Net SoapException. A SoapException only includes the <faultcode>, <faultstring>, <detail> and <faultactor>, when it is serialized into a SOAP message. So, we must make sure, when throwing the SoapException, that we only include the detail parameter as the main element of the <soap:Fault> and there is no invalid element within the <soap:Fault>
R1003	Within the <soap:Fault><detail>, there can be zero or more child elements with any qualified or unqualified namespace, except the name of the SOAP 1.1 envelope, http://schemas.xmlsoap.org/soap/envelope/	When coding attributes in the <detail> element for throwing Microsoft .Net SoapException, make sure that you do not use the namespace, http://schemas.xmlsoap.org/soap/envelope/
R1016	Within the fault message, we can specify a language other than English by using the attribute xml:lang in <soap:Fault> <faultstring xml:lang="fr">	Microsoft .Net SoapException class does not support including an xml:lang attribute on the faultstring element in the SOAP response.
R1005	Basic Profile 1.0 does not allow the use of <soap:encodingStyle> attributes on any elements whose namespace is http://schemas.xmlsoap.org/soap/envelope/	Do not use a SoapRpcMethodAttribute or SoapRpcServiceAttribute in Microsoft .Net. In addition, do not set the Use property to SoapBindingUse.Encoded when using SoapDocumentMethodAttribute or SoapDocumentServiceAttribute in Microsoft .Net.
R1006	Basic Profile 1.0 does not allow the use of <soap:encodingStyle> within the <soap:envelope> and the envelope's descendants; it does not allow RPC/Encoded message format.	
R1007	When using rpc-literal binding for a message, we must not use the <soap:encodingStyle> attribute on any elements that are descendants of <soap:body>.	Microsoft .Net does not support rpc-literal binding.

SOAP processing model

Table 8-3 SOAP processing model requirements to take note of in Microsoft .Net

Number	Rule	Comments
R1025	A receiver must check for mandatory headers in the message before processing the message.	Microsoft .Net Web service does not check for mandatory headers in the message before processing the message. It determines if the mandatory header is understood by checking if either the SoapHeaderAttribute is declared for the header or if the DidUnderstand property of the header is set to true.

HTTP in SOAP 1.1

Table 8-4 Http requirements to take note of in Microsoft .Net

Number	Rule	Comments
R1130	The consumer may automatically redirect a request when it encounters HTTP status code of "307 Temporary Redirect."	We must explicitly set the HTTP status code to 307 and add a Location HTTP header.
R1125	Basic Profile 1.0 states an instance must use a 4xx HTTP status code for responses that indicate invalid format of the request.	The Microsoft .Net Web service behavior, by default, is to return a 500 HTTP status code on error.

Service description

The WSDL describes the operations and binding to a protocol so that the sender and receiver know what to expect from the Web service.

Document structure

The document structure is defined in the WSDL 1.1 specification at <http://www.w3.org/TR/wsd1.html> and is used to describe Web services.

Table 8-5 WSDL document structure requirements to take note of in Microsoft .Net

Number	Rule	Comments
R2002	Basic Profile 1.0 requires the use of the XML Schema to "import" XML Schema Definitions.	The Microsoft .Net Web service uses WebServiceBindingAttribute to generate XML Schema import of a XML schema definition, provided the location attribute is specified with a non-empty string value.

Number	Rule	Comments
R2005	Basic Profile 1.0 requires that the targetNamespace attribute on the wsdl:definitions element of an imported WSDL description to have the same value as the namespace attribute on the wsdl:import element of the importing DESCRIPTION.	When we specify the WebServiceBindingAttribute in Microsoft .Net, the Microsoft .Net Web service includes a wsdl:import element, but we must make sure to specify the namespace property to have a value matching the targetNamespace attribute on the wsdl:definitions element of the WSDL description specified in the location property.

Messages

A WSDL message is an abstract definition of the data either presented as a document or arguments which can be used in a method invocation.

Table 8-6 WSDL messaging requirements to take note of in Microsoft .Net

Number	Rule	Comments
R2210	Basic Profile 1.0 requires that the corresponding abstract wsdl:message define zero or one wsdl:parts when a document-literal binding in a DESCRIPTION does not specify the parts attribute on a soapbind:body element.	The Microsoft .Net Web service generates the a WSDL description that typically defines a single wsdl:part for a wsdl:message when using document-literal binding. However, when using a SoapDocumentServiceAttribute or SoapDocumentMethodAttribute with the ParameterStyle property having a value of SoapParameterStyle.Bare, there can be no wsdl:part defined in the WSDL description. So, we must avoid using SoapParameterStyle.Bare to conform to Basic Profile 1.0.
R2203	Basic Profile 1.0 requires that an rpc-literal binding in the soapbind:body of a DESCRIPTION refer only to wsdl:part elements that have been defined using the type attribute.	The Microsoft .Net Web service does not generate a WSDL description with an rpc-literal binding. The Microsoft .Net Web service only supports the use of document-literal bindings.
R2211	Basic Profile 1.0 requires that a MESSAGE using rpc-literal binding not to have part accessors with the xsi:nil attribute set to 1 or true.	

Bindings

A WSDL binding defines the message format and protocol for operations and messages defined by a particular port-type. SOAP binding is the most used transport for WSDL in Web service and the only binding in the WS-I Basic Profile. The SOAP specification contains rules to map the abstract representation of data types, messages and operations to their physical representation.

Table 8-7 WSDL binding requirements to take note of in Microsoft .Net

Number	Rule	Comments
R2705	A wsdl:binding in a description must either be an rpc-literal binding or a document-literal binding.	The Microsoft .Net Web services generates WSDL description that uses document-literal binding. We must not use a SoapRpcMethodAttribute or SoapRpcServiceAttribute to avoid interoperability problems.
R2706	A wsdl:binding in a description must use the value of literal for the use attribute in all soapbind:body, soapbind:fault, soapbind:header, and soapbind:headerfault elements.	The Microsoft .Net Web service generates WSDL description that uses the literal value for the use attribute on all soapbind:body and soapbind:header elements. but the Web service does not generate soapbind:headerfault and soapbind:fault elements. We must not use SoapRpcMethodAttribute or SoapRpcServiceAttribute to avoid interoperability problem.
R2710	The operations in a wsdl:binding in a description must result in operation signatures that are different from one another.	The Microsoft .Net Web service generates a WSDL description having a wrapper element of the same name as the Webmethod to ensure unique operation signature. However, using SoapDocumentMethodAttribute with the ParameterStyle property set to SoapParameterStyle, we must then ensure that the parameters of our Webmethod are unique within the Web service.
R2717	Basic Profile 1.0 requires rpc-literal binding to have the namespace attribute specified on the soap:body elements and the namespace value to be an absolute URI.	The Microsoft .Net Web service generates a WSDL description with a document-literal binding only.
R2726	Basic Profile 1.0 requires rpc-literal binding not to have namespace be specified on soapbind:header, soapbind:headerfault and soapbind:fault elements.	
R2725	If an INSTANCE receives a message that is inconsistent with its WSDL description, it <i>must</i> check for VersionMismatch, MustUnderstand, and Client fault conditions in that order.	The Microsoft .Net Web service checks for a VersionMismatch error first. It then checks for a Client error after parsing the message for the header and parameter value. We must first check for MustUnderstand faults in our methods.

Number	Rule	Comments
R2738	A message must include all soapbind:headers specified on a wsdl:input or wsdl:output of a wsdl:operation of a wsdl:binding that describes it.	We must not set the Direction property of the SoapHeaderAttribute to SoapHeaderDirection.Fault, because the WSDL description includes the soap:header element on the wsdl:output while the response message does not.

8.2.3 Summary

Although these tables look rather forbidding, to quote the Microsoft article on interoperability (cited above on page 148) “... *it is apparent that interoperability using Web services developed on the Microsoft.NET Framework 1.1 and IBM WSAD 5.1.2 is most definitely achievable today. ... [It is] testament to how well both Microsoft and IBM have implemented the WS-I Basic Profile 1.0 for their implementations of Web services.*”

8.3 Interoperability standards: addressing

WS-Addressing was submitted to W3C in August 2004, but has not yet been approved as a standard. It is implemented in IBM's Emerging Technologies Toolkit (ETTK), available from alphaWorks. In this section, we will see how it enables dynamic change of endpoints, asynchronous and stateful communication.

The standard submitted to W3C can be found at:

<http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810>

8.3.1 Insurance example

Rather than simply working through the standard itself, we can best understand what it does and why it is useful if we use an example. Let's consider the insurance example developed in this book. Lord General Insurance (LGI) makes Web service calls to external assessors to establish whether they are available for particular assessments. In a simplistic version of this example, these calls would just be synchronous request-response messages. However, we will consider a more realistic scenario, as shown in Figure 8-2 on page 156.

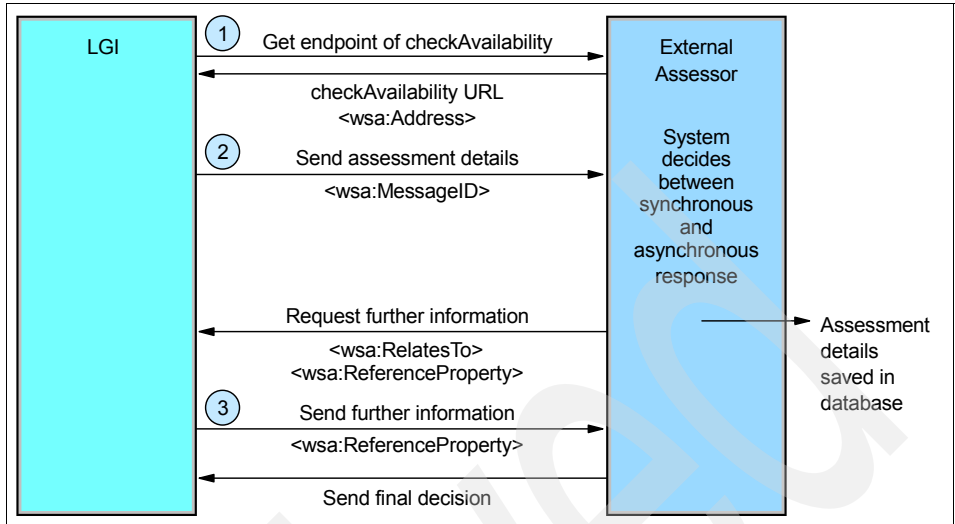


Figure 8-2 LGI communicating with external assessor, using WS-Addressing

In a simple scenario, the endpoint (that is, the address) that LGI is sending its messages to is fixed. This is fine for simple systems. However, suppose the assessor's server goes down and they want messages to be handled by their back-up system. If the endpoint is fixed, they have no way of doing this. However, WS-Addressing defines the `wsa:EndpointReference` element specifically so that we can dynamically change endpoints. In this more sophisticated scenario, LGI's initial call is simply to obtain the endpoint of the `checkAvailability` service. This information is returned in the SOAP header, like all information exchanged using WS-Addressing. This is because the information is meta-information: it is about the SOAP message, rather than being part of the message. Hence, the return message header might resemble Example 8-1.

Example 8-1 A Web service endpoint dynamically obtained using WS-Addressing

```

<wsa:EndpointReference>
  <wsa:Address>https://www.carAssessor123.com/checkAvailability</wsa:Address>
</wsa:EndpointReference>
  
```

This means that the assessor can change the endpoint of this service at will. This kind of flexibility means that in addition to re-routing to a backup system, a business can seamlessly switch from an old system to a new one, or can decide to outsource one of its services.

Having obtained the endpoint, LGI then sends a message to the assessor, asking if they are available for a particular assessment. In a simple

implementation, the assessor's system might just check their diary, and if they are free, will say that they are available. However, suppose that for some more complex assessments, the assessor system takes a long time to process the request and may need to ask for further details. In this situation, we would need a system that could respond either synchronously (if no complex processing is involved) or asynchronously (if it is). How can we do this?

When LGI sends in the assessment details, the SOAP header includes a message identification:

```
<wsa:MessageID>uuid:48454857</wsa:MessageID>
```

If the assessor's system can deal immediately with the availability check, it sends the response straight back. If the request is going to require more processing and further information, the system creates a database entry of the assessment details which it associates with a reference number. It then establishes what further information is required and sends a message back, the header of which contains the following elements.

Example 8-2 Use of wsa:RelatesTo and wsa:ReferenceProperty

```
<wsa:RelatesTo RelationshipType="wsa:ReplyTo">
  uuid:48454857
</wsa:RelatesTo>
<wsa:ReferenceProperty>
  <assessor:AssessmentID>193953785</assessor:AssessmentID>
</wsa:ReferenceProperty>
```

So, even though this reply is asynchronous, the LGI system knows what earlier message it refers to, because it uses the `wsa:RelatesTo` element and gives the unique identifier of the earlier message.

At this point, we need to stop and consider the consequences of this aspect of WS-Addressing. What we are describing is a dynamic decision to respond to a message either synchronously or asynchronously. However, for this to happen, the system which *sent* the message must be able to cope with this behavior. If, for example, a system has sent a SOAP message over Http, ordinarily the reply would come synchronously. Unless current systems are updated, if the immediate Http response is empty, and the real reply comes several minutes later, the systems may throw an error upon receiving the empty reply and not be listening for the real one. So, in order for WS-Addressing to work, when systems send messages, they must be capable of dealing with either a synchronous or an asynchronous response.

When the LGI system has obtained the extra information required, it sends it back to the assessor. How does the assessor system know which assessment it relates to? The LGI system can use the `wsa:ReferenceProperty` previously sent

back by the assessor. When the assessor receives this reference, they know which database entry it refers to. Thus, we see a third use of WS-Addressing: it enables stateful interaction in which, rather than each message exchange being one-off, messages can form a series. Note also that the LGI system does not have to understand the <assessor:AssessmentID> tag that is returned within the <wsa:ReferenceProperty>. The LGI system simply knows that this reference is being used to identify this particular communication session and that it must include it when it replies to the assessor. The introduction of just one new element (the wsa:ReferenceProperty) gives us the immense power of stateful communication.

8.3.2 Summary

We have seen that although WS-Addressing only introduces a few new XML elements, they give us a great deal of power. There are three main capabilities introduced by the standard:

- ▶ Dynamic change of endpoints
- ▶ Asynchronous messaging
- ▶ Stateful communication

We have also seen that for dynamic asynchronous messaging to work, the message sender must be able to cope with an asynchronous response.

8.4 Security

Security is essential for Web services to be widely adopted for e-commerce. In the following section, we discuss the need for additional Web service security standards above and beyond the security already available on the Internet, and the proposal from WS-I for a basic Web services security profile.

8.4.1 Why do we need more security specifications?

When we approach Web services security for the first time, we might be tempted to ask why we need more security specifications. Computer networks already use a number of security protocols, such as the Secure Sockets Layer (SSL) or Kerberos; why are these not sufficient?

Consider a very simple example, in which a client accesses a Web service across a network:

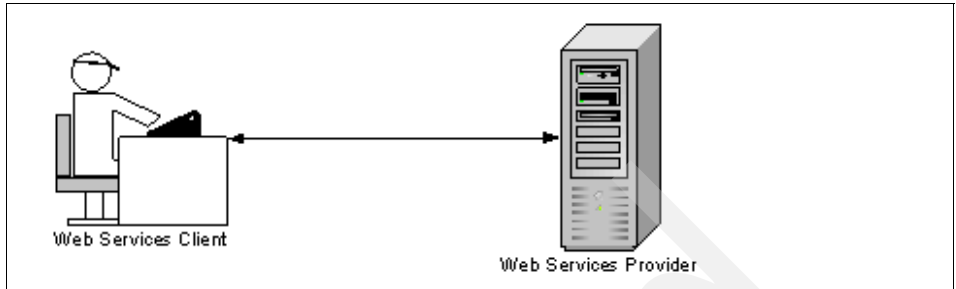


Figure 8-3 Client accessing a Web service

Why do we need to ensure that this communication is secure?

- ▶ Authentication and authorization. We need to be able to identify the client, so that we know what they are and are not allowed to do.
- ▶ Message integrity. How do we know that the message sent by the client has not been tampered with since they sent it? A hacker could have intercepted the message and changed it. To prevent this, we can digitally sign our message so that the recipient knows it has come from us.
- ▶ Message confidentiality. If we want the message to be private, we need to encrypt it.

So, why can't we use pre-established protocols to achieve these aims? For example, https is the version of http that uses SSL as a transport level protocol which allows us to encrypt any message we send. Web services security, on the other hand, is message level security. Let's compare transport level and message level encryption:

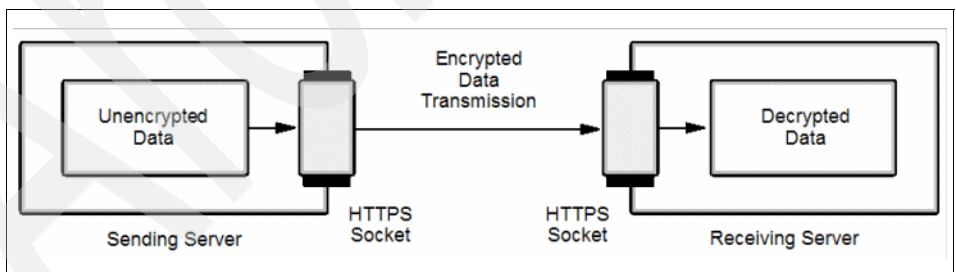


Figure 8-4 Transport level encryption using https

As we can see, in transport level encryption, the message is not encrypted before transport, but only as it is being sent across the network. As soon as it has reached its destination, the receiving http server is in possession of a decrypted message.

By contrast, with message level encryption, the encryption of the message is entirely separate from the process of sending it across the network.

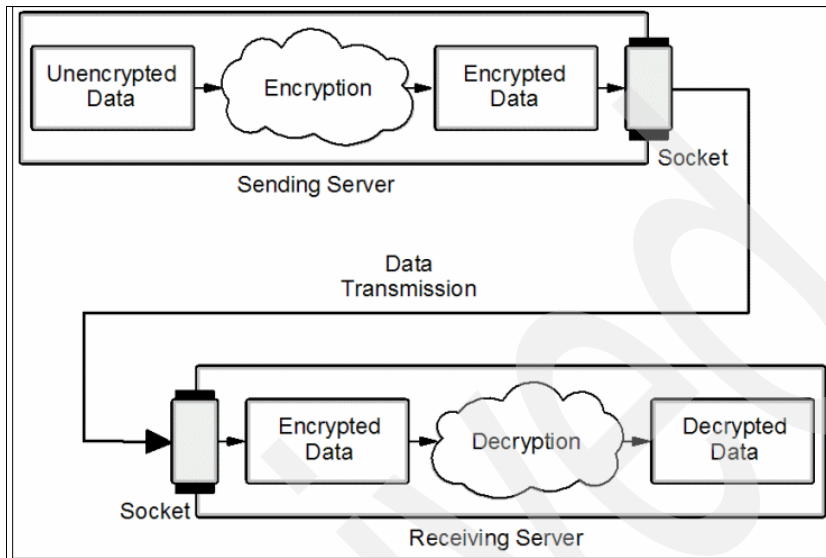


Figure 8-5 Message level encryption

We encrypt the message, then give it to a server to send. What the server starts off with is thus already encrypted. When it is received by a server at the other end, this server also sees the message as encrypted. A separate decryption stage will be required, utilizing the appropriate key.

What advantages does using this message level Web services security give us over simply transport level security?

- ▶ The message stays encrypted until it is explicitly decrypted. Contrast this with the transport level security in Figure 8-4 on page 159. In that scenario, once the message has been received by the second http server, it is already in decrypted form. If we perform further manipulation of the message, we need to consider its security. If the area behind our server is insecure in some way, for example not directly under our control, we have nullified the effect of encrypting our messages during transport, because a hacker can read them after they have been received.
- ▶ We can save processing time by choosing which sections of the message need to be encrypted. By contrast, with transport level encryption, we always have to encrypt the entire message.
- ▶ We can encrypt our sensitive data while at the same time leaving the routing information unencrypted. This means that we can send our messages via intermediaries such as firewalls. With transport level encryption, if we want a

server to forward our message on, it must decrypt the message and encrypt it again. With message level security, at each intermediary, the routing information can be read and the message routed correctly, while the data remains confidential.

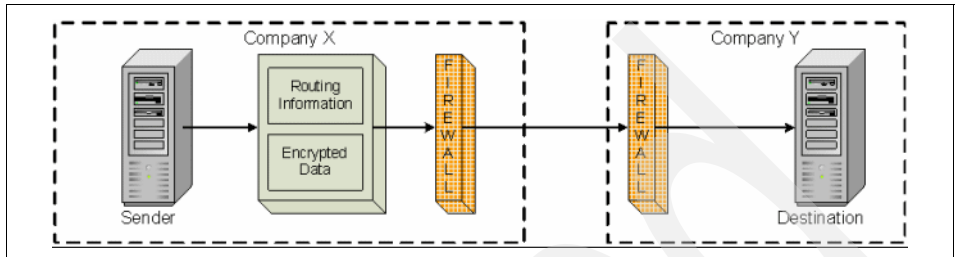


Figure 8-6 Example of selective encryption - data is encrypted, routing information is not

We can see that there are definite reasons why we need specifications for message-level Web services security. Now we will look at one of the most fundamental protocols.

8.4.2 WS-Security 2004

The WS-Security specification was first proposed in draft form by IBM, Microsoft and Verisign in April 2002. It was the first document to define standards for the three most basic security tasks: authentication, digital signature and encryption. It defines these standards for Web services calls made using SOAP. The Organization for the Advancement of Structured Information Standards (OASIS) group has recently approved several documents as part of the final version of this specification. At the time of writing, three documents have been published by OASIS:

- ▶ SOAP Message Security V1.0 (WS-Security 2004)
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>
- ▶ Username Token Profile V1.0
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf>
- ▶ X.509 Token Profile V1.0
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf>

First, we will examine what the standard defines for each of the three basic security tasks, then go on to look at some of the other definitions in WS-Security

2004 and what additional power and flexibility they give us in applying authentication, digital signature or encryption.

Authentication

WS-Security 2004 defines three methods of authentication:

1. Username only.

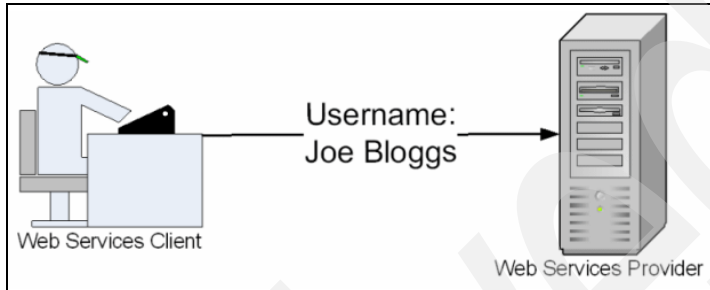


Figure 8-7 Authentication by username only

In a safe environment, such as inside a company firewall, we may want to provide services to people that are tailored to their needs, without needing the security afforded by passwords. For example, if none of the services being accessed use confidential information or can be used maliciously, identification purely by a username sent as plain text might be sufficient.

2. Username and password as plain text.

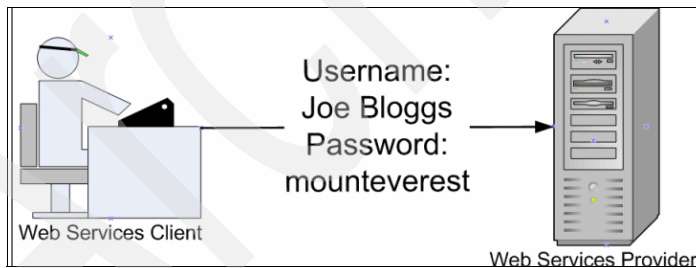


Figure 8-8 Authentication by username and password sent as plain text

The next level of security is to require users to have passwords and to send them as plain text to the server. This should only be used when we know that the message cannot be intercepted to read the password.

3. Username and password digest.

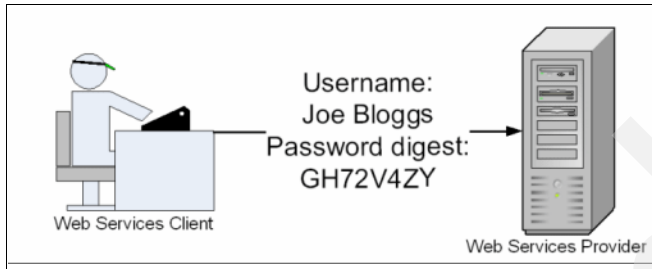


Figure 8-9 Authentication using password digest

This is the most secure form of authentication defined by WS-Security 2004. Rather than sending the password in plain text, the sender takes the SHA-1 hash of the password before sending it. The SHA-1 algorithm is defined by the US National Institute for Standards and Technology (NIST).

Digital signature

A message is digitally signed by taking a hash of the message to create a digest, then encrypting this digest with the sender's private key and attaching the result to the message. When the recipient receives the message, they decrypt the digest, using the sender's public key, then take their own hash of the message and compare the two. If the message has been tampered with, the new digest will not match the decrypted one.

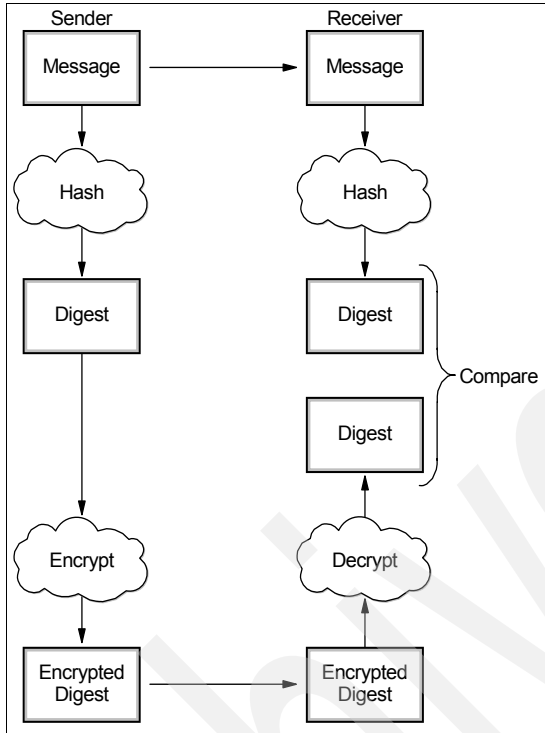


Figure 8-10 The process of creating and verifying a digital signature

WS-Security 2004 builds on the earlier XML Signature standard proposed by w3c. As such, the algorithms specified are the same:

- ▶ For creating the digest, the SHA-1 hash algorithm is used. The algorithm must be identified by declaring it with the element:

```
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmlsig#sha1"/>
```

- ▶ Two methods of encrypting the digest are allowed: the Digital Signature Algorithm (DSA) and RSA. They must be identified by using the following elements:

```
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmlsig#dsa-sha1"/>
```

```
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmlsig#rsa-sha1"/>
```

Encryption

WS-Security 2004 allows either symmetric encryption with a secret key, or asymmetric encryption with an encrypted session key. Let's review what each of these processes involves. In symmetric encryption, the sender encrypts the message with a secret key. The recipient must use this same key to decrypt the message.

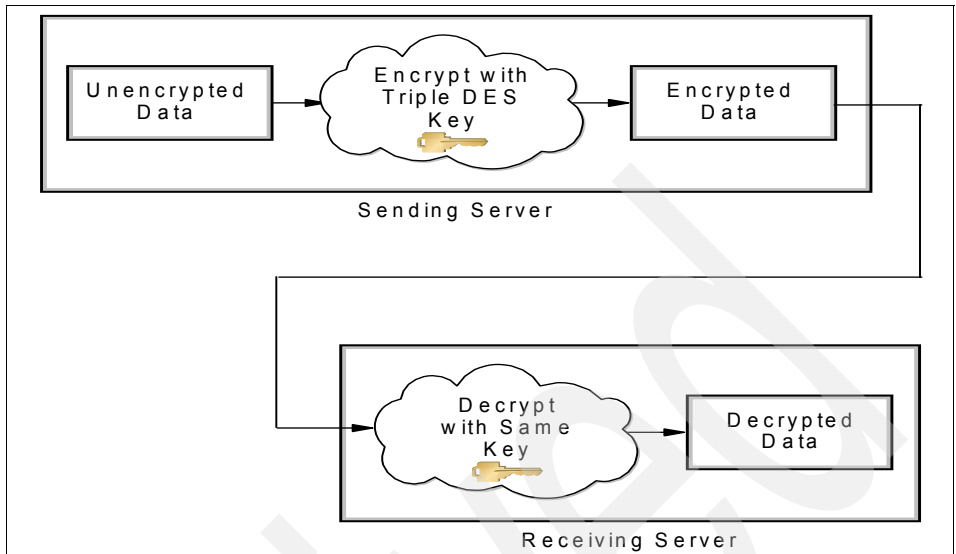


Figure 8-11 Symmetric encryption using a triple DES key

However, there is a problem with this method: how does the sender securely give a copy of the key to the recipient? If the key is copied by a third party while in transit, they will be able to read all of the encrypted information. To get around this problem, the public-private key mechanism was developed. In this system, keys exist in pairs. If one is used to encrypt a message, the other must be used to decrypt it. So, anyone wishing to receive confidential information generates a key pair. They keep one key private and make the other key publicly available. If we wish to send someone a message, we use their public key to encrypt, knowing that only they possess the private key necessary to decrypt it.

Asymmetric encryption is secure, but slow. To get the best of both worlds, most encryption is now done using a hybrid of the symmetric and asymmetric systems.

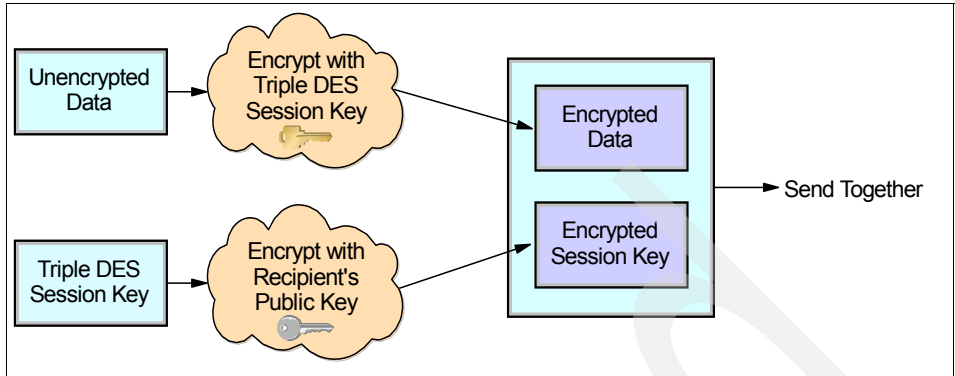


Figure 8-12 Hybrid of symmetric and asymmetric encryption

In this system, the sender first generates a symmetric key. This key will only be used for one communication session, so it is usually referred to as the session key. The sender uses it to encrypt the message. Then the sender encrypts this key itself, using the public key of the message recipient, and attaches the encrypted key to the message. Then, when the recipient gets the message, they use their private key to decrypt the session key and then use that to decrypt the message. Because the session key is only used once, even if someone managed to discover it, they would only be able to decrypt one message.

WS-Security 2004 has full support for these types of encryption. Again, it is based on an earlier standard: XML Encryption from W3C. A variety of encryption algorithms are allowed.

- ▶ Symmetric encryption using the triple-DES algorithm should be identified with the element:


```
<EncryptionMethod
  Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc">
```
- ▶ Encryption of a triple-DES session key can be identified with the element:


```
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
```

Now that we have looked at the fundamentals of the three security tasks defined by WS-Security 2004, let's examine some of the related concepts it defines and how they allow us to leverage the standard.

Security tokens

We need a method to encode binary security information such as a certificate. For example, when digitally signing a message, one's public key certificate is usually included with the message, so that the recipient can use it to decrypt the

digest. The SOAP Message Security document only defines the generic format for binary security tokens.

Example 8-3 Binary security token format

```
<wsse:BinarySecurityToken wsu:Id=...  
    EncodingType=...  
    ValueType=.../>
```

The specification of individual types of binary security token is intended to be dealt with by other documents. So, for example, the X.509 Token Profile defines three kinds of X.509 tokens, as shown in Table 8-8.

Table 8-8 Defined X.509 binary security tokens

Token	ValueType URI	Description
Single certificate	#X509v3	An X.509 certificate
Certificate path	#X509PKIPathv1	An ordered list of X.509 certificates packaged in a PKIPath
Set of certificates and certificate revocation lists (CRLs)	#PKCS7	A list of X.509 certificates and CRLs

A separate specification for Kerberos tokens is currently in draft form.

Encryption ReferenceList

The encryption ReferenceList allows one to define which sections of the SOAP message are to be encrypted. We saw that selective encryption is one of the advantages of message-level security and this standard enables this feature.

Actors

The specifications for SOAP define the concept of an actor. Let's examine how this concept applies to WS-Security 2004. If we want to have a message read and processed by one server, then passed on to other servers for further processing, and do all this securely, we want to be able to target different security information at these different recipients. The WS-Security 2004 standard enables you to have multiple security headers in a single SOAP message. Each of these headers can identify which recipient (or actor) needs to process it. Then, when a particular recipient has processed the message and is ready to forward it, it can choose to modify any of the existing security headers, or add another one.

Let's consider how actors might be used in the insurance example developed in this book. LGI, the insurance company, wants to request a specific external assessor to compile a report for a particular car insurance claim.

1. LGI sends a Web services message to the assessor, and digitally signs it so that the assessor can verify that it is genuine.
2. The assessor's scheduling system receives the message and books the time in the assessor's diary.
3. It then adds a second signature (its own) to the message and forwards it to the assessor's report database.
4. This system verifies that the message has genuinely come from the scheduler by checking the digital signature.
5. It then takes the car details from the message and uses them to create a skeleton report.

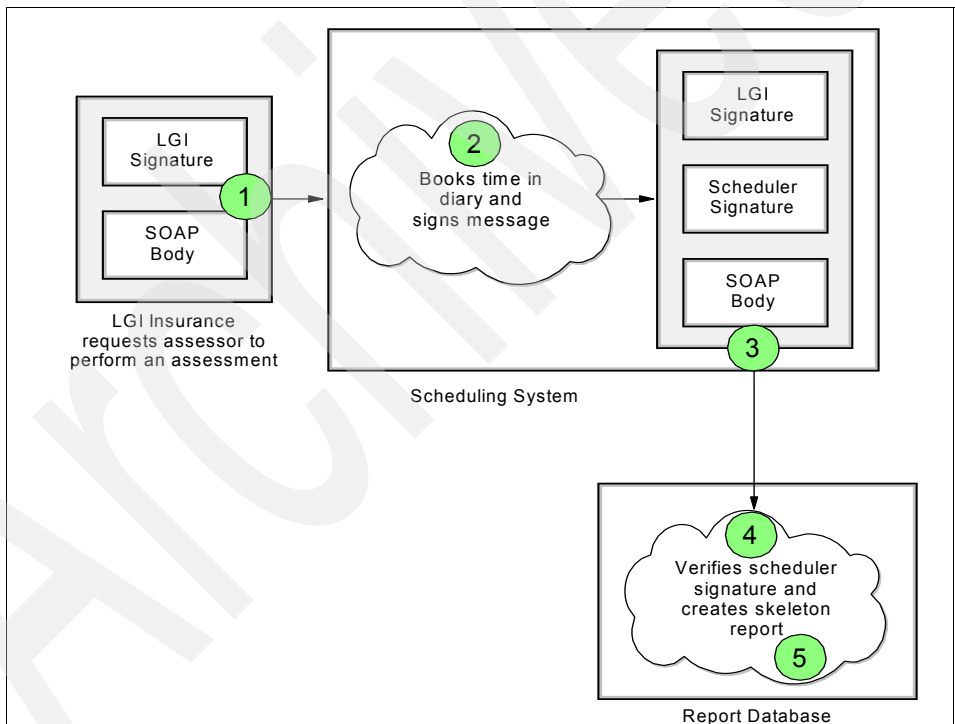


Figure 8-13 Use of actors in WS-Security 2004

This demonstrates how a message may be processed by multiple systems (in this case, the scheduler and the report database) while at the same time ensuring that each stage of the message processing and transfer is secure.

WebSphere makes use of this technique by having a Web Services Gateway, which provides a single endpoint that external messages are sent to. It then routes them to the correct WebSphere Web service. For more information, see *Employ the IBM WebSphere Web Services Gateway* by Michael Ellis, found at:

<http://www-106.ibm.com/developerworks/library/ws-routing/?ca=dnt-537>

Errors

In order to react appropriately when a secure communication fails, we need defined error messages. SOAP Message Security defines two messages for unsupported actions and five for failures. These messages are returned using the fault mechanism that is part of the SOAP standard.

Table 8-9 Unsupported action errors

Error	Fault message
An unsupported token was provided.	wsse:UnsupportedSecurityToken
An unsupported signature or encryption algorithm was used.	wsse:UnsupportedAlgorithm

Table 8-10 Failure errors

Error	Fault message
An error was discovered processing the <wsse:Security> header.	wsse:InvalidSecurity
An invalid security token was provided.	wsse:InvalidSecurityToken
The security token could not be authenticated or authorized.	wsse:FailedAuthentication
The signature or decryption was invalid.	wsse:FailedCheck
Referenced security token could not be retrieved.	wsse:SecurityTokenUnavailable

This brings us to the end of our examination of the WS-Security 2004 standard. We can see that it contains a number of important definitions, allowing us to perform authentication, digital signature, encryption and processing by multiple recipients. However, this specification by itself does not directly address interoperability. We will now go on to examine one that does.

8.4.3 WS-I Security Profile

The Web Services Interoperability Organization (WS-I) is a multi-vendor group, whose aim is to ensure that different implementations of Web services can interoperate. It published the draft Security Profile 1.0 on 12 May 2004:

<http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2004-05-12.html>

One might suppose that, providing all vendors have implemented a particular standard, such as WS-Security, their Web services will interoperate. However, there are two reasons why this might not be the case:

1. There may be ambiguities in the original standard which were not noticed when it was agreed.
2. There may be flexibility in the standard which hinders interoperability. For example, suppose one section of a standard defines two valid ways of doing something. If one vendor uses one by default, while another vendor uses the other, their implementations will fail to work together, even though both are correctly implementing the standard.

It was for these reasons that the WS-I was set up. Clearly, to solve the first problem, the WS-I has to remove the ambiguity. To resolve the second, it can either declare that one of the options is not valid if a vendor wishes to claim WS-I conformance, or it can say that both vendors must be able to cope with both alternatives.

Conformance to the Profile

The WS-I Security Profile has a number of sections. Some are considered part of the base profile, others are extensibility points. Of course, some vendors may choose not to implement the extensible parts of the profile, and one should not assume they are present in an implementation unless it explicitly documents them. Furthermore, even when we exclude the extensibility points, the remainder of the profile is still subdivided into four sections, so that vendors can explicitly state which sections they conform to. The division is as follows (this is not intended as an exhaustive list):

1. Core:
 - Transport layer security
 - Security tokens
 - Security token references
 - Timestamps
 - References
 - Processing orders
 - Signature
 - Encryption
 - Algorithms

2. Username tokens
3. X.509 Certificate Tokens
4. Attachments

Let's now examine the clarifications made by the profile, and why they have been necessary.

Transport layer security

We know that in addition to using Web services message level security, we can also use https to ensure our messages have transport level security. However, there are some ways of using https which are insecure. Two usages are specifically banned by the profile for this reason:

1. SSL V2.0. This has known security flaws.
2. The use of a SOAPACTION header. When a SOAP message is sent over http, the message contains a SOAP header and SOAP body, both contained in a SOAP envelope. These elements are all part of the http message. However, the protocol for http transport means that the http message *itself* has a header:

Example 8-4 An http message header with a SOAPACTION line

```
POST /authorizationsample/weblogservice.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: localhost
Cache-Control: no-cache
Pragma: no-cache
SOAPACTION: "http://www.onlinebookstore.com/orderBook"
Content-Length: 1094
```

We can see that this header contains a line entitled SOAPACTION. This line affects how the message will be processed when it is received. Ordinarily, this would not cause any problems. However, if we are sending a secure message using https, the http header will not be encrypted. Hence the SOAPACTION line could be changed. Even though the message itself may be encrypted, changing the header could affect how it is processed. In order to keep our message completely secure, the profile forbids the use of a SOAPACTION header in messages sent using https.

Security tokens

Specification of binary security tokens is made more explicit by two rules:

1. Base64Encoding must be specified as the encoding type.
2. The ValueType of the binary security token must be specified. For example, X.509v3.

Security token references

With a very simple use of a binary security token, such as encrypting the entire message, the token is only used once. Hence one might assume that we could insert the token into the message inline, just before it is used. However, consider a situation in which the same token is used several times in one message. For example, if we sign and encrypt different sections of the message. We can see that it is impractical to repeat the token. Rather, we need a way to reference the token whenever it is needed. While these references are very useful, there are a large number of ambiguities related to using them, so the profile places constraints on their use. Let's look at the more important constraints:

1. You must use the `wsse:SecurityTokenReference` tag for your references. Using the reference tags defined by older standards such as the XML Digital Signature standard is not allowed. The `wsse:SecurityTokenReference` must have a `ValueType` attribute.
2. When using a `wsse:SecurityTokenReference` tag, the recommended way to refer to it is by using an `XPointer` to refer to its `wsu:Id` attribute. A `XPointer` is the traditional method of reference used in html documents to refer to other sections of the same document. For example, `URI=#MyCertificate`.
3. If direct reference is not possible, two alternatives are permitted:
 - a. Embedding the security token directly within the reference.
 - b. Using a `wsse:KeyIdentifier`. It must have a `ValueType` attribute with one of the specified values.
4. A key name should never be used to identify a key. (This was allowed in the XML Digital Signature standard.)
5. A binary security token must precede the first `wsse:SecurityTokenReference` to it. This ensures that the binary token is readily available when it is required.

Timestamps

By including a timestamp with each of our messages, we can ensure that no one can intercept our message and re-send it later. The timestamp precludes this because it contains the time when the message was created and when it expires. If a message is received outside of these times, the recipient knows to ignore it. The profile specifies the following format for a timestamp:

Example 8-5 A valid timestamp

```
<wsu:Timestamp wsu:Id="timestamp">
  <wsu:Created>2001-09-13T08:42:00Z</wsu:Created>
  <wsu:Expires>2001-10-13T09:00:00Z</wsu:Expires>
</wsu:Timestamp>
```

References

References are the mechanism we use to refer to another part of the SOAP message. Sections of the message are identified by Id values so that we can refer to them. Of course, if several sections of the message had the same Id value, the reference would be ambiguous. Hence the profile specifies:

- ▶ All Id values must be unique.

Processing order

Consider what happens when we both encrypt and digitally sign a message. The recipient needs to know in what order the two processes were done. The profile specifies the constraint:

- ▶ The receiver must get the correct result if they process the elements in the order in which they appear in the security header.

Digital signature

1. Enveloping signatures, as defined by the XML Signature specification, are not allowed. An enveloping signature is where a signature signs all of the data contained within the XML tags of the signature element. This contrasts with a detached signature, which references a separate section of the XML document that it is signing. Enveloping signatures limit the ability of intermediaries to process the message, since they cannot alter the information contained within the signature tags.

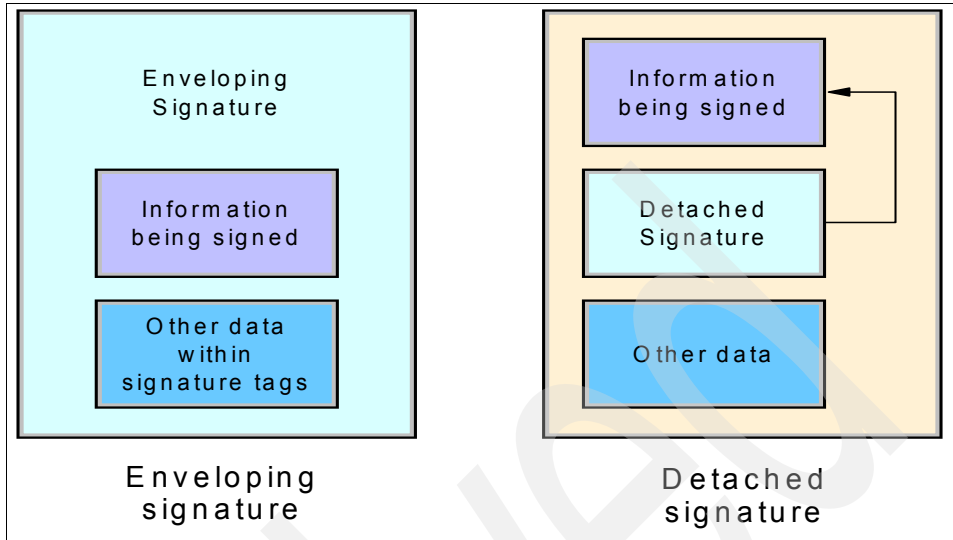


Figure 8-14 The difference between enveloping and detached signatures

2. References saying which part of the message is being signed should be made by using an XPointer to refer to an Id attribute. If the necessary Id attribute does not exist, the XPath Filter 2.0 transform must be used to form the reference.
3. If the public key certificate is included in the message to allow the recipient to easily verify the signature, it must be referred to by a `ds:Reference` within the signature, to prevent substitution of the certificate by another with the same key. (If the certificate was substituted for a certificate with a different key, the digital signature verification would fail. But if a certificate with the same public key, yet purporting to come from a different person, was substituted, the signature verification would succeed.)

