

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL

S-O
I-C-S

WS-Addressing • WS-Policy
WSDL • XML Schema
SOAP • Namespaces
Web Services
WS-* • SOA

*"The New Bible for WS-
Design and SOA."*

Sascha Kuhlmann
Global Program
Lead Enterprise
Architecture
SAP

Web Service Contract Design & Versioning for SOA

Foreword by
David Chappell

 PRENTICE
HALL

Thomas Erl, Anish Karmarkar, Priscilla Walmsley,
Hugo Haas, Umit Yalcinalp, Canyang Kevin Liu,
David Orchard, Andre Tost, James Pasley

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: www.informit.com/ph

Library of Congress Cataloging-in-Publication Data:

Web service contract design and versioning for SOA / Thomas Erl ... [et al.]
p. cm.

ISBN 0-13-613517-X (hbk. : alk. paper) 1. Web services. 2. Computer network architectures. 3. Computer architecture. I. Erl, Thomas. II. Title: Web service contract design and versioning for Service Oriented Architecture.

TK5105.88813.E76 2008
006.7'8—dc22

2008025188

Copyright © 2009 SOA Systems Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671 3447

ISBN-13: 978-0-13-613517-3

ISBN-10: 0-13-613517-X

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.
First printing September 2008

Editor-in-Chief

Mark Taub

Managing Editor

Kristy Hart

Copy Editor

Language Logistics

Development Editors

Dmitry Kirsanov

Songlin Qiu

Indexer

Cheryl Lenser

Proofreader

Williams Woods Publishing

Composition

Jake McFarland

Bumpy Design

Graphics

Zuzana Cappova

Tami Young

Photos

Thomas Erl

Johan Odendaal

Foreword

I have often commented that learning the Web services set of specifications (WS-*) by reading them is like trying to understand the English language by reading Webster's dictionary. This is because every concept, every word is expressed in terms of other words that have yet to be understood, resulting ultimately in cyclical definitions. How could one effectively learn a verbal language that way?

Nearly ten years in the making, the WS-* specifications are the result of an industry-wide effort to define interoperable interfaces, transports, and metadata, using XML and XML Schema as the means for doing so. The specifications are intentionally broken up into small atomic units so that individual applications or infrastructure offerings from commercial vendors and open source may implement them in order to ease adoption and move us all closer to platform-independent interoperability. They are also composable with each other in that they may be used together. But how?

In all fairness, the WS-* specifications are self-explanatory enough to get a rough understanding of the purpose of each one if you read them long and hard enough. Through arduous scrutiny of jargon and angle brackets, eventually there is enough that sinks in to reach a critical mass of basic understanding.

Grasping their purpose and using the WS-* specifications as a reference for implementation is only the first level of knowledge. In fact most programmers and architects by now are familiar with the common trio (WSDL, XML Schema, and SOAP). This does not, however, imply that the use of these specifications is intuitively obvious. And what about the other specs in the stack? At last count, there were nearly 50. Can they all be learned together? How does one sort through them all and figure out which ones are the

most important to pay attention to and which will have the most impact on their organization if used properly? What is the impact if used improperly? Even if the details are partially hidden behind an SOA platform, there are still architectural issues to be considered.

Enter this book. It is about technologies and practices related to designing and governing Web service contracts in support of SOA and service-orientation. Written by experts who have themselves contributed to the creation and development of the WS-* specifications, this book provides guidance to demonstrate how the most relevant Web service contract technologies work together as a framework—insight that all too often gets left behind on the cutting room floor after countless conference calls and face-to-face meetings among working groups, technical committees, task forces, and sub-committees. The book further helps the reader to understand how the WS-* technologies can be best leveraged and evolved in support of SOA and service-orientation.

This book provides the reader with common understanding across a broad range of WS-* technologies as they pertain to service contracts, service versioning, policy management, and SOA governance in order to enable the reader to leverage the full capabilities that are part of modern WS-* platforms. Issues such as the pros and cons and proper use of the many standardized interoperable message exchange patterns are explored, as well as how to take full advantage of WS-Addressing. In addition, this book provides in-depth discussion on governance issues, particularly as it pertains to service contract design, service versioning, and service policy. Insight and guidance is provided on when and how to design WSDL operations that are intelligently combined with XML Schema and WS-Policy statements to produce service contracts that are conducive to effective versioning.