Encryption

1. `xenc:EncryptedKey` elements must precede the data they have been used to encrypt. This is to ensure that when the SOAP message is being processed, the relevant decryption keys are to hand.
2. `xenc:EncryptedKey` and `xenc:EncryptedData` elements must specify their encryption method by using the `xenc:EncryptionMethod` child element.
3. SOAP envelope, header or body elements must not be encrypted. These elements are not part of the message, rather they contain the message. The SOAP envelope, header and body elements will be parsed before their contents. Hence, if they were encrypted, when the parser reached them, it would not know which elements they were. So it would not know to extract the

decryption information from the information within them. Hence it could not process the message.

4. A digest value for data that is subsequently encrypted must also be encrypted.

Algorithms

The algorithm most frequently used for symmetric encryption in recent years is the triple DES algorithm. In this system, data is encrypted with one key, decrypted with another and encrypted with a third. However, this system is really just a short term solution to the disadvantages of using the aging DES algorithm. The Advanced Encryption Standard (AES) was chosen in 2001 by the US government to be used for encrypting all non-classified data and is now gaining wide acceptance. The WS-I profile hence makes the following recommendation:

- ▶ Transport level security should use the AES algorithm.

Username tokens

We have seen that three different types of username tokens are defined by the WS-Security 2004 specification. The following constraints are placed upon their use:

1. Each `wsse:Password` element must have a `Type` attribute, which must be one of the allowed types.
2. If a password digest is used, it must be calculated by the following method¹:

¹ A NONCE is a *N*umber than can only be used *ONCE*. (Actually the true derivation is from Anglo-Saxon <http://dictionary.reference.com/search?q=nonce>) A Nonce used correctly prevents replay attacks.

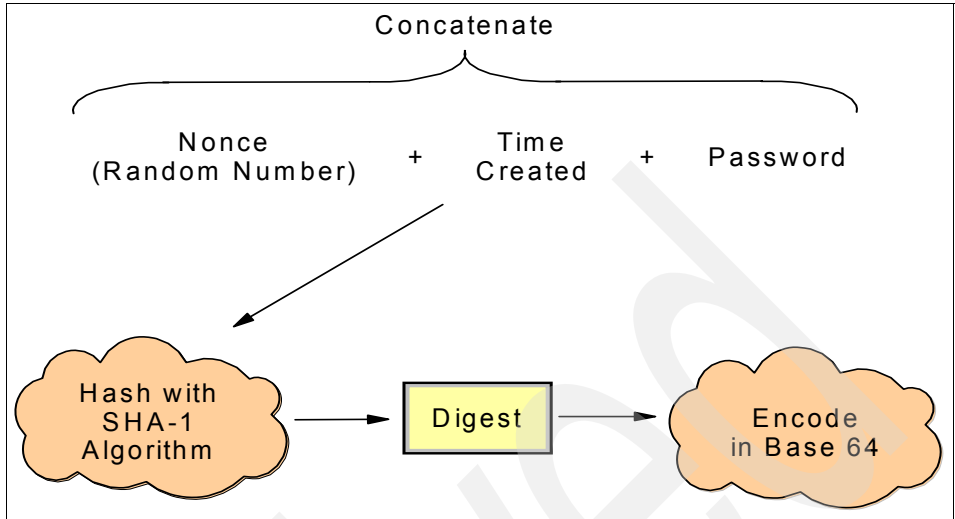


Figure 8-15 The mandated method for creating a password digest

X.509 Certificates

When a certificate is used for encryption or digital signature, the recipient must be able to confirm it is genuine by tracing the certification path back to a trusted certification authority. However, there are multiple ways this information can be encoded in X.509 certificates. Only two are allowed:

- ▶ X509PKIPathv1 (preferred)
- ▶ PKCS7

Attachments

The WS-I has a separate attachments profile. Hence the specification simply requires that profile is obeyed:

- ▶ WS-I Attachments profile must be implemented.
- ▶ All attachments that are encrypted or signed must be referenced by a `wsse:Security` reference.

8.4.4 Summary

We have seen that although many security standards already exist, such as transport level security using SSL, message level Web services security offers several advantages:

- ▶ Messages are secure until explicitly processed.
- ▶ We can save time by only applying security to sections of our messages.
- ▶ We can route messages via intermediaries and still keep them secure.

The two most fundamental security specifications are WS-Security 2004 and the WS-I Security Profile. WS-Security 2004 defines standards for basic security processes such as authentication, digital signature and encryption. Two related documents published by OASIS define username tokens and X.509 security tokens.

The WS-I Security Profile clarifies ambiguities in many of the OASIS standards, including:

- ▶ Transport layer security
- ▶ Security tokens
- ▶ Security token references
- ▶ Signature
- ▶ Encryption
- ▶ Algorithms
- ▶ Username tokens
- ▶ X.509 Certificate Tokens

Vendors should implement the WS-I Security Profile to ensure their products will interoperate.

8.5 WS-Coordination

WS-Coordination specification is part of the Web service transaction layer; it has been authored by IBM, Microsoft and BEA and current release is dated September 16, 2003.

WS-Coordination addresses all business processes requiring a composition of multiple Web services in a single workflow; the resulting artifact is subsequently exposed as a new single Web Service. This context is typically related to the implementation of a Service-Oriented Architecture in which Web services are considered as basic building blocks for new applications development. In such an environment, each Web service part of the whole process must be coordinated with the other ones to guarantee a consistent final state after process completion both in case of success and fault.

WS-Coordination standard achieves Web services coordination by means of a specific coordination service, also known as coordinator; it enables participants to reach consistent agreement on the outcome of distributed activities. Agreements are based on coordination protocols which are suited for specific activities.

Coordinator services are themselves exposed as Web Service. They are shown in the following list:

- ▶ ActivationService
 - CreateCoordinationContextRequest message
 - CreateCoordinationContextResponse message
- ▶ RegistrationService
 - RegisterRequest message
 - RegisterResponse message

The Activation Service creates the CoordinationContext object which is then returned to the service requestor, while Registration Service registers the application for a specific application protocol.

WS-Coordination specification details all request and response messages for the activation and registration service and the CoordinationContext type structure. As shown in Figure 8-16 on page 179, CoordinationContext is composed by the following three main objects:

- ▶ An activity identifier
- ▶ The endpoint to the registration service
- ▶ A Coordination type. Each coordination type can support multiple protocols

The WS-Coordination framework is considered extensible as new protocols or new extension elements to current protocols can be added to the framework itself. However, coordination types are detailed only in the WS-Transaction specification.

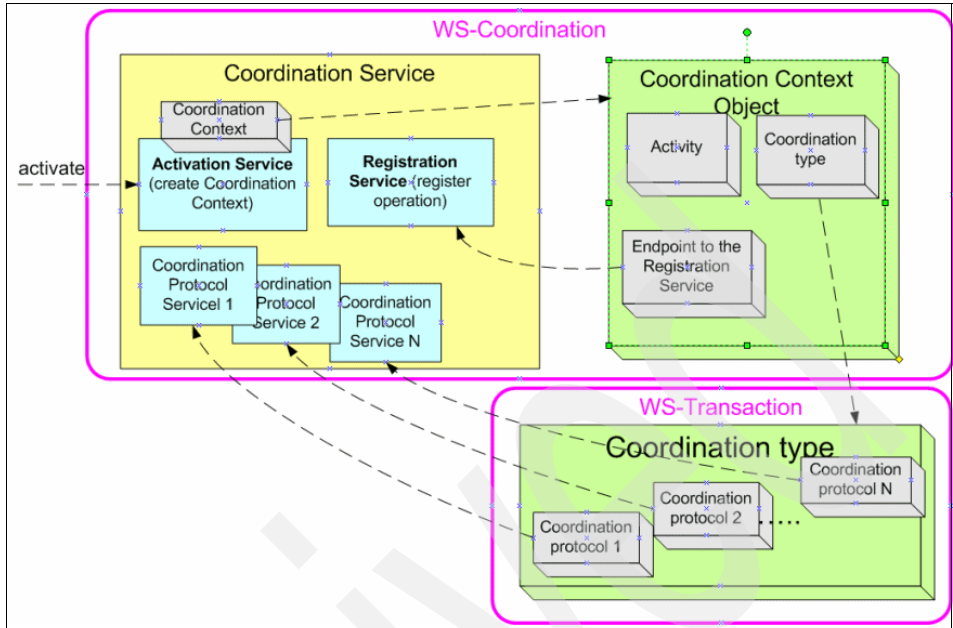


Figure 8-16 WS-Coordination and WS-Transaction objects

Currently, neither the WebSphere nor the Microsoft platforms implement the WS-Coordination standard.

8.6 WS-Transactions

WS-Transaction specification extends the WS-Coordination defining coordination types. A coordination type is a defined set of coordination behaviors, called coordination protocols specifying how the Coordinator should complete the task or the process. Coordination types use the WS-Coordination framework to define rules which both the Coordinator and participants must adhere to during their communications.

Current WS-Transaction specification defines two coordination types:

1. Atomic Transaction (AT)
2. Business Agreement (BA)

Each of these coordination types contain a number of coordination protocols. For example, the Atomic Transaction coordination type contains the following coordination protocols:

- ▶ Completion
- ▶ CompletionWithAck
- ▶ Volatile2PC (phaseZero protocol)
- ▶ Durable2PC (two-phase commit protocol)
- ▶ OutcomeNotification

At the time of writing this book, a WS-AT implementation is provided for WebSphere 5.1 as technology preview downloadable from IBM alphaWorks site:

<http://www.alphaworks.ibm.com/tech/wsat>

As detailed in the previous section, WS-Coordination is a Web service itself. This means that both client and server using SOAP messages for requests and response do not depend upon knowing each others development environment. Therefore WS-AtomicTransaction must be able to interface with any other transaction service coded using any programming language which supports WS-AtomicTransaction.

Interoperability of WS-AtomicTransaction across transaction services and programming languages was shown at a demo hosted by IBM and Microsoft. The demo application architecture is shown in Figure 8-17 on page 181. A Microsoft .Net application server beginning a non-JTA transaction making Web Service invocations to two WebSphere Application Servers and another Microsoft .Net server. Each of the application servers use their underlying transaction service to perform transactional work. Every time you invoke a Web Service you switch to using WS-Transaction. When the originator completes the transaction, you use the WS-Transaction technology to coordinate each of the participants to ensure that they all complete as if they were a single unit of work.

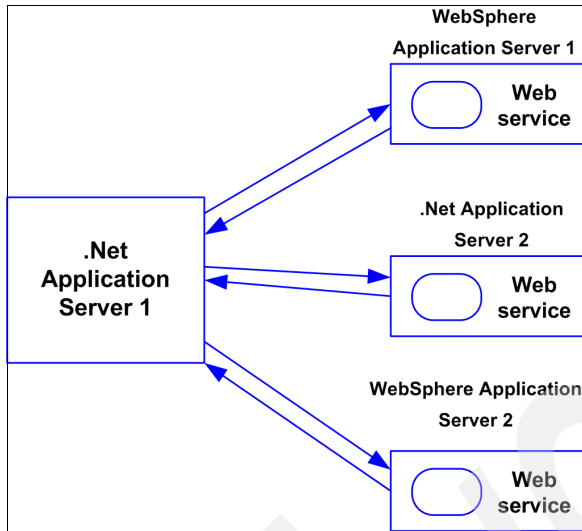


Figure 8-17 WS-AtomicTransaction interoperability demo between WebSphere and Microsoft .Net

8.6.1 WS-Transaction in a WebSphere environment

The WS-AT for WebSphere Application Server is a technology preview that provides transactional support for Web services. It allows distributed Web Service applications, and the resources they use, to take part in distributed global transactions. A transaction is a set of operations that must be executed as a single unit, often called a logical unit of work. A transaction is either completed in its entirety or not at all; it is indivisible or “atomic.”

WS-AT for WebSphere Application Server uses the standard JTA support in the J2EE programming model to scope transactions. JTA transactions are interpreted by the WS-AT for WebSphere Application Server runtime environment into CoordinationContexts such that a WS-AT representation of the current JTA transaction is made to flow upon Web service application requests.

If WS-AT for WebSphere Application Server is the system hosting the target endpoint, it automatically establishes a JTA transaction in the target's runtime environment, which becomes the transactional context under which the target Web service application will run. When the Web service request enters the target server, WS-COOR is used to register for participation in the 2PC protocol. The 2PC protocol is driven by the caller's WS-AT coordinator at completion of transaction.

No explicit registration of participants is required by the application developer. The WebSphere Application Server runtime environment takes responsibility for the registration of WS-AT participants, in the same way as it does the registration of XAResources in the JTA transaction to which the WS-AT transaction is federated. When the transaction is completed, all XAResources and WS-AT participants are atomically coordinated by the WebSphere Application Server Transaction Manager.

If a JTA transaction is active on the thread when a Web Service Application request is made, the transaction is propagated across on the SOAP/HTTP request and established in the target's environment. This is analogous to the distribution of transaction context over IOP as described in the EJB specification. Any transactional work performed in the target environment becomes part of the same global transaction.

8.6.2 WS transaction in a Microsoft .Net environment

Please refer to the IBM Redbook *WebSphere MQ Solutions in a Microsoft .Net Environment*, SG24-7012 for details about Web Services transactions implemented in Microsoft .Net platform.

8.7 Reliable messaging

Reliable messaging is important for Web services. Businesses need to be sure that critical message exchanges can be completed without loss, compromise or duplication of messages, and that messages have been delivered to the right recipient, in the right order and by a certain time.

Currently the burden is placed on Web service applications to ensure that exchanges are successfully completed. But this puts a lot of complexity onto the shoulders of the application writer. Assuring message delivery needs cooperation between the sender and receiver, part of that complexity involves getting agreement about how to acknowledge message delivery. Without a reliable messaging standard businesses are faced with designing and maintaining different reliable messaging protocols for different Web services.

One solution is the WS-ReliableMessaging specification from BEA, IBM, Microsoft and TIBCO. The specification is still under review and evaluation. In common with some other WS-* specifications that are being jointly developed the vendors hold regular workshops to thrash out a practical implementable specification and to demonstrate its interoperability. The last WS-ReliableMessaging workshop was held in May 2004. See <http://www-106.ibm.com/developerworks/offers/WS-Specworkshops/ws-rm200405.html> for details of this workshop.

There is an alternative Reliable Messaging specification being developed through OASIS by a number of other vendors. Currently the two protocols do not interoperate.

There is also the tactical solution of binding SOAP to a reliable messaging transport such as JMS. We look at SOAP/JMS in the next section of this chapter.

8.7.1 What is WS-ReliableMessaging?

WS-ReliableMessaging is part of the Web services solution for delivering messages reliably between enterprises. The intention of the WS-ReliableMessaging specification is to be composable with other WS-* specifications such as WS-Addressing, WS-Security, WS-Transaction, WS-Policy to enable vendors to deliver a complete solution to the overall objective of delivering messages reliably. As well as being efficient and interoperable between vendors, the challenge for the architects defining the ReliableMessaging specification is that it can be combined with these other WS-* specifications to meet the goals of a reliable enterprise messaging solution in support of a Service-Oriented Architecture.

The scope of the WS-ReliableMessaging specification itself is limited to defining protocol between sender and receiver endpoints that ensures delivery. The protocol is based on the three legged handshake protocol, adjusted to meet various different assurances of delivery.

- ▶ At-least-once delivery
- ▶ At-most-once delivery
- ▶ Exactly-once delivery (= At-least-once + At-most-once)
- ▶ In-order-delivery

8.7.2 The three legged handshake protocol

The three legged handshake protocol is simple because it doesn't require any coordinator, unlike the WS-Transaction specifications.

Figure 8-18 on page 184 shows how the protocol is used to establish a session between the sender and the receiver. The goal of the protocol is to establish a point in the interaction for each participant when a participant knows both its and its partners state. Either participant can then proceed with further interactions knowing that it has established a unique session with its partner.

In the diagram, this point is represented by the dark dotted line; more precisely, for the sender in Figure 8-18 on page 184 it is point *1a* and for the receiver, point *1b*. The session can be torn down by either party. In the example the sender knows at point *2a* that the session is deleted, and the receiver at point *2b*.

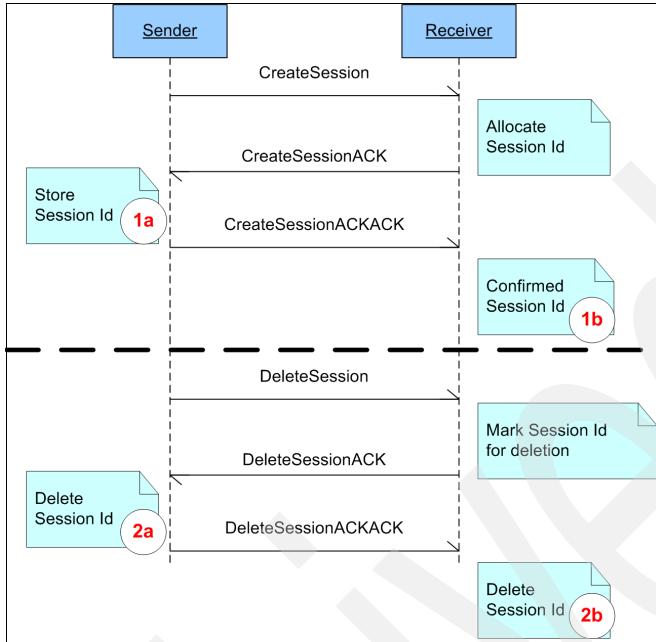


Figure 8-18 Three legged handshake

8.7.3 WS-ReliableMessaging Protocol

Figure 8-19 on page 185 shows use of the three legged protocol model to send a message exactly once using the WS-ReliableMessaging protocol. Before the protocol can start the endpoints need to be established, their mutual capabilities checked, and security exchanges completed to establish trust. Then the protocol steps are:

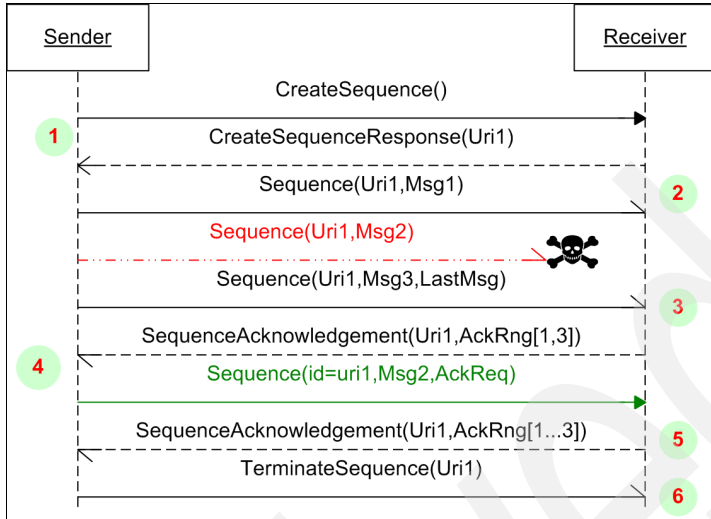


Figure 8-19 WS-ReliableMessaging protocol

1. The sender has sent a `CreateSequence` request and received a unique sequence token in the form of a URI. This is the `CreateSessionACK` in the three legged handshake model confirming the receiver is ready to play.
2. The `sessionACKACK` (`Sequence(Uri1,Msg1)`) is signalled by labelling the first message in the sequence 1. Each of the messages has a unique sequence number within the sequence
3. The receiver receives message 3 (`Sequence(Uri1,Msg3,LastMsg)`) with a `LastMsg` flag. This is effectively a `DeleteSession` request from the sender. It signals transfer is complete.
(In the example, message 2 was sent and not received).
4. The receiver responds listing all the messages it has received (`SequenceAcknowledgement(Uri1,AckRng[1,3])`) - This is effectively the `DeleteSessionACK`
The sender notes that message 2 was not received. It responds by re-sending message 2 again requesting an ACK, rather than sending a `TerminateSequence` (a `DeleteSessionACKACK`).
5. The receiver responds, indicating the range of messages it has received
6. This time, the sender is happy the receiver has got all the messages it sent and sends the receiver a `TerminateSequence` (a `DeleteSessionACKACK`). Both participants have now ended the exchange.

8.7.4 Reliable messaging requirements

IBM and Microsoft have published a white paper “Reliable Message Delivery in a Web Service World: A Proposed Architecture and Roadmap”, March, 2003 found at <http://www-128.ibm.com/developerworks/library/ws-rmdev/>. The white paper examines a scenario based on a distributor who wants to transfer inventory and account information to a supplier using Web service over the Internet. Based on this scenario five specification requirements are identified,

1. Transferring responsibility for reliable message delivery from the application developer to the Web service infrastructure.
2. Load balancing in a cross organizational context using WS-Addressing
3. Dealing with unreliable message delivery by using WS-ReliableMessaging
4. Dealing with different systems capabilities by exchanging meta-data between the Organizations
5. Dealing with message confidentiality and authenticity using WS-Security specifications
6. Dealing with system availability and peak load problems using WS-TransmissionControl

An implementation of an enterprise strength reliable messaging solution has to consider other requirements that are beyond the scope of the reliable messaging specification:

1. Providing a persistence mechanism to satisfy the delivery assurances
2. Increasing availability of Web services by de-coupling the Web service requester and provider from the cross-organizational messaging process.
3. Increasing throughput by multiplexing Web service requests between different requesters and providers onto single network and reliable messaging protocol sessions.
4. Reducing the application’s burden of managing undeliverable messages by transferring responsibility for undeliverable messages to the infrastructure
5. Increasing availability by Transferring a stalled “in-flight” message sequence to an alternative machine - not necessarily to a hot-failover on the same network transport or address.
6. Including the dispatch and receipt of the Web service within two different transactional contexts so that the end-to-end delivery of data can be fully transactional.

The WS-ReliableMessaging specification sets out to address requirements one and three. The WS-ReliableMessaging workshops look at the composition of Web services to tackle requirements one to five by testing interoperability of solutions.

Step 1. Transfer responsibility for delivery to the infrastructure

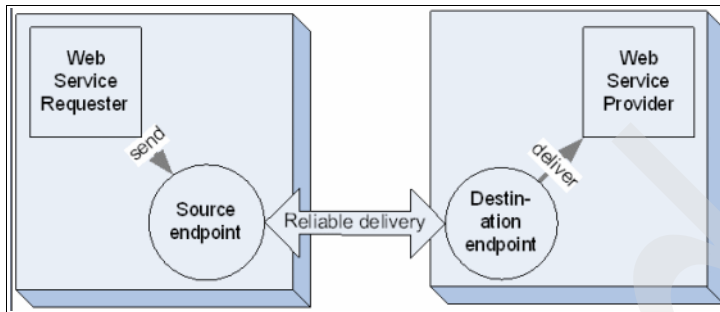


Figure 8-20 WS-ReliableMessaging endpoints

The reliable messaging model introduces a reliable messaging source and destination endpoint. The application requester and provider use the guarantee that a message sent to the reliable messaging source will be delivered to the destination endpoint. The responsibility for managing the exchange is taken on by the reliable messaging nodes and not by the application requester and provider.

Step 2. Load balancing

One of the characteristics of the WS-ReliableMessaging protocol is that it sets up a session between the source and destination endpoints. In a load balancing environment the destination endpoint node that receives the createSequence request may want to pass the request onto a different machine which will then handle the rest of the session. An affinity is created between the source and destination endpoint that must be respected by routing subsequent interactions within the protocol to the same destination endpoint.

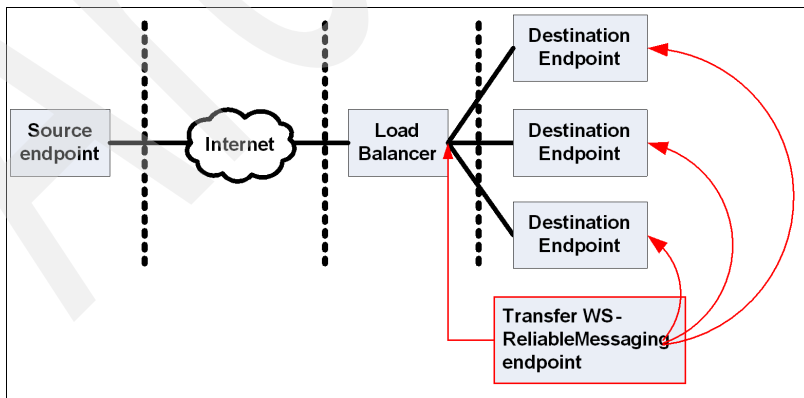


Figure 8-21 Using WS-Addressing to transfer WS-ReliableMessaging endpoints

The affinity management needs to have the same persistence qualities as the messages that are being transferred - that is, survive session, machine and storage failure. WS-ReliableMessaging can be combined with WS-Addressing to generate and pass endpoint address information inside the SOAP envelope with the same quality of service as the message itself.

The load-balancing node selects a new destination endpoint on the createSequence interaction, and inserts the address of the endpoint in a WS-Addressing header in the SOAP message. Interactions between sender and receive pass this address backwards and forwards to route subsequent messages to the right destination endpoint node.

8.8 SOAP/JMS and SOAP/MQ

WS-ReliableMessaging is still undergoing development and evaluation. There are already Web services applications in production that depend on a reliable SOAP transport. They are using SOAP over WebSphere MQSeries, or over JMS implemented either by WebSphere MQSeries or other messaging providers. There is a redbook, *WebSphere MQ Solutions in a Microsoft .NET Environment*, SG24-7012 that demonstrates how to configure an WebSphere MQSeries transport and SOAP handler for a WebSphere Application Server, Microsoft .Net and standalone WebSphere MQSeries environment.

The WebSphere MQSeries SOAP support is currently delivered as a Supportpac available from:

<http://www-3.ibm.com/software/integration/support/supportpacs/individual/ma0r.html>

The overall architecture is shown in Figure 8-22 on page 189. The implementation uses the standard SOAP engine in the environments in which it operates, exploiting the pluggability of SOAP engines.

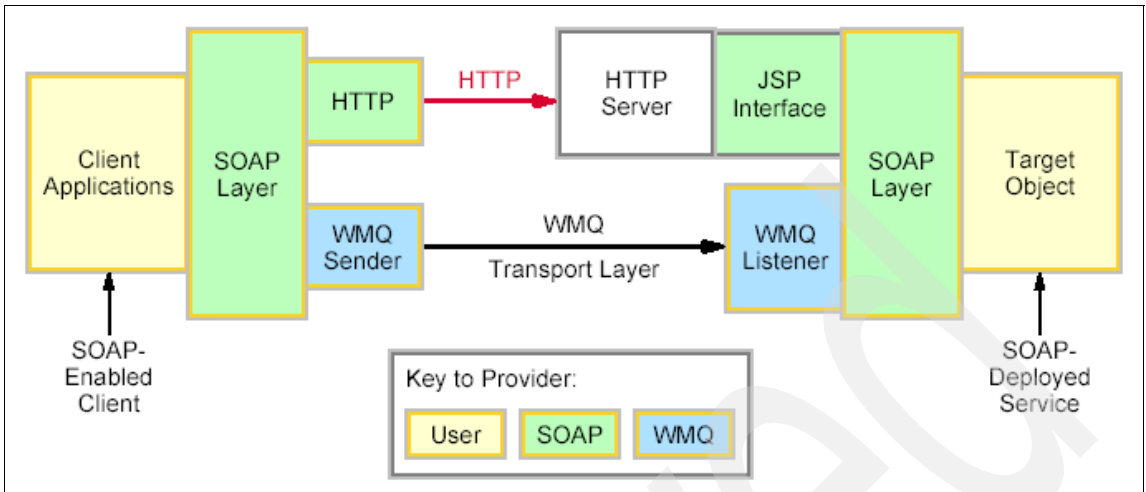


Figure 8-22 Overview of WebSphere MQSeries transport for SOAP

The solution is fully interoperable: SOAP messages can be flowed between WebSphere Application Server and Microsoft .Net using WebSphere MQSeries.

8.8.1 Interoperability of SOAP/JMS and SOAP/MQ

WebSphere MQSeries has already solved the problem of providing a robust message transport between diverse platforms. By using a single vendor to supply the SOAP bindings to the messaging layer at both the client and server one can build an interoperable, robust SOAP solution.

A word of caution, however. This is interoperability between platforms, not between vendors. If the focus is on creating a Web service that can be published and deployed just like any other, but with a robust transport option, then we do run into an interoperability problem. Today, there is no standard way to define a SOAP/JMS or SOAP/MQ binding, or map the resulting WSDL to a SOAP envelope implemented as a message. Both ends of the Web service have to use the same vendor to map the SOAP message to the SOAP stack at either end as well as using the same messaging transport. With the ubiquity of WebSphere MQSeries and the possibility of using JMS with an alternative messaging provider the use of a common messaging transport is generally acceptable. However the requirement to use the same vendor to map the Web services stack is an inhibitor to deploying SOAP/JMS or SOAP/MQ outside the enterprise to connect business partners - such as the Claims Assessors in our scenario.

The solution is to bind SOAP messages to messaging providers in a common format. That way the requester and provider can be developed independently,

and only when the solution is deployed does a decision need to be made on which common message provider to employ. The question is how to define a message mapping common to different proprietary messaging providers.

Figure 8-23 shows the problem trying to get WebSphere Application Server and another SOAP server to interoperate using SOAP/JMS. Without a common SOAP binding, the WebSphere Studio Application Developer and “A. Tool” are going to implement the SOAP binding differently. As a result the solution doesn’t interoperate.

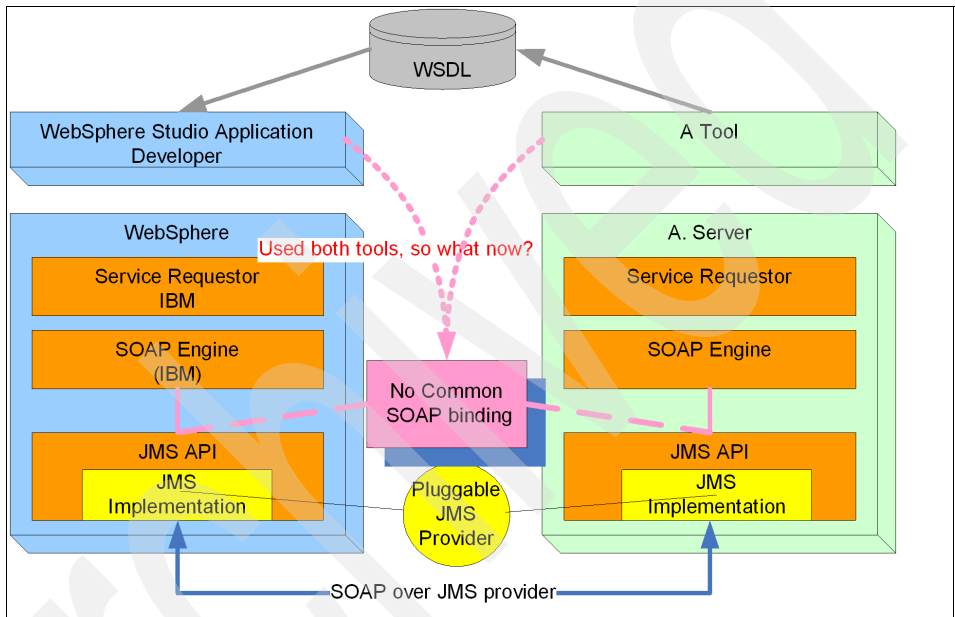


Figure 8-23 SOAP/JMS Interoperability problem

The problem is currently being studied by a number of customers and vendors. Solving the problem for SOAP over JMS is one way forward. Rather than defining the binding of SOAP to proprietary message formats, the SOAP/JMS binding would define the mapping of SOAP to the JMS API. The implementation of the SOAP/JMS layer would then be independent of the JMS transport provider.

Web services in Microsoft .Net and WebSphere

In this chapter, we describe the architecture and implementation of Web services in Microsoft .Net and WebSphere 5.1.2 Java 2 Enterprise Edition environments. In a separate section at the end of the chapter, we look at the implementation of secure Web services in Microsoft .Net and WebSphere Application Server.

This chapter is a brief summary. Both IBM and Microsoft have written books comparing J2EE and .Net. They are both available online:

- ▶ IBM Redbook: *WebSphere and Microsoft .Net Coexistence*, SG24-7027, found at:
<http://publib-b.boulder.ibm.com/abstracts/sg247027.html?open>
- ▶ Microsoft, Patterns and Practices series, *Application Interoperability Microsoft .Net and J2EE*, found at:
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/jdni.asp>

9.1 Microsoft .Net architecture

Web service in Microsoft .Net uses the Microsoft Distributed Internet Applications (DNA) design patterns and architecture. This DNA architecture partitions a Web service application into three different layers, Presentation, Business Logic and Data layers. Each layer exposes its interface to the next layer such that they are loosely coupled with one another.

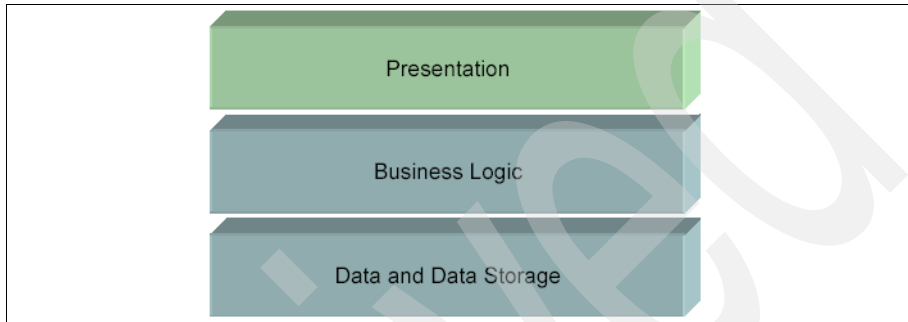


Figure 9-1 DNA three-layer architecture of Web service application

Many different parts make up the DNA architecture. The Windows operating system, COM+ components, Internet Information Services, Active Server Pages, and SQL Servers can be used in combination to implement a Microsoft .Net Web service application. The Internet Information Services is a Web server and application server that supports hosting of dynamic Web applications. It performs functions similar to both WebSphere Application Server and Apache Web Server. Active Server Pages are HTML pages with embedded Visual Basic scripts and they are similar to Java Server Page (JSP) with its embedded tags in the HTML pages. COM+ Components can be written in different languages such as Visual Basic or Visual C++ and they provide business logic components for a Web service application. Active Data Objects (ADO) is a set of COM objects used for accessing relational databases. SQL Server provides the Relational Data Base Management System including storage and management of data.

Microsoft built the Microsoft .Net framework based on the DNA architecture and its Windows operating system with enhancements to include support of different new features. Microsoft .Net encapsulates the Windows operating system and its quality of service mechanisms using industry specifications such as WS-I Profile 1.0 and others involved in Web services. It also provides a runtime environment for application software with services like garbage collection, exception management, transaction management and namespace support. There is a rich framework of useful classes to develop robust enterprise applications.

As shown in Figure 9-2, the way Microsoft .Net provides these values is through implementation of a Common Language Runtime and the Microsoft .Net framework on top of COM+ and the Windows operating system. The purpose of the Common Language Runtime is to provide language independence and execution code management. It is similar to a JVM. It has the capability of

- ▶ JIT compilation of Microsoft Intermediate Language (MSIL) to native code
- ▶ Support of multiple languages (Visual Basic .Net, C#, Managed C++, JScript, J#, Perl, Eiffel, Python, Pascal, FORTRAN)
- ▶ Thread, exception and memory management
- ▶ .Net Remoting
- ▶ Garbage collection
- ▶ Security, including code signing and using “Strong Names” to overcome the problem of DLL conflicts where different programs use different versions of the same DLL. Multiple versions of the same DLL can be loaded at the same time.
- ▶ Runtime type checking supporting cross language type checking
- ▶ Debugging - bring up a debug dialog in a runtime system when code is not being run in a development debugging environment.

The .Net framework provides access to the underlying quality of service, consuming Web services, .Net Remoting and other features.

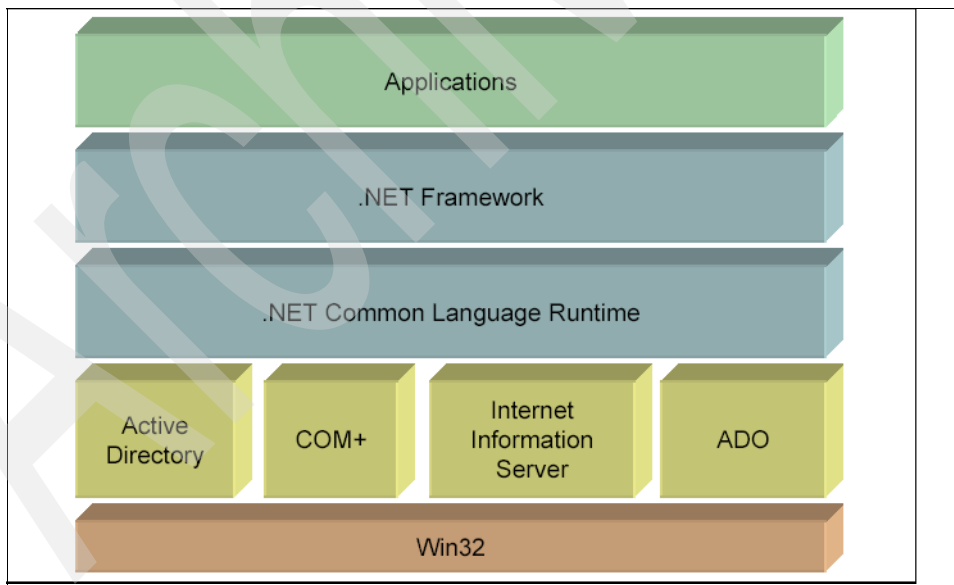


Figure 9-2 Microsoft .Net framework and runtime enhances the DNA components

Microsoft has also enhanced the supporting DNA components. Microsoft has released Windows Server 2003 and upgraded COM+ from v1.0 to v1.5 and calls

it *Enterprise Service* under Microsoft .Net. It has enhanced ADO and named it ADO.NET. It has upgraded IIS from V5.0 to V6.0. Microsoft has built Microsoft .Net from scratch to take full advantage of the Windows operating system.

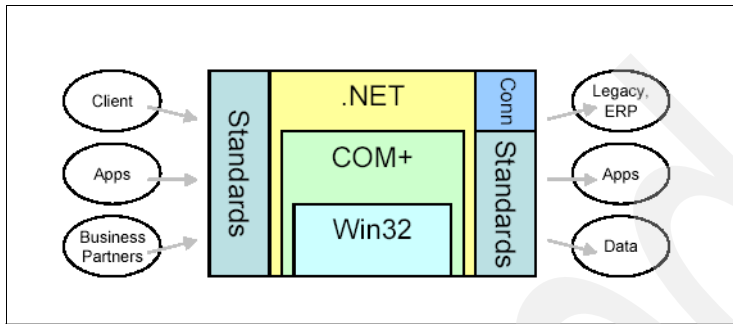


Figure 9-3 The building block of Microsoft .Net

Microsoft .Net is very tightly coupled with the Windows operating system, while allowing different languages to be used to develop the Web service in its Microsoft .Net Framework. We can use Visual Basic .NET, Visual C#, J#. and other languages to write Microsoft .Net applications. Microsoft .Net makes use of the underlying runtime and Quality of Service mechanisms such as COM+ and provides the implementation of standard interfaces such as Web-based access via HTTP, XML and Web services in the Windows platform.

9.1.1 Microsoft .Net Web service application architecture

The Microsoft .Net Web service application architecture is a standard three tier model with presentation, business and data layers.

Presentation layer

The presentation layer interacts with the user. It consists of the visual forms that are created in Active Server Page .NET (ASP.NET) with embedded tags and client-side scripting such as VBScript or JScript. In Microsoft .Net, this Web form contains a code-behind page with extension of .aspx.cs for C# and .aspx.vb for Visual Basic .NET. The forms are populated with drag-and-drop widgets such as buttons, text and labels from the toolbox of the Microsoft Visual Studio .Net 2003. The Web form in ASP.NET has the extension of .aspx.

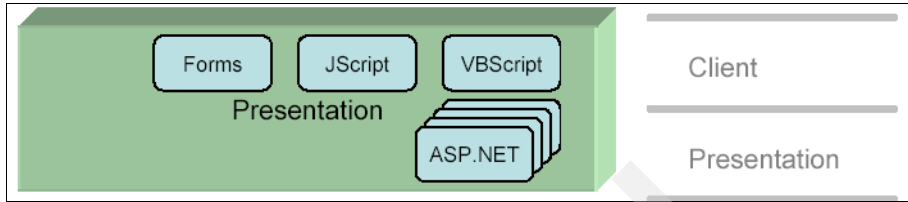


Figure 9-4 Presentation Layer with forms created in ASP.NET and scripts

Business layer

The business layer provides services to the presentation layer. These services wrap business objects running under the Microsoft .Net Common language runtime. They can also wrap COM+ components in the Windows operating system without the support of the Microsoft .Net Common Language Runtime. These objects can be created using Visual Basic Microsoft .Net, C# or other languages. They are classes containing methods and variables. If they are managed code¹ they also have attributes to extend their functionality to use the services provided by the Microsoft .Net Framework. If the classes are not compiled into managed code, they make use of the services and libraries provided by the Windows operating system. Classes written in C# have the file extension .cs. If they are written in Visual Basic Microsoft .Net, they have the file extension of .vb.

When business objects reside in different machines, Microsoft .Net uses .NET Remoting to facilitate the invocation between them. .NET Remoting is similar to Java 2 Enterprise Edition Remote Method Invocation (RMI/IIOP) for invoking objects in different machines. .NET Remoting supports the invocation of remote objects via SOAP or of a proprietary binary over TCP/IP. Objects within the business layer can work with each other in various layers of abstraction, such as providing a service via an XML Web service interface.

¹ What is managed code? Managed code is written in one of many high level languages and is compiled into IL (intermediate language). It executes in a managed execution environment that ensures type safety, array bound and index checking, exception handling, and garbage collection very similar to Java running in a JVM. See

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_m/directx/what_ismanagedcode.asp

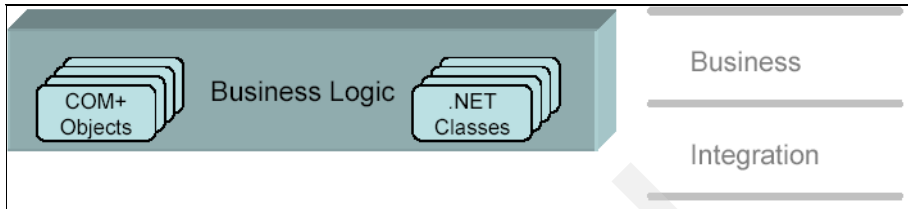


Figure 9-5 Business Layer consists of COM+ or Microsoft .Net objects

Data layer

The data layer abstracts data access to the business layer by providing data access services to the business layer. The data access service can include access to databases or other resources such as queues or resource adapters to legacy applications. The data layer can use Microsoft SQL Server as the database management system or other database through ODBC data sources or using the Microsoft JDBC driver. Microsoft .Net uses ADO.NET to access the data in the database. In ADO.NET, we have connection, dataset, data adapter, which allows disconnected access to the database.

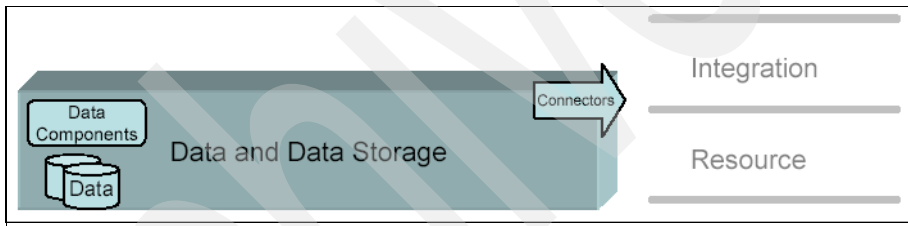


Figure 9-6 Data layer

9.1.2 Developing software using Microsoft Visual Studio .Net 2003

Microsoft Visual Studio .Net 2003 organizes development work in projects. The opening panel gives us a choice of different types of project and different types of language types to work on, such as building class library or an ASP.NET Web service using C#:

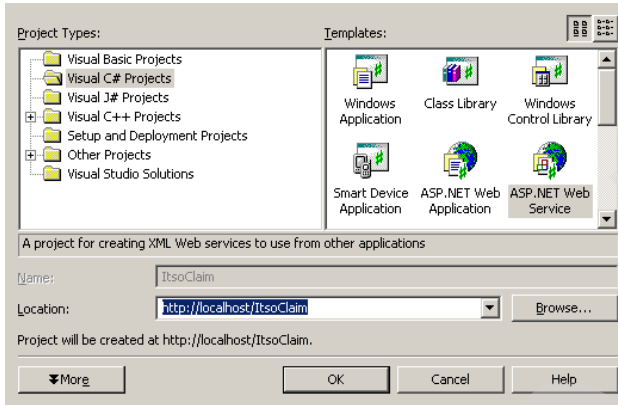


Figure 9-7 Selecting a language and project type with Microsoft Visual Studio .Net 2003²

Developing an ASP.NET Web service

Having decided to build a ASP.NET Web service project type, Microsoft Visual Studio .Net 2003 prompts you by bringing up the component designer,

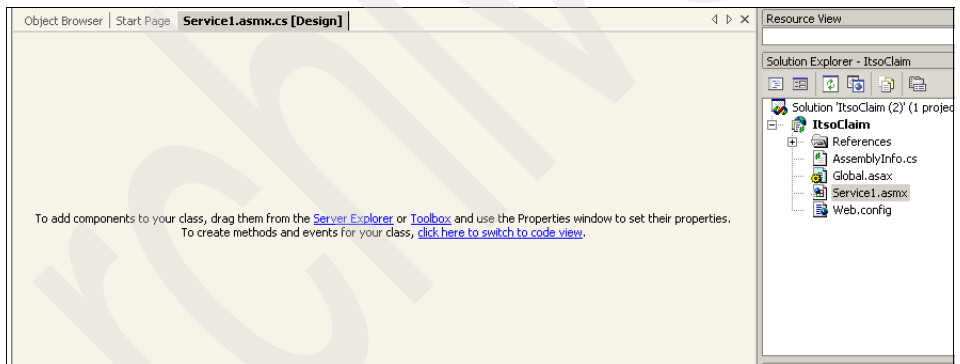


Figure 9-8 Component design in Microsoft Visual Studio .Net 2003

This creates a default Web service entry point (service1.asmx) and a form (see Figure 9-8) to add components, or to switch to the code view to start writing code - typically kept in a .asmx.cs “code behind” file.