We hope you enjoy this body of work. Your SOA projects are likely already under way or are about to begin. You are building new architectures for your organization that are going to be in place for decades to come. Many years of hard work have gone into the WS-* set of specifications, and this book will help you to reap the benefits of that work to create truly loosely-coupled and flexible service contracts that will mitigate the impact of change on your organization.

—*David A. Chappell, Technologist and author of several books and papers on interoperability, SOA, and Enterprise Service Bus*

Preface

After we completed this manuscript, I checked the schedule and noticed our original start date. From the initial kick-off call during which everyone was given the green light to begin writing their chapters to the day I had to hand over the manuscript to Prentice Hall for indexing spanned a period of about 32 months. I initially didn't think too much of it because I already knew this project had taken over two years. But when I looked at that number again sometime later, it struck me.

The time it has taken for this book to be developed and authored is actually comparable to the time it originally took for several of the XML and Web services-based technology specifications covered in this book to be developed into fully ratified standards.

Though a curious statistic, this comparison doesn't do the subject matter justice. The development processes these technology standards were subject to are on entirely different levels, in that they were vastly complex both from human and technology perspectives.

There's the human element that emerges in the technical committee that is tasked with the responsibility of producing a standard. Such a committee will be comprised of members with different agendas, different perceptions, and different personalities. So many differences can turn a standards development process into a rollercoaster of group dynamics, ranging from strong teamwork to stages of scrutiny, confrontation, and even raw tension. Trying to achieve a consensus in an active technical committee is not for the weak at heart.

And then there's the technology element, which is reflected in the deliverables produced by the committee. Technical specifications are meticulously crafted and worded and revised and reworded in continuous, patient, and sometimes mind-numbingly tedious

cycles. But despite best efforts, creating a new language or vocabulary that will meet the ever-escalating needs and expectations of the industry as a whole is a daunting prospect. Not to mention that there is a constant possibility that the particular standard a committee might have spent a good part of their lives working on will be overshadowed by a competing effort or perhaps even rejected by the industry altogether.

But amidst these challenges, there have been many success stories. In a way, this book is a testament of this in that it documents a collection of respected and widely-recognized de facto standards that have established themselves as important IT milestones.

Ultimately, though, this book is about you, the reader. It was written for you to fully leverage what these technology standards have to offer. As successful as these technologies have been, what counts in the end is how effective they are for you.

—*Thomas Erl*

Chapter 4



Anatomy of a Web Service Contract

- 4.1 What is a Web Service Contract?
- 4.2 The Parts of a Web Service Contract
- 4.3 Technologies Used to Create Web Service Contracts
- 4.4 Guidelines for Using Web Service Contract Technologies

Web service contracts can range in content, depth, and complexity. To fully appreciate the intricacies and design options of how Web service contracts can be structured, we first need to decompose this structure in order to understand its individual parts and the mechanics that make these parts work together.

The purpose of this chapter is to explain the Web service contract from a conceptual and structural perspective without yet getting into the details of how the contract is actually developed through markup code.

We start exploring contract structure by breaking the contract down into a set of primary parts. This allows us to describe what these parts are, what technologies can be used to create them, and how they can relate to each other. Subsequent chapters will then drill down into each of the mentioned parts and technologies.

4.1 What is a Web Service Contract?

A Web service contract is essentially a collection of metadata that describes various aspects of an underlying software program, including:

- the purpose and function of its operations
- the messages that need to be exchanged in order to engage the operations
- data models used to define the structure of the messages (and associated validation rules used to ensure the integrity of data passed to and from the messages)
- a set of conditions under which the operations are provided
- information about how and where the service can be accessed

Several different and cooperating technologies are required to formally define this metadata, as explained later in the *Technologies Used to Create Web Service Contracts* section.

Fundamental Structure

Web service contracts are organized into a basic structure that reflects a relatively clear separation of “what,” “how,” and “where” as follows:

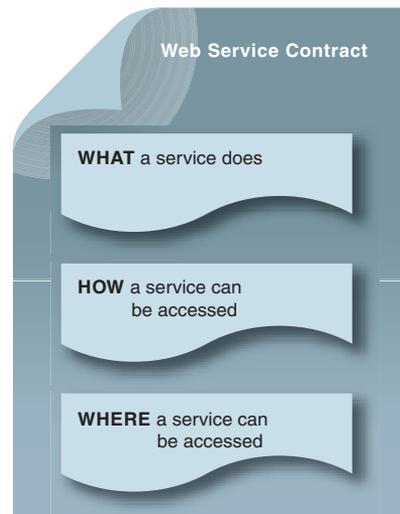
- **What** is the purpose of the service and its capabilities?
- **How** can the service be accessed?
- **Where** can the service be accessed?

When potential consumer program designers evaluate a service, they need to know *what* the service is capable of doing and under what conditions it can carry out its capabilities. If what’s offered is what consumer designers need, then they must be able to determine *how* and *where* to access the service.

As illustrated in Figure 4.1, the service contract is organized into sections that individually address these three questions.

Figure 4.1

A Web service contract defines what a service offers and how and where it can be accessed.



In addition to providing flexibility as to how a service can be located and consumed, this clean separation allows different parts of the contract to be owned and developed at different stages of the service delivery lifecycle by different members of a project team.

For example, architects and analysts can focus solely on the “what” part of a service when it is being conceptualized and designed. The “how” and “where” parts don’t usually become relevant until developers actually build and deploy the contract as part of the overall service implementation.

Abstract and Concrete Descriptions

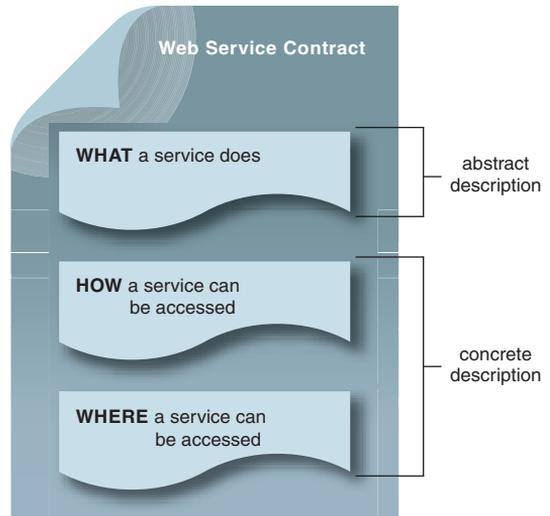
There are formal terms used to represent the three fundamental parts we just covered. As shown in Figure 4.2, the “what” portion of the Web service contract is referred to as the *abstract description*. This part is essentially responsible for expressing the contract’s public interface (or API).

The balance of the Web service contract is represented by the *concrete description*, providing the implementation and communication details necessary for consumer programs to locate and use the service at runtime.

The concrete description encompasses the previously described “how” and “where” parts. An example of a characteristic that would be considered a “how” part is the wire format and protocol necessary to exchange messages with the service.

Figure 4.2

An abstract description establishes the technical interface independent of implementation details. A concrete description adds deployment-specific details about how and where to access a service.



NOTE

The terms “abstract description” and “concrete description” originated with the W3C, the standards organization responsible for developing most of the technologies covered in this book. From an implementation perspective, the abstract description is also very much concrete in that it ends up forming the actual physical interface that service consumer programs end up connecting to.

SUMMARY OF KEY POINTS

- The term “abstract description” is commonly used to refer to the “what” part of the contract, whereas the term “concrete description” represents both the “how” and “where” parts.
 - Abstract and concrete descriptions are frequently created during different stages of the service delivery lifecycle by different members of the project team.
-

4.2 The Parts of a Web Service Contract

Having just established a high-level structure of a Web service contract, let’s drill down a bit to explore how abstract and concrete descriptions are further sub-divided into smaller parts.

NOTE

The following sections cover only the primary parts of the Web service contract. There are many other, more specialized definitions that will be introduced and explored in subsequent chapters.

Primary Parts of the Abstract Description

Figure 4.3 provides us with a *logical* view of the parts that comprise the abstract description. This means that when this section of a contract is actually created, these parts will be physically structured differently (as shown later in the *A Physical View of the Abstract Description* section).

The following sections are numbered to correspond with the numbers on Figure 4.3.

1. Port Type (Interface) Definition

The definition of the *port type* is the cornerstone of a Web service contract. Think of a port as a place of entry (such as a port used to receive ships for transferring cargo). A Web service contract can have one or more “types” of ports. Each port type essentially represents a technical interface or access point.

Note that for reasons we’ll explain later, it is sometimes more appropriate to refer to the port type definition as the “interface definition.”