In Microsoft Visual Studio .Net 2003 the development and production environments are the same. This is different to WebSphere Studio Application Developer Rather which embeds test servers in the development environment. As a consequence, Internet Information Service must be running to develop a

² This and other screenshots from Microsoft products are reprinted by permission from Microsoft Corporation

Web service project because the visual studio wizard also creates and deploys the project on the Web server at the same time as it is created in Microsoft Visual Studio .Net 2003.

As with the rest of the .Net framework the development of a new Web service is highly integrated with the rest of the Windows operating system and relies upon setting up the Windows to support separate development, testing and production environments - for example by using access control to restrict those who can modify the development, test and production environments.

Web services Service Interface pattern

The pattern of integration used by Microsoft Visual Studio .Net 2003 to build interoperable Web services is the *Web services Service Interface pattern*, found at:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/DesServiceInterface.asp>

This is like a Patterns for e-business [P4eb] Runtime pattern, and we show it using the [P4eb] notation below.

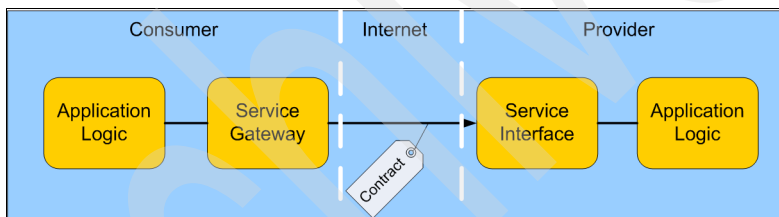


Figure 9-9 Service Interface pattern

Microsoft Visual Studio .Net 2003 provides a code template to implement the Service Interface. Each method in the class that is exposed as a Web service should have a `Public` declaration and be marked with the attribute `[WebMethod]`.

ASP.NET handles the creation and parsing of SOAP Web service requests and responses.

Microsoft Visual Studio .Net 2003 Web services client

Just as Microsoft Visual Studio .Net 2003 provides a project template for a Web services service, it also has a project template for a Web services client.

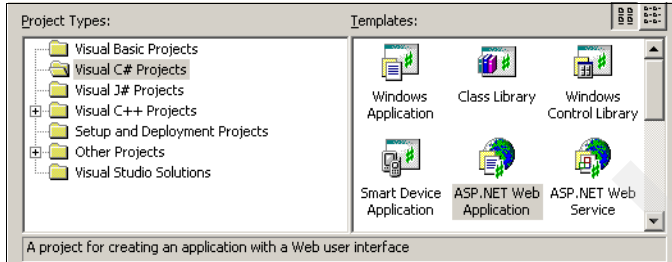


Figure 9-10 Creating a Web services client using Microsoft Visual Studio .Net 2003

The template generates a Web service proxy class and uses .NET Framework for all the Service Gateway coding.

The next step is to add a Web reference to the project. **Add Web reference...** is a right-mouse click in the solution explorer and brings up a Web service browser:



Figure 9-11 Microsoft Visual Studio .Net 2003 Web services browser

Once a Web service is selected Microsoft Visual Studio .Net 2003 automatically generates the client Web service proxy class. This class should not be customized in case the Web reference is updated which will cause a new proxy class to be generated overwriting the old one.

Alternatively if the Web service is not accessible at the time of development Microsoft Visual Studio .Net 2003 provides a tool, WSDL.exe to create the proxy class from WSDL. This has the advantage you can customize the proxy class as its management is under manual control.

The proxy class namespace must be added to the client application, and the client code needs to create an instance of the proxy class and then invokes the appropriate methods in the proxy class.

By default the URL of the Web service is statically coded into the client proxy. By changing the URL behavior of the Web reference to Dynamic the URL is read from the application's configuration file.

9.1.3 Microsoft secure Web services implementation

In this section we will examine the Microsoft .Net method of building secure Web services and what standards are implemented by Microsoft Web Service Enhancements v2.0.

Building secure Microsoft .Net Web services

When we use Microsoft .Net to write a secure Web service, all of the security configuration is done in the code itself. We do not have the split between code and configuration files that exists when writing a Java 2 Enterprise Edition Web service. Let's have a look at an example Web service client, written in Visual Basic .NET:

Example 9-1 A Visual Basic .NET Web service client sending an encrypted message

```
Dim WebService As New XYZElectronics.placeOrder
Dim encryptionCertificate As X509Certificate

Dim store As X509CertificateStore
store = X509CertificateStore.CurrentUserStore(X509CertificateStore.MyStore)
store.OpenRead()

For Each cert As X509Certificate In store.Certificates
    If (cert.GetName.IndexOf("XYZElectronics") > -1) Then
        encryptionCertificate = cert
    End If
Next cert

Dim encryptionSecurityToken As New X509SecurityToken(encryptionCertificate)
Dim encryptedData As New EncryptedData(encryptionSecurityToken)
WebService.RequestSoapContext.Security.Elements.Add(encryptedData)

WebService.placeOrder(Input.Text)
```

This client begins by creating an instance of the XYZElectronics.placeOrder class, which has been generated at development time from the WSDL of the Web service. The client then searches through the certificate store on the machine it is running on and locates the public key certificate of XYZElectronics. It uses this to create an X509SecurityToken, and from this, an EncryptedData object. It then adds this EncryptedData object to the security elements in the

outgoing SOAP message. (A message sent from a client to a Web service is often called a request, the one returned is a response.)

Microsoft Web Service Enhancements V2.0

Microsoft Web Service Enhancements v2.0 implements WS-Security 2004, WS-Policy, WS-SecurityPolicy, WS-Trust and WS-SecureConversation. It was released before the WS-I Security Profile was drafted, so it does not implement this. However, because it implements WS-Security 2004, we should expect reasonable interoperability between secure Web services built with it and Java Web services built with Rational Application Developer v6.0.

For further information see the article *WS-Security Drilldown in Web services Enhancements 2.0* by Don Smith:

<http://msdn.microsoft.com/webservices/building/wse/default.aspx?pull=/library/en-us/dnwse/html/wssecdrill.asp>

9.2 WebSphere Java 2 Enterprise Edition architecture

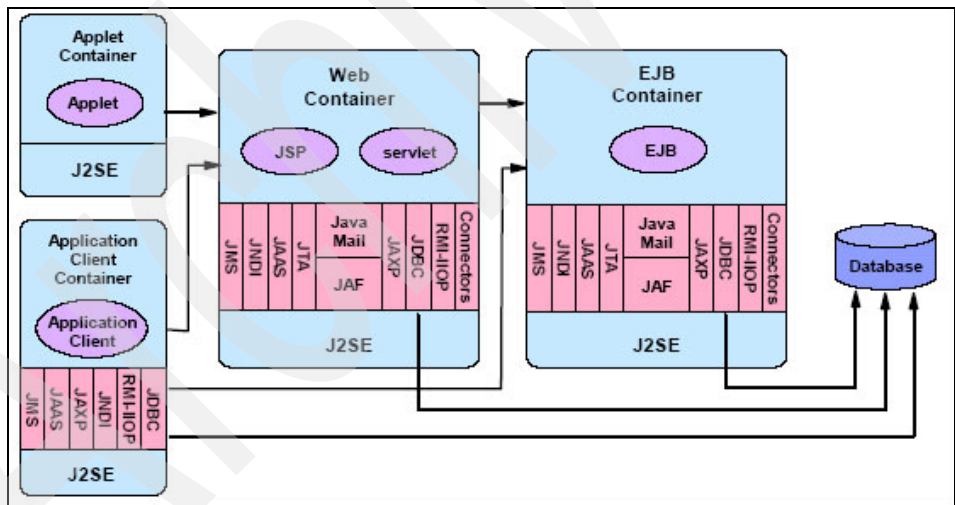


Figure 9-12 Java 2 Enterprise Edition containers, components and services

In the Java 2 Enterprise Edition programming model, there are application client and applet containers, and the Web and Enterprise JavaBean (EJB) containers.

The application clients have access to the services of the Java 2 Enterprise Edition application client container. These services include Java Messaging Service (JMS), Java Authentication and Authorization Service (JAAS), Java for

XML Parsing (JAXP), Java Naming and Directory Interface (JNDI), Java Remote Method Invocation running over Internet Inter-Orb Protocol (RMI/IIOP) and Java Database Connectivity (JDBC).

Unlike the Java application clients, applets, typically running in a browser, have restricted access to system resources and are prohibited from reading and writing files, and making network connections except to the originating host.

The Web container components include Java Server Pages (JSP) and Java servlets. JSP pages are used to construct dynamic Web pages and Java servlets are implement control logic. JSPs implement the view and servlets the controller in the Model-View-Controller pattern of a Java 2 Enterprise Edition implementation.

The model component of the pattern is implemented by Enterprise JavaBeans running in the EJB container of the Java 2 Enterprise Edition application server. There are three types of EJBs - Session, Entity and Message driven beans (MDB).

Session beans can be stateless or stateful. In most design patterns stateless session beans are used to implement stateless objects and entity beans to implement stateful objects. Entity bean state (or persistence) can either be container managed (CMP) or bean managed (BMP). Persistence is normally implemented using a relational database connected using data sources which manage pools of JDBC connections.

Message Driven Beans use JMS or IBM WebSphere MQSeries to implement asynchronous patterns of interaction. Both point-to-point and publish-subscribe styles of messaging are supported, as well as multiple degrees of message persistence and performance.

The Java 2 Enterprise Edition defines the standardized external protocols used for process and application communication and integration. RMI/IIOP is used for synchronous connection between beans, and JMS for asynchronous connection. Java Connector Architecture (JCA) defines how non Java 2 Enterprise Edition applications connect to the application server. In JCA 2.0, supported in Java 2 Enterprise Edition 1.4 the connection can be initiated in either direction, and asynchronous connection, using JMS, is also supported. So for example, in a typical Integration pattern an Enterprise Information System (EIS) such as SAP can trigger a one-way message from SAP to the Java 2 Enterprise Edition server in response to a modification of an SAP business object. The application server can then coordinate updating other EISs that need to be synchronized with SAP.

Java 2 Enterprise Edition embeds the Java 2 Standard Edition Java Virtual Machine runtime environment in which all the Java code runs.

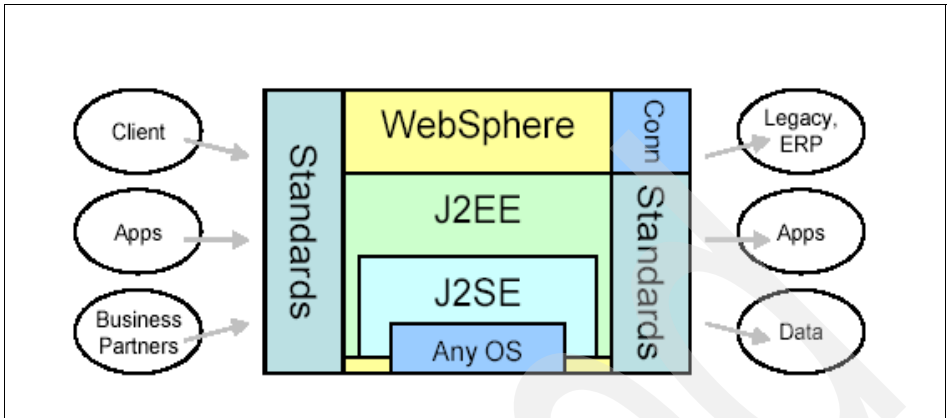


Figure 9-13 The building block of WebSphere Application Server

Java 2 Enterprise Edition Web service application development is restricted to the use of Java language, but allows development in different operating systems such as Unix, Windows, or IBM Z/OS.

WebSphere Application Server is the implementation of the Java 2 Enterprise Edition specification. It also includes IBM's extension to provide security and other binding services. There are a number of different versions of WebSphere Application Server providing different levels of capability that build upon one another.

Table 9-1 Capabilities of different versions of WebSphere Application Server

Server	key Capabilities
WebSphere Application Server Express	Dynamic Web pages, Web services,
WebSphere Application Server	Full Java 2 Enterprise Edition compliance and Web services support
WebSphere Application Server Network Deployment	Support for managing clusters (cells) of application servers for scalability and reliability. Support for publishing Web services using the Web service gateway and UDDI server.
WebSphere Application Server Extended Deployment	Support for administering multiple WebSphere Application Server ND cells

Server	key Capabilities
WebSphere Business Integration Server Foundation	Provides additional business integration capabilities and extensions to the Java 2 Enterprise Edition programming model that have are not in the current finalized level of the Java 2 Enterprise Edition specification (currently 1.4). For example BPEL4WS

WebSphere

9.2.1 Java 2 Enterprise Edition Web service architecture

JSR 101 defines the Java API for XML-based Remote Procedure Call (JAX-RPC) style programming model. It mandates both client and server side requirements.

JSR 101 formalizes the procedure for invoking Web services in an RPC-like manner in a Java programming environment. It is a required part of the J2EE 1.4 specification, but due to huge demand for this programming model, IBM also provides it on the J2EE 1.3 platform. This support has been provided out-of-the-box in WebSphere 5.0.2 and later.

JSR 101 provides for interoperability of the Web services Java API between different Web service vendors' development tools. As long as the implementations of JSR 101 are compliant to the SOAP interoperability specifications then clients and servers will interoperate with other Java and non-Java platforms such as Microsoft .Net.

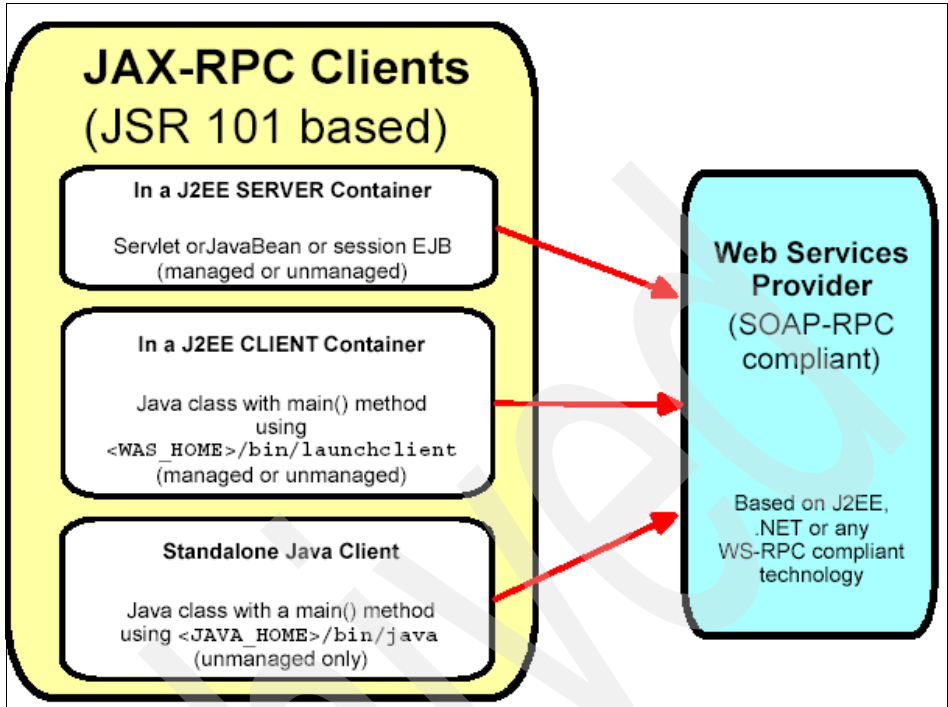


Figure 9-14 JAX-RPC clients interact with SOAP-RPC Compliant server

JSR 109 standardizes the process of deploying a Web service in a Java 2 Enterprise Edition platform to achieve interoperability and portability across different Java 2 Enterprise Edition compliant platforms. JSR 109 is only defined for the implementation of a stateless session EJB in an EJB container and Java class in a Web container. The specifications detail the programming model for Java 2 Enterprise Edition components with Web services, the assembly of the components with Web services and the deployment of these components as Web services components.

Java 2 Enterprise Edition Version 1.4 mandates JSR 101 Java API for XML-RPC (JAX-RPC) and conformity to the JSR 109 Web services implementation specification.

For more details about JSR 101 and JSR 109 refer to *WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook*, SG24-6891.

9.2.2 Developing J2EE applications using WebSphere Studio Application Developer

WebSphere Studio Application Developer 5.1.2 contains built-in Java 2 Enterprise Edition Version 1.3, but allows the flexibility to use Java 2 Enterprise Edition Version 1.4. WebSphere Application Server 5.1.1 conforms to Java 2 Enterprise Edition 1.3, and also implements parts of Java 2 Enterprise Edition 1.4 including JSR 101 and 109. Full 1.4 support is provided by WebSphere Application Server 6.0.

WebSphere Studio Application Developer 5.1.2 has different perspectives that ease implementation of the different Java 2 Enterprise Edition Web service layers. Figure 9-15 shows using the Java 2 Enterprise Edition perspective in WebSphere Studio Application Developer to edit different layers in the application.

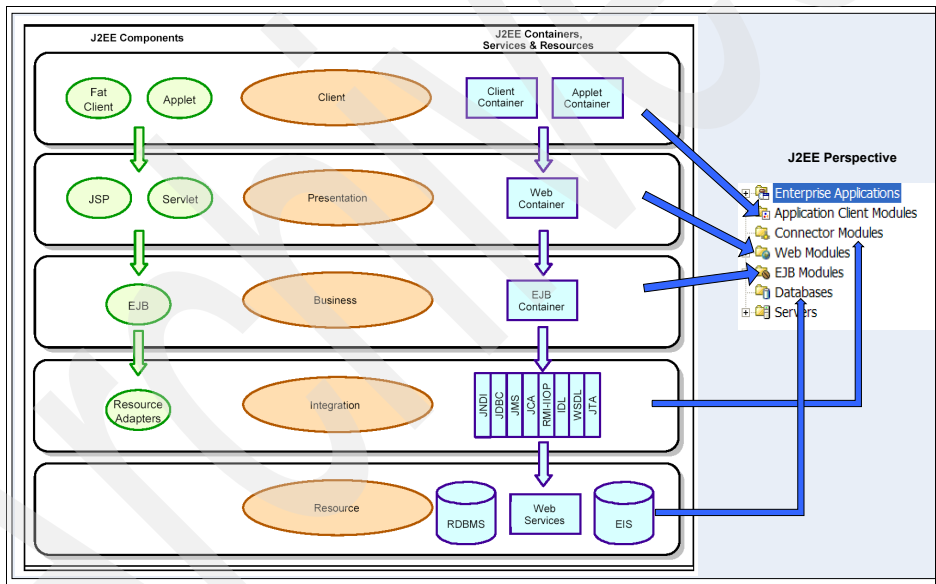


Figure 9-15 J2EE Logical Application layers mapped to WebSphere Studio Application Developer

Client

We can implement a simple Java application client or applet client to access the presentation layer. We use the WebSphere Studio Application Developer visual editor to create the client application, which can reside in the same as or different machine from the server.

Presentation layer

We usually use the Web perspective with the Java 2 Enterprise Edition perspective to develop Dynamic Web pages. We can use WebSphere Studio Application Developer Visual Editor to design text boxes, buttons and list boxes, creating the JSP and HTML pages. We can easily convert the EJB into Web service using the Web service wizard. WebSphere Studio Application Developer also provides the Universal Test Client and built-in UDDI registry for publishing Web service and testing the Web service application.

We use the web.xml file in Web-Content to configure the Web pages. The layer is packaged in the Web Application Resource (.war) file for distribution. WebSphere Studio Application Developer automatically wraps the .war file with the default Enterprise Application Resource (.ear) file. WebSphere Studio Application Developer also includes the Apache Struts framework for developing robust, loosely-coupled presentation layer by using the Model-View-Controller pattern, where the servlets are usually used as the controller or dispatcher.

Business layer

We use the Java 2 Enterprise Edition perspective in WebSphere Studio Application Developer 5.1.2 to assist in the implementation of the business layer. We use Session beans as the business facade to group different operations into a single interface and we use the entity beans to map class attributes to the data fields in the database. We implement asynchronous Web service using the JMS based Message driven beans with the support of either WebSphere MQSeries, the embedded WebSphere Application Server messaging or a third party messaging provider.

Integration layer

In the Integration Layer, WebSphere Studio Application Developer can be used to bridge to legacy resources such as CICS, IMS™ or Batch systems. The integration layer is where Java 2 Enterprise Edition services and other integration middleware such as message queueing software and enterprise information system connectors reside.

Resource/Data layer

WebSphere Studio Application Developer 5.1.2 has a data perspective which is used to automate development of the data layer. In WebSphere Studio Application Developer. We can connect to databases, develop schemes and develop SQL statements and stored procedures to access the database.

This layer includes the DB2® database, CICS, SAP, JDEdwards, and other resources which a distributed application may connect to them.

Developing a J2EE Web service from an EJB

WebSphere Studio Application Developer has wizards to build Web services from different types of applications. In 12.1.5, “Create a Web service from Enterprise JavaBeans” on page 262 we work through building a Web service from an EJB. The steps required are:

1. Select the level of WS-I compliance conformance required
2. Select the **New Web service** wizard.
3. Select the different attributes of the Web service that are required and how the Web service is to be tested
4. Select the EJB project and the EJB EAR file that contains the EJBs to be converted into Web services, and the “Router” project that will contain the resulting Web service.
5. Select the runtime environment it is to be targeted at
6. Select the EJBs to be included in the Web service.
7. Select the methods to be included in the Web service

The wizard will generate the Web service, issue any appropriate warnings about WS-I compliance, generate a test client, deploy the EAR file to the test server, and start the test server. There are no manual steps involved for EJBs using simple datatypes. The level of WS-I compliance is reported if requested.

Building a J2SE Web service client

Once the WSDL file for the Web service has been imported the method to build a standard Java client to call a Web service works regardless of where the WSDL file was obtained from. The steps are laid out in detail in 12.3, “Building the Web services clients” on page 293. In brief the steps are:

1. Create a new Java project
2. Import the WSDL file
3. Test the client using the Web services explorer
4. Select Java proxy as client proxy type and generate the proxy classes
5. Test the client proxy by launching the Run Java Application - the generated test code will only handle simple data types.

9.2.3 IBM secure Web services implementation

In this section, we begin by examining what different ways of building Web services we have available when using IBM products and how this affects their security configuration. We then go on to look at exactly what security standards are implemented in versions 5.1 and 6 of WebSphere Application Server.

Building secure Java Web services and clients

We first look at building a secure service provider, and then how to build a secure service requester.

Web services

A Java Web service, like any server application, runs in a Java 2 Enterprise Edition container. When we configure the security settings, we use the `webservices.xml` file. Thus we have a split between the Java code that is implementing the Web service, and the configuration files that specify the security settings. The Java code has no mention of security. We do not need to manually change this file, because WebSphere Studio Application Developer provides a graphical editor. The figure below shows us using the `webservices.xml` file to specify that incoming messages must be encrypted.

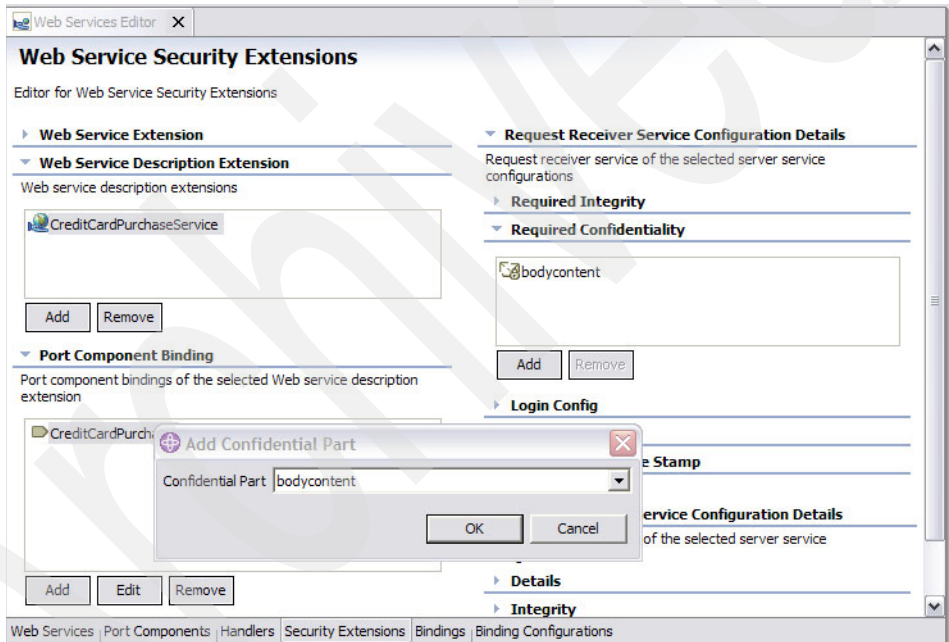


Figure 9-16 Using the `webservices.xml` file to specify that incoming messages must be encrypted

Web service clients

When writing Java Web service clients, we have a number of options to choose from, including:

- ▶ J2SE ServiceLocator client - this is the simplest type of client. It is created from code stubs created ahead of time and has the endpoint of the service hard coded into it. It is very easy to use, but is IBM specific.

- ▶ JAX-RPC ServiceFactory. This is the JSR 101 standard for J2SE Web service clients. There are three variants:
 - a. Static stub. A Service object is created at development time, with the service URL hard coded into it.
 - b. Dynamic proxy. The location of the Web service WSDL must be known at development time, but the proxy object is created dynamically at run time. This has the advantage that if the service definition has changed since the last use of the client, the appropriate proxy will be generated from the updated WSDL.
 - c. Dynamic Invocation Interface (DII). In this implementation, not only is the proxy object created dynamically at run time, but we do not even need to know the WSDL URL at development time. Rather, we can, for example, perform a UDDI lookup to locate the service information. This method is the most flexible, but also the most complex.
- ▶ Java 2 Enterprise Edition Container managed clients, as defined by JSR 109. This standard builds upon JSR 101. Hence we have the same client types available, but the clients are packaged into Enterprise Archive files (EARs), which also contain Java 2 Enterprise Edition deployment descriptors, such as the webservicelient.xml file. Several types of Java 2 Enterprise Edition clients are possible:
 - a. A Java class, which runs as a Java Bean in the Web container.
 - b. An Enterprise JavaBean (EJB), running in the EJB container.
 - c. An Application Client, running in the Application Client container.

So, if we wish to create secure clients, we must use Java 2 Enterprise Edition container management and configure their security using the webservicelient.xml file. This file is designed to be very similar to the webservicelient.xml file:

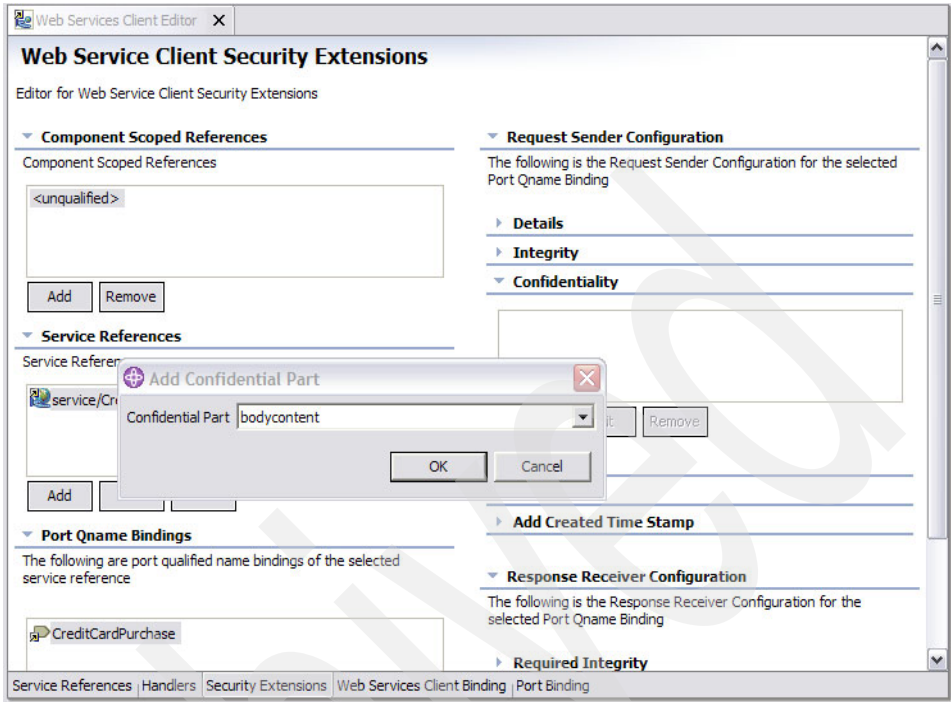


Figure 9-17 We configure client security in a similar way to Web service security

For a more detailed discussion of WebSphere client types, see *Invoking Web services with Java clients* by Bertrand Portier:

<http://www-106.ibm.com/developerworks/webservices/library/ws-javaclient>

For general information about Web services security configuration in WebSphere, see the WebSphere Studio Application Developer infocenter:

http://publib.boulder.ibm.com/infocenter/wasinfo/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/rwbs_index.html

WebSphere Application Server V5.1

WebSphere Application Server v5.1 and WebSphere Studio Application Developer v5.1 were released before the WS-Security 2004 specification was agreed. As such, their security implementation is based on the WS-Security draft. Although the original proposal was made in 2002, by 2003, OASIS was working on the draft. This means that the security namespaces have the value:

```
xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
```

As opposed to the namespace for WS-Security 2004:

```
xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-secext-1.0.xsd"
```

Several other namespaces are different. Of course, some functionality is also different between the two standards. Because of this, we should not expect interoperability of secure Web services with Microsoft Web services Enhancements v 2.0, since it implements the WS-Security 2004 standard. This will not affect the interoperability of Web services that do not use security.

WebSphere Application Server V6.0

WebSphere Application Server v6.0 and Rational Application Developer v6.0 implement WS-Security 2004 and the draft WS-I Security Profile. Thus secure Web services should interoperate with Microsoft Web Service Enhancements 2.0. We hope to publish a revision this redbook during 2005 to demonstrate how to make secure Web services interoperate using the to-be finalized WS-I security profile.

9.2.4 Summary

The requester solicits a separate response, usually implemented as a second synchronous communication originating from the responder. The requester must have some means of correlating the callback with the original request by passing some token or address in the original request which is returned in the callback.

Security

We have seen that when creating secure Web services using WebSphere, we use the `webservices.xml` and `webservicesclient.xml` files. Security configuration is completely separate to the business logic of the Web services. Since the files are Java 2 Enterprise Edition deployment descriptors, our Web services clients must be container managed Java 2 Enterprise Edition clients, not J2SE clients.

When using Microsoft .Net, security settings are done in the code of the Web service or client. For example, when coding a client, we create an object that represents the Web service, by using the code stubs generated from a Web reference. Then we configure security for the request and response messages by calling methods on the `RequestSoapContext` and `ResponseSoapContext` members of this object.

Development tools

WebSphere Studio Application Developer provides some useful wizards and test tools to generate Web services. In particular the automatic WS-I compliance tool

in WebSphere Studio Application Developer is a feature Microsoft Visual Studio .Net 2003 is lacking.

Testing and deployment

The other main difference in the experience in using the tools is that the WebSphere Studio Application Developer packages a test environment into the tooling, and has very explicit deployment steps to publish a new application into a production environment, whereas with Microsoft .Net the familiar facilities of the Windows platform are used to manage the development, test and production environment.

Are these differences significant? Probably not greatly - they reflect that Microsoft Visual Studio .Net 2003 supports development on a single platform, whereas WebSphere Studio Application Developer supports multiple platforms. The difference in approach does not have much bearing on the number of tasks that need to be performed for a large enterprise to deploy a Web service into a production environment. The fact that Microsoft Visual Studio .Net 2003 doesn't require an additional deploy step doesn't mean that compared with WebSphere, no one needs to be employed to plan and manage deployment.

For successful management of development, testing and deployment on both platforms, enterprises need to employ skilled professionals who understand their enterprise, their IT infrastructure, their solutions and the tools being used. Both a complex Microsoft .Net environment and Java 2 Enterprise Edition present equivalent challenges.

The only meaningful comparison between the platforms is the one *you* make as to how well Microsoft .Net or WebSphere match the needs of your enterprise based on the solutions you aim to implement. Web services is only part of the story.

Archived



Deploying Web services

This chapter describes the Web service deployment models and runtime architecture for WebSphere and Microsoft .Net platforms.

We cover using a UDDI registry and configuring Web services runtime architectures on Microsoft .Net and WebSphere. It is beyond the scope of this redbook to look at high availability, high performance and highly secure Web sites. Our goal is to provide an introduction to the basic runtime architecture recommended by IBM and Microsoft in their publications.

10.1 Overview

As described in the business scenarios, integration between WebSphere and Microsoft .Net platforms can be achieved in an intranet (corporate) or Internet (B2B or B2C) environment.

In the first case, all service consumers belong to the same corporation; usually, in such environments, proxying, firewalling or security protections are advisable, even if they are not considered mandatory.

In the second case, Web services consumers are outside corporate boundaries and the same Web service represents the means by which two or more companies do business together. In such environments, service providers must provide security and accomplish more non-functional requirements; the most important are listed below:

- ▶ Establish access policies to prevent unauthorized access
- ▶ Mask intranet Web service endpoint
- ▶ Provide a reliable service in terms of high service availability and low service fault
- ▶ Assure the provisioning of interoperable service discovering, binding and invoking mechanisms

The Web services deployment architecture assumes a great importance and must be suitably planned and executed.

10.1.1 Web services publishing

The reference architectures we show later in this chapter use an internal UDDI service to publish Web services on the Internet. This is not the only choice we have: companies publishing Web services on the Internet may take one of the following three actions:

- ▶ Do not publish the Web service. In this case, only static binding can be used from Web service clients to invoke the service. This also means that the service provider must provide the WSDL file directly to the service consumer.
- ▶ Publish the Web service in an internal UDDI registry server. An internal UDDI server is typically a product implementing UDDI specifications and can be exposed on the Internet by means of the company Internet Web server.
- ▶ Publish the Web service in one of the public UDDI business registries (UBR) listed in the UDDI OASIS Web site, such as the one provided by IBM at <http://uddi.ibm.com> or Microsoft at <http://uddi.microsoft.com>. A UBR is a group of Web-based UDDI nodes, which together form a UDDI registry. All

UDDI nodes will replicate each other daily, so that all registries remain current.

The choice to use an internal or an external UDDI server is based on how much we want to make the service public. Most times, just as in our External Claim Assessors scenario, Web services are used in B2B applications and only accessed by a restricted team of corporates having special agreements. In this case, the internal UDDI registry or simply the direct provisioning of the WSDL file is usually preferred.

From an interoperability point of view, the key difference between the use of a public or private UDDI registry is that, in the case of public UDDI Business Registries, both IBM and Microsoft have their own registry which are kept synchronized with each other; this means that at any time, Microsoft Web services clients can locate services in the Microsoft UDDI registry while IBM Web services clients can locate services in the IBM UDDI registry. There is no need to check interoperability between the IBM UDDI client and the Microsoft UDDI server, or between the Microsoft UDDI client and the IBM UDDI server.

Interoperability problems may arise when private UDDI registries are used; in this case, we cannot assume to have both Microsoft and IBM platform providing synchronized UDDI registries; only one of them would reasonably be implemented. A client may be based on a different platform than the server and we must be sure that client and server components implemented from different vendors are interoperable.

10.2 WebSphere Web services deployment model

WebSphere platform provides a fully secure, scalable and reliable architecture for publishing Web services; this architecture can address all requirements listed in the previous section. Service deployment and related system network architecture have been defined with the objective of building a full Service-Oriented Architecture.

The main software components needed are the UDDI registry and the Web Services Gateway. Both are product features implemented in WebSphere Application Server Network Deployment. In the following sections, we first describe the Web Services Gateway and UDDI Registry features and then provide the overall deployment architecture.

10.2.1 Web Services Gateway

Web Services Gateway works in two directions. For services deployed inside the enterprise, it is a reverse proxy server between an external Web service client

and the corresponding Web service provider. Web Services Gateway enables clients from outside the corporate intranet to use Web services that are deployed in the corporate intranet without directly connecting to them; the proxy configuration is also supported to enable client from the corporate intranet to consume services exposed on the Internet network.

In a WebSphere environment, a good base level of security can be reached with a basic three-tier architecture where the only process running in the DMZ layer is the HTTP server and the WebSphere plug-in. Using EJB to implement Web services, we decouple the Web layer from the integration layer. In fact, in this case, no business code is contained in the included WAR file and the mapping between the EJB and the corresponding Web Service is obtained by means of configuration files.

The Web Services Gateway provides a boundary layer with a complete set of functions to publish, secure, decouple and adapt an intranet Web service to an external Internet environment. Using the Web Services Gateway administration tool, we can:

- ▶ Register Web services to make:
 - Internal Web services available outside the intranet
 - External Web services available inside the intranet

When the service registration is performed, a new WSDL file is generated from the original one provided with the Web service. This generated WSDL file is similar to the original one except for the service endpoint. The original endpoint is masked with the new one provided by the gateway. The generated WSDL file is managed by the gateway and exposed outside the internal network. External clients consume the internal service through the Web service reverse proxy installed at the gateway.

- ▶ Modify Web service channel configuration

For example, a SOAP/JMS internal Web service can be registered on the Internet as a SOAP/HTTP Web service; protocol conversion is implemented on the gateway.

- ▶ Apply custom filters or JAX-RPC handlers before or after service execution

For example, filters can implement logging as an alternative to developing custom logging functionality in each Web service implementation. This can be useful when exporting existing code as a Web service and needing to add some additional management capability.

Filters are still present in WebSphere V5.1, but their use is deprecated. All new gateway installations should use JAX-RPC handlers rather than gateway filters, for the following reasons:

- JAX-RPC is part of the proposed Java 2 Platform, Enterprise Edition (J2EE) 1.4, and JAX-RPC handlers are emerging as the standard approach in Java for intercepting and filtering service messages.
- JAX-RPC handlers are already being widely implemented - and any JAX-RPC handlers you write for use in other systems can also be deployed to the gateway.
- JAX-RPC handlers are already accepted as the standard approach in Java for managing message-level security as defined by the Web Services Security (WS-Security) specification.

Another difference between filters and JAX-RPC handlers is that while filters are applied at the level of the gateway service, JAX-RPC handlers are applied to:

- The gateway service and the channel (for messages passing between the service requester and the gateway).
- The target service and the target service port (for messages passing between the gateway and the target service).

▶ Manage UDDI registries

UDDI publishing of the server is managed within the Web Services Gateway administration.

▶ Manage security

The gateway can secure the communication between the service requester and the gateway, and between the gateway and the target service. Security can be applied at different levels from transport level security to message level security:

- Web service security (WS-Security)
- Gateway-level authentication
- Operation-level authorization
- SSL protocol using HTTPS
- Proxy authentication

The interaction between a Web service client and the corresponding service implementation registered on the gateway is shown in Figure 10-1 on page 220.

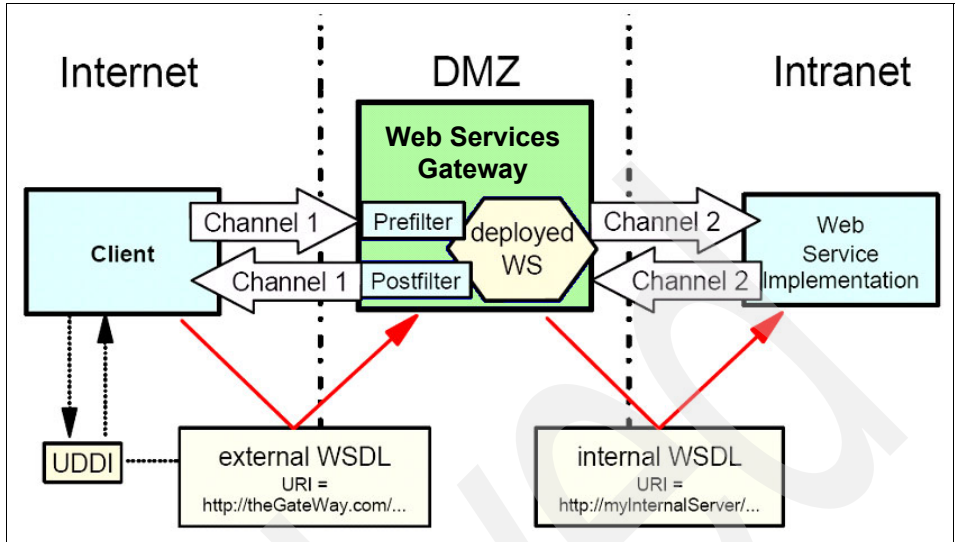


Figure 10-1 Web service consuming through the Web Services Gateway

Please refer to the IBM Redbook *WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook*, SG24-6891 for details about Web Services Gateway administration.

10.2.2 IBM UDDI registry

The Universal Description, Discovery, and Integration (UDDI) registry provided by the IBM WebSphere platform implements the UDDI specifications version 2.

UDDI is delivered with WebSphere Network Deployment. It can be installed both in a node which is part of a deployment manager cell or in a separate, standalone application server. The database manager supported to keep UDDI registry information is DB/2 or Cloudscape™.

UDDI registry is a J2EE application and it is deployed as an EAR. So, any performance or security considerations can be applied on it as on any other J2EE application running on WebSphere.

10.2.3 Deployment architecture

The reference Web service deployment architecture in a WebSphere environment is shown in Figure 10-2 on page 221. The nodes and applications configuration we chose to represent the architecture form a general purpose configuration; in fact, both the UDDI registry and Web Services Gateway could

be installed in a standalone WebSphere Application Server not part of a deployment manager cell, and there is no specific need to keep them in different nodes.

The advantages of such a configuration are as follows:

► **Network and protocol decoupling between service consumer and service provider**

Internal Web services are not directly exposed on the Internet; furthermore, the gateway can change the protocol used by the internal Web service with the HTTP protocol, which is more commonly used for Internet connections.

► **Network protection of all externalized services, included the gateway and the UDDI**

Both firewalls can be configured to allow only incoming HTTP requests on port 80 and HTTPS requests on port 443.

► **A fully secure DMZ**

No application code runs on the HTTP Server except the HTTP server itself and the WebSphere Application Server plug-in.

► **A centralized administration tool**

The gateway is also used for UDDI registry publishing.

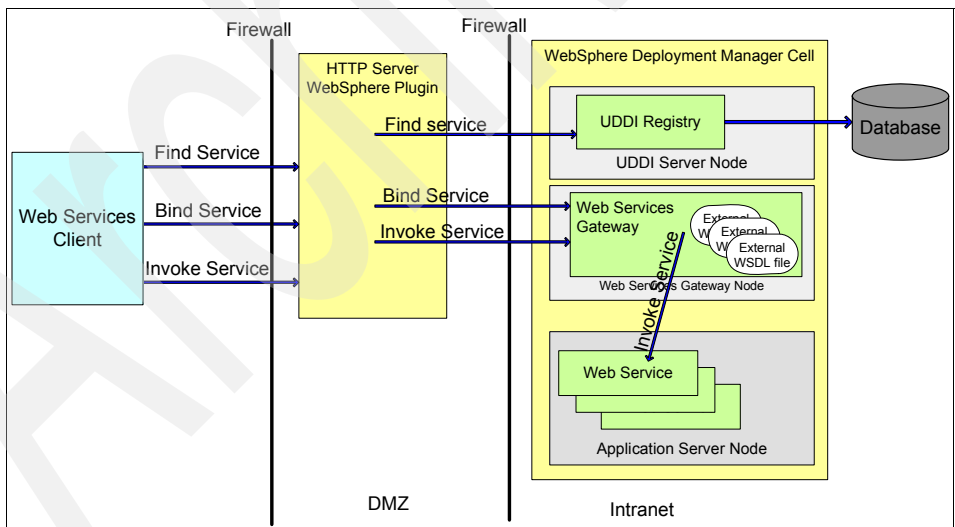


Figure 10-2 Web services deployment architecture in a WebSphere environment

In order to deploy a single Web service in the proposed environment, only the following two deployment steps must be executed:

1. Deploy Web service EAR file into the target application server.
2. Register the Web service on the Web Services Gateway with all required options (protocol conversion, JAX-RPC handling, security, etc.) included in the entry published to the UDDI registry.

The Web service consuming process is based on the following steps:

1. The client locates the service using the find service provided by the UDDI registry. Because the UDDI is a J2EE application, client requests are managed by the presentation layer provided with the UDDI registry, which is a Web application exposed on the Internet through the WebSphere plug-in of the HTTP server. The UDDI registry returns the found service, indicating as the service endpoint the external one provided by the gateway.
2. The client performs the service binding asking the Gateway for the WSDL. The gateway returns its own WSDL file.
3. The client sends the service invocation request to the Gateway, which routes the invocation to the real internal Web service after applying all handlers and protocol conversion, when needed. The response from the Web service is then rerouted to the original client; also in this case, handlers and protocol conversion are applied as needed.

10.3 Microsoft .Net Web service deployment model

Configuration of Microsoft .Net XML Web services follows the same paradigm used by all ASP.NET Web applications, so the same concepts of deploying, configuring, scaling, remoting and securing ASP.NET Web Applications can also be applied for Web services.

The main software components are the Microsoft .Net platform itself and the UDDI registry. In the following sections we first describe the UDDI Registry features and then provide the overall deployment architecture.

10.3.1 Microsoft UDDI registry

Microsoft Enterprise Universal Description, Discovery, and Integration (UDDI) Services is included in Windows Server 2003. Its main features are as follows:

- ▶ Supports versions 1.0 and 2.0 of the UDDI Programmer's API
- ▶ Has been developed using Microsoft ASP.NET and the Microsoft .Net Framework
- ▶ Includes a Web interface with searching, publishing, and coordination features

- ▶ Takes advantage of the Active Directory service, providing the authentication and authorization backbone for UDDI Services

Microsoft UDDI core entity names differ from the ones provided in the UDDI specification. The relationship between names is shown in Table 10-1.

Table 10-1 Microsoft UDDI Core entity names

UDDI.org Specification	Microsoft UDDI Services
businessEntity	Provider
businessService	Service
bindingTemplate	Binding
tModel	tModel

The installation consists of three components:

- ▶ Microsoft Internet Information Server 6.0 (IIS 6.0) - This is the Web server housing the ASP.NET front end which is used to browse UDDI Services.
- ▶ Microsoft SQL Server 2000 or Microsoft SQL Server 2000 Desktop Engine (MSDE) - This is the database to store the UDDI Services information.
- ▶ The Microsoft Management Console (MMC) management component, which can be used to manage multiple UDDI Services from one administrative console.

These components can be installed on the same machine or can be distributed over the network. The distributed installation is available only with the Datacenter or Enterprise Edition of Windows Server 2003 and does not support MSDE. A distributed installation is configurable with different scalability topologies for each component. For example, it is possible to have multiple Web servers with only one database providing shared information.

10.3.2 Deployment architecture

The proposed architecture expects at least one classic front-end firewall with packet filtering capabilities. This firewall must be configured, as usual, to allow incoming HTTP requests on port 80 and HTTPS requests on port 443. More robust firewall services able to inspect the HTTP and Web service request can be obtained either by installing a more capable external firewall, an XML firewall, or by using the URLScan tool from Microsoft.

Microsoft's recommendation is: *"A firewall should exist anywhere you interact with an untrusted network, especially the Internet. It is also recommended that you separate your Web servers from downstream application and database*

servers with an internal firewall.” (from *Improving Web Application Security, Threats and Countermeasures*, p.413).

In general, it is undesirable to install and run application code, such as a Web service, on a server in the DMZ because access to DMZ servers needs to be very tightly controlled. Administrators must have confidence that the machines are not compromised by unauthorized programs or people. Allowing application code onto a DMZ machine could result in more people needing access to the DMZ machines. More care and attention needs to be given to the procedures to authorize people to access the servers, to deploy application code and to authenticate it.

The concern is not about errant employee programmers compromising the DMZ servers, but about opening up the DMZ server to more people, via more connection methods, and having no one team fully understanding the implications of the additional software running on a DMZ server; this increases vulnerability to attack.

Usually, the Web service which is exposed on the Internet network is an .asmx file located in the Web server platform (Microsoft IIS). This deployment architecture is shown in Figure 10-3 on page 225. By virtue of being deployed in the DMZ, a Web service has an IP address that can be addressed externally and it can also be protected by the external firewall and other security measures. Chapter 19 of *Threats and Countermeasures* describes how to lock down the resources used by the Web service to minimize the opportunities for anyone who gets illicit access to the Web server machine to make use of the same resources to access the intranet.

Our recommendation is to write Web services using the Service Interface pattern, so the Web service has a limited number of connections inside the intranet which can be monitored with an internal firewall.

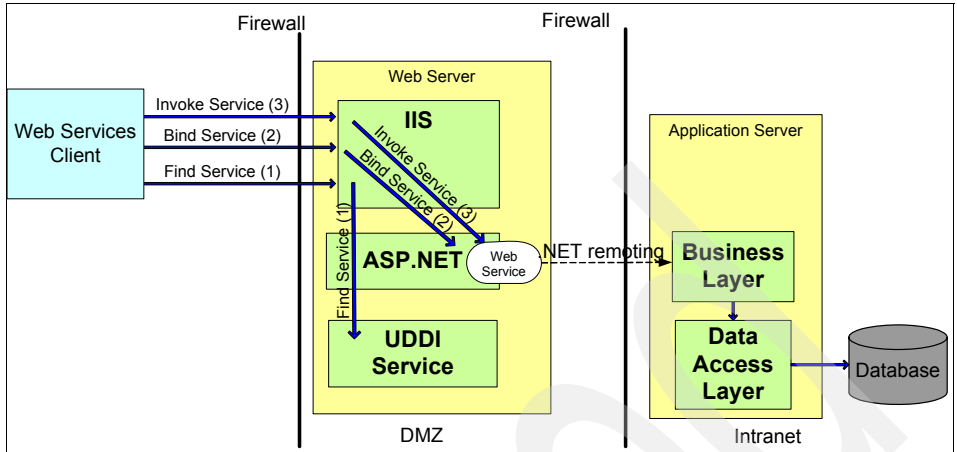


Figure 10-3 Internet Web services deployment architecture in Microsoft .Net environment

According to Microsoft guides, a Web service can also be deployed on the internal Application Server; this happens when the Web service client is internal too. Remote Web services are used as an alternative to .NET remoting.

Figure 10-4 shows a classic Web client connecting to an ASP.NET Web application. To accomplish the client request, the Web application must invoke the remote business layer, located in the application server; an ASP.NET Web service is then used to connect the ASP.NET Web application, acting as a Web service client, to the remote business layer.

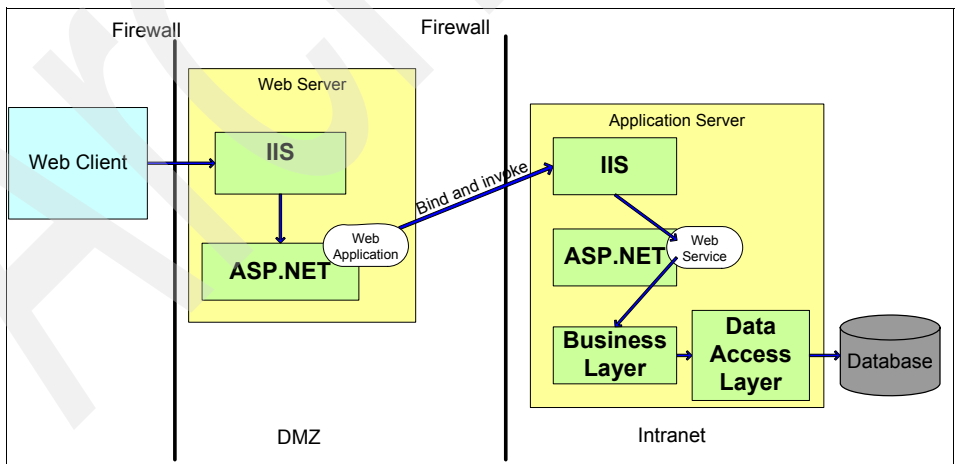


Figure 10-4 Intranet Web services deployment architecture in Microsoft .Net environment

All other non-functional requirements such as load balancing, fail-over capabilities and server clustering can be added using external equipment and operating system built-in capabilities. The Microsoft paper “Application Architecture for .NET: Designing Applications and Services” includes a large number of different deployment patterns for Web services; it can be found at:

<http://www.microsoft.com/downloads/details.aspx?FamilyId=A08E4A09-7AE3-4942-B466-CC778A3BAB34&displaylang=en>

Detailed information about security in a Microsoft environment can be found in the Microsoft white paper *Securing ASP.NET Web Services*, found at:

<http://www.microsoft.com/technet/itsolutions/net/maintain/secnetws.msp>

10.4 Summary

We have described how a Web service is deployed on the WebSphere and Microsoft .Net platforms and summarized the runtime architecture guidance from Microsoft and IBM. The architecture builds on the infrastructure used for Web sites today.

The guidance on deploying Web services from Microsoft makes use of capabilities in the Windows platform that will be familiar to Windows System Programmers to secure the new services.

The IBM approach focuses on managing Internet access to Web services by introducing a new cross-platform component, the Web Services Gateway. The Web Services Gateway provides a single point of control for Web services across multiple platforms that can be managed by a Systems Programmer with WebSphere skills.

Both approaches share the same objective: exposing as little as possible of the IT infrastructure to outside attention.



Part 3

Claims scenario

In this part, we build the claims scenario using WebSphere Studio Application Developer and Microsoft Visual Studio .Net 2003.

Our goal is to show how to use the build and deployment capabilities of the integrated development environment to realize the merger of the existing claims applications as Web services.

Archived

Designing the scenarios

This chapter describes the scenarios, use cases and analysis of the types of data being used. We analyze two scenarios. The first is an intranet scenario and will be implemented without security. The second is an Internet scenario which uses WS-Security. Only the first scenario has been implemented in this edition of the redbook. The goal is to use WS-I security profile 1.1 when it is finalized to implement the second scenario to demonstrate the use of secure Web service between Microsoft .Net and WebSphere.

This chapter includes the following topics:

- ▶ Mergers and Acquisitions scenario
- ▶ External Claims Assessors scenario
- ▶ LGI and DCI Insurance claims applications: table schema
- ▶ XML Schema Data Types as common denominator

11.1 Mergers and Acquisitions scenario

The first step in the solution building process is the use case definition. In this redbook, we decided to design and implement only those use cases that are most useful in looking at interoperability between the WebSphere and Microsoft .Net implementations of Web services technology.

Furthermore, we decided to select use cases having different levels of complexity. The aim is to give the correct answer to designers and developers in the following two cases:

- ▶ When the interoperability is very simple to achieve and detail is needed only at development time. In this case, a typical problem is the use of a specific SOAP format or a certain *type* for an input/output variable because of the different programming languages supported by WebSphere and Microsoft .Net.
- ▶ When some decisions and design choices have an impact across the application environment and decisions must be taken earlier during the design phase.

This can happen when more complex specifications are used, for example WS-Security. When using more complex specifications, it is necessary to verify that they can be considered interoperable or whether there are some limits in the interoperability which may impact the application design.

11.1.1 Use cases overview

For the Mergers and Acquisitions scenario, as part of the ClaimProcess application, we identify the following actors:

- ▶ **Customer** - The insurance policy owner
- ▶ **Customer service** - An LGI employee who works in the contact center and gives remote assistance to customers by phone
- ▶ **Agent** - An LGI employee who works in a generic LGI branch office and gives assistance to local customers

We also identify a use case:

- ▶ **Register claim** - All actors listed above can register a claim using an Internet form provided on the LGI Internet site. The difference between the three actors is that, while the customer executes the claim registration by himself, the customer service or agent executes the claim registration on behalf of a customer but using their own authentication on the system.

Using the Rational XDE tool, which is a plug-in of WebSphere Studio Application Developer, we design the use case model as shown in Figure 11-1 on page 231.

Note: Explaining how to use WebSphere Studio Application Developer and Rational XDE plug-in to design and implement scenarios' use cases is beyond the scope of this publication. More information can be found in *WebSphere Version 5 Application Development Handbook*, SG24-6993-00

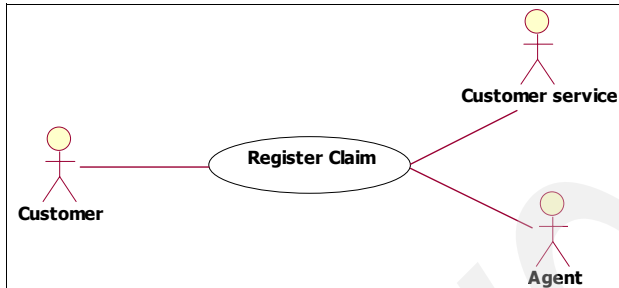


Figure 11-1 Mergers and Acquisitions scenario use case model

11.1.2 Actors

Table 11-1 provides details about the customer actor.

Table 11-1 Customer actor details

Actor name	Customer
Brief description	Customer: a person who has signed a policy with LGI or DCI.
Status	Primary
Relationships	
Associations to use cases	Use case 001: Register claim

Table 11-2 on page 232 provides details about the customer service actor.

Table 11-2 Customer service actor details

Actor name	Customer service
Brief description	An LGI or DCI employee who works in the customer service department answering to customers phone calls and providing them remote assistance
Status	Primary
Relationships	
Associations to use cases	Use case 001: Register claim

Table 11-3 provides details about the agent actor.

Table 11-3 Agent actor details

Actor name	Agent
Brief description	An LGI or DCI employee who is responsible to contact customers, propose and sign policies, and provide local assistance to the customer during all policy validation time
Status	Primary
Relationships	
Associations to use cases	Use case 001: Register claim

11.1.3 Use case 001: Register claim

Table 11-4 provides details about the register claim use case.

Table 11-4 Use case 001: Claim registration

Use case name	Use case 001: Register claim
Subject area	Claim system
Business event	A claim is submitted following up a car accident occurred to a LGI or DCI customer.
Actors	<ul style="list-style-type: none"> ▶ Customer ▶ Customer service ▶ Agent

Preconditions	<ul style="list-style-type: none"> ▶ The customer who refers the car accident owns a valid policy signed with LGI or DCI. ▶ The user who submit the claim is already connected to the new LGI Web site and authenticated to submit a claim (user authentication is supposed to be provided by an LDAP server which contains all information regarding registered Web users).
Steps	<ol style="list-style-type: none"> 1. User selects Register Claim from the menu. 2. The system displays the claim registration form (if the user is a customer, the value in the customer id field is prefilled and fixed) 3. User fills out the form with all required information and performs the form submission. 4. After validating the inputs, the system performs the following two steps: <ol style="list-style-type: none"> a. Asks each back-end system if the claim owner belongs to its customers list and holds a valid policy. b. Performs the claim submission to the back-end system which gives an affirmative answer to the previous question. If no system returns an affirmative answer, the claim is rejected. 5. The system displays the result of submission process.
Termination outcome 1	The claim is registered on the LGI or DCI back-end system with the state validation.
Notes	Web users are not required to be policy owners. We assume that the person who submits the claim is not intended to be the same user as the claimant; in fact customer service and agents can also perform a claim registration on behalf of a customer. For this reason customer identification is part of the claim information.

A use case activity diagram is shown in Figure 11-2 on page 234.

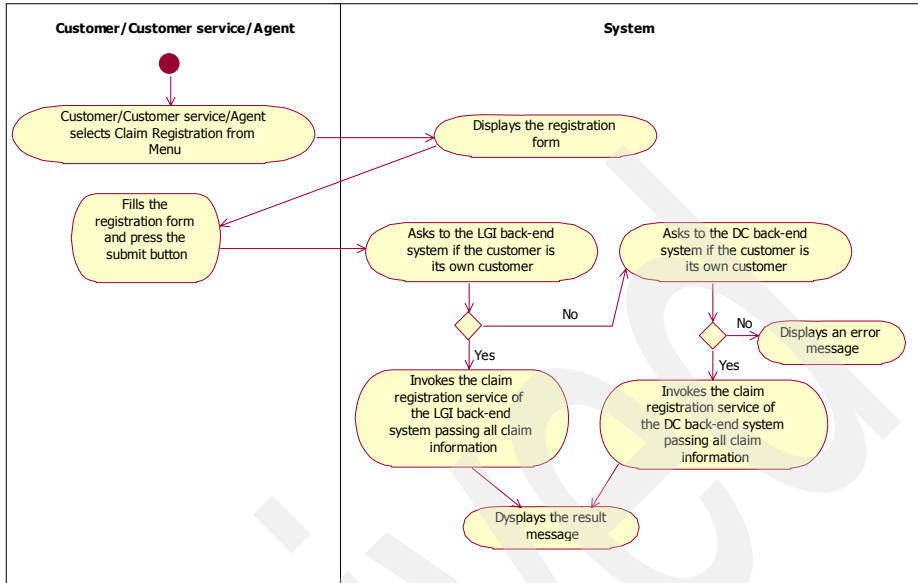


Figure 11-2 Activity diagram for the register claim use case

11.1.4 Realizing the use case

Once the use case model is completed, we start realizing the use case. The use case sequence diagram that is shown in Figure 11-3 on page 235 represents objects and interactions.

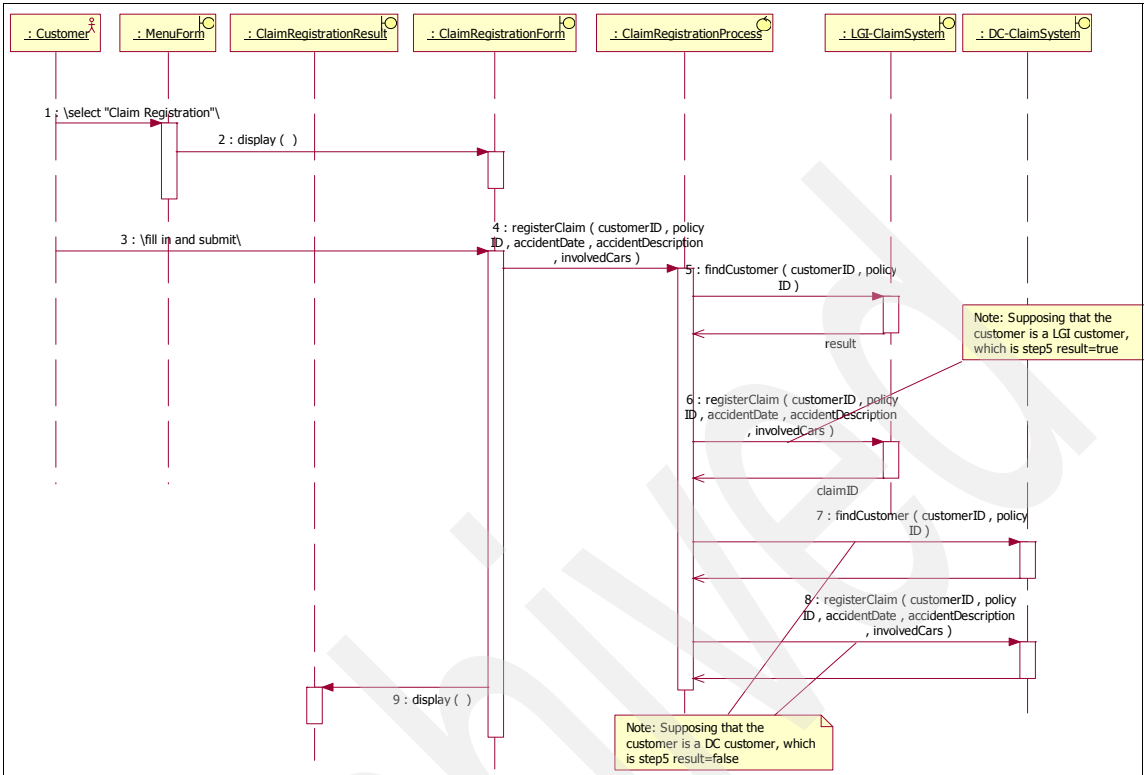


Figure 11-3 Sequence diagram for the register claim use case

Objects defined for the use case implementation are as follows:

- ▶ MenuForm - a boundary class responsible for calling and displaying the ClaimRegistrationForm
- ▶ ClaimRegistrationForm - a boundary class responsible for collecting all required user information and calling the claim registration process
- ▶ ClaimRegistrationProcess - a control class responsible for managing the claim registration process
- ▶ LGIClaimSystem - a boundary class responsible for finding an LGI customer and registering a claim in the LGI back-end system
- ▶ DCClaimSystem - a boundary class responsible for finding a DCI customer and registering a claim in the DCI back-end system
- ▶ ClaimRegistrationResultForm - a boundary class which displays the claim registration process result

Application architecture and design model

The layered modular design of the ClaimProcess application is based on the J2EE architecture.

The ClaimProcess application design is logically split into four layers:

- ▶ Presentation layer - Containing all presentation related implementation modules.
- ▶ Business layer - Containing the reusable business logic components.
- ▶ Integration layer - Including components to integrate any external system such as data sources or services outside the system boundary.
- ▶ EIS (enterprise information system) - Providing the information infrastructure of an enterprise, including relational databases, mainframe transaction processing systems, and legacy database systems.

Following the main purpose of this redbook, which is to show how to realize an interoperable solution based on Web services technology, in the rest of the chapter we focus only on the design and implementation of the communication between the business layer and the integration layer. This implementation is provided via Web services. All other components required to complete the examples are supposed to be already developed and working.

The design model structure represents the different application layers. The design layer packages are derived from the application architecture previously described. The design model is represented in Figure 11-4.

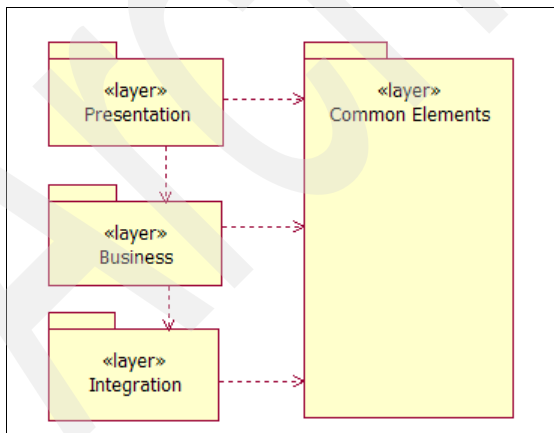


Figure 11-4 High-level design diagram

Integration layer design

Based on the system analysis, we have to design integration components for the two external ClaimProcess services, LGIClaimSystem and DCClaimSystem, which are used to realize the register claim use case.

Figure 11-5 shows the register claim analysis diagram with the external LGIClaimSystem and DCClaimSystem boundary classes as a result of the application analysis.

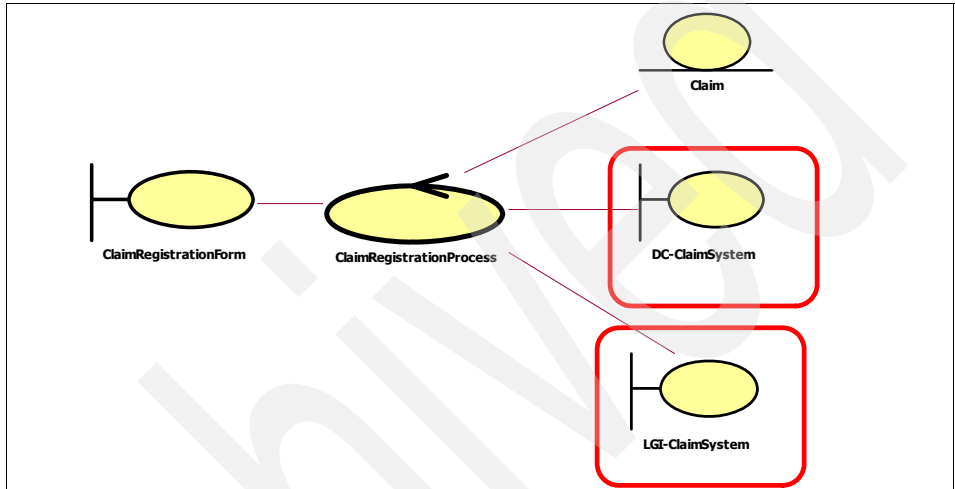


Figure 11-5 Register claim analysis diagram

First of all, we create two new subsystems within the integration layer design package, as shown in Figure 11-6 on page 238.

We must keep in mind that the two subsystems refer to different IT environments; so, even if it is providing the same services, the application development must produce different code.

- ▶ Java code for LGIClaimSystem
- ▶ C# code for DCClaimSystem

Because we are now in the design environment, we decide to use the Rational XDE™ tool for modelling both layers. The development phase will instead be carried out with the specific development tools:

- ▶ WebSphere Studio Application Developer for the LGIClaimSystem
- ▶ Microsoft Visual Studio .Net 2003 for the DCClaimSystem

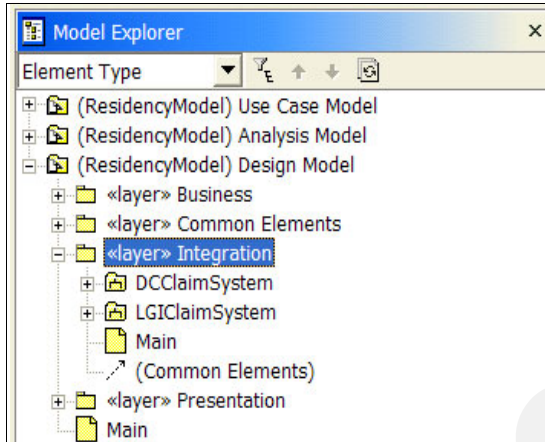


Figure 11-6 Register claim integration layer package

Web services design

For both previously defined subsystems, we have a simplified class diagram containing only data access objects and Web services objects; data access objects adapt the FindCustomer service and the RegisterClaim service to LGI and DCI back-end systems.

Table 11-5 lists the FindCustomer and RegisterClaim Web services integration design classes. The class names and description are considered valid for both WebSphere and Microsoft .Net environments. We use the data access objects model because it is a general pattern for objects providing data from back-end systems.

Table 11-5 Register claim integration layer simplified class diagram

Class name	Description
DataAccessException	This exception is raised during a generic DataAccessObject methods execution.
CustomerDataAccessObject	This is the data access object that encapsulates the data manipulation of the Customer object.
ClaimDataAccessObject	This is the data access object that encapsulates the data manipulation of the Claim object.
ClaimWebService	This represents the ClaimRegistration Web service implementation class.

Class name	Description
ClaimException	This exception is used by an integration service to notify of an error during execution of the external Claim Web service.

The final step is to provide the specifications of Web service exposed methods, as shown in Figure 11-7.

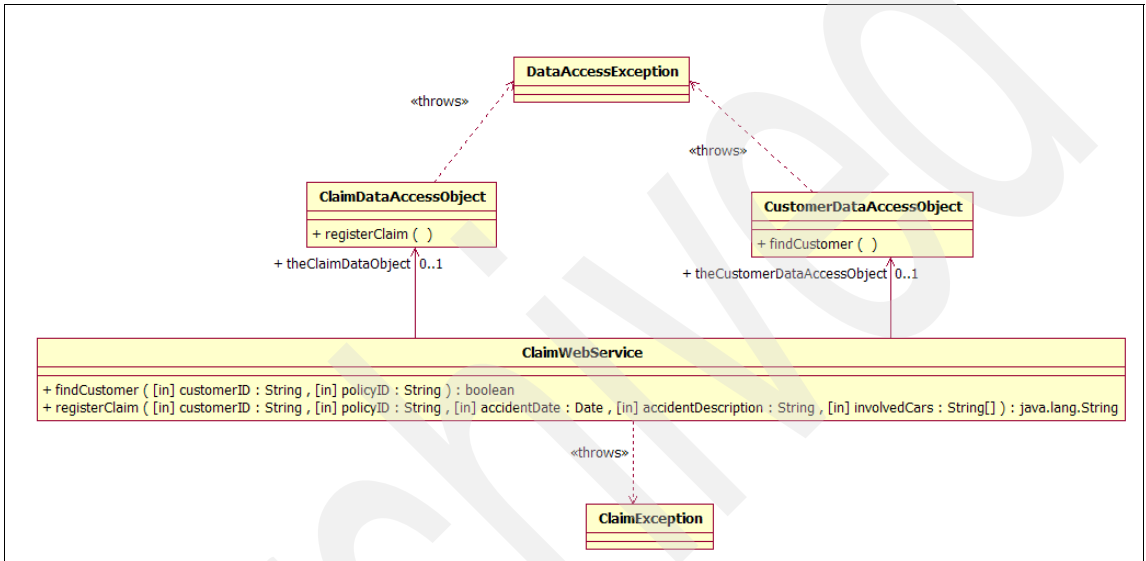


Figure 11-7 Register claim scenario design

The ClaimWebService will provide the following two methods:

1. findCustomer, which accepts the following inputs:

- *customerID*, type string
- *policyID*, type string

and returns a boolean stating whether the corresponding customer has been found in the system. A ClaimException is the fault returned if an error occurs during the Web service execution.

2. registerClaim, which accepts the following inputs:

- a string object containing the *customerID*, type string
- a string object containing the *policyID*, type string
- a date object containing the *accidentDate*, type dateTime
- a string object containing the *accidentDescription*, type string

- an array of strings containing the *involvedCars* list, type string with unbounded maxOccurs

and returns a string containing the generated claim code. A *ClaimException* is the fault returned if an error occurs during the Web service execution.

After this design step, all further design and implementation details must be considered platform/technology dependent and must be carried out separately within the two different environments. In any case, our purpose is not to focus on coding data access objects; they can be EJBs or Java classes in the case of a WebSphere platform or any piece of Microsoft .Net compatible code (C#, VisualBasic, etc.) in the case of a platform. The examples we provide are focused on:

- ▶ How to turn these data objects into Web services or how to create a Web service which is able to invoke them
- ▶ How to build a corresponding Web service client, deploy and test the developed code

11.2 External Claims Assessors scenario

This section details the analysis and design process for the second scenario.

11.2.1 Use cases overview

For the External Claim Assessors scenario, part of the ClaimProcess solution, we identify the following actor:

- ▶ **Claims handler** - An LGI employee who is responsible for managing claim process

We also identify a use case:

- ▶ **Manage external claim assessor** - The Claim BPM System, during the claim investigation process, selects an external assessor to investigate the claim and produce the assessment report. The selected external assessor sends back the produced report.

11.2.2 Actors

Table 11-6 on page 241 provides details about the claim BPM system actor.

Table 11-6 Claims handler actor details

Actor name	Claims handler
Brief description	An LGI employee who manages the claim processes. He uses a client application showing all processing claims and their status. The application also provides some work lists with all claims in a specific status which need a manual activity. A claims handler can choose whether a claim should be investigated by LGI or by an external assessor.
Status	Primary
Relationships	
Associations to use cases	Use case 002: Manage external assessor process

Table 11-7 provides details about the external assessor system actor.

Table 11-7 External assessor actor details

Actor name	External Assessor System
Brief description	The external assessor back-end system which is able to answer to assessment availability requests, to accept assessments requests and to send assessment reports.
Status	Primary
Relationships	
Associations to use cases	Use case 002: Manage external assessor process

11.2.3 Use case 002: Manage external claim assessors

Table 11-4 on page 232 provides details about the manage external claim assessors use case.

Table 11-8 Use case 002: Manage external claim assessors

Use case name	Use case 002: Manage external claim assessor
Subject area	Claim system
Business event	A valid claim must be investigated before judgement. The investigation process is delegated to an external independent assessor.
Actors	► Claims handler

<p>Preconditions</p>	<ol style="list-style-type: none"> 1. A claim is submitted to the system. 2. The claim is validated (policy not expired, valid driver insurance, provided claim details are accurate and correct, etc.). 3. The claims handler requests the current work list. The work list is displayed containing all claims in the “investigation” state; that is all claims needing an assessment.
<p>Steps</p>	<ol style="list-style-type: none"> 1. The claims handler selects a specific claim that needs to be investigated by an external assessor and clicks the button External Assessment. 2. The Assessor Business Process Management (Assessor BPM) receives this information and starts the process associated with an external assessment. 3. The Assessor BPM asks the Assessor Management System the list of eligible external assessors passing some information about the claim to be assessed (post code, car type, etc.). 4. After receiving the response from the Assessor Management System, the Assessor BPM needs to ask each assessor in the returned list for their current availability. Availability is requested to the External Assessor System providing some information about the claim to be assessed. 5. After receiving the availability response from each external assessor, the Assessor BPM builds a new list containing only available assessors and passes this list to the Assessor Management Business Rules to select an assessor. 6. The Assessor BPM asks the External Assessor System for an assessment from the selected assessor. 7. After completing the assessment, the External Assessor System sends the final assessment report to the Assessor BMP. 8. The Assessor BPM receives the assessment reports and saves it in the document management system.
<p>Termination outcome 1</p>	<p>The claim assessment is saved in the Document Management System and the claim status is judge.</p>

Using the Rational XDE tool, we design the use case model is shown in Figure 11-8 on page 243; the activity diagram is shown in Figure 11-9 on page 243.

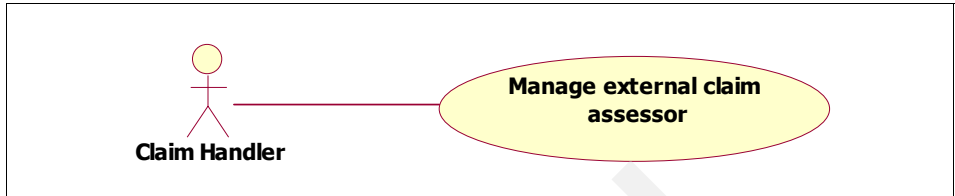


Figure 11-8 External Claim Assessors scenario use case model

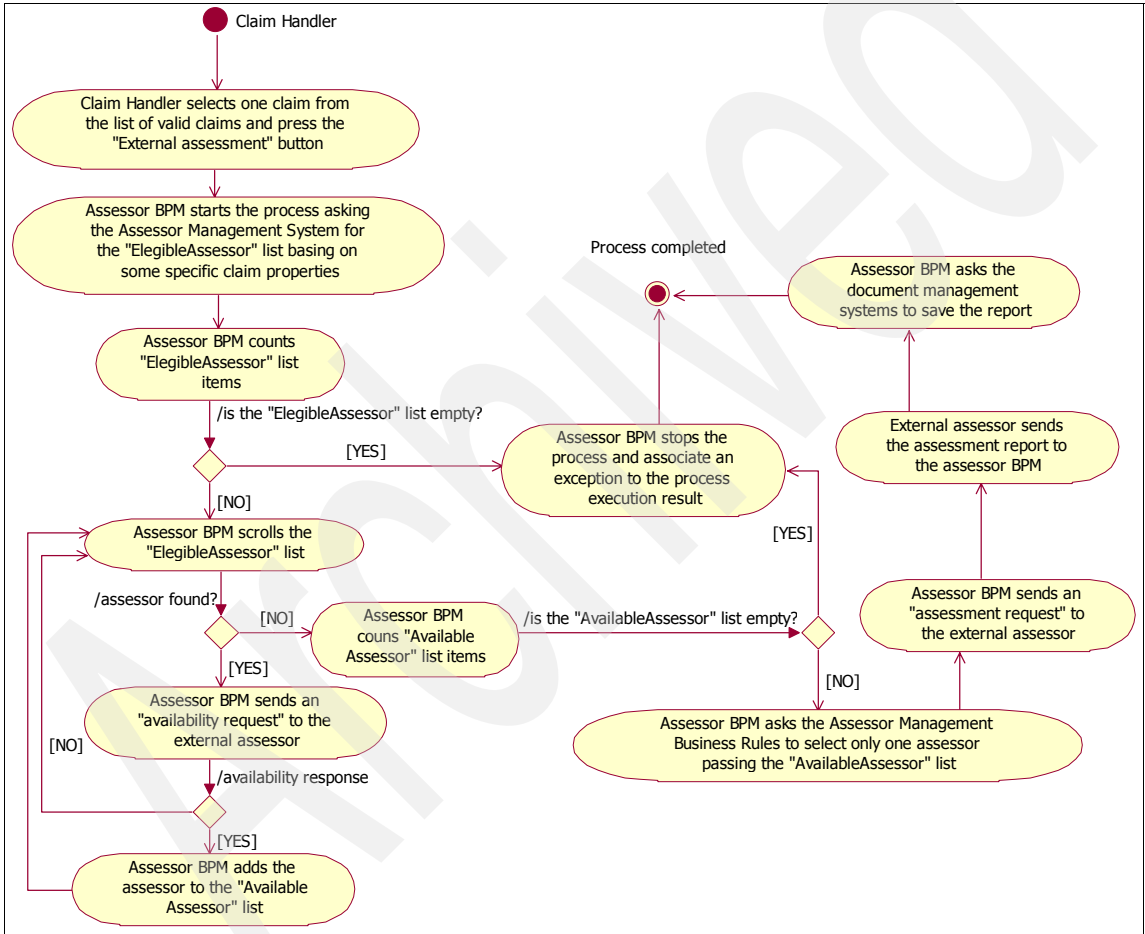


Figure 11-9 Activity diagram for the manage external claim assessor use case

11.2.4 Realizing the use case

Once we have completed the use case model, we start realizing the use case; the use case sequence diagram shown in Figure 11-10 represents objects and interactions.

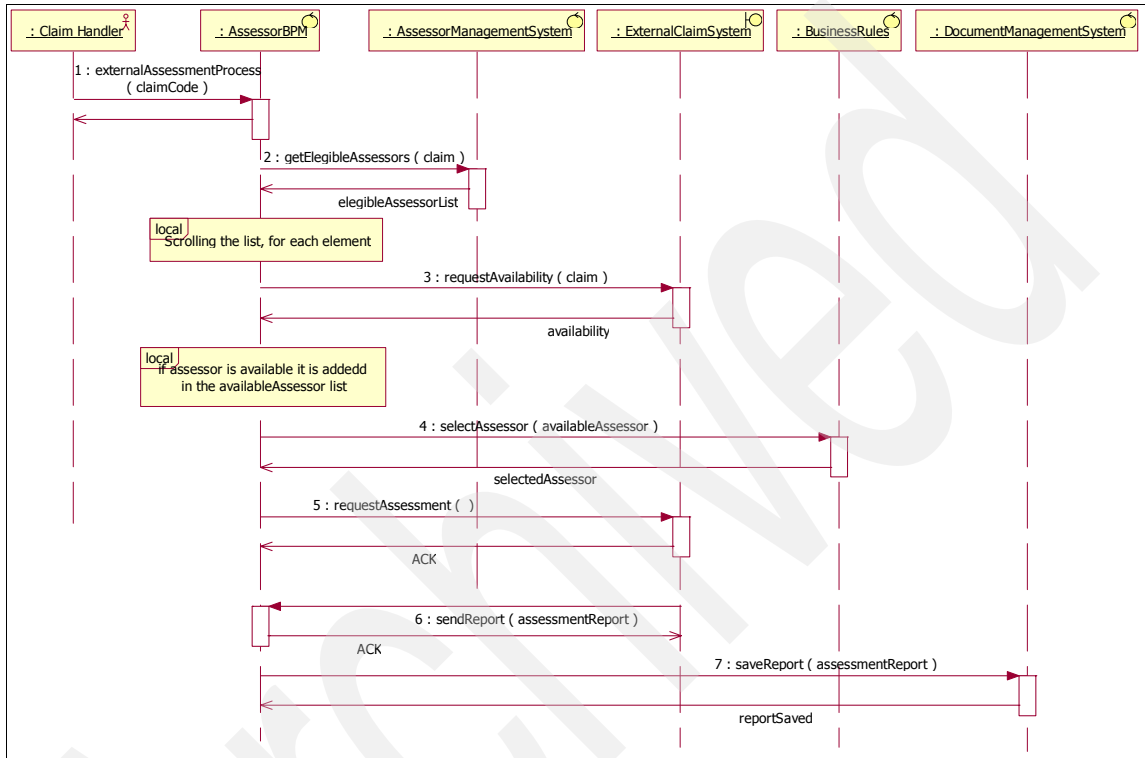


Figure 11-10 Sequence diagram for the manage external claim assessor use case

Objects defined for the use case implementation are as follows:

- ▶ Assessor Business Process Management - a control class responsible for managing the claim assessment process.
- ▶ Assessor Management System - a control class responsible for managing the list of registered external assessors and providing a list of ones who are eligible to perform the assessment for a specific claim.
- ▶ Assessor Management Business Rules - a control class responsible for choosing only one assessor among those who stated their availability. The choice is based on specific business rules.

- ▶ Document Management System - a control class responsible for managing documents such as the assessment report. This use cases asks the class to save the assessment report.
- ▶ External Assessor System - a boundary class responsible for answering an availability request for a specific claim, receiving assessment requests and sending assessment reports.

Application architecture and design model

The application architecture and design model is the same as the one detailed for the previous scenario. See “Application architecture and design model” on page 236.

Web services design

The Web services design is postponed until we start work on implementing the scenario when the WS-I security profile is finalized and supported by Microsoft .Net and WebSphere Application Server 6.0.

11.3 Claim applications: table schema

We have implemented simple LGI and DCI register claim sample applications to use in the demonstration of the interoperability between the Microsoft .Net Common Language Runtime and WebSphere runtime environments.

The sample application is implemented both in Microsoft Visual Studio .Net 2003, and in WebSphere Studio Application Developer.

The DCI application uses the Data Access model, where the Web service is the business layer and the data layer is represented by the DataAccessClass and ClaimAccessClass. Microsoft. Table 11-9 summarizes the fields and field types in the table schema.

Table 11-9 Sample application fields and field types - table schema

Name of fields	Type of field
Customer ID	String
Policy ID	String
Accident Date	DateTime
Accident Description	String
Vehicles involved in the accident	Array of String

Name of fields	Type of field
Return Claim Code	String
Return status	Boolean

11.4 XML schema data types as common denominator

Implementation interoperability between Microsoft .Net and WebSphere Web services require that we understand the automatic conversion of data types in both environments. Microsoft .Net may represent data types differently than WebSphere. However, XML data types, validated by the XSD (XML Schema), are used as the common denominator. Microsoft .Net converts its data types to XML data types before sending the SOAP message to WebSphere. When WebSphere receives the message, it converts the message from XML back to WebSphere Java data types. In the sample application, we are using Microsoft .Net DateTime, String Array, String and Exception.

11.4.1 Data type mapping

The sample application is implemented using String, Boolean, Array of String and Exception classes. Table 11-10 summarizes the available data type mappings. However, not all the types are interoperable between Microsoft .Net and WebSphere. When merging a Microsoft .Net system to a WebSphere system, we may need to take this difference into consideration. The problematic types can be wrapped in a new class.

Table 11-10 Data Type mapping between Microsoft .Net and WebSphere

XML Data Type	Microsoft .Net 2003 Data Type	WebSphere Data Type
AnyUri	System.Uri	Java.net.URL
base64Binary	Byte[]	Byte[]
Boolean	Boolean	boolean
Byte	SByte	byte
DateTime	DateTime	java.util.date
Decimal	Decimal	java.math.BigDecimal
Double	Double	Double
Float	Single	Float

XML Data Type	Microsoft .Net 2003 Data Type	WebSphere Data Type
HexBinary	Byte[]	byte[]
Int	Int32	int
Long	Int64	long
NegativeInteger	String	int
nonNegativeInteger	System.Decimal	int
nonPositiveInteger	System.Decimal	int
Short	Int16	short
unsignedInt	UInt32	int
Map	IList	java.util.HashMap
Vector	IList	java.util.Vector
Array	IList	array of built-in data types
Element	DataSet	org.w3c.dom.Element

11.4.2 SOAP message for registerClaim()

When the registerClaim() Web service method is invoked, it sends a SOAP request from WebSphere to Microsoft .Net. The SOAP request looks like that shown in Example 11-1. Within the SOAP message, we see that the simple string type and date type are interoperable between Microsoft .Net and WebSphere. This SOAP message conforms to the WS-I Basic profile 1.1.

Example 11-1 SOAP request for registerClaim() from WebSphere to Microsoft .Net

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:q0="http://tempuri.org/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <q0:registerClaim>
      <q0:customerID>ABC123455</q0:customerID>
      <q0:policyID>1234567890</q0:policyID>
      <q0:accidentDate>2004-09-24T00:00:00.00Z</q0:accidentDate>
      <q0:accidentDescription>At the corner of Springfield Blvd
      </q0:accidentDescription>
      <q0:involvedCars>
        <q0:string>William</q0:string>
        <q0:string>Francesca</q0:string>
      </q0:involvedCars>
    </q0:registerClaim>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
        </q0:involvedCars>
    </q0:registerClaim>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

When the request is received in WebSphere, the SOAP response looks like that shown in Example 11-2. A claim code of type String is returned and is the concatenation of the Customer ID, Policy ID and Accident Date. Note that the SOAP response has the element registerClaimResponse. This SOAP message also conforms to the Basic profile 1.1 and it is interoperable between Microsoft .Net and WebSphere.