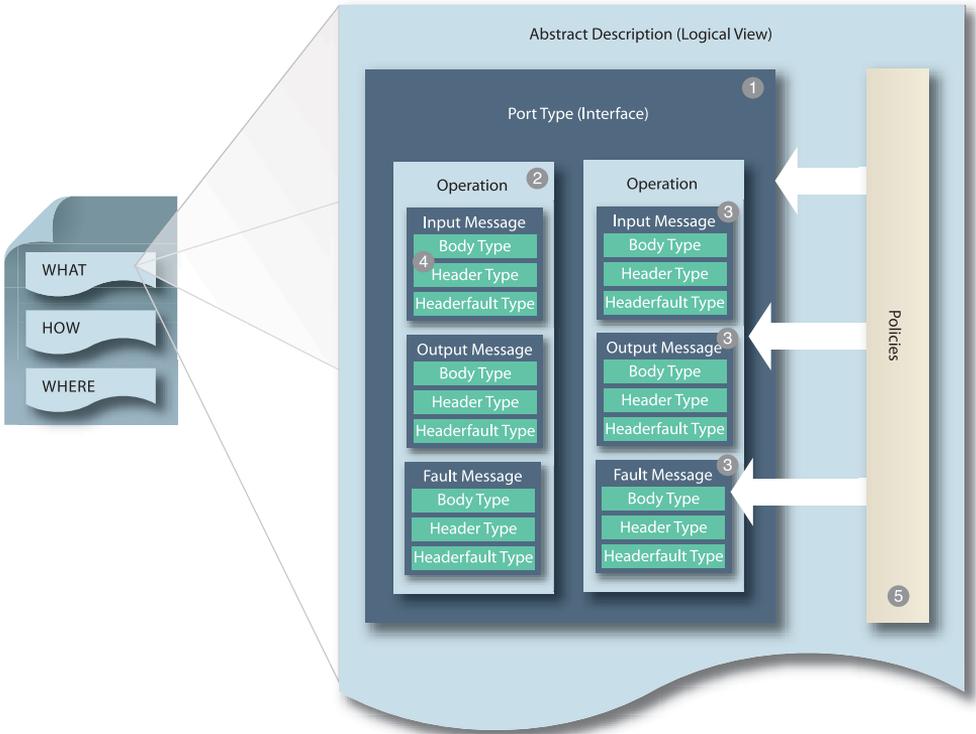


Figure 4.3
A logical view of the primary parts of an abstract description.

2. Operation Definition

For a Web service to be used by another program, it needs to expose externally consumable capabilities or functions. These are referred to as operations and are expressed as *operation definitions*.

A port type acts as a container for a set of related operations.

3. Message Definition

When a Web service is being invoked via one of its operations, the consumer program needs to exchange data with it in order for the Web service to perform the requested function. A *message definition* essentially establishes a “mini-structure” for the data to be transmitted.

There are three types of message definitions:

- *Input* – a message sent to the Web service by a consumer program
- *Output* – a message sent by the Web service to the consumer program
- *Fault* – an error notification message sent by the Web service to the consumer program

An operation can contain one or all of these message definitions but would generally not contain only an output or a fault message.

4. Type Definition

The data for a given input, output, or fault message needs to have a pre-defined structure of its own. That way, the Web service knows whether or not it is receiving or sending valid data. This data structure is established in a *body type definition*.

Messages further support the inclusion of message-specific metadata (information about the message that accompanies the message). For this form of supplementary data, a *header type definition* is also provided.

Finally, just as message transmissions can encounter error conditions, so can the processing of message metadata. Therefore, an additional *headerfault type definition* is available for when a response to a metadata-related error condition needs to be sent.

NOTE

The use of the headerfault type definition is not common in practice.

5. Policy Definition

All of the previous definitions in the abstract description have collectively defined the functional aspects of the Web service contract. *Policy definitions* can be created to extend the technical interface by expressing further behavioral requirements and characteristics of the service.

A Web service contract can have any number of supplementary policies. The horizontal arrows in Figures 4.3 and 4.4 indicate that policies can be specific to certain parts of the abstract description.

Note that policies can also be applied to parts of the concrete description.

A Physical View of the Abstract Description

So far we've been focusing on the logical view of the abstract description because it is conceptually important to understand it from this perspective. However, this is a book about technology, and the hundreds of pages that follow this chapter are all concerned with how Web service contracts are physically built.