Example 11-2 SOAP response for registerClaim() from Microsoft .Net

```
<?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>
    <registerClaimResponse xmlns="http://tempuri.org/">
      <registerClaimResult>ABC123455 1234567890 9/23/2004 5:00:00
PM</registerClaimResult>
    </registerClaimResponse>
  </soap:Body>
</soap:Envelope>
```

11.4.3 SOAP message for findCustomer()

The LGI Insurance application also provides the Web service to find a customer. If the customer is LGI's customer, then the findCustomer() method will return true. Otherwise, it will return false. In addition, if the customer ID is longer than seven characters, an ItsoClaimException will be thrown.

The SOAP request message sent from WebSphere to Microsoft .Net looks like that shown in Example 11-3. This SOAP request message conforms to the WS-I Basic profile 1.1.

Example 11-3 SOAP request message for findCustomer() sent from WebSphere

```
<?xml version="1.0" encoding="UTF-8" ?>
  <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:q0="http://tempuri.org/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <q0:findCustomer>
      <q0:customerID>ABC1234</q0:customerID>
      <q0:policyID>1234567890</q0:policyID>
```

```
    </q0:findCustomer>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP response message sent from Microsoft .Net to WebSphere looks like that shown in Example 11-4. Again, an element `findCustomerResponse` is included in the SOAP body. The SOAP message conforms to the Basic profile 1.1 and provides interoperability between Microsoft .Net and WebSphere.

Example 11-4 SOAP response message for `findCustomer()` from Microsoft .Net

```
<?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>
    <findCustomerResponse xmlns="http://tempuri.org/">
      <findCustomerResult>true</findCustomerResult>
    </findCustomerResponse>
  </soap:Body>
</soap:Envelope>
```

11.4.4 SOAP exception for `findCustomer()`

When the Customer ID is longer than seven characters for `findCustomer()`, it generates an `ItsoClaimException` with an `Invalid Customer` message. The SOAP exception message looks like the one shown in Example 11-6 on page 250. Microsoft .Net generates a SOAP exception for any exception that occurs in the Microsoft .Net Web service.

Example 11-5 SOAP request for `findCustomer()` with invalid string sent to Microsoft .Net

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:q0="http://tempuri.org/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <q0:findCustomer>
      <q0:customerID>ABC123456</q0:customerID>
      <q0:policyID>1234567890</q0:policyID>
    </q0:findCustomer>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP exception has fault tags in the SOAP body of the message, which is required to conform to the WS-I profile 1.1.

Example 11-6 SOAP exception response for findCustomer() from Microsoft .Net

```
<?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>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>Server was unable to process request. --> Invalid
Customer</faultstring>
      <detail />
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

11.5 Summary

This chapter provides two scenarios with the use case and actors illustrated in different diagrams. We only implement the first scenario, leaving the second scenario for a future update. We summarize the different field types which are used in the implementation of the first scenario, map different data types between Microsoft .Net and WebSphere, and briefly introduce the SOAP messages that are passed between Microsoft .Net and WebSphere, since these data types are simple ones and have no problem in representation by both Microsoft .Net and WebSphere. However, some other types may not be interoperable and we may need to wrap them in new classes.

Building the claims scenario

As discussed in previous chapters, we have gone through two scenarios, that is, Mergers and Acquisitions and External Claims Assessor, from business case and patterns selection to a discussion of implementation. In this section, we will demonstrate implementation and execution of the Mergers and Acquisitions register claim scenario using WebSphere Studio Application Developer, which will be followed by implementation using Microsoft .Net in the next section. Further, we will investigate interoperability across both WebSphere and Microsoft .Net platforms by implementing a common front end client in WebSphere interfacing to both platforms.

12.1 Building the scenario for WebSphere

In this scenario, Lord General Insurance (LGI) has acquired a typical modern dot.com auto insurance company, DirectCarInsure.com (DCI). There are existing business logics and processes that need to be accessed using a common front end by users of both the companies. In this section, we will consider how to make LGI's business logic, which is in the form of Enterprise JavaBeans, available for access by the common front end using Web services. In the next section, we will perform a similar step for the business logics and processes used by DCI. In the last section, we will implement two J2SE Java client prototypes which, in the complete implementation, would be incorporated into a Java 2 Enterprise Edition servlet to give the register claim Web page access to the Web services.

In summary, we will:

- ▶ Create Web services for LGI's register claim application from existing Enterprise JavaBeans
- ▶ Test the Web services using the Test Client auto-generated by WebSphere Studio Application Developer
- ▶ Deploy the Web services in WebSphere Application Server

12.1.1 Problem definition

LGI's business logics are available in the form of ItsClaim.ear which needs to be exposed to a common front end to process a user request.

12.1.2 Solution

We will expose the business logic by creating of Web service from the existing Enterprise JavaBeans in ItsClaim.ear.

We will perform the following tasks in the given order:

1. Import Enterprise JavaBeans
2. Test imported Enterprise JavaBeans
3. Create Web service from Enterprise JavaBeans
4. Test created Web service

12.1.3 Import Enterprise JavaBeans

In order to create a Web service from an existing Enterprise JavaBean (EJB), the first step is to import the EJB in WebSphere Studio Application Developer's development environment.

1. Start WebSphere Studio Application Developer 5.1.2 with a new workspace and then select the session bean **LGIClaimRegistration** in the EJB Modules and **File** → **Import** from the menu bar to invoke the Import wizard. Select **Import source as EAR file** as shown in Figure 12-1.

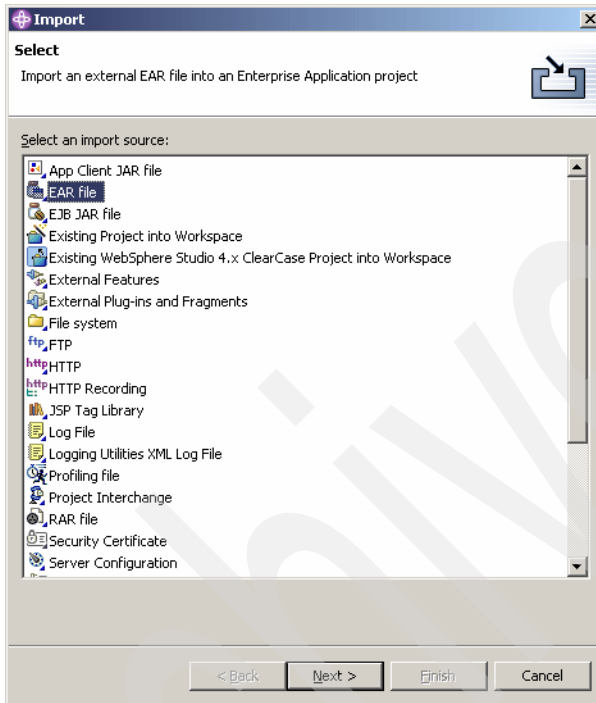


Figure 12-1 Import wizard: import source

2. Select **ItsoClaim.ear** at the Import wizard's Enterprise Application Import interface using the Browse button at the EAR file item.
3. Make sure to select the utility JAR **ItsoClaimCommon.jar** to be imported as a utility project, as shown is Figure 12-2 on page 254.

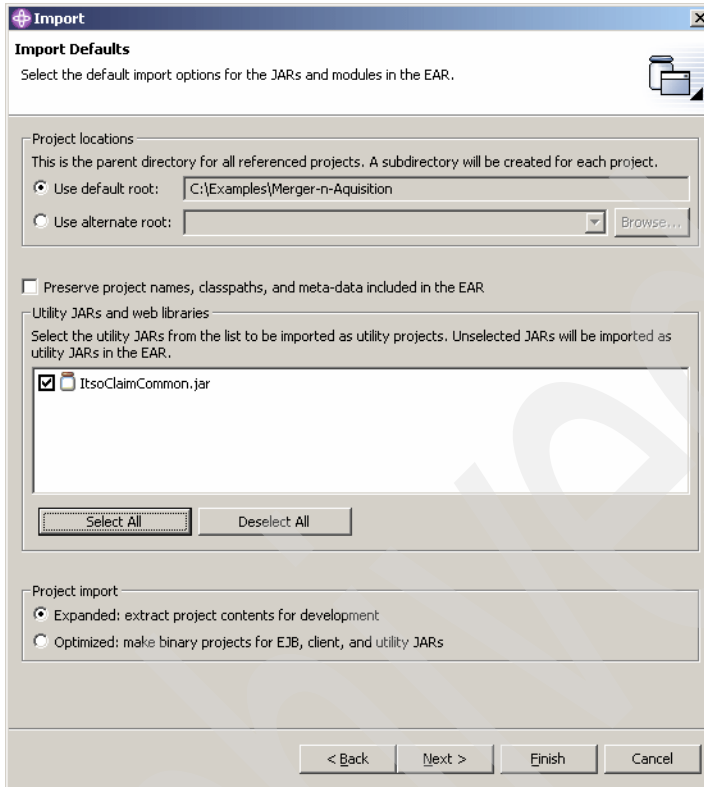


Figure 12-2 Import wizard: selecting utility JAR as utility project

4. In the next panel, Manifest Class-Path JARs and other module files will be displayed with available dependent JARs. In this example, there will be two JARs and a WAR, namely ItsoClaimEJB.jar, ItsoClaimCommon.jar and ItsoClaimWeb.war, which will be displayed in the left pane of the interface.
5. After successfully importing the .ear file, open the project navigator view. The workspace should look like Figure 12-3. Note that ItsoClaimCommon.jar is imported as a separate utility project.

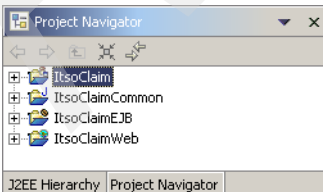


Figure 12-3 Workspace after import

Source code can be examined by navigating through the classes on the navigator pane. As given, we have a session bean `itso.examples.claim.ejb.LGIClaimRegistration` with class and interfaces as shown in Figure 12-4.

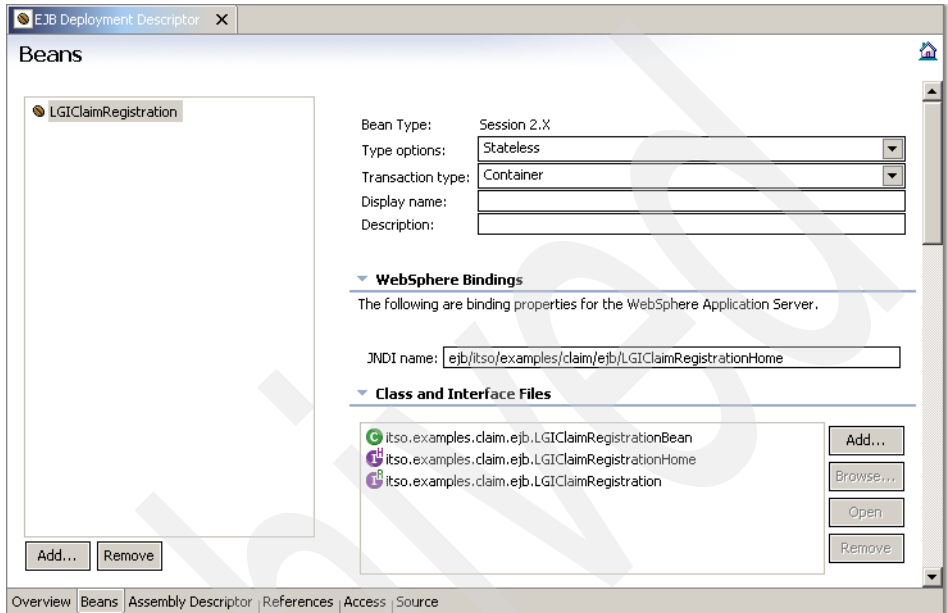


Figure 12-4 Given session bean `itso.examples.claim.ejb.LGIClaimRegistration`

To visualize interface and classes, Figure 12-5 on page 256 illustrates the outline.

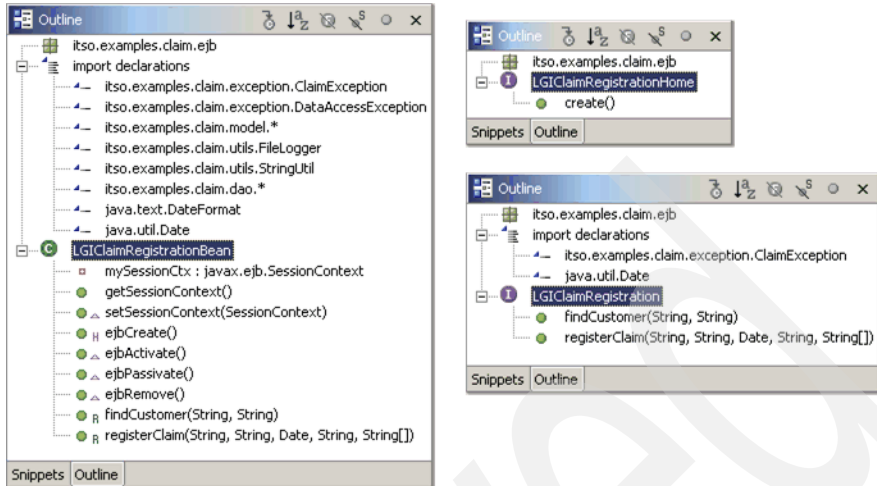


Figure 12-5 Outline of EJB

12.1.4 Test imported Enterprise JavaBeans

Now we will test the EJB using WebSphere Test Environment to ensure that it has been imported successfully and the business logic is working properly. Figure 12-6 shows the navigator in the resource perspective.

1. Select the session bean **LGIClaimRegistration** and right-click and select **Run on Server**.

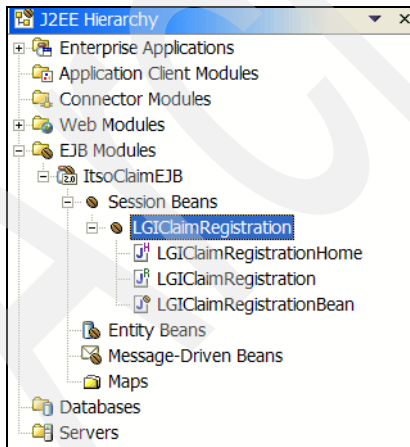


Figure 12-6 Selecting the LGIClaimRegistration session bean to test

This activity will perform the following tasks:

- Create a test server
 - Deploy the EJB on the server
 - Generate Test Client
 - Invoke IBM Universal Test Client browser to test the EJB
2. Upon clicking **Run on Server**, as described above, a Server Selection wizard will appear. Since there is no server at the moment, this wizard will suggest the default to create new server. Select the server type as **WebSphere Server 5.1** → **Test Environment** as illustrated in Figure 12-7.

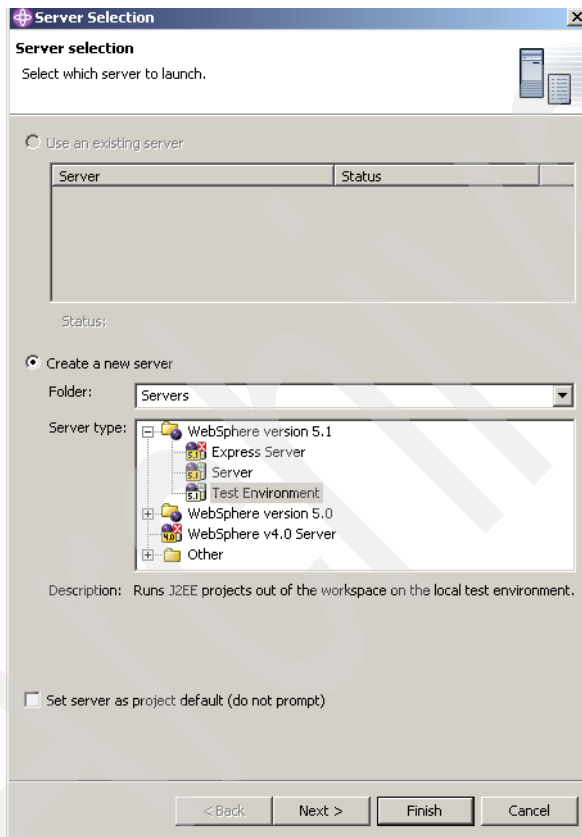


Figure 12-7 Create a new Test Server

3. In the next panel, WebSphere Server Configuration Settings, there is only one item to check: HTTP Port Number; select **Use default port numbers**, for instance, 9080.

4. After clicking **Next**, the Select Tasks interface will appear with a task option for ItsoClaimEJB; check **Deploy EJB Beans** as shown in Figure 12-8. This task will generate relevant deployment objects for ItsoClaimEJB.

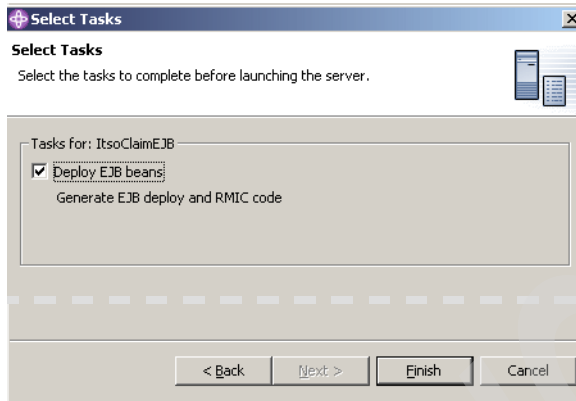


Figure 12-8 Check Deploy EJB beans

5. After processing the tasks chosen above, a server WebSphere v5.1 Test Environment will be created with deployed Enterprise application ItsoClaim, as shown in Figure 12-9.

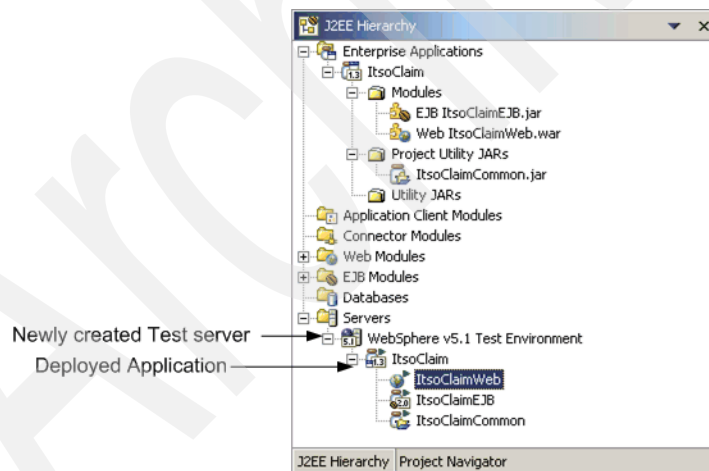


Figure 12-9 Test Server with deployed application

The IBM Universal Test Client will be launched with LGIClaimRegistration displayed under EJB References, as shown in Figure 12-10 on page 259.

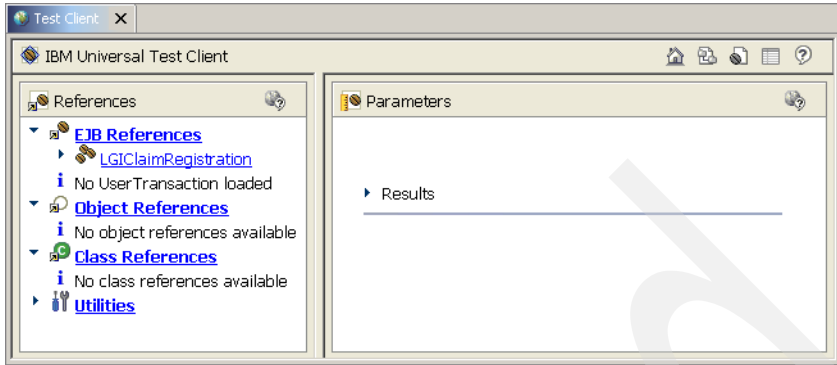


Figure 12-10 IBM Universal Test Client

- Expand **LGIClaimRegistration** → **LGIClaimRegistrationHome** and click **LGIClaimRegistration create()** in the IBM Universal Test Client, as illustrated in Figure 12-11.

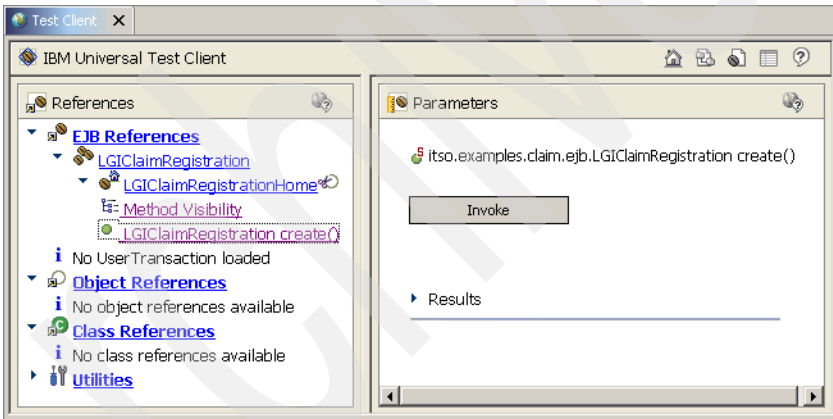


Figure 12-11 Expanded view of EJB LGIClaimRegistration

- Click **Invoke** to create the object LGIClaimRegistration. As a result, the new object LGIClaimRegistration, along with a button called *Work with Object*, will be displayed in the Result section, as shown in Figure 12-12 on page 260.

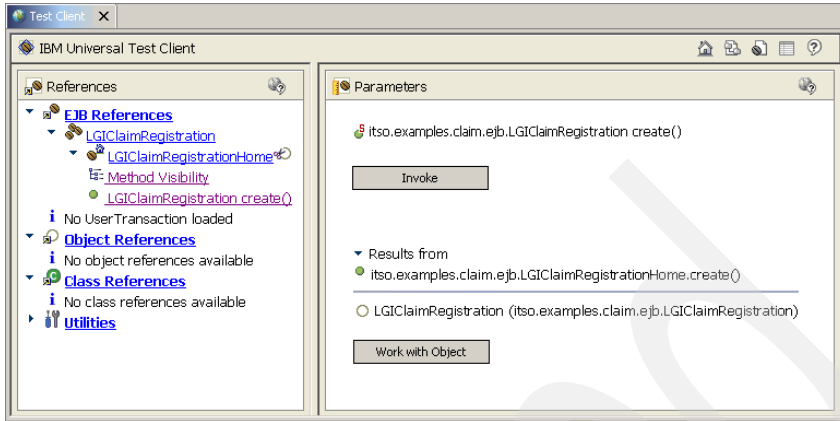


Figure 12-12 Created Object LGIClaimRegistration

8. In order to test the methods of the Object LGIClaimRegistration, click **Work with Object**. As a result, both methods findCustomer and registerClaim will be displayed under Method Visibility. You will need to expand instance LGIClaimRegistration 1, as shown in Figure 12-13.



Figure 12-13 Invoked object LGIClaimRegistration

9. To test the method findCustomer, click **findCustomer**. The right pane will show an interface to accept required input parameters for the method findCustomer, as shown in Figure 12-14 on page 261.

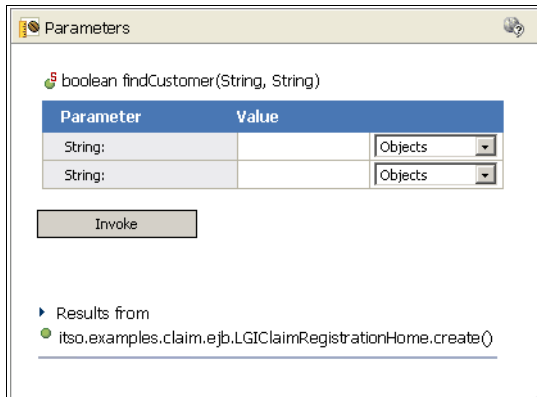


Figure 12-14 Parameters for method findCustomer

Do not type anything in either parameter. An exception with message customerID is null will be thrown if the method is invoked with both parameters, customerID and policyID, passed as a null string, as illustrated in Figure 12-15.

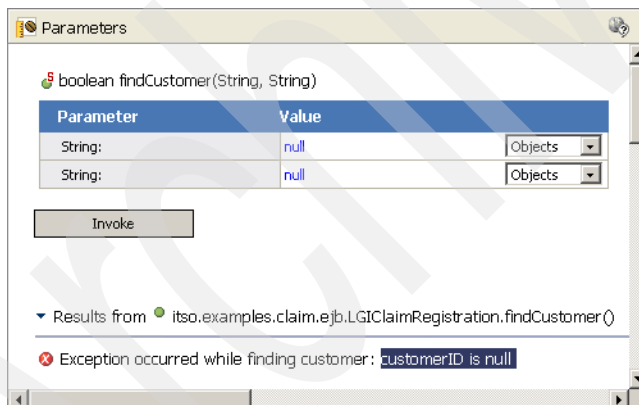


Figure 12-15 Exception with message customerID is null

Similarly, an exception with message policyID is null will be thrown when only policyID is passed as a null string. If both customerID and policyID are passed with non-null values but do not match any existing customer details, then false (boolean) will be passed as the result value by the method findCustomer. If both parameters customerID and policyID match the customer details, then true (boolean) will be passed as a result by the method findCustomer, as shown in Figure 12-16 on page 262.

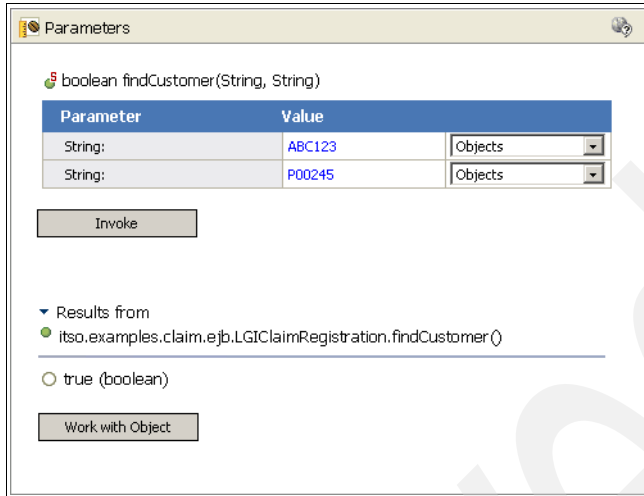


Figure 12-16 Output as *true* (boolean) on matched customer details

This testing shows that test results are as per the business logic/process of the EJB and ensure successful import and proper behavior of the EJB.

12.1.5 Create a Web service from Enterprise JavaBeans

We will create a Web service for the `ItsoClaimRegistration` EJB using the Web Service wizard of WebSphere Studio Application Developer. The wizard will also create a WSDL document, a deployment descriptor, proxy classes and a test client for the created Web service. The Web service proxy and test client are created in a separate Web projects.

Select the **ItsoClaimEJB** in the EJB Modules and then click **File** → **New** → **Other**. Select **Web Services** to display the various Web service wizards. Select **Web Service** and click **Next** to start the Web Service wizard. Go through all the pages of the wizard. Click **Next** on each page to get to the next page.

1. In the Web service type drop-down menu, select **EJB Web service** and ensure that the boxes selected in Figure 12-17 are checked; then click **Next**.

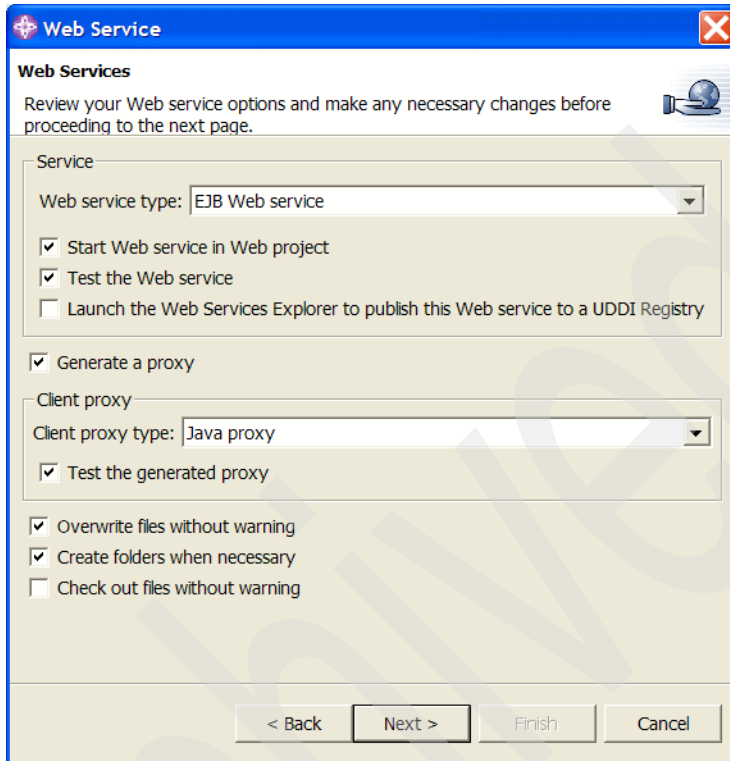


Figure 12-17 Web service options

2. Service Deployment configuration (Figure 12-18 on page 264) follows.

Select the service project and the EAR project with which you want it to be associated. If an EAR or project does not exist or is currently unassociated, it will be created and associated as required when you click Next.

Server-Side Deployment Selection:

Web service runtime: IBM WebSphere V5
 Server: WebSphere v5.1 Test Environment
 Edit...

EJB project: ItsoClaimEJB
 EJB project EAR: ItsoClaim
 Router project: ItsoClaimRouterWeb

Client-Side Environment Selection:

Web service runtime: IBM WebSphere V5
 Server: WebSphere v5.1 Test Environment
 Edit...

Client type: Web
 Client project: ItsoClaimClient
 Client project EAR: ItsoClaimClient

The client project and EAR project will be created and associated with one

Figure 12-18 Naming Web service projects

- Change Router project name to ItsoClaimRouterWeb.
- Change Client project EAR to ItsoClaimClient to avoid any runtime errors while invoking the Web service

Click **Next**.

3. Select the Web service EJB (Figure 12-19).

EAR projects ItsoClaim

EJB bean

Stateless EJB beans	EJB project
LGIClaimRegistration	ItsoClaimEJB

Figure 12-19 Select EJB to convert into a Web service

Click **Browse EJB Beans** then select **EJB bean LGIClaimRegistration**; click **Next**.

The classes will be completed by the wizard as in Figure 12-20.

JNDI provider URL:	iiop://localhost:2809	
JNDI initial context factory:	com.ibm.websphere.naming.WsnInitialContextFactory	
EJB JNDI name:	ejb/itsso/examples/claim/ejb/LGIClaimRegistrationHome	
EJB home interface class name:	itsso.examples.claim.ejb.LGIClaimRegistrationHome	Browse...
EJB remote interface class name:	itsso.examples.claim.ejb.LGIClaimRegistration	Browse...

Figure 12-20 Classes and names selected by Web service wizard

4. Web service Java Bean identity:

- Make sure that methods **findCustomer** and **registerClaim** are checked.
- Select Document/Literal as the default for Style and Use. The security configuration should be set to No Security as this Web service is for internal use only. Click **Next**.

5. In the Web service Test Page, click **Launch** to invoke the Web service browser, then click **Next**.

6. In the Web service Proxy Page, make sure that **Generate proxy** is checked, that the Output folder is /ItssoClaimEJBClient/JavaSource and that Security Configuration is selected as No Security since it is for internal access; click **Next**.

7. In the Web Service Client Test, make sure that following methods are checked:

Methods	
<input checked="" type="checkbox"/>	findCustomer(java.lang.String,java.lang.String)
<input checked="" type="checkbox"/>	getLGIClaimRegistration()
<input checked="" type="checkbox"/>	setEndpoint(java.lang.String)
<input checked="" type="checkbox"/>	registerClaim(java.lang.String,java.lang.String,java.util.Calendar,java.lang.String,java.lang.String[])
<input checked="" type="checkbox"/>	useJNDI(boolean)
<input checked="" type="checkbox"/>	getEndpoint()

Select All Deselect All

Figure 12-21 Client methods

Also ensure that **Test the generated proxy** and **Run test on server** are checked, then click **Next**.

8. In the Web service Publication window, do not check any option since we will not be publishing the Web service for a while.

Two Web service Warning windows will pop up.

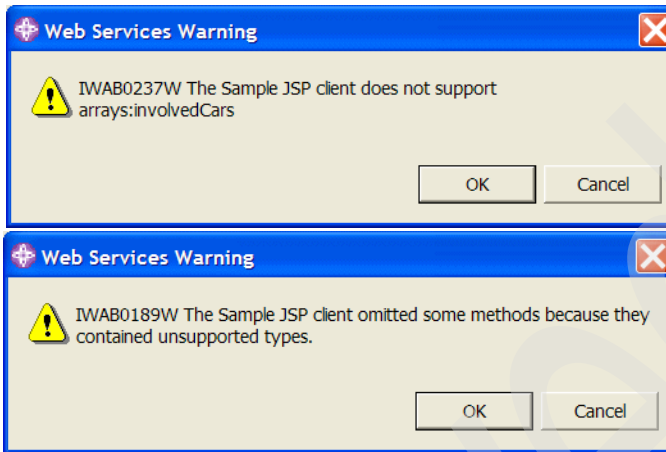


Figure 12-22 Warning messages

The `array:involvedCars` is being used in the methods `registerClaim` and hence method `registerClaim` will not be available in the JSP client.

After processing, Test Client will display the following methods to test:

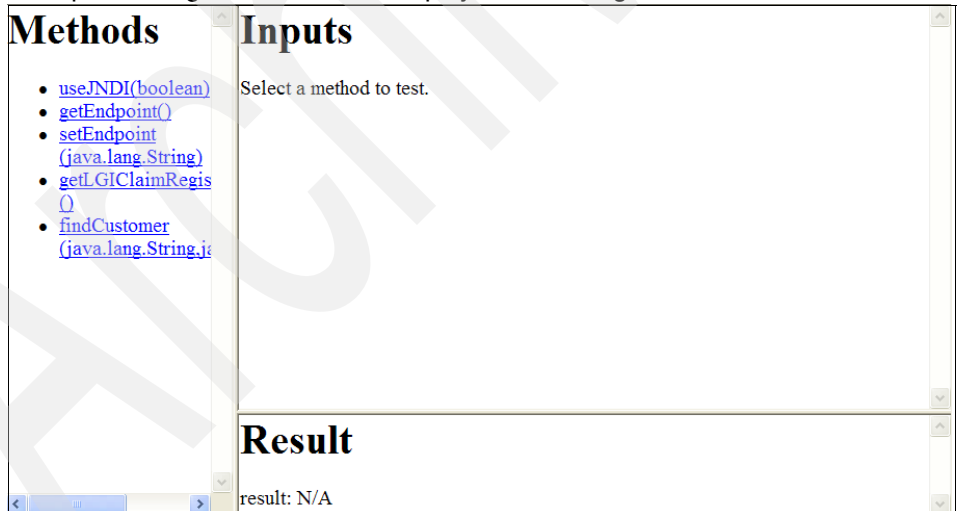


Figure 12-23 Generated test client

Note: The method registerClaim is not available to the Test Client due to array.involvedCars not being supported in the JSP client. However, the method registerClaim is available in the Web Service Explorer.

Three new projects will be created as shown in Figure 12-24. Project ItssoClaimRouterWeb contains the WSDL file for generated Web service and ItssoClaimEJBClient contains teh Test JSP Client.

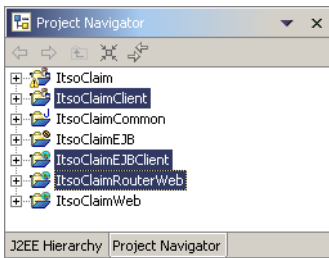


Figure 12-24 Three new projects created

Generated files

After generation of the Web service, the generated files are shown in Figure 12-25 on page 268

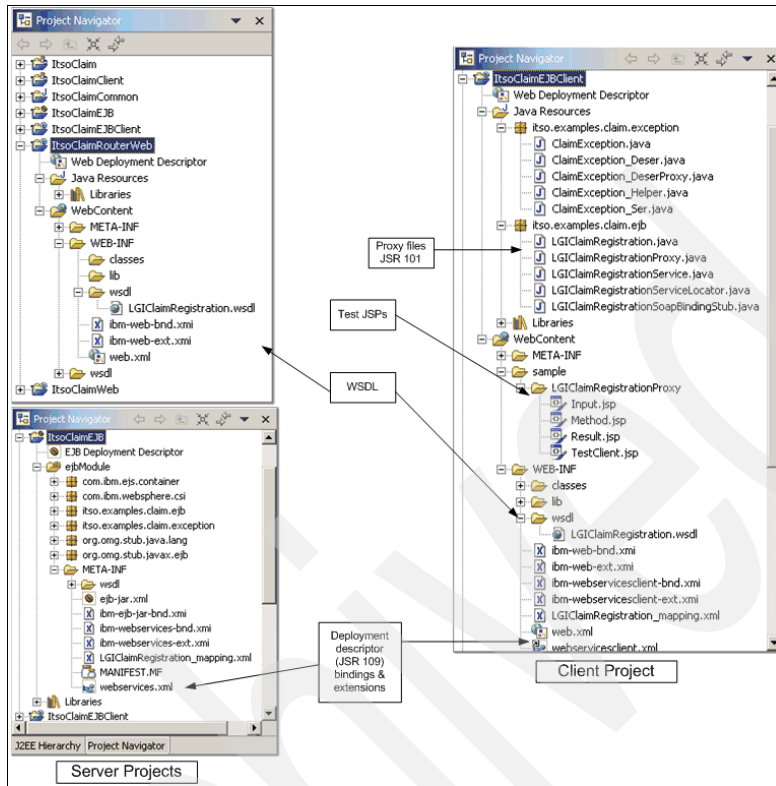


Figure 12-25 Project view after Web service generation

Files generated in the server-side Web projects

The following files have been created as per the options selected in the Web Service wizard.

- ▶ In **ItsoClaimRouterWeb**
 - The WSDL file `\WebContent\WEB-INF\wsdl\LGIClaimRegistration.wsdl` describes the Web service to the clients. A copy is also in the WebContent folder of client project `ItsoClaimEJBClient`.
- ▶ In **ItsoClaimEJB**
 - Deployment descriptor: `webservices.xml`, `ibm-webservices-ext.xml`, and `ibm-webservices-bnd.xml`. These files describe the Web service according to the Web services for J2EE style (JSR 109).
 - Service endpoint interface (SEI): `itso.examples.claim.ejb.LGIClaimRegistration_SEI.java` is the interface defining the methods of the Web service.

Files generated in the client-side Web project

Two packages are generated in the ItsoClaimEJBClient project by selecting the option to create a client-side proxy:

- ▶ Proxy classes are generated in packages `itso.example.claim.ejb` and `itso.example.claim.exception`. These classes are used by the client to make remote calls as per JSR 101. With the help of these classes, the client can instantiate local representations of the remote classes. The generated test JSPs also use these proxy classes. For more information about Web service generation using WebSphere Studio Application Developer, refer to *WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook*, SG24-6891.
- ▶ Test client: JSPs to test each method exposed as a Web service. The test client is generated in the `WebContent/sample/LGIClaimRegistryProxy` folder.
- ▶ Deployment descriptor: `webservicesclient.xml` and extension. These files describe the Web service in the client according to the Web services for J2EE style (JSR 109).
- ▶ A copy of the WSDL file (in `WEB-INF\wsdl`).

12.1.6 Test the created Web service

We can test the Web service using the generated Test JSP Client or the Web service Explorer. Since we selected the **Launch Web service Explorer** option during Web service creation, it will already be launched and available, as shown in Figure 12-26 on page 270.

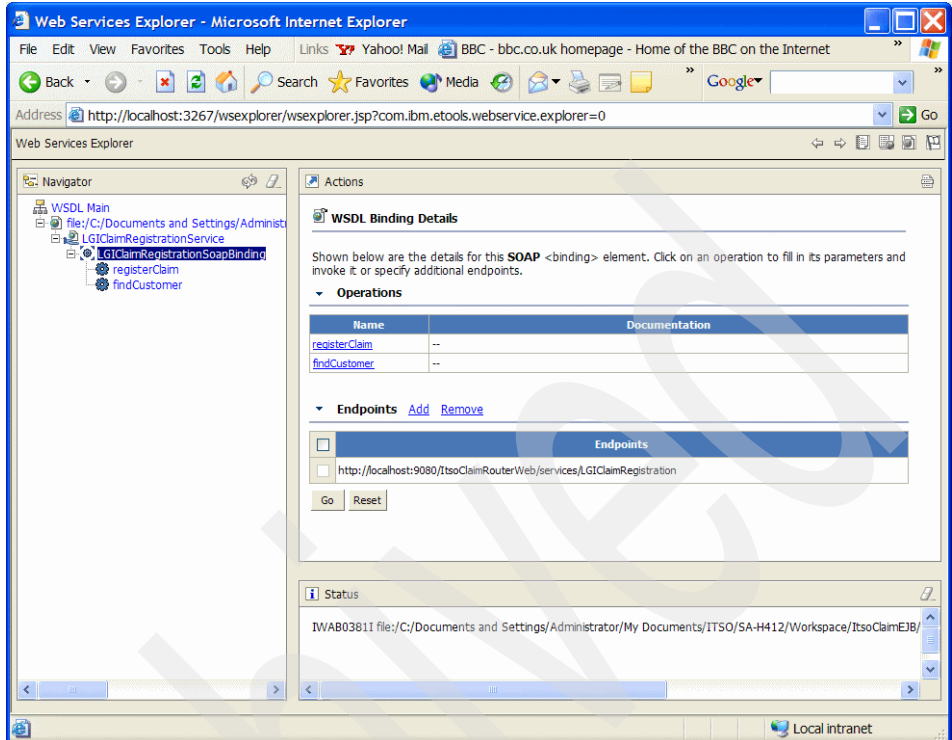


Figure 12-26 Web services explorer

The Explorer can also be launched later; use the following steps:

1. Navigate to the WSDL file of the Web service. In this scenario, this file is LGIClaimRegistration.wsdl.
2. Select the WSDL file then right-click and select **Web Services** → **Test with Web Services Explorer**. This launches the tool, as shown in Figure 12-26.
3. The Explorer will display both the methods `findCustomer` and `registerClaim`. Select the method **findCustomer**.
4. Select customer ID **ABC123** and policyID **P00245**, then click **Go**.
`findCustomerReturn (boolean): 1` will be displayed as `findCustomerResponse` in the Status pane.
5. Similarly, other test cases related to the generated Web service can be tested in the Web service Explorer.

After successfully testing all test cases, we conclude that the generated Web service provides all business logic/processes of the EJB LGIClaimRegistration.

This business logic/ processes can be accessed by other applications using the generated Web service.

12.1.7 Deploy the created Web service

In order to deploy the Web service on the application server, we will first export the application to an EAR file and then install the EAR file on WebSphere Application Server.

Exporting the application to an EAR file

1. Select the **ItsoClaim** project .
2. Select **File** →**Export**.
3. Select the EAR file as the destination, click **Next**.
4. Click **Browse** to locate the target directory. By default, the output file name populated is ItsoClaim, to distinguish it from the existing EAR. Change the file name to ItsoClaimWS and click **Save**.
5. ItsoClaimWS.ear will be generated in the target directory.

Install the EAR file on WebSphere Application Server

1. Open the WebSphere administrative console using a Web browser with the URL:
`http://<server-hostname>:9090/admin`
2. Log in with your user ID.
3. Select **Applications** →**Install New Application**.
4. Prepare for the application install interface.

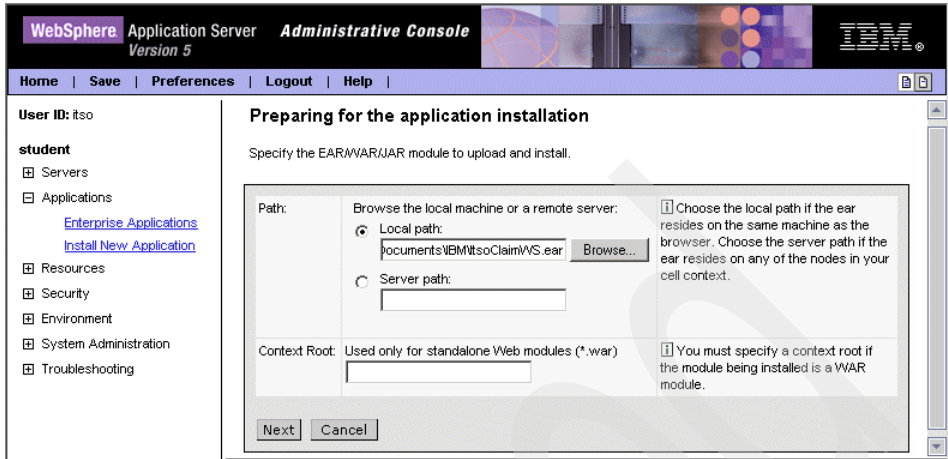


Figure 12-27 Installing ItsoClaimWS

- a. Specify the full path name of the enterprise application file (EAR file), then click **Next**.
 - b. Check **Generate Default Bindings**.
 - c. Ensure other selections are defaults, as follows:
 - Do not specify a unique prefix for beans
 - Do not override existing bindings
 - Use the default virtual host name for Web module, default_host
 - d. **Click Next**
5. Install the new application interface using the following steps.

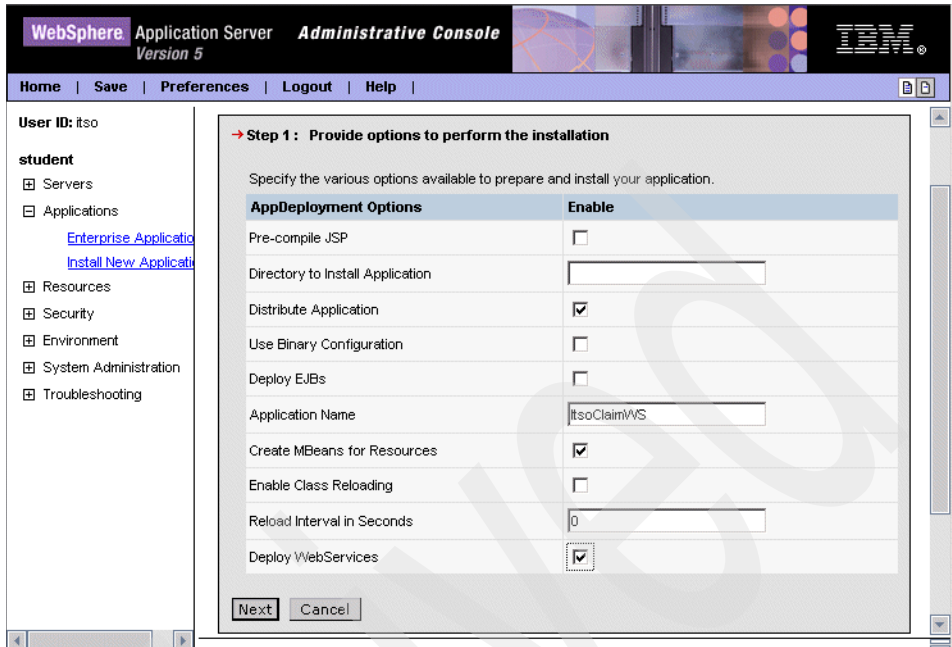


Figure 12-28 Installing ItsoclaimWS - step 1 - deploy as Web service

- a. Phase 1: first, make sure the following steps are taken:
 - Change the Application Name to ItsoClaimWS.
 - Check **Deploy WebServices**.
 - Ensure that **Distribute Application** and **Create MBeans for Resources** are checked.
- b. Phase 2: **JNDI Name** will be selected by default, so click **Next**.
- c. Phase 3: Virtual Host will be default t_host; click **Next**.
- d. Phase 4: modules will be mapped to server1 by default; click **Next**.
- e. Phase 5: there will be no security role; click **Next**.
- f. Phase 6: no change; click **Next**.
- g. Phase 7: in this step, the interface will show a summary of installation options as shown in Figure 12-29 on page 274; click **Finish**.

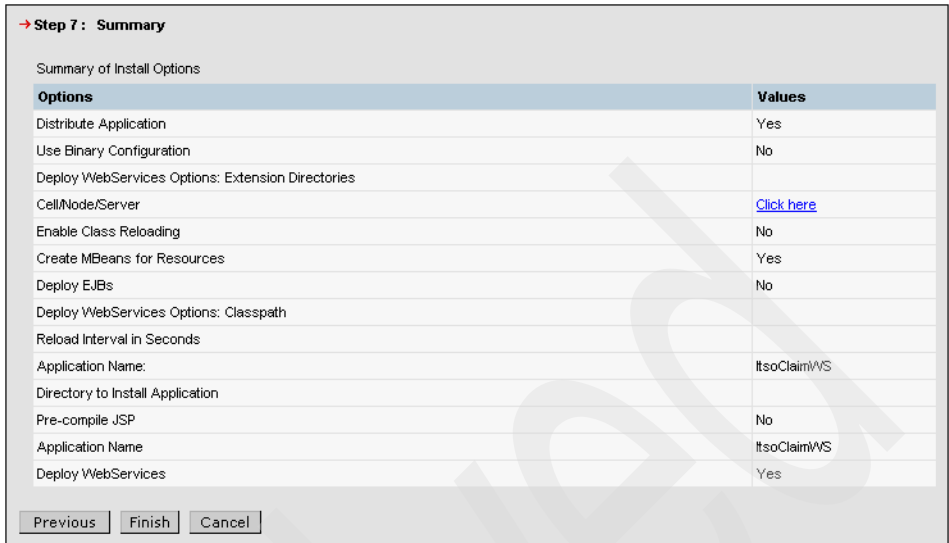


Figure 12-29 Summary of installation options

6. A success message, Application ItsoClaimWS installed Successfully, will be displayed along with the link to save the Master Configuration; click **Save to Master Configuration**.

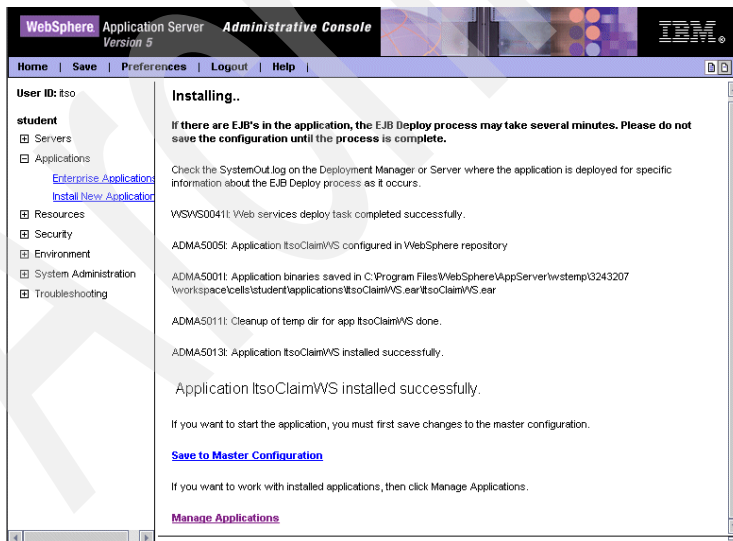


Figure 12-30 Save ItsoClaimWS to the master configuration

7. In the next interface, changed items can be viewed after expanding **View items with changes**; click **Save**.
8. Start the application as follows.

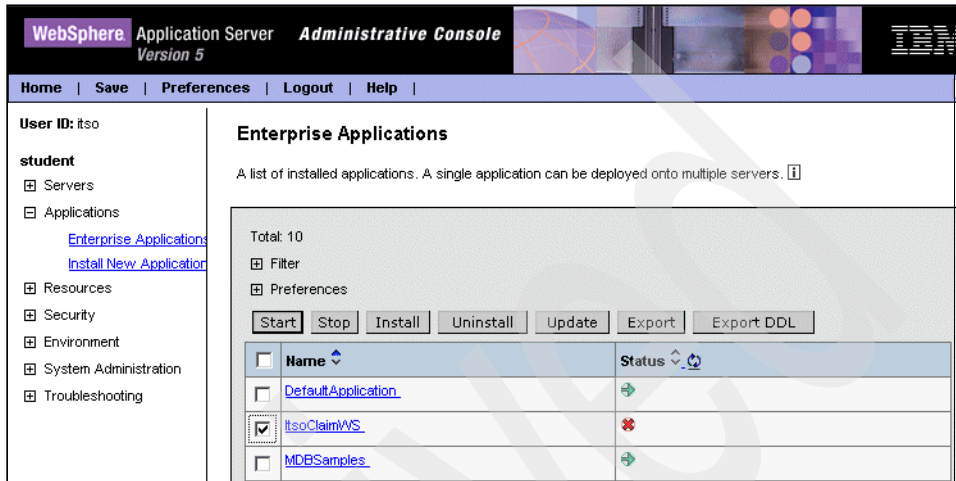


Figure 12-31 Starting ItsoClaimWS

- a. Select **Enterprise Application**.
- b. Check **ItsoClaimWS**.
- c. Click **Start**.

Now the Web service is ready to be consumed by client interface.

Test the deployed Web service using Web Service Explorer

The deployed Web service can be tested using Web Service Explorer in the development environment with WebSphere Application Server on the Test Server.

1. Follow instructions up to step 2 as described in 12.1.6, “Test the created Web service” on page 269.
2. Select **Add** to add a new Endpoint server host. Select the new Endpoint as `http://<host>:9080/ItsoClaimRouterWeb/services/LGIClaimRegistration` where <host> is the server with WebSphere Application Server on which the Web service is being deployed.
3. Select the method to test, for instance, **findCustomer** and then select **Endpoint** with the WebSphere Application Server host.

Successful execution of the test cases will verify the deployment onto WebSphere Application Server and give confidence that the Web service can be used by other Web service clients.

12.2 Building the scenario for Windows Server 2003

The next stage in the example is to create the Microsoft .Net Web service for the DCI register claim application. As with the LGI example, the register claim application is already implemented and in use with DCI's own Web site. The merger task is to wrap the existing application classes in a Web service and deploy it onto the Windows 2003 server.

12.2.1 Prerequisites to run the Web service application

The environment we set up ran the Web service application using Microsoft Visual Studio .Net 2003 on a Windows 2003 server. We used IIS V6.0 on an IBM ThinkCenter with 1GHz CPU, 1GB of main memory and 12 GB of DASD. The scenario may run on other Windows configurations, but we only tested on Windows Server 2003.

12.2.2 Create the Web Service

In Microsoft Visual Studio .Net 2003, we create an ASP .NET Web service project before we can create the Web service class and add other existing classes from other visual studio project. Start up the Microsoft Visual Studio .Net 2003.

To create a new project:

1. Select **File** → **New** in the menu at the top.
2. Select **Visual C# project** on the left-hand side.
3. Select **ASP.NET Web Service** on the right-hand side.
4. Rename the Web Service project name in Location from `http://localhost/WebService1` to `http://localhost/ItsoClaim`.
5. Click **OK**.

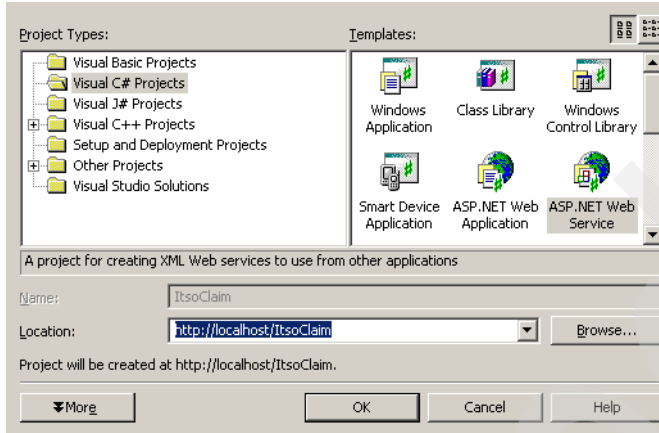


Figure 12-32 Create New Project for ItsoClaim ASP.NET Web service using C#¹

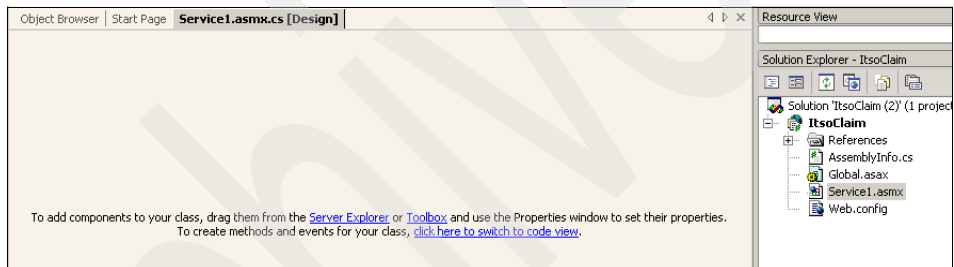


Figure 12-33 ItsoClaim.aspx.cs [Design] with default Service1 class name

The Web service page opens up and has the extension .asmx.cs[Design] as shown in Figure 12-33. The screen does not show any code, so we need to click the line **Click here to switch to code view**.

Select all the default generated code in ItsoClaim.aspx and replace it with the code in Itsoclaimpaste.txt which was prepared earlier and is shown in Example 12-1 on page 278.

The example includes two Web services that call the existing application classes:

- ▶ findCustomer (String customerID, String policyID) returns Boolean
- ▶ registerClaim (String customerID, String policyID, DateTime accidentDate, String accidentDescription, String [] involvedCars) returns claim code of type String

¹ Screen shot(s) reprinted by permission from Microsoft Corporation.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
namespace ItsoClaim {
    public class ItsoClaim: System.Web.Services.WebService{
        public ItsoClaim(){
            InitializeComponent();
        }
        #region Component Designer generated code
            private IContainer components = null;
        private void InitializeComponent(){
        }
        protected override void Dispose( bool disposing ){
            if(disposing && components != null){
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        #endregion
        [WebMethod]
        public Boolean findCustomer(String customerID,String policyID ){
            CustomerDataAccess customerObj = new CustomerDataAccess();
            try {
                return customerObj.getCustomer(customerID, policyID);
            }
            catch (DataException de){
                throw new ClaimException(de.Message);
            }
        }
        [WebMethod]
        public string registerClaim(String customerID,String policyID, DateTime
        accidentDate, String accidentDescription, String [] involvedCars){
            ClaimDataAccess claimObj = new ClaimDataAccess(customerID, policyID,
            accidentDate, accidentDescription, involvedCars);
            try {
                return claimObj.getClaimCode();
            }
            catch (DataException de1){
                throw new ClaimException(de1.Message);
            }
        }
    }
}}
```

Rename the file from Service1.asmx to ItsoClaim.asmx as in Figure 12-34.

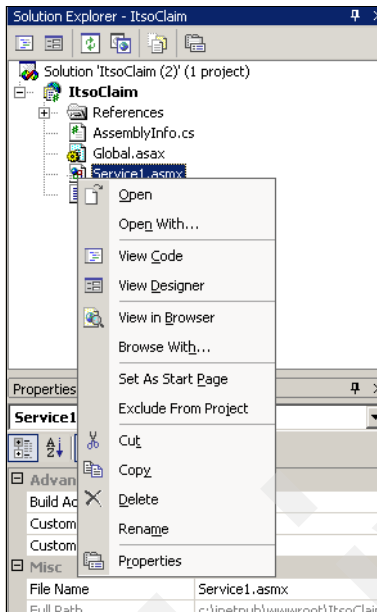


Figure 12-34 Rename Web Service

12.2.3 Import the existing classes

The next step is to import the existing application classes into the project.

Import the Existing ClaimDataAccess class

We encapsulate the access to the database by using the ClaimDataClass, even though in the example we are hard-coding the data. We could change the class to use a database without needing to change the Web service class. The ClaimDataAccess class already exists and is imported from the DirectCarInsure.com company, which is being acquired by LGI as described in the scenario use case.

To import an existing class into the project:

1. Select **File** → **Add Existing Item** from the top bar as in Figure 12-35 on page 280.

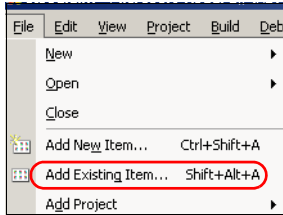


Figure 12-35 Select File and Add Existing Item

2. Navigate to the directory containing the file.
3. Select the file **ClaimDataAccess** as shown in Figure 12-36.
4. Click **Open**.

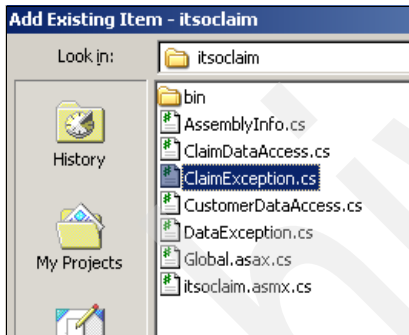


Figure 12-36 Import an existing class to the project

Example 12-2 shows the `ClaimDataAccess` class. We will instantiate the `ClaimDataAccess` class in the Web service class, `Itsoclaim.asmx`. The returned claim code is the concatenation of the `CustomerID`, `PolicyID` and `AccidentDate`.

Example 12-2 Existing ClaimDataAccess class

```
using System.Text;
namespace Itsoclaim {
    public class ClaimDataAccess {
        private String l_customerID;
        private String l_policyID;
        private DateTime l_accidentDate;
        private String l_accidentDesc;
        private String [] l_involvedCars = null;
        public ClaimDataAccess( String customerID, String policyID, DateTime
accidentDate,String accidentDesc, String [] involvedCars) {
            l_customerID = customerID;
            l_policyID = policyID;
            l_accidentDate = accidentDate;
```

```

        l_accidentDesc = accidentDesc;
        l_involvedCars = new String[involvedCars.Length];
        for (int i = 0; i < involvedCars.Length; i++) {
            l_involvedCars[i] = involvedCars[i];
        }
    }
    public String getClaimCode() {
        StringBuilder claimCode = new StringBuilder(l_customerID);
        claimCode.Append(l_policyID);
        claimCode.Append(l_accidentDate.ToShortDateString());
        return claimCode.ToString();
    }
}
}
}

```

Import the Existing CustomerDataAccess class

Similarly, we import the existing Customer Data class by adding an existing item into the project. We can modify the class without affecting the Web service class later. A `DataException` is thrown when the Customer ID is longer than eight characters. Our sample test data is hard-coded in the `CustomerDataAccess` class below.

Example 12-3 Existing CustomerDataAccess class

```

using System;
namespace ItsoClaim {
    public class CustomerDataAccess {
        public CustomerDataAccess() {
        }
        public Boolean getCustomer(String custID, String polID) {
            DataException de = new DataException("Invalid Customer");
            if (custID.Equals( "ABC1234") && polID.Equals( "1234567890"))
                return true;
            else
                if (custID.Equals( "ABC1235") && polID.Equals( "1234567891"))
                    return true;
                else
                    if (custID.Equals( "ABC1236") && polID.Equals("1234567892"))
                        return true;
                    else
                        if (custID.Length > 8)
                            throw de;
                        else
                            return false;
        }
    }
}
}

```

Import DataException and ClaimException classes

We use DataException to simulate data access errors and ClaimException to simulate exception generated by the Web service, separating the data layer from the business layer. We extend the Exception class and simply call the constructors of the parent class.

Example 12-4 Existing DataException class

```
using System;
namespace ItsoClaim {
    public class DataException:Exception {
        public DataException() : base(){}
        public DataException (String msg) : base(msg) {}
    }
}
```

The ClaimException class, just like the DataException class, can be expanded, to simulate an exception generated by the Web service..

Example 12-5 Existing ClaimException class

```
using System;
namespace ItsoClaim {
    public class ClaimException:Exception {
        public ClaimException():base() {}
        public ClaimException(String msg):base(msg) {}
    }
}
```

12.2.4 Build the Web service

The Build process generates proxy classes and the WSDL file that describes the Web service.

To build the project:

1. Select **Build** from the top bar.
2. Select **Build Solution**.

Microsoft .Net only uses Soap Document/Literal binding in the WSDL description. Even though WS-I profile 1.1 allows RPC/Literal binding, WebSphere Studio Application Developer 5.1.2, by default, generates Document/Literal binding in the WSDL description.

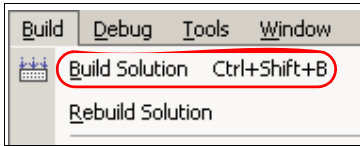


Figure 12-37 Build the Web Service

12.2.5 Microsoft Internet Information Services (IIS)

When Microsoft Visual Studio .Net 2003 rebuilds the Web service, it also publishes the Web service to the Internet Information Server, which resides in the `c:\inetpub\wwwroot` directory, as shown in Figure 12-38 on page 284. We are using Internet Information Services V6.0 with Windows 2003 Server. Internet Information Services can be separately installed by adding the Application Server Windows components in Windows 2003. There are default Web sites and virtual directories that are created when Internet Information Services are installed. We can also use the Internet Information Services Manager to create a new Web site or virtual directory.

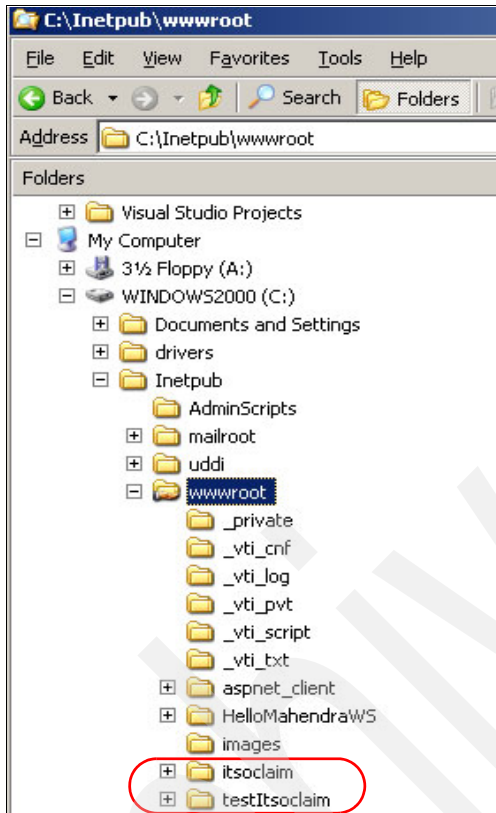


Figure 12-38 *inetpub/wwwroot* Internet Information Services directory

Open IIS Manager

To start the Internet Information Services Manager:

1. Click the **Start** menu in the bottom left corner of the desktop.
2. Select **All Programs**.
3. Select **Administrative Tools**.
4. Select **Internet Service (IIS) Manager**.

Alternatively, to start the Internet Information Services Manager:

1. Click the **Start** menu.
2. Select **Run**.
3. Type **INETMGR**.
4. Click **OK**.

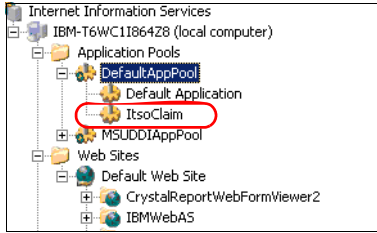


Figure 12-39 IIS Manager shows ItsoClaim on the DefaultAppPool and DefaultWebSite

If the IIS Manager does not show ItsoClaim on the DefaultAppPool and Default Web Site, right-click **DefaultAppPool** to select **Refresh**. Similarly, right-click **Default Web Site** to select **Refresh**.

12.2.6 Create Microsoft .Net Test Client

The default Microsoft Visual Studio .Net 2003 Test Client does not allow testing of applications with DateTime fields. So, we create a simple test client to make sure we can access the Web Service on Windows 2003 first. We create an ASP.NET Web form with labels and buttons. The test involves hard-coding the CustomerID and PolicyID. We also change it to have the length of the CustomerID exceed eight characters to generate an exception.

To create the test client:

1. Select **File** → **New** → **Project**.

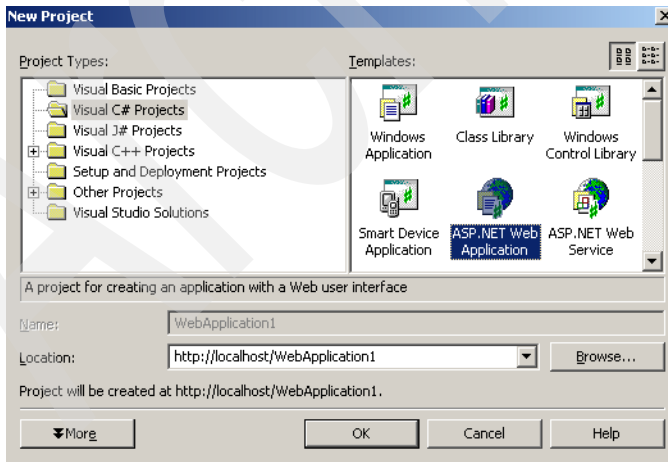


Figure 12-40 Create ASP.NET Web Application to test Web service

2. Select **Visual C# Project** in the left window pane.

3. Select **ASP.NET Web Application** in the right window pane.

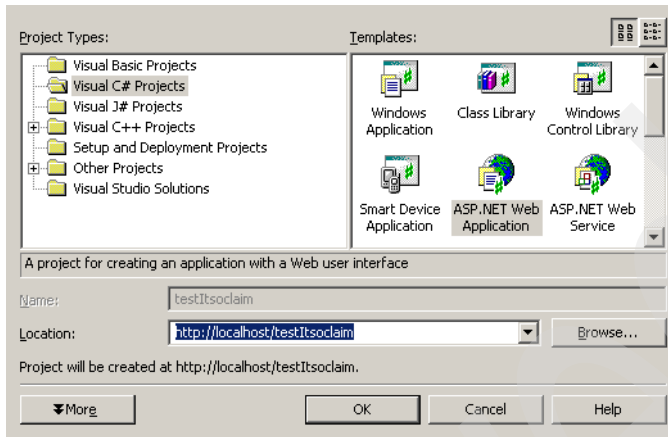


Figure 12-41 Rename Location to `http://localhost/testItsoclaim`

4. Change the location name to `http://localhost/testItsoclaim`.
5. Click **OK**.

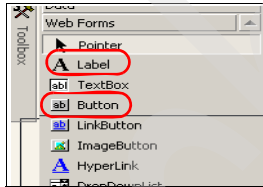


Figure 12-42 Toolbox containing buttons, labels and other widgets

6. Drag and drop two buttons and four labels from the toolbox on the left bar as in Figure 12-42.
7. Create the Web form as in Figure 12-43 on page 287.

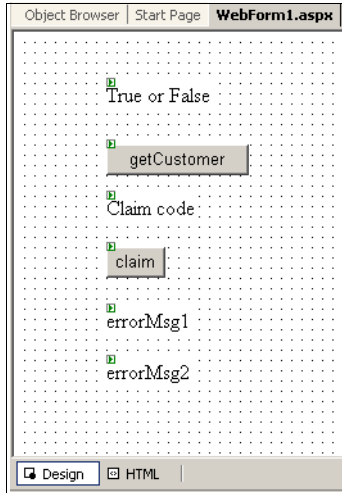


Figure 12-43 testItsoclaim form designer to test the Web service

8. Rename button1's ID and Text in the Properties window (Figure 12-44) to getCustomer.
9. Rename button2's ID and Text in the Properties window to claim.
10. Rename label1's Text in the Properties window to True or False.
11. Rename label2's Text in the Properties window to Claim Code.
12. Rename label3's ID and Text in the Properties window to errorMsg1.
13. Rename label4's ID and Text in the Properties window to errorMsg2.

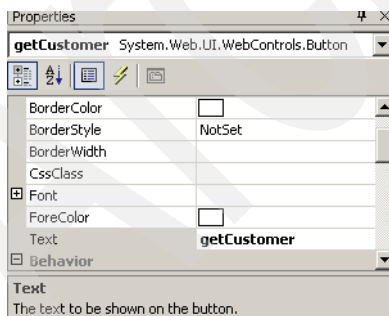


Figure 12-44 Properties window where we rename ID and Text of button and label

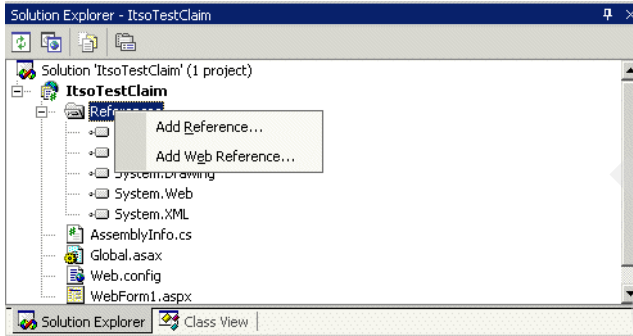


Figure 12-45 Right click Reference and select Add Web Reference

14. To add a Web reference to the ItsoClaim Web service, right-click **References** to select **Add Web Reference** in the Solution Explorer in the right pane, as shown in Figure 12-45.

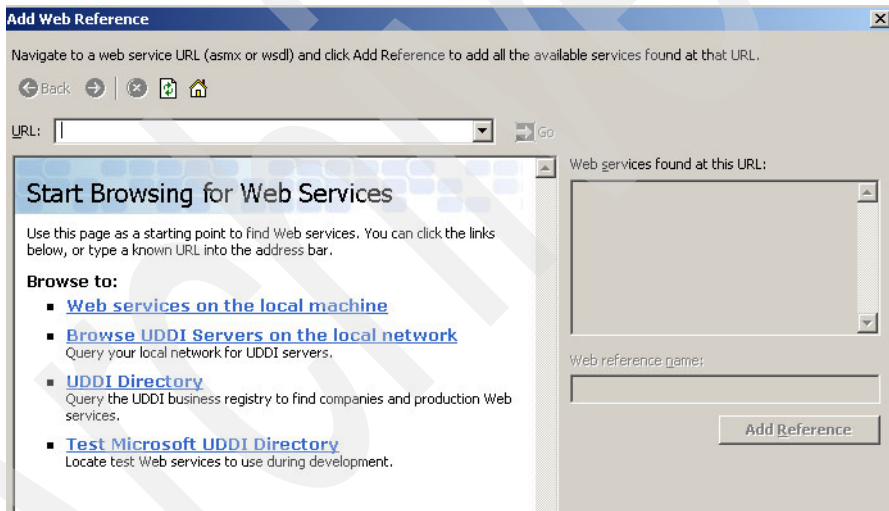


Figure 12-46 Add Web Reference

15. Select **Web services on local machine** as shown in Figure 12-46.

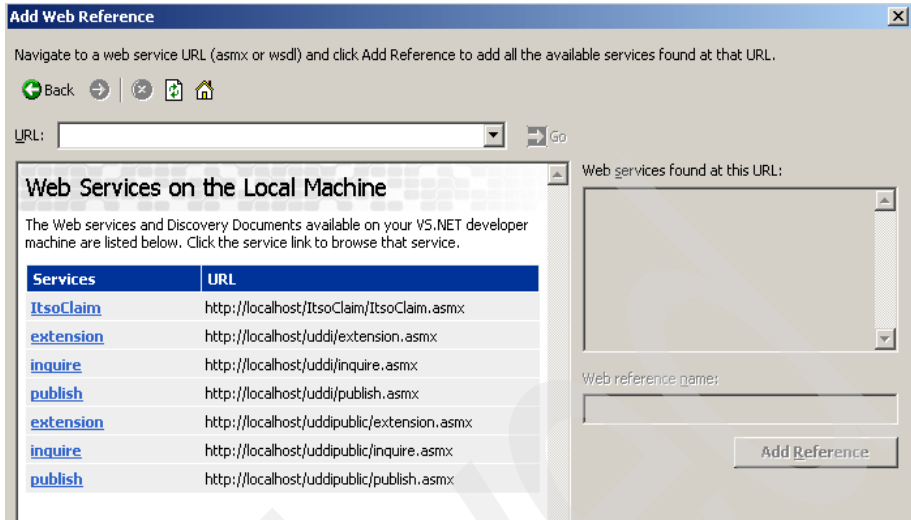


Figure 12-47 List of Web services on local machine

16. Select **ItsoClaim** from the list of Web services (Figure 12-47)

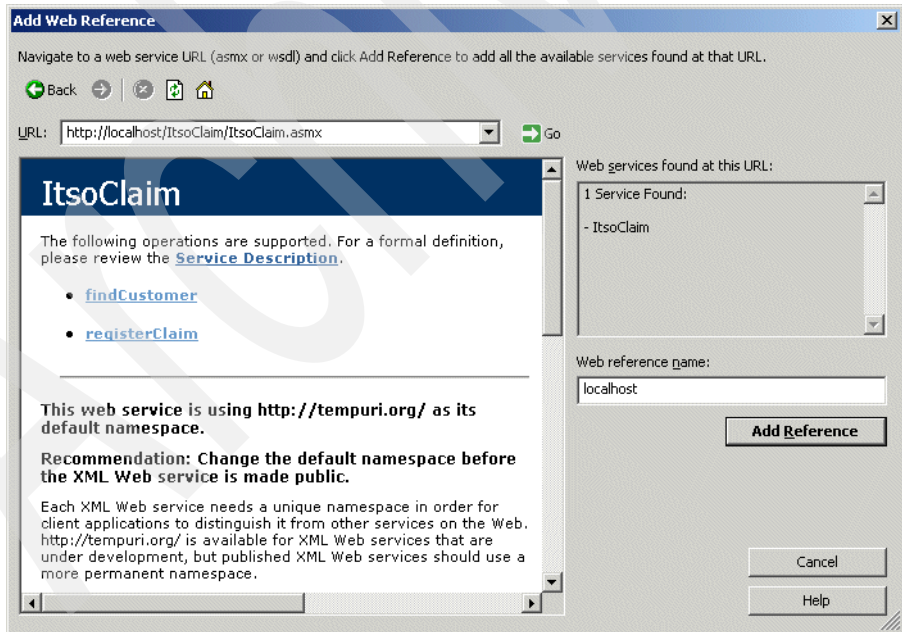


Figure 12-48 List of methods in ItsoClaim Web service

17. Click **Add Reference**.

Next, we have to add the code behind the Web form.

18. Double-click the **getCustomer** button in the form designer to view and edit the codes.
19. Select and cut the **getCustomer()** codes from the testItsoClaim.txt and paste it within the getCustomer_Click() method.
20. Similarly, go back to the form designer and double-click the **Claim** button to view and edit the codes.
21. Select and cut the claim codes from testItsoClaim.txt and paste them within the claim_Click() method.

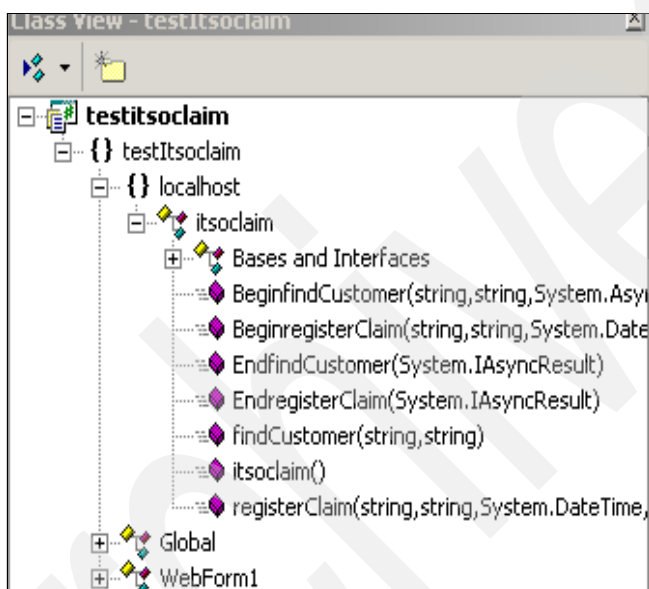


Figure 12-49 Class View shows the Web service proxy class

22. Build the solution.

Figure 12-49 shows that the Web service proxy class is localhost.ItsoClaim.

The codes that accompany the window are as follows:

Example 12-6 testItsoClaim.aspx.cs test client codes

```
namespace testItsoClaim {
    public class WebForm1 : System.Web.UI.Page {
//.....Microsoft Visual Studio .Net 2003 generates extra codes here
// Cut and paste the testItsoClaim.txt and paste it within WebForm1 class
// below other VS .NET automatically generated codes
```

```

private void getCustomer_Click(object sender, System.EventArgs e) {
    localhost.ItsoClaim myClaim = new localhost.ItsoClaim();
    try {
        getCustomer.Text=myClaim.findCustomer("ABC1234",
            "1234567890").ToString();
    }
    catch (Exception exc) {
        errorMsg1.Text = "You have got an ERROR";
        errorMsg2.Text = exc.Message;
    }
}
private void claim_Click(object sender, System.EventArgs e) {
    String [] myArray = new String[] {"me","you","him"};
    DateTime myDate = new DateTime(2004,09,26);
    localhost.ItsoClaim myClaim = new localhost.ItsoClaim();
    claim.Text=myClaim.registerClaim("ABC1234","1234567890",
        DateTime.Today, "Just an accident",myArray);
}
}
}

```

To run the Test client, click **Debug** → **Start Without Debugging** in the top menu bar.

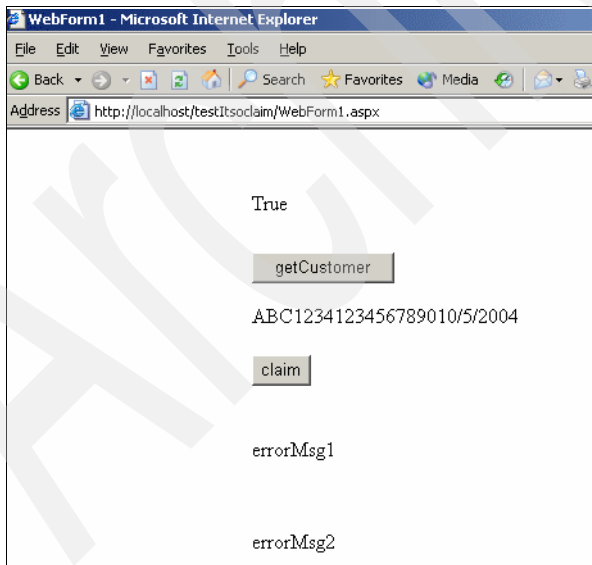


Figure 12-50 Test result when Customer ID is ABCD1234, policy ID is 1234567890

Click the **getCustomer** button and **Claim** button. Figure 12-50 shows the result screen indicating that the test passes and there is no exception.

Stop the application. Go back to the `getCustomer_Click()` method in the editor and change the Customer ID to ABC1234999 then run the application again. Click the **getCustomer** button and you should get an exception as shown in Figure 12-51.

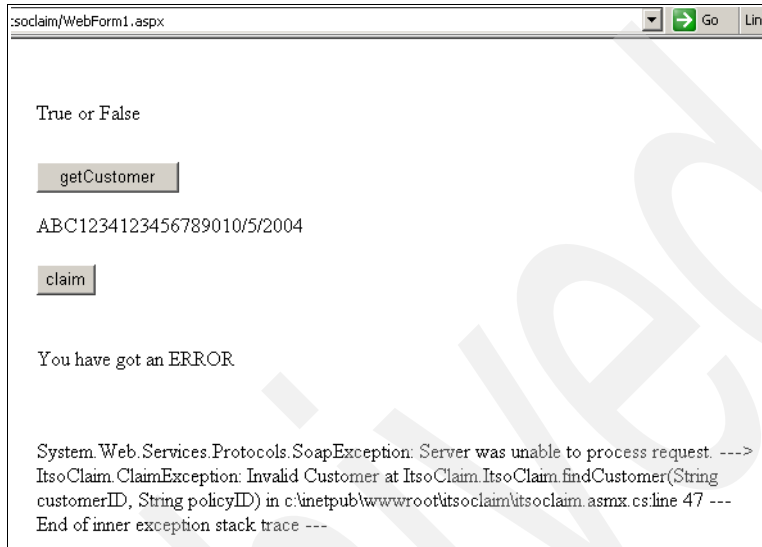


Figure 12-51 Exception is thrown when CustomerID is greater than 8 characters

We get the exception when we code the customer ID such that it is longer than eight characters. Figure 12-51 indicates that an exception has occurred because the customer ID is longer than eight characters. In a complex application, the exception can be the result of database access failure due to the record locking or the database server not running.

12.2.7 Summary

We have created the Web service class and imported its existing supporting classes. The Rebuild generates the Web service proxies and the WSDL file. We created the simple Test client to make sure that the Web service runs properly in the Microsoft .Net platform. We also briefly touched on the Microsoft Internet Information Services where Microsoft Visual Studio .Net 2003 automatically deploys its Web service and through which we can confirm our Web applications are correctly deployed.

12.3 Building the Web services clients

Now that we have built a and test wrappers for the insurance claims application in Microsoft .Net and WebSphere, we can build the new Web application that will make use of the two Web services.

A Java 2 Enterprise Edition Web service client, also known as a service consumer or service requestor, is an application component acting as a client for a specific Web service. As in all classical remoting implementations, such as CORBA or RMI, the Web service client component is implemented as a proxy object able to wrap the remote invocation to the rest of the business logic.

As already described in the previous chapters, a Web service client communicates with the Web service provider using SOAP messages. Both the SOAP request and SOAP response messages must follow the format specified in the WSDL file associated to the Web service. The WSDL file gives a complete set of specifications about the information to be put in the SOAP request and response, including all information regarding operations exposed by the Web service, input and output variables, variable types, document encoding type and, last but not least, the Web service location. Therefore, the WSDL file is both necessary and sufficient information to develop a client for the corresponding Web service.

Both WebSphere and Microsoft .Net platforms provide automatic tools for generating a proxy class wrapping a Web service. After building the proxy, each application having to access a Web service need only instantiate the proxy class and invoke the required operation on it.

In this section, we show how it is possible to build two WebSphere Web service clients: one for a WebSphere Web service and the other one for a Microsoft .Net Web service. The Web services we want to invoke are the same as we created in the previous two sections.

In both cases, we start from the WSDL file provided by the Web service. We can opt to download the WSDL file and copy it in the WebSphere Studio Application Developer workspace or, if we can remotely access the WSDL file from the development environment, we can simply provide the remote URI address when required by the Web service proxy wizard. The only advantage in copying the WSDL file in the local workspace is that we can test it using the Web Service Explorer even before developing any code.

Note: More information about how to use WebSphere Studio Application Developer to build an test Web services client or proxy can be found in the redbook *WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook*, SG24-6891-01.

In this example, we suppose that WSDL files have been copied to the local disk and that the Web service client is required to be part of the Web application that will give LGI's and DCI's combined customers a single Web interface to register an insurance claim.

Rather than build the entire Web application, we will only build the Java code that is going to act as a Web services client to the LGI and DCI Web services. The Java code will be part of the Web application's servlet. To test the code in isolation, we will build it as a simple Java client. We will not build the servlet or the rest of the Web application.

12.3.1 Web service client for the WebSphere Web service

To create a proxy class and all other related classes, the same steps are required for both a WebSphere Web service and a Microsoft .Net Web service. Nevertheless, since the WSDL files are different, the results are also different and must be individually discussed before a comparison can be made. Steps to create a client for a WebSphere Web service are detailed below:

1. Open a new workspace in the WebSphere Studio Application Developer tool and set the Java perspective as a current active perspective. If the WebSphere Studio Application Developer tool has already been used to develop the Web service, a new workspace is preferable, instead of having the Web service code in the same workspace, in order to show the effective execution of a remote object.
2. Create a new Java project:
 - a. Select **File** → **New** → **Project** → **Java** → **Java Project** and click **Next**.
 - b. Specify `ItsoClaimWasWSCClient` as the project name and click **Next**.
 - c. Click **Add Folder** to open the Source Folder Selection window and click **Create New Folder**.
 - d. Specify source as folder name and click **OK**.
 - e. Click **OK** to close the Source Folder Selection window.
 - f. Click **Yes** to the following question.
 - g. Click **Finish**.
3. Import the `LGIClaimRegistration.wsdl` file in the root project directory (as stated before, it is not mandatory to have a local WSDL file).
 - a. Select the `ItsoClaimWasWSCClient`.
 - b. Select **File** → **Import**.
 - c. Select **File system** as the input source and click **Next**.
 - d. Click **Browse** to select the directory where the WSDL file is located.

- e. Select the WSDL file in the left window under the Browse button and click **Finish**. If the Web Services server is deployed on a different machine, be sure that the address location in the WSDL file refers correctly to the remote machine; otherwise, the address can be replaced with the correct one.

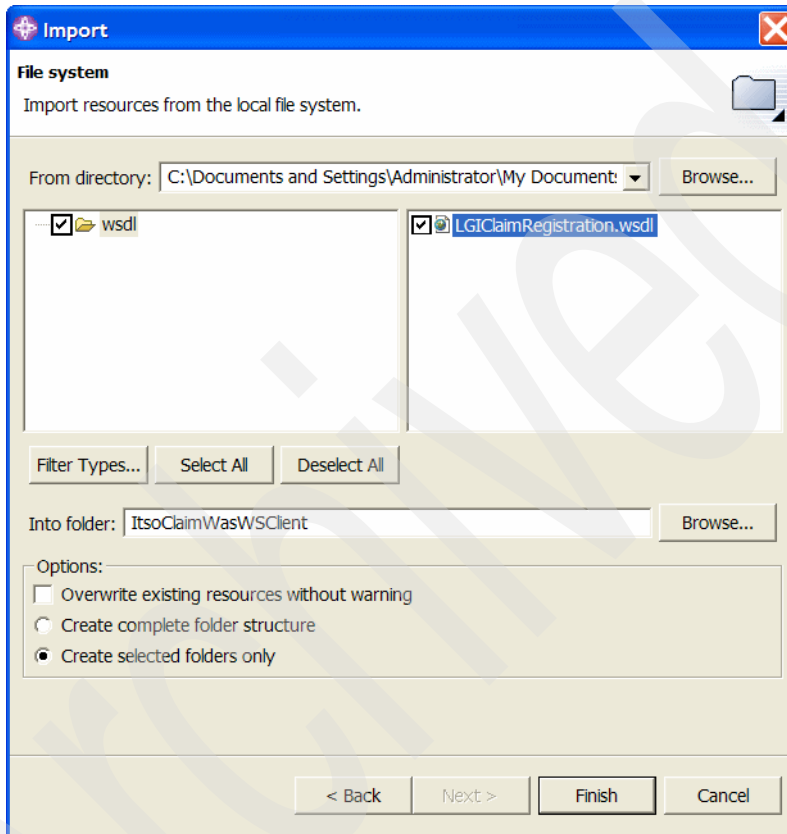


Figure 12-52 Importing WSDL file into Web service client project

- f. Now the Web Service can be tested, even before creating the proxy: select the WSDL file, open the context menu and select **Web Services** → **Test with Web Services Explorer**.
4. Now we have two choices to run the Web service client wizard:
- Select **File** → **New** → **Other** → **Web Services** → **Web Service Client** and click **Next**.
 - or alternatively:

- Select the WSDL file, open the context menu and click **Web Services** → **Generate Client**.
- 5. Select **Java proxy** as the client proxy type and click **Next** (do not select **Test the generated proxy**).
- 6. Verify that the Client project is `ItsoClaimWasWSCClient` and click .
- 7. Click **Browse** to select the WSDL file if not yet selected and click **Next**.
- 8. Click **Finish**.

The final result is shown in Figure 12-53.

Tip: Make sure the plug-ins shown in Figure 12-53 are listed; they should load automatically. If not, check that you have selected the WebSphere 5.1 test server as your target server correctly, either as a default under Preferences, or as shown in Figure 12-7 on page 257.

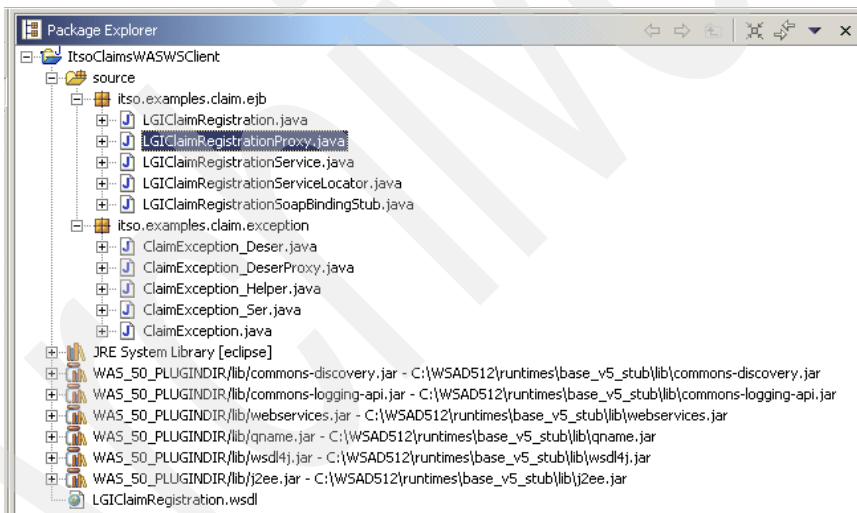


Figure 12-53 Web service client Java classes for a WebSphere Web service

Besides the Proxy class, four other proxy classes have been generated to support the proxy invocation:

- ▶ Service interface *LGIClaimRegistrationService* - defines the service methods of the locator class (for example, retrieving the SEI)
- ▶ Service locator class *LGIClaimRegistrationServiceLocator* - implements the service interface (provides access to the SEI)

- ▶ Service endpoint interface (SEI) *LGIClaimRegistration* - defines the method signatures of the Web service
- ▶ Binding stub *LGIClaimRegistrationSoapBindingStub* - implements the SEI (makes the actual calls to the Web service)

The Claim Exception class is automatically generated to support the SOAP fault described in the WSDL files.

At runtime, the client instantiates the service locator class, calls it to retrieve the SEI (actually the binding stub), then calls the SEI to invoke the Web service. Figure 12-54 shows the calling sequence in a Java implementation.

1. The client instantiates the service locator.
2. The client calls the service locator to retrieve the SEI (an instance of the client stub that implements the SEI is returned).
3. The client invokes a Web service through the SEI.

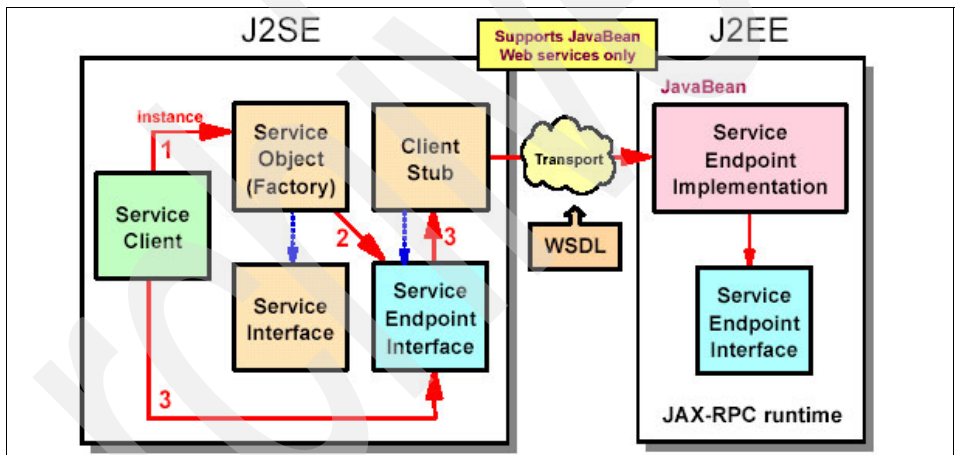


Figure 12-54 JAX-RPC static client calling sequence

If we want to test the generated code invoking a remote Web service method, we must simply create an instance of the proxy and invoke the corresponding method exposed by the proxy object. Proxy invocation can be done from a Java class, an EJB, a Java Servlet or any other Java component, even a JSP. For example, if we want to invoke the `findCustomer` method, the client code should be the one listed in the following example:

Example 12-7 Web service proxy invocation from a Java client