Therefore, it's time that we start learning about how the organization of the various definitions we've described so far differs in a real-world contract document. As shown in Figure 4.4, the purpose behind the physical structure is to promote reuse in support of flexible and normalized Web service contracts.

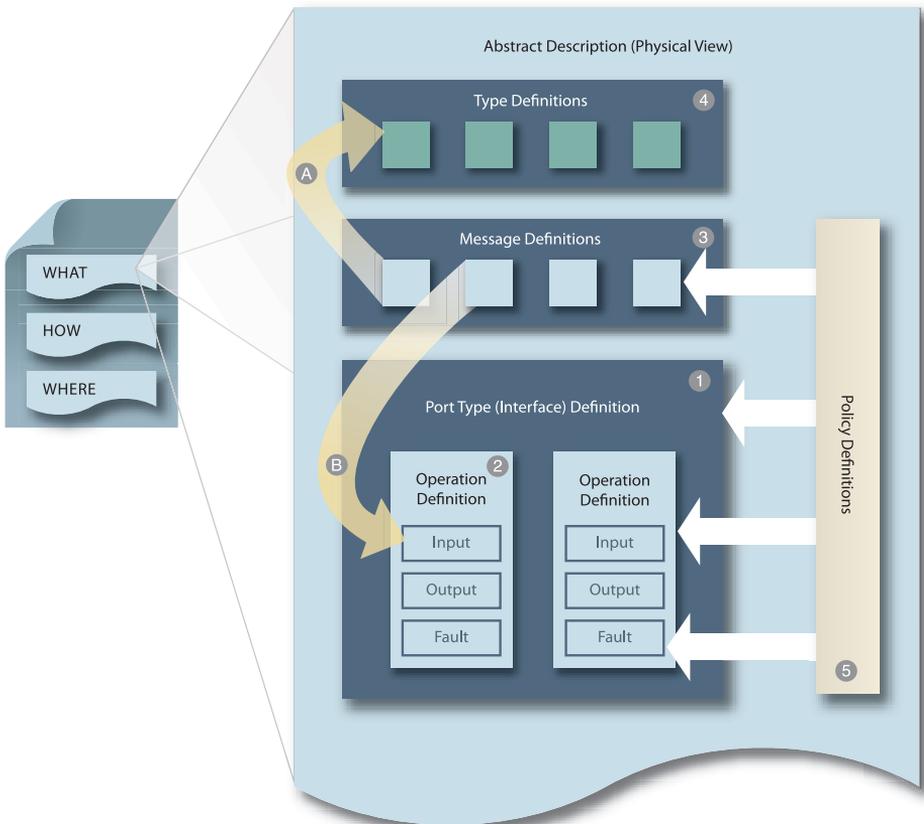


Figure 4.4
A physical view of the primary parts of an abstract description.

By following the numbers in Figure 4.4 we can trace back the actual location of the previously described definitions. While the port type definition (1) and policy definitions (5) are pretty much in the same place, we can see that the message definitions (3) and the type definitions (4) are no longer part of the port type or operation definitions (2).

Type and message definitions are positioned in separate parts of the contract so that they can be reused as follows:

- A. Each type definition can be used by multiple message definitions.
- B. Each message definition can be used by multiple operations (as input, output, and fault messages).

Although Figure 4.4 shows a series of available types, it does not make a distinction between a body, header, or headerfault type. All three types can exist as individual type definitions (4) but are then bundled together into one message definition (3). The body portion of this message is then associated with an input, output, or fault message as part of the operation definition (2).

NOTE

The header and headerfault parts are assigned to messages in the concrete description, the reason being that the messaging protocol to which the abstract description will be bound may or may not support their use.

WSDL 2.0 Changes to Physical View

We've intentionally avoided mentioning specific technologies so far in this chapter in order to establish as much of an abstract perspective of Web service contracts as possible. But it is worth noting that this book covers two different versions of the WSDL language used to express the Web service contract structure.

WSDL is explained later in this chapter. At this stage, we just need to point out that the physical view we just described complies with WSDL version 1.1. In version 2.0, the message definitions section (3) is removed, and operation definitions (2) refer directly to individual type definitions (4).

NOTE

In the upcoming sections dedicated to the parts of the concrete description, there is no distinction between a logical and physical view.

Primary Parts of the Concrete Description (Part I)

A concrete description supplements the abstract description with implementation-specific details required for the Web service to be invoked and communicated with by service consumer programs.

As we established earlier, the concrete description corresponds to the “how” and “where” sections of a Web service contract. We’ll focus on these sections individually, starting with the “how” considerations addressed by the binding-related definitions illustrated in Figure 4.5.

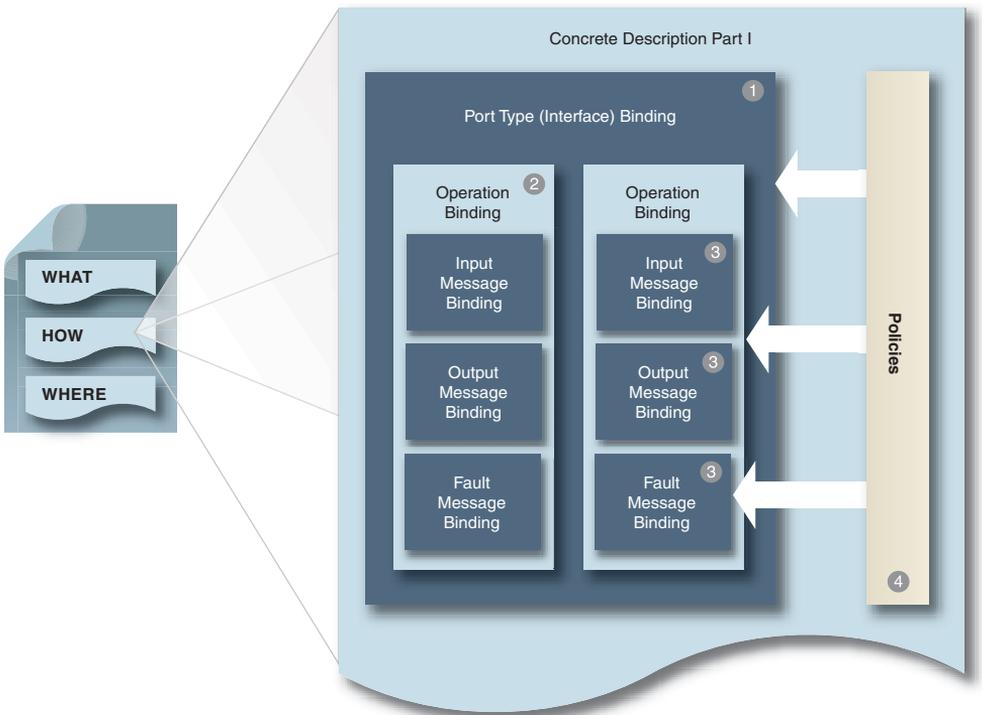


Figure 4.5 The primary parts of the binding portion of the concrete description. (Note that this represents both logical and physical views.)

The following sections are numbered to correspond with the numbers on Figure 4.5.

1. Port Type (Interface) Binding Definition

A *binding definition* details the communication technology that can be used by consumer programs to invoke and interact with the Web service.

A port type definition within an abstract description can be associated with one or more binding definitions, each of which contains the following two pieces of information:

- *Messaging Protocol Binding* – Defines a technology or industry standard (the message protocol) that specifies the format in which a message should be packaged by the sender and then unpackaged by the receiver.
- *Transport Protocol Binding* – Defines the communications technology (the transport protocol) with which the message should be transmitted across a network.

2. Operation Binding Definition

An *operating binding definition* represents the same type of binding definition as the port type binding, except that it is specific to the operation only. If the operation does not have this binding specified, it inherits the binding settings from its parent port type.

3. Message Binding Definitions

As with the operation binding, the *message binding definition* also represents a granular form of binding definition that, in this case, is specific to an input or output (or fault) message. If a message-specific binding is not provided, the message inherits the binding settings from its parent operation or its parent port type (if an operation binding is not present).

It is also here in the message binding definition that the header and headerfault parts of a message get associated with an input, output, or fault message that was defined in the abstract description. The reason again for making this association here is because not all message protocols support the use of header and headerfault parts.

4. Policy Definition

As with the definitions of an abstract description, policies can also be defined for the individual parts of a concrete description. Policies for binding definitions tend to be related to the configuration and runtime requirements of a particular messaging or transport protocol.

Primary Parts of the Concrete Description (Part II)

This final portion of the Web service contract structure focuses on the “where” section, as shown in Figure 4.6.