```
LGIClaimRegistrationProxy proxy = new LGIClaimRegistrationProxy();
boolean result = proxy.findCustomer(aCustomerID, aPolicyID);
```

where both `aCustomerID` and `aPolicyID` are the input variables. Input variable values and related expected results are discussed in the section dealing with the creation of the Web service.

The additional material provided with this redbook includes a complete Java class to test the Web service. To import the Java class, follow these steps:

1. Select the **source** folder in the Java project.
2. Select **File** → **Import**.
3. Select **File system** as the input source and click **Next**.
4. Click **Browse** to select the **WSWASClient** directory located in the additional material root directory and click **OK**.
5. Check the **WSWASClient** in the left window under the Browse button and click **Finish**.

The `itso.examples.claim.test` package is created under the source folder containing the `TestWASWebServiceJava` class. The new package and class are shown in Figure 12-55.

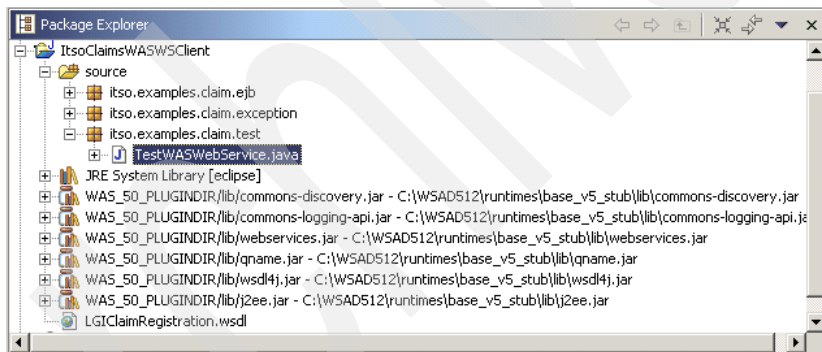


Figure 12-55 Test Web services client class

You must run the class as a Java application, providing the inputs for the method invocation. Running the Java class without any input or help about how to provide the inputs is displayed in the console.

To run the class, follow these steps:

1. Select the class.

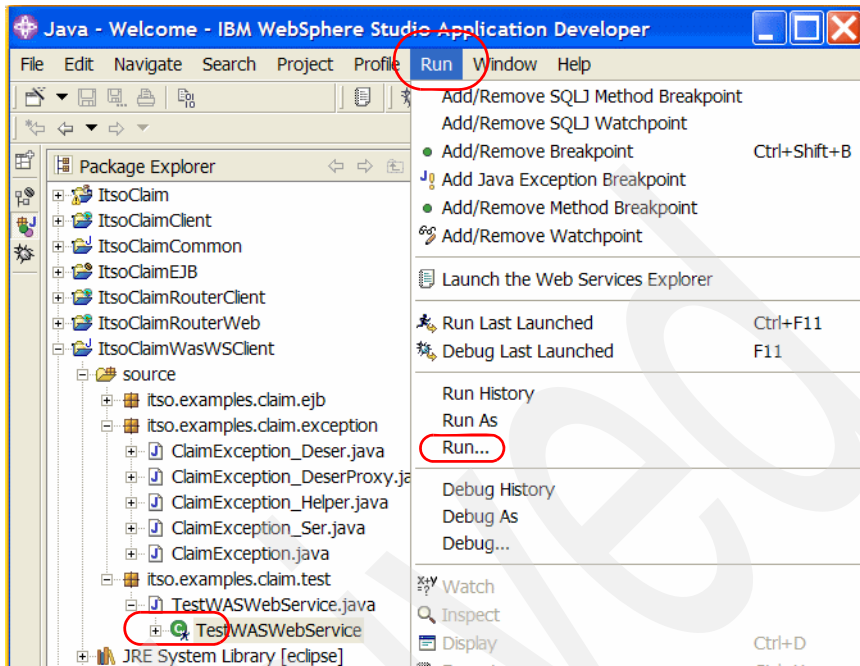


Figure 12-56 Starting the test client

2. Click **Run** → **Run...** as in Figure 12-56.
3. Select **Java Application** in the left side navigator.
4. Click **New**.
5. A new tabbed panel opens on the right side. Select the **Arguments** tab.

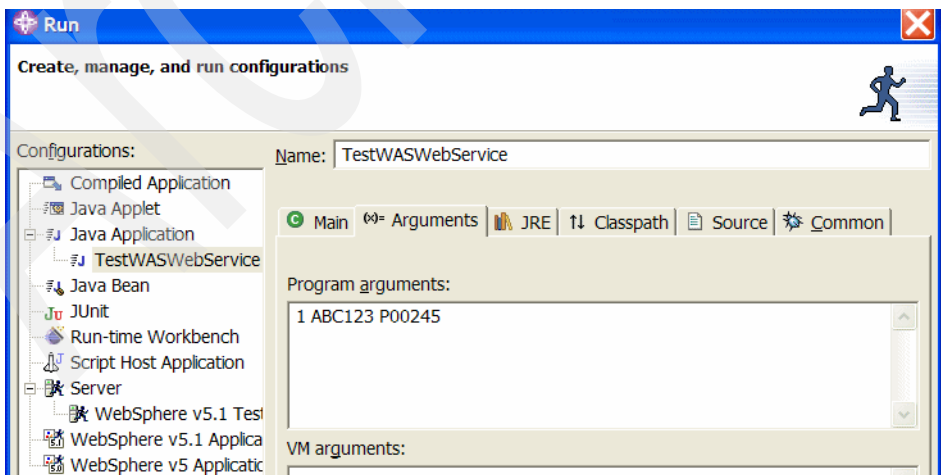


Figure 12-57 Supply the parameters to the test Web service

6. In the *program arguments* text window, insert for example the following text: 1 ABC123 P00245.
7. Click **Run**.

The client starts and the console reports the message as shown in Figure 12-58:

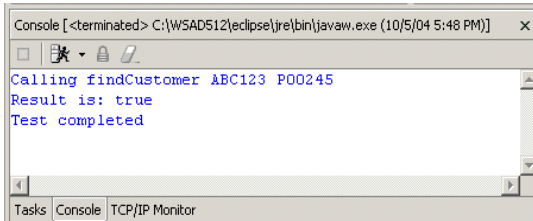


Figure 12-58 Console message after the execution of the test client

12.3.2 Web service client for the Microsoft .Net Web service

To create a client for a Microsoft .Net Web service, follow these steps:

1. Open the WebSphere Studio Application Developer tool and set the Java perspective as the current active perspective.
2. Create a new Java project:
 - a. Select **File** → **New** → **Project** → **Java** → **Java Project** and click **Next**.
 - b. Specify `ItsoClaimDotNetWSCClient` as the project name and click **Next**.
 - c. Click **Add Folder** to open the Source Folder Selection window and click **Create New Folder**.
 - d. Specify source as the folder name and click **Ok**.
 - e. Click **Ok** to close the Source Folder Selection window.
 - f. Click **Yes** to the following question.
 - g. Click **Finish**.
3. Import the `itsocclaim.wsdl` file in the root project directory. As stated before, it is not mandatory to have a local WSDL file; if the local file is missing, follow steps from a. to f., otherwise skip to step g.
 - a. Be sure you have the Microsoft .Net Web service up and running.
 - b. Be sure you have the service endpoint. We assume it is:
`http://dotnethost.itso.ibm.com/ItsoClaim/ItsoClaim.asmx`.
 - c. Open a Web browser and go to the following link:
<http://dotnethost.itso.ibm.com/ItsoClaim/ItsoClaim.asmx?WSDL>

- d. The WSDL file is shown in the browser.
 - e. Select **File** → **Save As...** to save the WSDL file in the local disk.
 - f. Select **ItsoClaimDotNetWSClient**.
 - g. Select **File** → **Import**.
 - h. Select **File system** as the input source and click **Next**.
 - i. Click **Browse** to select the directory where the WSDL file is located.
 - j. Select the WSDL file in the left window under the Browse button and click **Finish**. If the Web Services server is deployed on a different machine, be sure that the address location in the WSDL file refers correctly to the remote machine; otherwise, the address can be replaced with the correct one.
 - k. Now the Web service can be tested, even before creating the proxy: select the WSDL file, open the context menu and select **Web Services** → **Test with Web Services Explorer**.
4. Now we have two choices to run the Web service client wizard:
 - Select **File** → **New** → **Other** → **Web Services** → **Web Service Client** and click **Next**.
 - or alternatively:
 - Select the WSDL file, open the context menu and click **Web Services** → **Generate Client**.
 5. Select **Java proxy** as the client proxy type and click **Next** (do not select **Test the generated proxy**).
 6. Verify that the Client project is ItsoClaimWasWSClient and click **Next**.
 7. Click **Browse** to select the WSDL file if not yet selected and click **Next**.
 1. Click **Finish**.

The final result is shown in Figure 12-59 on page 302.

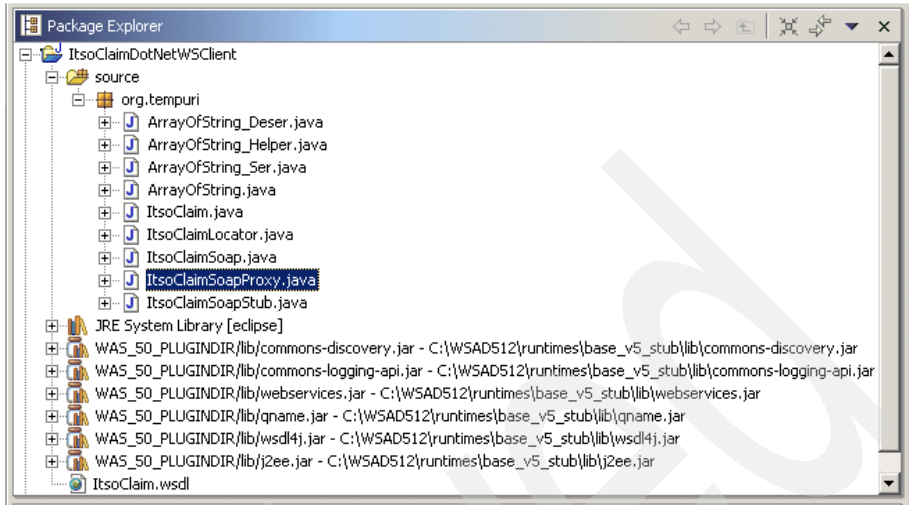


Figure 12-59 Web service client Java classes for a 0738492302 Microsoft .Net Web service

As in the previous example, the only code needed in the client application to invoke the findCustomer method is:

Example 12-8 Web service Proxy invocation from a Java client

```
ItsoClaimSoapProxy proxy = new ItsoClaimSoapProxy();
boolean result = proxy.findCustomer(customerID,policyID);
```

where both aCustomerID and aPolicyID are the input variables. Inputs variable values and related expected results are discussed in the section dealing with the creation of the Web service.

The additional material provided with this redbook includes a complete Java class to test the Web service. To import the Java class, follow these steps:

1. Select the **source** folder in the Java project.
2. Select **File** → **Import**.
3. Select **File system** as the input source and click **Next**.
4. Click **Browse** to select the **WSDotNETClient** directory located in the additional material root directory and click **OK**.
5. Check the **WSDotNETClient** in the left window under the Browse button and click **Finish**.

The `itso.examples.claim.test` package is created under the source folder. It contains the `TestDotNETWebServiceJava` class. You must run the class as Java application, providing the inputs for the method invocation. Running the Java class without any input will help about how to provide the inputs is displayed in the console.

12.3.3 Microsoft .Net

The steps required to build a Web service client in Microsoft Visual Studio .Net 2003 are the same as already described to build the test client for the Microsoft .Net Web service.

12.3.4 Differences between the two Web services and conclusions

The main differences between the Microsoft .Net generated WSDL file and the WebSphere one are:

- ▶ Exception handling
- ▶ Object array management
- ▶ Parameter multiplicity specification

For each difference, we provide a specific description in the following sections.

Exception handling

Business logic component methods belonging to both development environments throw a `ClaimException` to report some errors that may have occurred during the method execution.

No SOAP fault information is included in the Microsoft .Net WSDL file. The lack of detail is probably related to the fact that WSDL files generated in Microsoft .Net start from a C# class; since C# does not have an analog of the Java throws clause in method signatures, the method signature does not contain any information about exceptions thrown during the method execution.

In the WebSphere Studio generated WSDL file, a complex type is defined to map the `ClaimException` and the SOAP fault is associated with this type. The following example shows the exception handling in the WebSphere Studio generated WSDL file:

Example 12-9 Exception handling in the WebSphere Studio generated WSDL file

```
.....  
<wsdl:types>  
.....  
  <schema elementFormDefault="qualified".....>  
    <complexType name="ClaimException">
```

```

        <sequence>
            <element name="message" nillable="true" type="xsd:string" />
        </sequence>
    </complexType>
    <element name="ClaimException" nillable="true" type="tns2:ClaimException"
    />
</schema>
</wsdl:types>
....
<wsdl:message name="ClaimException">
    <wsdl:part element="tns2:ClaimException" name="fault" />
</wsdl:message>
.....
<wsdl:portType name="LGIClaimRegistration">
    <wsdl:operation name="findCustomer">
        <wsdl:input message="intf:findCustomerRequest"
        name="findCustomerRequest" />
        <wsdl:output message="intf:findCustomerResponse"
        name="findCustomerResponse" />
        <wsdl:fault message="intf:ClaimException" name="ClaimException" />
    </wsdl:operation>
    <wsdl:operation name="registerClaim">
        <wsdl:input message="intf:registerClaimRequest"
        name="registerClaimRequest" />
        <wsdl:output message="intf:registerClaimResponse"
        name="registerClaimResponse" />
        <wsdl:fault message="intf:ClaimException" name="ClaimException" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="LGIClaimRegistrationSoapBinding"
type="intf:LGIClaimRegistration">
    <wsdl:soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="findCustomer">
        <wsdl:soap:operation SOAPACTION="" />
        <wsdl:input name="findCustomerRequest">
            <wsdl:soap:body use="literal" />
        </wsdl:input>
        <wsdl:output name="findCustomerResponse">
            <wsdl:soap:body use="literal" />
        </wsdl:output>
        <wsdl:fault name="ClaimException">
            <wsdl:soap:fault name="ClaimException" use="literal" />
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="registerClaim">
        <wsdl:soap:operation SOAPACTION="" />
        <wsdl:input name="registerClaimRequest">
            <wsdl:soap:body use="literal" />
        </wsdl:input>

```

```

        </wsdl:input>
        <wsdl:output name="registerClaimResponse">
            <wsdlsoap:body use="literal" />
        </wsdl:output>
        <wsdl:fault name="ClaimException">
            <wsdlsoap:fault name="ClaimException" use="literal" />
        </wsdl:fault>
    </wsdl:operation>
</wsdl:binding>
.....

```

Starting from the previously described WSDL file, the Web service proxy wizard generates an `itso.examples.claim.exception` package, a `ClaimException` class and all other related classes needed for the SOAP serialization and deserialization of the `ClaimException` itself.

The `SoapBindingStub` class which wraps the methods exposed by the Web service throws both a `java.rmi.RemoteException` and a `itso.examples.claim.exception.ClaimException`.

Part of the code implementing the generated method is shown in the following example:

Example 12-10 Exception handling in the stub generated from a WebSphere WSDL file

```

try {
    .....
} catch (com.ibm.ws.webservices.engine.WebServicesFault wsf) {
    Exception e = wsf.getUserException();
    if (e != null) {
        if (e instanceof itso.examples.claim.exception.ClaimException) {
            throw (itso.examples.claim.exception.ClaimException) e;
        }
    }
    throw wsf;
}

```

If we compare the code listed in Example 12-10 with the corresponding code in Example 12-11 on page 306 generated from the Microsoft .Net WSDL file, we can observe that in the second case, the only handled exception is the standard `WebServicesFault`. Both the `findCustomer` and `registerClaim` methods throw only a `java.rmi.RemoteException`.

Example 12-11 Exception handling in the stub generated from a Microsoft .Net WSDL file

```
try {  
    .....  
} catch (com.ibm.ws.webservices.engine.WebServicesFault wsf) {  
    throw wsf;  
}
```

The conclusion is that it is not possible, while developing a Microsoft .Net Web service, to generate specific exceptions with the client. Microsoft .Net Studio does not add any SOAP fault message in the WSDL files and all exceptions thrown by the Web service are managed as simple SOAP server fault code. This different implementation of the exception management in the SOAP message, however, does not impact the interoperability between the two platforms.

To produce a detailed SOAP fault report from a Microsoft Web service requires some coding. Some good advice is given in the MSDN article “*Using SOAP Faults*” (Scott Seely, Microsoft Corporation, September 20, 2002) found at:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnservice/html/service09172002.asp>

Object array management

In the registerClaim method, an array of strings is required as an input parameter. In both cases, the Web service source code is developed using the basic String[] type, but, if we compare the two generated WSDL files, we can find a different parameter declaration approach.

The WebSphere Studio generated WSDL file uses the basic type xsd:string with the maxOccurs property set to unbounded; the related part of the WSDL file is shown in Example 12-12.

Example 12-12 Object array type specification in WebSphere Studio generated WSDL file

```
...  
    <element maxOccurs="unbounded" name="involvedCars" type="xsd:string"/>  
....
```

The Microsoft Visual Studio .Net 2003 generated WSDL file, instead, uses the complex type ArrayOfString as shown in Example 12-13 on page 307.

Example 12-13 Object array type specification in Microsoft Visual Studio .Net 2003 generated WSDL file

```
...
<s:element name="registerClaim">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="customerID"
        type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="policyID"
        type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="accidentDate"
        type="s:dateTime" />
      <s:element minOccurs="0" maxOccurs="1" name="accidentDescription"
        type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="involvedCars"
        type="s0:ArrayOfString" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:complexType name="ArrayOfString">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="string"
      nillable="true" type="s:string" />
  </s:sequence>
</s:complexType>
...
```

Starting from the Microsoft Visual Studio .Net 2003 generated WSDL file, the WebSphere Studio wizard generates an `ArrayOfString` class and all other related classes used to manage SOAP serialization and deserialization; this means that `ArrayOfString` is managed as a non-standard object.

The difference in the SOAP request is shown in the following two examples where Example 12-14 refers to the SOAP request to the WebSphere Web service while Example 12-15 on page 308 refers to the SOAP request to the Microsoft .Net Web service.

Example 12-14 SOAP request to the WebSphere Web service

```
<?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>
    <registerClaim xmlns="http://ejb.claim.examples.itso">
      <customerID>ABC123</customerID>
      <policyID>P00245</policyID>
    </registerClaim>
  </soapenv:Body>
</soapenv:Envelope>
```

```

    <accidentDate>2004-09-26T04:00:00.000Z</accidentDate>
    <accidentDescription>Car crash</accidentDescription>
    <involvedCars>NC-SH1</involvedCars>
    <involvedCars>SA-NUM2-00</involvedCars>
    <involvedCars>DH-CICS3</involvedCars>
  </registerClaim>
</soapenv:Body>
</soapenv:Envelope>

```

Example 12-15 SOAP request to the Microsoft .Net Web Service

```

<?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>
  <registerClaim xmlns="http://tempuri.org/">
    <customerID>AAAA</customerID>
    <policyID>BBBB</policyID>
    <accidentDate>2004-09-26T04:00:00.000Z</accidentDate>
    <accidentDescription>CCCC</accidentDescription>
    <involvedCars>
      <string>SH1</string>
      <string>NUM2</string>
      <string>CICS3</string>
    </involvedCars>
  </registerClaim>
</soapenv:Body>
</soapenv:Envelope>

```

According to section 4.3.3 of the WS-I BasicProfile 1.1, we found the following recommendation (see *R2112* in Table 8-5 on page 152).

- ▶ In a *description*, elements should not be named using the convention *ArrayOfXXX*
- ▶ The correct way to define arrays is to define a basic type with `maxOccurs=unbounded`

There is no specific unrespected **MUST** in the Microsoft .Net WSDL file and the WebSphere Studio wizard is able to generate the correct client; the interoperability is then guaranteed between the two platforms.

Parameter multiplicity specification

In both platform generated WSDL files, the methods' input parameters are considered optional: `minoccurs` is set to 0 in the Microsoft .Net WSDL file while `nillable` is set to `true` for the WebSphere WSDL file.

The difference in type declaration does not influence the proxy generation wizard in WebSphere Studio and Microsoft Visual Studio .Net 2003 and neither tool shows any problem in generating the Web service proxy starting from a WSDL file generated with a different platform.

We also tried a manual update of the WSDL file, forcing the value of `minoccurs` to 1 and the `nillable` to `false`. The aim was for a client to be able to raise an exception before invoking a service if null values were set for mandatory inputs. However, even if we regenerated the proxy, we were not able to obtain such a behavior; the proxy generation is not influenced by these new values and we were able to invoke the service even passing null values for mandatory inputs and receiving an exception raised from the service. The lesson is that WSDL definitions should not be taken as a guaranteed precondition of how a service behaves. The author of a Web service must check input arguments, even invalid values that are not allowed in the WSDL file.

Archived



Web service interoperability implementation guidance

This chapter provides some guidance when coding Web services consumer and provider classes in both Microsoft .Net and WebSphere Studio Application Developer. For a different perspective, also refer to:

<http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnbda/html/wsinteroprecsibm-final.asp>

13.1 Web service interoperability guidance

During the course of development using WebSphere Studio Application Developer 5.1.2 and Microsoft Visual Studio .Net 2003 (Microsoft .Net Framework 1.1), we encountered some errors because the development tools generate code assuming different naming conventions. Duplicate names for classes in Microsoft Visual Studio .Net 2003 can become a problem when the client proxy classes are generated in the WebSphere Studio Application Developer.

13.2 WebSphere client

There are a number of changes that may need to be made when using WSDL generated in Microsoft .Net to generate a Web services client using WebSphere Studio Application Developer.

Duplicate Web service name

The first case is the generation of duplicate Web service names for namespaces having same domain name in Microsoft .Net.

If we want to access more than one Microsoft .Net service within the same WebSphere Web service client, we want to make sure that we use a unique service name in Microsoft .Net or that the namespace does not have the same domain name.

WebSphere Studio Application Developer 5.1.2 maps the domain portion of the namespace into its package name, resulting in duplicate proxy classes for the two Microsoft .Net services, as shown in Figure 13-1.

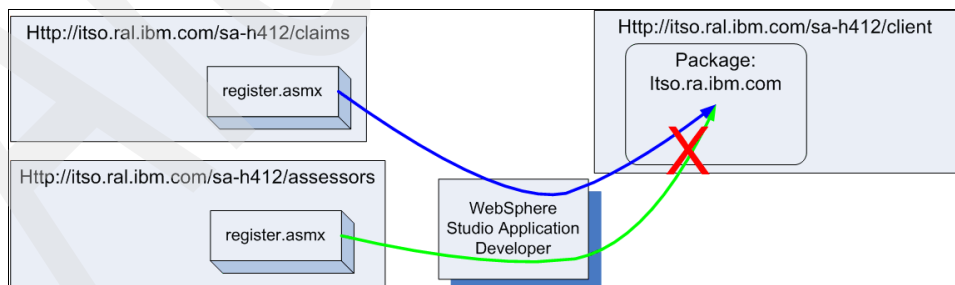


Figure 13-1 Clash of package names

For example, two Web services are named register.asmx, but are created in different namespaces such as http://itso.ral.ibm.com/sa-h412/claims and

<http://itso.ral.ibm.com/sa-h412/assessors>. Web Sphere Studio Application Developer will generate proxy classes of register.java in the package itso.ral.ibm.com, resulting in duplicate proxy classes. Only the first proxy class will be kept and the second one will be lost.

Incorrect reference of namespace for array types

When we create two Web services (two .asmx) in Microsoft .Net project, a common data type, such as the Customer class, is often shared between the services. However, when we use an array of the Customer class as a parameter, WebSphere Studio Application Developer puts the generated client proxy into the shared package but binds the class to the first namespace of the Web service for which the proxy client is generated. The proxy class generated for the second Web service in WebSphere Studio Application Developer is then incorrect.

We can either fix the problem in the proxy class by changing the namespace to point to the correct one or we can generate the proxy classes for the first Web service and then move the classes to a unique package by using the WebSphere Studio Application Developer refactor function. When we generate the proxy classes for the second Web service, we move the classes to another unique package. We then edit the proxy classes referencing the array of Customer to point to the right package.

dateTime comparison

A Microsoft .Net Web service having dateTime data type is deserialized into java.util.Calendar in WebSphere Studio Application Developer. Using == to compare this date to another date field that is defined in WebSphere Studio Application Developer will not get the correct result of true, even when both date values are the same. So, instead of using ==, we must use compareTo().

Array of class type as parameter

When consuming a Microsoft .Net Web service that requires an array argument, we have to use the array type generated in the proxy classes by WebSphere Studio Application Developer. If Microsoft .Net defines a method taking a Customer array parameter Customer[], we cannot simply call the method by passing Customer[] as argument. We have to instantiate the ArrayOfCustomer type as it is generated in the proxy classes by WebSphere Studio Application Developer. The correct coding is as shown in Example 13-1 on page 314.

```
Customer[] customerArray = createCustomers();
ArrayOfCustomer mCustomer = new ArrayOfCustomer();
mCustomer.setMessageType(customerArray);
service.getCustomers(mCustomer);
```

Return array with null value or empty array

When a WebSphere client receives an array with null values or an empty array from Microsoft .Net Web service, it will find a default message exists instead of a null value.

13.3 WebSphere Web service

The following hints concern the behavior of WebSphere and WebSphere Studio Application Developer, and do not specifically have to do with interoperability with Microsoft .Net. However, the conventions may not be familiar to Microsoft .Net programmers who are using WebSphere.

ClassCastException for two Web services with the same name

This issue concerns throwing `ClassCastException` for two Web services with the same name across packages.

In WebSphere Studio Application Developer, even though two Web services with the same name in different packages are accepted, they cause a `java.lang.ClassCastException` during deployment. So, it is advisable to either keep the Web service name unique across projects or only create one Web service per project.

Case sensitivity problem in getter methods

When generating Web service classes, WebSphere Studio Application Developer keeps to a convention of naming getter methods. Valid getter methods are `getCustomer()` and `getOrder()`, and invalid method names are `GetCustomer()` and `GetCustomer()`.

In addition, do not use an underscore for method attributes, such as `_customer_` or in method names, which would become, for example, the `get_customer_()` getter method, resulting in a `ClassNotFoundException`. WebSphere Studio Application Developer will generate and look for `getCustomer()` in its proxy classes.

Problem with Boolean getter method

When using a Boolean method attribute and generating Web services from the EJB beans, be aware that WebSphere Studio Application Developer does not generate `getCustomerExist()`, but instead generates `isCustomerExist()` in its proxy classes.

13.4 Microsoft .Net client

There are a number of changes that may need to be made when using a Microsoft .Net client to talk to a Web service generated in WebSphere Studio Application Developer.

Return null date

When a Web service developed with WebSphere Studio Application Developer returns a null date that is going to be used by a Web service client programmed with Microsoft Visual Studio .Net 2003, the null date will generate a `System.Format.Exception` whether or not the date field is within a class. In WebSphere Studio Application Developer, `java.util.Date` and `java.util.Calendar` are passed by reference and can have null as a value, but in Microsoft .Net, `System.DateTime` is considered to be a value type and should not include null.

Return null Array

When a Web service created using WebSphere Studio Application Developer returns a null array, Microsoft .Net treats the null as a null element, not as a null array.

Return array with null values or empty array

When Microsoft .Net receives an array with null values or an empty array from a Web service developed by WebSphere Studio Application Developer, it will find a default message exists instead of a null value.

13.5 Summary

This chapter provides some implementation guidance for coding Web service producers and consumers in both development environments. It is always best to code one Web service per project to avoid name collision. However, sometimes, there is a need to group more than one Web service class in one project. It is helpful to pay attention to this guide in naming classes, passing parameters and returning values of type array.

Archived



Part 4

Appendixes

Archiving

Archived

Installation and setup

This appendix provides instructions for installing the products to prepare the WebSphere environment and Microsoft .Net environment as used in this publication; these are:

- ▶ IBM WebSphere Application Server Version 5.1.0, Fixpack 5.1.1 and Cumulative Fix 5.1.1.1
- ▶ IBM WebSphere Studio Application Developer 5.1.2
- ▶ IBM Universal Database DB2 Version 8.1

Installation planning for the WebSphere environment

This section provides installation planning information for the products to prepare the WebSphere environment as used in this publication.

WebSphere Application Server V5.1.1.1 requirements

This is also known as WebSphere Application Server 5.1.1 (cumulative fix pack 1).

IBM WebSphere Application Server has the following hardware and software requirements. For updated information about the requirements, please refer to the WebSphere InfoCenter and documentation:

<http://www-3.ibm.com/software/webservers/appserv/infocenter.html>
<http://www-3.ibm.com/software/webservers/appserv/doc/latest/prereq.html>

Hardware

Hardware requirements for Windows servers include:

- ▶ Intel® Pentium® processor at 500 MHz, or faster
- ▶ Minimum 600 MB free disk space for installation of Version 5.1.1.1
- ▶ Minimum 256 MB memory; 512 MB or more recommended
- ▶ CD-ROM drive
- ▶ Support for a communications adapter

Software

The installation requires the following software to be installed for a Windows server:

- ▶ Windows NT® Server V4.0, SP 6a or later, Windows 2000 Server or Advanced Server SP 3, Windows Server 2003, Windows XP Pro SP 1
- ▶ Netscape Communicator 4.79 or Microsoft Internet Explorer 5.5 SP 2 or 6.0
- ▶ Web browser that supports HTML 4 and CSS

For detailed hardware and software requirements, go to:

http://www-306.ibm.com/software/webservers/appserv/doc/v51/prereqs/was_v511.htm

Database support

For the WebSphere installation, the database does not have to be configured. Cloudscape can be used in the test environment, but other databases are required for the production environment.

Note: This publication includes sample code that uses a database and a data source server configuration that is for DB2 UDB V8.1.

Installing WebSphere Application Server 5.1.1.1

In order to install WebSphere Application Server V5.1.1.1, first we install V5.1.1 and then install cumulative fix 5.1.1.1.

Installation process for the V5.1 base product

We start the LaunchPad (launchpad.bat) to access the product overview, the ReadMe file and installation guides.

Select **Install the product** to launch the installation wizard.¹

After confirming that you agree with the license agreement, choose between two installation choices: Full and Custom. Full installs the entire product, whereas the Custom installation option allows you to deselect components you do not want to install. We chose the **Full** installation.

The installation directories for the selected components are entered in the next window. We chose:

```
c:\WebSphere\AppServer  
c:\WebSphere\IBMHttpServer  
c:\WebSphere\WebSphere MQ
```

In the following panel, enter a node name and host name or IP address. In addition, you can choose to install both WebSphere Application Server and IBM HTTP Server as a service on Windows NT, 2000, 2003 and XP.

After the Summary window, the installation starts.

The FirstSteps window is started automatically at the end of the installation.

Verifying the installation

Installation verification can be started from the menu. In Windows 2000, click **Start → IBM → WebSphere Application Server v5.1 → First Steps**. Then select **Verify Installation**. You can also start with the command `ivc localhost`.

If the install was successful, you should see messages similar to the following:

```
IVTL0095I: defaulting to host <node> and port 9080  
IVTL0010I: Connecting to the WebSphere Application Server <node> on port: 9080
```

¹ If installing on Windows Server 2003, we found we needed to change the properties of the installation .exe files (setup and install) to be Windows 2000 compatible.

```
IVTL0020I: Could not connect to Application Server, waiting for server to start
IVTL0025I: Attempting to start the Application Server
IVTL0030I: Running cmd.exe /c "C:\WebSphere\AppServer\bin\startServer" server1
>ADMU0116I: Tool information is being logged in file
>
> C:\WebSphere\AppServer\logs\server1\startServer.log
>ADMU3100I: Reading configuration for server: server1
>ADMU3200I: Server launched. Waiting for initialization status.
>ADMU3000I: Server server1 open for e-business; process id is 3056
IVTL0050I: Servlet Engine Verification Status - Passed
IVTL0055I: JSP Verification Status - Passed
IVTL0060I: EJB Verification Status - Passed
IVTL0070I: IVT Verification Succeeded
IVTL0080I: Installation Verification is complete
```

Fixpack

After successful installation of version 5.1, we install fixpack 5.1.1 and then cumulative fix 5.1.1.1. This will upgrade WebSphere Application Server to version 5.1.1.1. It is available from:

<http://www-1.ibm.com/support/docview.wss?rs=180&context=SSEQTP&uid=swg24007753>

To install fixpack 5.1.1, we will follow the steps below:

1. Launch Update Wizard (updateWizard.bat).
2. Select a language.
3. First, a welcome window will appear and then the next window will display currently installed products as shown in Figure A-1 on page 323; click **Next**.

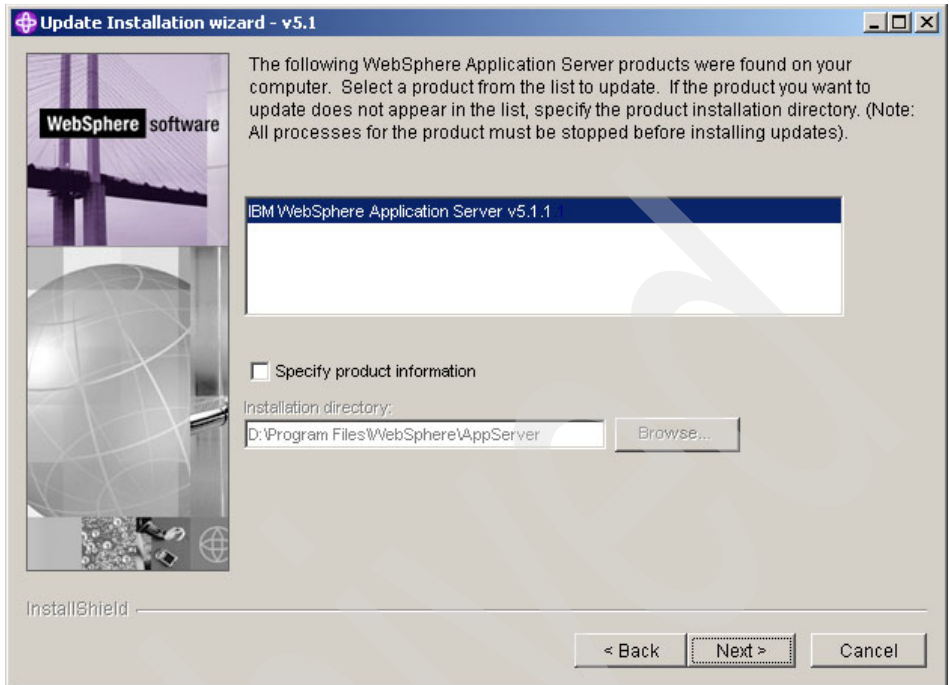


Figure A-1 updateWizard - currently installed product

4. Select **Install Fixpack**.
5. Select the directory where fix packs are located.
6. After scanning for installable fix packs, the next window will display the fix pack to install; click **Next**.
7. A successful message will confirm installation completion.

Installation of Application Developer 5.1.2

To install WebSphere Studio Application Developer 5.1.2, perform the following steps:

1. Double-click **setup.exe** and the Installation Launcher window appears.
2. Select **Install IBM WebSphere Studio Application Developer**.
3. In the Welcome window, click **Next**.
4. In the License Agreement window, accept the agreement, then click **Next**.

5. In the Destination Folder window, select a folder of your choice and click **Next**. We used the default folder as the installation folder:
c:\Program Files\IBM\WebSphere Studio\Application Developer\v5.1.2
6. In the Custom Setup window, accept the defaults, then click **Next**.
7. In the next window, click **Install**.
8. After a rather long time period, the next window tells you of the success of the installation. Click **Finish**.
9. The last window allows you to specify the location of your workspace. We use:
c:\Examples\Merger-n-Acquisition
for the workspace location for the first Mergers and Acquisitions scenario.

Fixpack

Install Interim Fix 004 using Install/Update Perspective, as follows:

1. Download the ZIP file (wsappdev512_interim_fix004.zip) from the following site:
ftp://www3.software.ibm.com/software/websphere/studiotools/zips/512/wsappdev512_interim_fix004.zip
2. Unzip it in the local file system.
3. Open the Install/Update Perspective in WebSphere Studio Application Developer.
4. Expand the Feature Updates pane as shown in Figure A-2, then click **Install Now**.

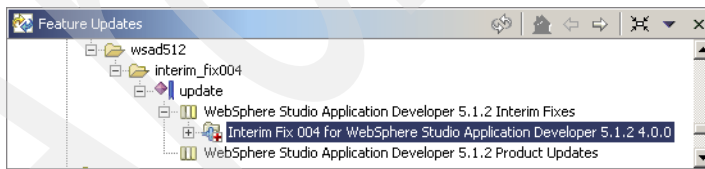


Figure A-2 Feature updates - Interim fixpack 004

5. Follow Install wizard to complete the installation.

Installing optional components

The optional components are not required to follow the samples in this publication.

You can install these components:

- ▶ IBM Agent Controller: if you want to test or debug applications running in a real WebSphere Application Server on the same or another machine.
- ▶ Embedded messaging client and server: if you want to develop applications using WebSphere MQ (the message driven bean we use for Web services can be tested using the built-in MQ Simulator).
- ▶ Rational ClearCase® LT: for team development as an alternative to Common Versions Systems (CVS).

Starting Application Developer with a dedicated workspace

You can create icons to start Application Developer with multiple workspaces. In the Properties window of the icon, enter the target with the `-data` flag to indicate the workspace location, for example:

```
C:\<WSAD-HOME>\wsappdev.exe -data C:\Examples\Merger-n-Acquisition
```

Installation planning for the Microsoft .Net environment

- ▶ Windows 2003 Server
Install Windows 2003 Server on the computer.
- ▶ Microsoft Visual Studio .Net 2003
Install the prerequisite disk and visual studio disk and perform the update.
- ▶ Microsoft .Net Framework 1.1
This comes with Microsoft Visual Studio .Net 2003 or you can download it from the Microsoft Web site.
- ▶ IIS 6.0
After installing Windows 2003 Server, we need to separately install the IIS 6.0. Go to **Settings** → **Control Panel** → **Add/Remove Programs** and select **Add/Remove Window Components**. Select **Application Server** and click **Next**. Click **Finish** to complete the process.

Archived

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246395>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds to the redbook form number, SG24-6395.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
SG246395.zip	Zipped Code Samples
sa-h412 readme.htm	This document describes the contents of the zip file

System requirements for downloading and running the Web material

The following system configuration is recommended:

- ▶ Minimum 1GHz Intel Pentium or equivalent
- ▶ Minimum 1 GB RAM
- ▶ 40 GB Disk
- ▶ WebSphere Studio Application Developer 5.1
- ▶ WebSphere Application Server 5.1.1.1 (optional)
- ▶ Windows 2000 with upgrades or better, capable of running:
 - Microsoft Visual Studio .Net 2003
 - Microsoft .Net Framework 1.1
 - IIS 6.0

Note: The Microsoft .Net Web service Extensions are not required for this version of the redbook. They will be required for the security samples.

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Instructions for installing the system software are in Appendix A, “Installation and setup” on page 319, and instructions for using the material in Chapter 12, “Building the claims scenario” on page 251.

In addition to the source materials and .ear files needed for the samples, for those familiar with using .Net there are also binaries for .Net which can be deployed directly rather than building with Microsoft Visual Studio .Net 2003.

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 333. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook*, SG24-6891
- ▶ *Patterns: Service Oriented Architecture and Web Services*, SG24-6303
- ▶ *Using Web Services for Business Integration*, SG24-6583
- ▶ *WebSphere Web Services Information Roadmap*, REDP-3854-00
- ▶ *WebSphere Version 5 Application Development Handbook*, SG24-6993-00
- ▶ *WebSphere and Microsoft .Net Coexistence*, SG24-7027
- ▶ *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346

Online resources

These Web sites and URLs are also relevant as further information sources:

developerWorks and other IBM articles

- ▶ *Merging disparate IT systems: Build a single integrated view for users quickly and with minimal disruption*, IBM developerWorks, found at:
<http://www-106.ibm.com/developerworks/ibm/library/i-merge.html>
- ▶ *Specifications and Standards*, IBM developerWorks, found at:
<http://www-106.ibm.com/developerworks/views/webservices/standards.jsp>
- ▶ *Discover SOAP encoding's impact on Web service performance*, by Frank Cohen, found at:
<http://www-128.ibm.com/developerworks/webservices/library/ws-soapenc/>

- ▶ *IBM Patterns for e-business*, found at:
<http://www-106.ibm.com/developerworks/patterns/>
- ▶ *The hidden impact of WS-Addressing on SOAP*, by Doug Davis, IBM developerWorks, found at:
<http://www-106.ibm.com/developerworks/webservices/library/ws-address.html>
- ▶ *Security in a Web Services World: A Proposed Architecture and Roadmap*, IBM and Microsoft 2002, found at:
<http://www-128.ibm.com/developerworks/webservices/library/ws-secmap/>
- ▶ *An overview of the Web Services Inspection Language*, by Peter Brittenham, found at:
<http://www-106.ibm.com/developerworks/webservices/library/ws-wslover1/>
- ▶ *Enterprise Privacy Authorization Language*, IBM Zurich Labs, found at:
<http://www.zurich.ibm.com/security/enterprise-privacy/epal/>
- ▶ *Declarative Privacy Monitoring for Tivoli Privacy Manager*, IBM alphaWorks, found at:
<http://www.alphaworks.ibm.com/tech/dpm>
- ▶ *Web Services Atomic Transaction for WebSphere Application Server*, IBM alphaWorks, October 2003, found at:
<http://www.alphaworks.ibm.com/tech/wsat>
- ▶ *WebSphere MQSeries SOAP Supportpac*, found at:
<http://www-3.ibm.com/software/integration/support/supportpacs/individual/major.html>
- ▶ *Invoking Web services with Java clients* by Bertrand Portier, found at:
<http://www-106.ibm.com/developerworks/webservices/library/ws-javaclient>

MSDN articles

- ▶ *Web services specifications*, Microsoft MSDN, found at:
<http://msdn.microsoft.com/webservices/understanding/specs/default.aspx>
- ▶ *Application Interoperability: Microsoft .Net and J2EE*, found at:
http://download.microsoft.com/download/7/2/6/7269f183-639a-4e99-bd84-cc3e6515af86/PnP_J2EE_Interop_V1.pdf
- ▶ *Understanding Web services*, Microsoft MSDN, found at:
<http://msdn.microsoft.com/webservices/understanding/default.aspx>
- ▶ *How ASP.NET Web Services Work*, Aaron Skinnard, found at:
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/howwebmeth.asp>

- ▶ *Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation*, MSDN Magazine, December 2002, found at:
<http://msdn.microsoft.com/msdnmag/issues/02/12/DIME/default.aspx>
- ▶ *Web Services Interoperability Guidance (WSIG): IBM WebSphere Application Developer 5.1.2*, MSDN, found at:
<http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnnda/html/wsinteroprecsibm-final.asp>
- ▶ *Building Interoperable Web services, WS-I Basic Profile 1.0, V1.0*, Microsoft 2003, found at:
<http://www.microsoft.com/downloads/details.aspx?FamilyId=60080CA9-2466-43E4-A19C-8A9DE724ABA8&displaylang=en>
- ▶ *WS-Security Drilldown in Web services Enhancements 2.0* by Don Smith, found at:
<http://msdn.microsoft.com/webservices/building/wse/default.aspx?pull=/library/en-us/dnwse/html/wssecdrill.asp>
- ▶ *What Is Managed Code?*, MSDN, found at:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_m/directx/whatismanagedcode.asp
- ▶ *Improving Web Application Security, Threats and Countermeasures*, Microsoft, found at:
<http://www.microsoft.com/downloads/details.aspx?FamilyId=E9C4BFAA-AF88-4AA5-88D4-0DEA898C31B9&displaylang=en>
- ▶ *Application Architecture for .NET: Designing Applications and Services*, Microsoft, found at:
<http://www.microsoft.com/downloads/details.aspx?FamilyId=A08E4A09-7AE3-4942-B466-CC778A3BAB34&displaylang=en>
- ▶ *Securing ASP.NET Web Services*, Microsoft, found at:
<http://www.microsoft.com/technet/itsolutions/net/maintain/secnetws.mspx>
- ▶ *The argument against SOAP encoding*, Tim Ewald, MSDN Oct 2002, found at:
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap/html/argsoape.asp>

Standards bodies

- ▶ For a list of all WS-* specifications, refer to Table 7-2 on page 116.

WS-I

- ▶ WS-I Basic Profile base specifications are in Table 8-1 on page 147.
- ▶ *WS-I Basic Profile Version 1.0*, found at:
<http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html>
- ▶ *WS-I Basic Profile 1.1*, found at:
<http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html#references>

W3C

- ▶ *Simple Object Access Protocol (SOAP) 1.1*, W3C note 8 May 2000, found at:
<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- ▶ WSDL 1.1 is available from W3C at:
[-http://www.w3.org/TR/2001/NOTE-wsd1-20010315](http://www.w3.org/TR/2001/NOTE-wsd1-20010315)
- ▶ *Web services Architecture*, W3C 2004, found at:
<http://www.w3c.org/TR/ws-arch/>

OAGIS

- ▶ *eXtensible Access Control Markup Language (XACML)*, OAGIS, found at:
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
- ▶ *Business Transaction Protocol 1.0*, OAGIS 2002, found at:
http://www.oasis-open.org/committees/download.php/1184/2002-06-03.BTP_cttee_spec_1.0.pdf

OASIS

- ▶ OASIS - Web Services pages, found at:
http://www.oasis-open.org/committees/tc_cat.php?cat=ws
- ▶ *ASOAP Message Security V1.0 (WS-Security 2004)*, OASIS, found at:
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>
- ▶ *Username Token Profile V1.0*, OASIS, found at:
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf>
- ▶ *X.509 Token Profile V1.0*, OASIS, found at:
<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf>

Others

- ▶ *Predicts 2003: SOA is Changing Software*, Roy Schulte (Gartner, Inc.), found at:
<http://www.gartner.com/resources/111900/111987/111987.pdf>
- ▶ *Identifying best-of-breed characteristics in Enterprise Services Buses (ESBs)*, Steve Craggs, June 2003, found at:
http://www.sonicsoftware.com/products/whitepapers/docs/best_of_breed_esbs.pdf
- ▶ *Hype Cycle for Web Services, 2003* and *Hype Cycle for Web Services, 2004*, W.Andrews, D. Smith, C. Abrams, R. Wagner, R. Valdes, C. Haight, M. Govekar, Gartner Strategic Analysis Report

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

- ▶ IBM Support and downloads
ibm.com/support
- ▶ IBM Global Services
ibm.com/services

Archived

Abbreviations and acronyms

FORTRN

.Net	“dot Net” Web service platform on Windows (Microsoft)	DMZ	Demilitarized Zone
ADO	Active Data Object: COM object used to access database (Microsoft)	DSA	Digital Signature Algorithm
AES	Advanced Encryption Standard	EAI	Enterprise Application Integrations
AKAMAI	Company that hosts a about 15% of all Internet traffic	EDI	Electronic Data Interchange
B2B	Business to Business	EIS	Enterprise Information System
B2C	Business to Consumer	EJB	Enterprise JavaBean
BEA	Software company. Named after Bill Coleman, Ed Scott, and Alfred Chuang	EPAL	Enterprise Privacy Automation Language
BMP	Bean Managed Persistence	EPAL	Enterprise Privacy Authorization Language
BPEL	Business Process Execution Language	EPR	Endpoint Reference
BPEL4WS	Business Process Execution Language for Web service	ESB	Enterprise Service Bus
BPM	Business Process Management	ETTK	Emerging Technologies Toolkit (IBM)
CLR	Common Language Runtime (Microsoft)	FORTRN	Formula Translation (Programming language)
CMP	Container Managed Persistence	GRID	Technologies for sharing remote computer resources
COM+	Current version of the Component Object Model (Microsoft)	Http	Hypertext transfer protocol
CORBA	Common Object Request Broker Architecture	Https	Secure Http
DB/2	Database (IBM)	IBM	International Business Machines Corporation
DCOM	Distributed Common Object Model	IDE	Integrated Development Environment
DII	Dynamic Invocation Interface	IDL	Interface Definition Language
DIME	Direct Internet Messaging Encapsulation (Microsoft)	IETF	Internet Engineering Taskforce
		IIOP	Internet Inter Operable Protocol or Internet Inter-ORB protocol (CORBA)
		IIS	Internet Information Server ((Microsoft)

IMS	Information Management System (IBM)	MSFT	Microsoft (Share listing name)
INETMGR	Program that runs IIS ((Microsoft)	MSIL	Microsoft Intermediate Language
IONA	Irish Software company originally associated with CORBA	MTOM	Message Transmission Optimization Mechanism
IP	Internet Protocol	MUWS	Management using Web Services
ISO	International standards Organization	NIST	National Institute for Standards and Technology (US)
IT	Information Technology	NONCE	Number that can only be used ONCE
ITSO	International Technical Support Organization	OAGIS	Open Applications Group Interface Specification
J#	J-Sharp (Java on Windows)	OASIS	Organization for the Advancement of Structured Information Standards
J2EE	Java 2 Enterprise Edition	ODBC	Open Database Connectivity
J2SE	Java 2 Standard Edition	P4eb	Patterns for e-Business (IBM)
JAAS	Java Authentication and Authorization Service	PKCS7	Type of X.509 security certification
JAXP	Java for XML Parsing	RFC	Request for Comment
JAX-RPC	Java for XML Remote Procedure Call	RM/IIOP	Remote Method Invocation over Inter Operable Object Protocol
JCA	Java Connector Architecture	RPC	Remote Procedure Call
JCP	Java Community Process	RPSS	Reverse Proxy Security Server
JDBC	Java Database Connectivity	RSA	Software security company founded by Ron Rivest, Adi Shamir and Len Adleman to commercialize their discovery of an asymmetric encryption algorithm
JNDI	Java Naming and Directory Interface	RUP	Rational Unified Process
JScript	Java Script	S/390	System 390 (IBM)
JSP	Java Servlet Page	SAP	Systeme, Anwendungen, Produkte in der Datenverarbeitung -Software company
JSR	Java Specification Request		
JTA	Java Transaction Architecture		
JVM	Java Virtual Machine		
MDB	Message Driven Bean		
MIME	Multipurpose Internet Mail Extensions		
MMC	Microsoft Management Console		
MSDE	Microsoft SQL Server Desktop Engine		
MSDN	Microsoft Developer Network		

SEI	System Endpoint Interface	WSIL	Web service Inspection Language
SHA-1	Secure Hashing Algorithm - 1	WS-RM	Web services reliable messaging
SOA	Service-Oriented Architecture	WSRP	Web service for remote portals
SOAP	Simple Object Access Protocol (now simply SOAP)	WS-TXM	Web services transaction Management
SOAPACTION	SOAP parameter naming service to be called (Microsoft)	WSXL	Web services experience language
SPML	Service Provisioning Markup Language	WX-CF	Web service coordination framework
SQL	Structured Query Language	X.509	Standard for Public-Key Infrastructure
SSL	Secure Sockets Layer	XACML	Extensible Access Control Markup Language
SwA	SOAP with Attachments	XDE	Extended Development Environment (IBM)
TCP/IP	Transport Control Protocol/Internet Protocol	XMI	XML metadata interchange
TIBCO	The Information Bus Company	XML	Extensible Markup Language
TLS	Transport Layer Security	XOP	XML Binary Optimized Package
UDDI	Universal Data Definition Interface	XSD	XML Schema Definition
UERL	Universal Resource Locator		
UML	Unified Modelling Language		
URI	Universal Resource Identifier		
UTP-16	A universal 2 byte character encoding scheme		
UTP-8	A universal mixed one and two byte character encoding scheme		
W3C	World Wide Web Consortium		
WS-	Web service		
WS-CAF	Web service Composite Application Framework		
WS-CTX	Context		
WSDL	Web services definition language		
wsdl2java	Web services to Java (converts WSDL to Java object)		
WS-I	Web Services Interoperability Organization		

Archived

Index

Symbols

.asmx 22–23, 224
.ear 207
.Net Remoting 193
.war 207

A

Access integration pattern 99
Active Data Objects 192
Active Directory 223
Active Server Pages 192
adapter server 49
adapters xiii, 13, 101–102, 109, 111, 196
 bridges 46
administrators 5–6
ADO 192, 194, 196
 ADO.NET 194
AES
 Advanced Encryption Standard 175
affinity 64, 126, 187–188
Apache
 Axis 149
 Struts 207
 Web Server 192
Application Architecture for .NET
 Designing Applications and Services 331
Application Interoperability
 Microsoft .Net and J2E 330
application tier 96
architects 5
Articles
 An overview of the Web Services Inspection Language 136, 330
 Application Architecture for .NET
 Designing Applications and Services 226
 Application Architecture for .NET - Designing Applications and Services 226
 Building Interoperable Web services, WS-I Basic Profile 1.0, V1.0 331
 Declarative Privacy Monitoring for Tivoli Privacy Manager 130, 330
 Discover SOAP encoding's impact on Web service performance 20
 Discover SOAP encoding's impact on Web service performance 329
 Enterprise Privacy Authorization Language 330
 How ASP.NET Web Services Work 22
 Hype Cycle for Web Services, 2003 & Hype Cycle for Web Services, 2004 333
 Identifying best-of-breed characteristics in Enterprise Services Buses (ESBs) 333
 Improving Web Application Security, Threats and Countermeasures 224
 Invoking Web services with Java clients 330
 Merging disparate IT systems
 Build a single integrated view for users quickly and with minimal disruption 3, 329
Patterns
 Implementing an SOA Using an Enterprise Service Bus, SG24-6346 329
 Predicts 2003 - SOA is Changing Software 57, 333
 Securing ASP.NET Web Services 226
 Security in a Web Services World
 A Proposed Architecture and Roadmap 330
 Security in a Web Services World - A Proposed Architecture and Roadmap 129
 Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation 128, 331
 The argument against SOAP encoding 36, 331
 The hidden impact of WS-Addressing on SOAP 126, 330
 Using Web Services for Business Integration, SG24-6583 329
 Web Services Atomic Transaction for WebSphere Application Server 330
 Web Services-Specifications and Standards 329
 WebSphere and Microsoft .Net Coexistence, SG24-7027 329
 WebSphere Version 5 Application Development Handbook, SG24-6993-00 231, 329
 WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook, SG24-6891 329
 WebSphere Web Services Information Roadmap, REDP-3854-00 329

- What Is Managed Code? 331
- asynchronous
 - messaging 20
 - model 126
 - response 126, 157–158
- Atomic Transaction 179
- Attachments
 - base64Binary 123, 127–128, 246
 - binary attachment data 127
 - DIME 127–128, 171
 - Direct Internet Messaging Encapsulation 127
 - MIME 127
 - MTOM 128
 - SwA 127
- Authors
 - C. Abrams 115
 - C. Haight 115
 - D. Smith 115
 - Jeannine Hall Gailey 128
 - M. Govekar 115
 - Peter Brittenham 136
 - R. Valdes 115
 - R. Wagner 115
 - Tarak Modi 137
 - W.Andrews 115

B

- B2B 98, 139, 216–217
- B2C 216
- Batch systems 207
- BEA 116–117, 119–120, 122, 133, 177, 182
- Best practices 146
 - guidelines 94
- BPEL 64, 76, 116
 - BPEL4WS 53, 116, 138, 204
- BPM 79, 85, 88, 240, 242
- broadcast 109, 124
- broker 40–41, 47, 75, 109–110, 130
- browser 12
- building blocks 54, 95, 177
- bus topologies 56
- Business Layer 195, 207, 225
- business needs 137
- business objects 195
- Business Process 40, 53, 64, 72, 104, 138
- business processes xi, 2–3, 53, 72–75, 78, 98–99, 134, 177
- business requirements 5, 37, 51, 95, 107–108

- Business Transaction Protocol 1.0 132, 332

C

- C# 193
- call center 106
- Callback 125
- central point of control 56
- CICS 2, 41, 47, 61, 70–71, 103, 134, 207
- claims database 75
- class library 196
- ClassCastException 314
- client application 199, 206
- Cloudscape 220, 320
- CLR 124, 193, 195, 245
- IL
 - Intermediate Language 195
 - intermediate language 193, 195
 - Managed C++ 193
- clusters of servers 64
- coarse-grained 42
 - services 20
- code behind 197, 290
- COM+ 192
- Common Language Runtime 124, 193, 195, 245
- communication protocols 55
- complex business process interactions 40
- Computer Associates 117–118, 120
- connectors 46–47, 50, 83, 96, 207
- consultant xiv, 5
- container 46
- Coordination
 - CoordinationContext 178
- coordination 45, 117, 133–134, 139, 177, 179–180, 222
- CORBA 19
- Corporations
 - AKAMAI Technologies 117–118
 - ArjunaTechnologies 117
 - DevelopMentor 12
 - Fujitsu 117–118, 133, 137
 - Globus 117–119, 137
 - Hewlett-Packard 117–118
 - IONA 117, 133
 - JDEdwards 207
 - Layer 7 Technologies 119–120
 - Microsoft 12, 22
 - Oblix 119–120
 - Oracle 117, 133

- PeopleSoft 50
- SalCentral 137
- SAP 47, 50, 61, 117–118, 202, 207
- Sonic Software 117–118
- Sun 117–118, 133
- Talking Blocks 118, 140
- TIBCO 117–118, 182
- Tivoli 130
- VeriSign 117, 119–120, 161
- webMethods 118

Coupling 19, 46, 60, 90–91, 99, 186, 221

D

- data integrity 88
- Data Layer 196, 207
- data types 28, 43, 123, 153, 208, 229, 246–247, 250
 - array with null values 314–315
 - complex type 20
 - dateTime 35, 123, 239, 245–246, 277–278, 280, 285, 291, 307, 313, 315
 - empty array 314–315
 - hexBinary 123, 127, 247
 - null array 315
- databases 97, 192, 196, 207, 236, 320
- DB/2 220
- DCOM 12, 19
- Deployment
 - descriptor 262, 268–269
 - manager 220
 - models 215
 - Web services 273
- design model 236, 245
- developers, Roles, See also Roles 141
- developerWorks xiv, 2, 116–120, 126, 129, 136, 182, 186
- Digital Signature Algorithm
 - DSA 164
- DII
 - Dynamic Invocation Interface 210
- DIME
 - Direct Internet Messaging Encapsulation 127–128, 171
- Direct Internet Messaging Encapsulation 127
- distributed application 12, 42, 207
- distributed components 24
- Distributed Internet Applications 192
- divestment 72

- DNA 192–193
- domain name 312
- dot.com 70–71, 103, 135, 252
- DSA
 - Digital Signature Algorithm 164
- duplicate IT capabilities 71
- Duplicate names 312
- duplicate proxy classes 312
- dynamic clients 60
- dynamic invocation 89, 210
- Dynamic Invocation Interface 210
- Dynamic proxy 210

E

- EAI xiii, 78, 88
- Ease of use 46
- e-business xii–xiv, 50, 66–67, 71, 93–96, 100, 102, 112, 198, 322
 - applications 94
 - solutions 94
- EDI 19, 75–76
- Eiffel 193
- EIS 47–49, 62, 202, 236
- EJB
 - bean managed 202
 - BMP 202, 242
 - CMP 202
 - container 201–202, 205, 209–210, 212
 - entity beans 202, 207
 - session bean 253, 255–256
 - stateless session beans 202
- e-mail 19, 74, 85
- Emerging Technologies Toolkit 134, 155
- Enterprise application integration 50, 78
- Enterprise Application Resource 207
- Enterprise Information System 47, 202, 207, 236
- Enterprise Service Bus xi, 39, 41, 56, 58–66, 88, 90, 96, 109, 125
- EPAL 1.1 130
- ESB xi
- ETTK
 - Emerging Technologies Toolkit 134, 155
- Event-driven Architecture 90
- exception management 192, 306
- eXtensible Access Control Markup Language (XAC-ML) 332
- External Claims Assessors Scenario 229, 240
 - assessment report 84, 88–89, 240, 242, 245

assessor automation system 84
Assessor BPM 242
Assessor Business Process Management 242, 244
Assessor Management Business Rules 242, 244
Assessor Management System 87, 92, 242, 244
claims assessors 4, 70, 74, 80, 83, 86, 88, 90, 103, 189, 229, 240
claims handler 75–76, 78, 87–88, 107, 240–242
claims process 6, 70, 73–74, 83–84, 86, 107
claims supervisor 75–76
cost reduction 71, 76
Document Management System 88, 242, 245
External Assessor System 241–242, 245
External Claims Assessor 4, 70, 83–85, 88, 98, 103, 251
external independent assessor 241
external service provider 86
Identify assessors 87
Investigate claim 76, 78
Receive Assessment report 88
Select assessor 87
extranets 40

F

financial services sector 71
FORTRAN 193

G

garbage collection 192–193, 195
Gartner 45, 115
gateway service 219
Globalization 50
GRID 137

H

high availability 20
high-level design 69
horizontal integration 98
hot failover 64
How ASP.NET Web Services Work 330
http
 POST 21–23
Https 121, 156, 159, 171, 219, 221, 223
hub-and-spoke 56, 75

Hursley xiii, 2

I

IBM Agent Controller 325
IBM alphaWorks 134, 155, 180
IBM Patterns for e-business 93, 330
IBM System House Business Scenarios 2, 70
IBM UDDI registry 220
IDE 48
IDL 36
IETF 116
IIOF 12, 45, 55, 100, 182, 195, 202
IIS 6.0 111, 223, 325, 328
Improving Web Application Security, Threats and Countermeasures 331
IMS 207
INETMGR 284
In-order-delivery 183
Integrated Development Environments 48
integrated software platform 70
integration gaps 70
Integration Layer 77–78, 80, 207, 218, 236–238
Intentia 50
Internet Information Services 192, 283–284, 292
Internet scenario 229
Internet Service (IIS) Manager 284
intranet 24, 40, 74, 80, 90, 98, 110, 216, 218, 224, 229
IP address 224, 321
ISO 24
 layer 7 24

IT

infrastructures 12, 69–70, 72, 74
policies 71, 111
solutions 70–71
systems 3, 73

J

Java

Authentication and Authorization Service 201
Beans 79, 202, 252
Connector Architecture 47, 202
Database Connectivity 202
for XML Parsing 201
J# 193
J2EE xiii–xiv, 3, 51, 63, 76, 134, 181, 191, 204, 206, 208, 219–220, 222, 236, 268
J2EE and .Net

- comparison 191
- J2SE 123, 208–209, 212, 252
- JAAS 201
- Java 2 Enterprise Edition xii, 3–4, 46–48, 191, 195, 200–203, 206–207, 209–210, 212–213, 252, 293
 - application client container 201
 - programming model 201, 204–205
- java.lang.ClassCastException 314
- java2wsdl 149
- Java-RMI 12
- JAXP 202
- JAX-RPC 204–205, 210, 218–219, 222
- JCA 47–48, 202
- JCP 116
- JDBC 196, 202
- JNDI 202, 273
- JScript 193–194
- JSP 192, 202, 207, 266–267, 269, 297, 322
- JSR 101 116, 122, 124, 204–206, 210, 269
- JSR 109 205, 210, 268
- JSR109 116
- JTA 63, 180–182
- JVM 193, 195
- Naming and Directory Interface 202
- Remote Method Invocation 202
- Server Page 192
- servlets 202
- Transaction API 63
- JMS 189
- Judge claim 76, 78

L

- legacy system 40
- level of isolation 60
- load balancing 64, 125, 186–188, 226
- location transparency 51, 55
- locator class 296
- logging 63, 65, 218
- loosely coupled 19, 45, 51, 55, 80, 95, 192

M

- managed environment 47, 55
- Management using Web Services 140
- Management using Web services 140
- MDB 202
- Mergers and Acquisitions Scenario xi, 2–3, 70–73, 76, 78, 80–81, 91, 102, 104, 111–112, 227, 229–231, 251, 276, 324–325
 - administration costs 72, 83
 - auto-insurance 71
 - back-end system 73, 75, 80, 233, 235
 - business case 251
 - business events 70
 - Business goals 69–71, 83, 92
 - Business Layer 236, 245, 282
 - business needs 70
 - business problems 70
 - Business Process 78–79, 83, 85–86, 91, 242, 244
 - business requirements 69–70, 88
 - business vision 77
 - call center 75
 - claim agent 75
 - Claim Applications
 - Table Schema 245
 - claim number 75–76
 - claim references 78
 - claim registration 76, 230, 232–233, 235
 - ClaimDataAccess 278–280
 - ClaimDataAccessObject 238
 - ClaimException 239–240, 278, 282, 303–305
 - ClaimProcess application 230, 236
 - claims administration 72–73
 - claims applications -- Table Schema 229
 - claims assessors 3
 - claims database 76
 - claims information 75
 - claims registration scenario 104–106, 110
 - claims scenario 227
 - claims system 73–75, 77, 81, 84, 88
 - ClaimWebService 238–239
 - client application 75, 241, 302
 - CustomerDataAccessObject 238
 - Data Layer 245, 282
 - DataAccessException 238
 - DCClaimSystem 235, 237
 - Deployment descriptor 262, 268–269
 - disparate IT system 76
 - Enterprise application ItsClaim 258
 - findCustomer 23, 32, 238–239, 248–249, 260–261, 265, 270, 275, 277–278, 291, 297, 302, 304–305
 - LGIClaimSystem 235, 237
 - Lord General Insurance 71, 102, 155, 252
 - policy selector application 105, 107, 109–110
 - Register claim xii, 75, 77, 80, 230–235,

- 237–238, 245, 251–252, 276
- registerClaim 31–33, 35, 238–239, 247–248, 260, 265–266, 270, 277–278, 291, 304–308
- Message confidentiality 159
- Message Driven Beans 202
- message signatures 20
- Message Transmission Optimization Mechanism 128
- Message-Driven beans 202, 207
- Messages 28, 36
- Messaging
 - At-least-once delivery 183
 - At-most-once delivery 183
 - Exactly-once delivery 183
 - message hub 75
 - Message Information Headers 126
 - Message integrity 159
 - message level encryption 159–160
 - Message-driven architecture 90
 - One-way 21–22, 24, 124, 202
 - reliable messaging 132–134, 182–183, 186–187
 - request-response 21, 97, 124, 137, 155
- meta-data 35, 37
- Microsoft
 - .Net Framework 222
 - .Net Web service deployment model 222
 - ASP.NET 222
 - Intermediate Language 193
 - Management Console 223
 - MMC 223
 - MSDE 223
 - MSDN 36, 120, 122, 128, 195, 306
 - MSFT 116–117, 119–120
 - MSIL 193
 - patterns and practices 226
 - Server 2003 xii, 3
 - SQL Server 196, 223
 - SQL Server 2000 Desktop Engine 223
 - UDDI registry 222
 - Visual Studio .Net 2003 xi, 27, 111, 149, 194, 196–199, 213, 227, 237, 245, 276, 283, 285, 290, 292, 303, 306–307, 309, 312, 315, 325, 328
 - Web Service Enhancements 200–201, 212
- middleware 58, 90, 94, 96, 100, 207
- MIME 127
- MMC 223
- Model-View-Controller pattern 202, 207
- monitor and manage 72, 83

- MSDN Magazine 331
- MSIL 195
- Multi-hop 125
- Multipart message 127
- MUWS 140

N

- namespace 15–16, 22, 28, 30, 32, 123, 151, 153–154, 192, 199, 212, 278, 280–282, 290, 312–313
- naming conventions 136, 312
- National Institute for Standards and Technology 163
- network protocol 14, 42
- network transports 14
- new technology 74, 99
- nibblized 128
- NIST 163
- NONCE 175
- non-interruptible processes 79
- non-repudiation 88

O

- OASIS 116, 118–119, 131–133, 139–140, 143, 161, 177, 183, 211–212, 216
- ODBC 196
- off-the-shelf 75
- on demand xiv, 43, 80
- open standards 76, 85
- OpenNetwork Technologies 119–120
- open-source 44
- Oracle xiv
- Organization for the Advancement of Structured Information Standards 161
- out-of-band 147
- outsourcing 2, 70

P

- P4eb
 - Patterns for e-business xii, 66, 93–96, 100, 102, 112, 198
- package name 312
- Pascal 193
- Patterns
 - Application 96, 99, 107
 - Application Integration 100, 107, 109
 - Application integration 104–105

- Application interoperability 14
- Broker 109
- Business 94, 99, 103
- Collaboration 99
- Composite 94
- Direct Connection 98, 100–102, 107, 109–110
- Direct connection single adapter 100–101
- Extended Enterprise 98–99, 109
- Governance 47, 56, 63
- half adapter 101
- hub-and-spoke 107
- Integration 94, 99, 103
- P4eb
 - Patterns for e-business 198
- Product mappings 94, 96, 104, 111
- Runtime patterns 94–96, 99, 103, 108, 110–111
- Self-Service 96–99, 104
- Service-Oriented Architecture and Web Services, SG24-6303-00 329
- patterns
 - Information Aggregation 99
- patterns selection 251
- Perl 193
- physical format 36
- Ping Identity Corp 119–120
- PKCS7 176
- point-to-point architecture 56
- policy administration 72–74
- Portal 139
- portTypes 36
- presentation layer 24, 81, 194–195, 206–207, 222, 236
- presentation tier 96
- Problem determination 60, 65, 77
- Process Choreographer 79
- processing nodes 62
- producer 89, 114, 127, 140
- production applications 14
- profitability 71–72, 83
- programmers 5–6
- programming language 13, 24, 36, 43–44, 180
- property and casualty insurance company 70–71
- Protocol independent 51, 80
- publish/subscribe 109, 124
- Python 193

Q

- quality of service 24, 41, 46, 51–53, 55, 61, 88, 91, 96, 102, 108, 111, 134, 188, 192, 194

R

- Rational Unified Process 5
- Rational XDE 230–231, 237, 242
- Reactivity 119–120
- Redbooks Web site 333
- redeployment 60
- redundant network paths 64
- reliable messaging xii
- Remote Method Invocation 195, 202
- Remote portlets 139
- reprogramming 60
- Request assessment 88
- Request Availability 87
- Resource/Data Layer 207
- reusable business function 55
- reverse proxy 63, 108, 217–218
- RFC2246
 - The TLS Protocol Version 1.0 148
- RFC2459
 - Internet X 509 Public Key infrastructure Certificate and CRL Profile 148
- RFC2616
 - Hypertext Transfer Protocol - HTTP/1.1 147
- RFC2818
 - HTTP Over TLS 148
- RFC2965 148
- RMI/IIOP 195, 202
- Robustness 46, 60, 63
- Roles 5, 15, 70, 273
 - Actors 167–168, 230–232, 240–241, 250
 - administrators 224
 - analysts 70
 - application designer 16
 - architect 92
 - architects 44, 94, 125, 127, 183
 - component designer 197, 278
 - consultant 70, 92
 - developers 36, 44, 70, 117, 141, 230
 - programmers 19, 24, 224, 226, 314
 - staff roles 74
- routing 14, 29, 58, 75, 107, 125, 156, 160–161, 187
- RPSS
 - Reverse proxy security server 63
- Runtime pattern xii

S

- S/390 71, 103
- sample applications 146
- Scalability 60, 64, 203, 223
- scenarios xiii, 2–3, 65–66, 69–70, 92, 102, 114, 146, 216, 229, 231, 250–251
- Securing ASP.NET Web Services 331
- Security
 - asymmetric encryption 164
 - authentication 63, 88–89, 129, 159, 161–163, 169, 177, 201, 219, 223, 230, 233
 - autonomic capabilities 56
 - breaches 63
 - certificate 166
 - certificate revocation lists 167
 - digital signature 89, 128–130, 161, 163–164, 168–169, 172–174, 176–177
 - DMZ 90, 110, 218, 221, 224
 - Encryption 89, 129–130, 159–161, 164–167, 169–170, 174, 176–177
 - Enterprise Privacy Authorization Language 130
 - enveloping signature 173
 - EPAL 1.1 130
 - errant employee programmers 224
 - federated identity management 130
 - federated secured user directory 78
 - firewall 44, 162, 223–224
 - Gateway-level authentication 219
 - hacker attacks 108
 - Kerberos 120, 130–131, 135, 158, 167
 - Manage 219
 - message level encryption 159–160
 - Operation-level authorization 219
 - password digest 175
 - PKCS7 176
 - privacy 63, 130
 - privacy policies 130
 - private key 130, 163, 165–166
 - Proxy authentication 219
 - public key 163
 - public key certificate 174
 - public-private key mechanism 165
 - RPSS
 - Reverse proxy security server 63
 - RSA 117, 119–120, 164, 166
 - Secure Sockets Layer 158
 - SHA-1 163
 - SSL 158
 - SSL Protocol Version 3.0 148
 - Transport layer security 171
 - transport level encryption 159
 - triple-DES 166
 - vulnerability to attack 224
 - Web Trust Association 63
 - X509PKIPathv1 176
- SEI
 - Service Endpoint Implementation 268, 296–297
- self-describing 19
- Service Availability 63–64, 216
- service broker 40
- Services 30, 32, 36
 - service bus xi, 3, 39, 41, 56, 58–59, 65–66, 90, 96, 101–102, 109–110
 - service Description 27, 52, 140, 147, 152
 - service discovery 60–61
 - service endpoint interface 268, 297
 - service instance 147
 - service integration bus 95
 - service interface 40, 45, 89, 198, 224, 296
 - service level agreement 72
 - Service Oriented Architecture xi–xii, 5, 39, 45, 50–52, 54–55, 66, 69, 79–80, 90, 93, 95–96, 98, 177, 183, 217
 - service Provisioning Markup Language 143
 - service registry 40, 53
 - service requestor 40–41, 53, 59, 66, 178, 293
- servlet 23
- SHA-1 163
- signature element 173
- Simple Object Access Protocol
 - see SOAP
- Simple Object Access Protocol (SOAP) 1.1 147, 332
- simple query 40
- single customer view xi
- SOA
 - Service Oriented Architecture xi, 51–60, 62–63, 65–66, 69, 79–80, 88, 93, 95–96, 98, 100–102, 109–111
- SOAP
 - Bindings 36
 - Body 14, 16–18, 24, 29–30, 32–33, 151, 153–154, 171, 174, 247–250, 304–305, 307–308

- Document/Literal 18, 20, 30, 33, 149, 282
 - encodingStyle 15, 18, 151
 - endpoint 24, 29, 60, 62, 90, 126, 140, 142, 156, 169, 178, 181, 187–188, 209, 216, 218, 222, 268, 275, 297, 300
 - Endpoint Reference 126
 - Envelope 14, 17–18, 33, 127–128, 151, 171, 174, 188–189, 247–250, 307–308
 - EPR 126
 - extensibility 14, 25, 37, 43, 131, 147, 170, 178
 - Fault 14, 16–17, 28–30, 126, 150–151, 154–155, 169, 177, 216, 239–240, 249–250, 297, 303–306
 - Faultfactor 17, 151
 - Faultcode 17, 151, 250
 - Faultstring 17, 151, 250
 - header entry 14, 16
 - interaction patterns 21
 - operation name 20
 - part accessors 20
 - request-response 21–22, 24, 124
 - response message 21–22
 - RPC style 18–21
 - RPC/Encoded 18–19, 149, 151
 - RPC/Literal 18, 20, 30, 33–34, 149, 282
 - Service consumers 45, 53, 89, 96, 147
 - SOAP 1.1 15, 151–152
 - SOAP 1.2 16, 116
 - SOAP Body 16, 29, 171, 249
 - SOAP Envelope 14, 128, 171, 174, 189
 - SOAP exception 249
 - SOAP Header 14, 29, 171
 - SOAP message 12–14, 20, 29, 128, 151, 156–157, 161, 188–189, 248–250
 - SOAP node 15
 - SOAP/JMS 189
 - SOAP/MQ 189
 - SOAPACTION 22–23
 - software component 4–5, 39, 41–42, 45–47, 217, 222
 - Software environments 13–14, 47, 49, 66
 - Heterogeneity 46, 50–51
 - solution
 - architect 5
 - architecture xi, 67, 83, 91–92
 - architectures 3, 70
 - context 69, 72, 84
 - integration 25
 - level design 77, 85
 - Specifications 37
 - SPML
 - Service Provisioning Markup Language 143
 - SQL Servers 192
 - staffing 71, 85
 - stand-alone applications 45
 - Strong Names 193
 - subsystems 237–238
 - Sun xiv
 - Suppliers 41, 47, 56–58, 71, 111
 - symmetric encryption 164
 - synchronous 157
 - System House xiii–xiv, 2, 70, 108
 - System House Business Scenario 70
 - Systems Programmer 226
- T**
- TCP/IP 64, 195
 - Technical approach 78, 88
 - Technical constraints 69, 76, 85
 - Test client 207–208, 252, 257–259, 262, 266, 269, 285, 290–292, 299–300, 303
 - Test Environment 213, 256–258, 320
 - testers 70
 - three legged handshake 183
 - topologies 56, 223
 - touch point 60, 62
 - Trading Partner Agreements 139
 - transaction management 192
 - Transactional support 79, 181
 - Transactionality 63–64
 - Transparency 55, 59–60
 - transport 29–30, 32, 36
 - transport level protocol 159
 - two-phase commit 63, 180
 - type information 13, 18–20, 24, 36
- U**
- UDDI xii, 6–7, 14, 19, 40, 42–44, 50, 53, 63, 89–90, 116, 135–136, 140, 143, 147–148, 203, 207, 210, 215–217, 219–223
 - UDDI registry 147, 217
 - UML
 - boundary class 235, 237, 245
 - control class 235, 244–245
 - underscore for the method attributes 314
 - Understanding Web services 330
 - Unified administration 60

- unique service name 312
- Universal Database DB2 Version 8.1 319
- universal description, discovery, and integration
 - see UDDI
- Universal Test Client 207, 257–259
- unmanaged environment 47
- URI 15, 17, 22–23, 30, 123, 154, 167, 172, 185, 246, 293
- URLScan tool 223
- Use case
 - Manage external claim assessor 240–241, 243
- use case
 - realization 234, 244
- use case definition 230
- use cases xii, 70, 80, 83, 146, 229–232, 240–241, 245
- user request 97, 252
- UserLand 12
- Username Token Profile V1.0 161, 332
- Using SOAP Faults 306
- UTP-16 127
- UTP-8 127

V

- Validate claim 76, 78
- vendors 4, 24, 37, 45, 49–50, 57, 70, 80, 114, 121, 135, 143, 146, 170, 177, 182–183, 189–190, 204, 217
- virtual directory 283
- Visual Basic .Net 193–194, 200
- Visual Basic scripts 192
- Visual C++ 192

W

- W3C 7, 11, 27, 35, 41, 116, 120, 122, 124, 128, 135, 155, 164, 166, 247
- web application 73, 207, 222, 224–225, 285, 293–294
- Web reference 199, 212, 288
- Web service
 - composable 44
 - interoperable 5, 36, 41, 44, 58, 74, 82–83, 121, 146, 150, 183, 198, 236, 247
 - language-independent 44
- Web service Browser 269
- Web service Explorer 267, 269, 275, 293
- Web services
 - addressing information 62

- Co-existence 60
- consumer 51–52, 96, 114, 127, 140, 147, 152, 216, 221, 293, 311
- Deployment 273
- MUWS 140
- self-contained 40, 44
- self-describing 44
- specifications 43, 113, 330
- Web services Architecture 332
- Web services clients 293
- Web services description language
 - see WSDL
- Web Services Explorer 270, 295, 301
- Web Services Gateway 89–91, 169, 217–220, 222, 226
- Web Services Inspection Language 136
- Web Services Interoperability Guidance (WSIG)
 - IBM WebSphere Application Developer 5.1.2 331
- Web Services security 219
- web.xml 24, 207
- webservices.xml 24, 209–210, 212, 268
- webservicesclient.xml 210, 212, 269
- WebSphere Application Server xi–xii, xiv, 3, 5, 9, 23, 111, 134, 188–192, 203, 206–208, 211–212, 245, 252, 271, 275, 321, 325
 - Cumulative fix 5.1.1.1 319, 321–322
- WebSphere Application Server Network Deployment 217
- WebSphere Application Server Version 5.1.0, Fixpack 5.1.1 319
- WebSphere Business Integration Adapters 49
- WebSphere Business Integration Server Foundation 79, 204
- WebSphere MQSeries 2, 41, 55, 64, 70, 121, 134, 188–189, 202, 207
- WebSphere MQSeries SOAP Supportpac 330
- WebSphere plug-in 218, 222
- WebSphere Studio Application Developer xi–xii, 3, 20, 27, 30, 79, 111, 114, 148, 150, 190, 197, 206–207, 209, 211–213, 227, 230, 237, 245, 251–252, 262, 275, 311–315, 324, 328
- WebSphere Studio Application Developer 5.1.2 319
- Westbridge Technology 119–120
- Windows 2003 111, 276, 283, 285, 325
- Windows Server 2003 4, 193, 222–223, 276, 320–321
- wire format 36

workflow 41, 44, 53, 74–75, 77–78, 83, 85–87, 91, 107, 134, 139, 177
 work-list 87–88
 wrapper 20
 WS-Addressing 21, 64, 116, 126, 137, 155–158, 183, 186–188
 WS-AT 181
 WS-AtomicTransaction 133–134, 180
 WS-AtomicTransactions 116, 134
 WS-Attachments 116, 127
 WS-BaseNotification 117–118
 WS-BrokeredNotification 117–118
 WS-Business Agreement 133
 WS-BusinessActivity 117
 WS-CAF 117, 133
 WS-Coordination 64, 117, 133–134, 177, 179–180
 WS-CTX 117
 WSDL
 Definitions 31, 36
 Document/Literal 18, 20, 30, 33, 149, 282
 operation name 20
 RPC/Encoded 18–19, 149, 151
 RPC/Literal 18, 20, 30, 33–34, 149, 282
 WSDL (Web Services Description Language) 2.0 120
 wsdl2java 149
 WS-Eventing 117, 127
 WS-Experience Language 117
 WS-Federation
 Active Requestor Profile 117
 Passive Requestor Profile 117
 WS-Federation Language 117
 WS-I
 Basic Security Profile 1.0 4
 conformance 146
 conformant consumer 147
 conformant service instance 147
 WS-I Basic Profile 1.0 118, 143
 WS-I Basic Profile 1.1 118, 332
 WS-I Basic Profile Version 1.0 332
 WS-I Simple SOAP Binding Profile 1.0 118
 WSIL 50, 136
 WS-Inspection 1.0 118
 WS-Manageability 1.0 118
 WS-Manageability-Representation 118
 WS-Manageability - Concepts 118
 WS-MetadataExchange 118, 137
 WS-Notification 117–118, 126–127, 137
 WS-Policy 35, 37, 119, 130, 135, 137, 183, 201
 WS-PolicyAssertions 119, 135
 WS-PolicyAttachments 119
 WS-PolicyFramework 119
 WS-Privacy 130
 WS-Provisioning 119, 143
 WS-RM 119, 134
 WSRP 1.0 119
 WS-SecureConversation 119, 131, 201
 WS-Security 4, 80–81, 109, 119, 129, 131, 161–164, 166–170, 175, 177, 183, 186, 201, 211–212, 219, 229–230
 WS-Security 2004 119
 WS-Security Drilldown in Web services Enhancements 2.0 331
 WS-Security Kerberos Binding 120
 WS-SecurityPolicy 119, 130, 201
 WS-Topic 126
 WS-Topics 117–118
 WS-Transaction 181
 WS-Transaction 1.0 120
 WS-Trust 120, 130–131, 201
 WS-TXM 117
 WSXL 117
 WX-CF 117

X

X.509 Token Profile V1.0 161, 332
 X509PKIPathv1 176
 XML Binary Optimized Package 128
 XML infoset 128
 XML schemas 18, 24, 36, 53, 121
 XOP 128
 XSD 23, 33, 35–36, 123, 212, 246–250, 304, 306–308
 XSDs 36, 122

Z

zones 98

Archived



WebSphere and .Net Interoperability Using Web Services

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



WebSphere and .Net Interoperability Using Web Services

Examples and guidance for building interoperable Web services

Roadmap to Web services specifications

Using Service-Oriented patterns

IBM and Microsoft are strong supporters of the Web Services Interoperability Organization's (WS-I) efforts to make building solutions using software from different suppliers a reality. In this IBM Redbook, we take a practical look at building a solution with IBM WebSphere and Microsoft .Net components using Web services that are compliant with the WS-I organization's Basic Profile.

This redbook is aimed at customers who want to know how far the reality of Web services has caught up with the hype; it is for customers who want a redbook to help them decide whether Web services are right for them now.

The book provides an introduction to SOAP, WSDL and the rest of the Web services concepts. It provides a review of the many Web service specifications. Which ones are most important to building a practical solution?

We use a scenario based on work IBM has been doing with the insurance industry to demonstrate how to design a service-based solution and then implement it using the latest programming tools from IBM and Microsoft.

Based on our experience, we identify areas where extra effort up front will be rewarded with an easier implementation.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks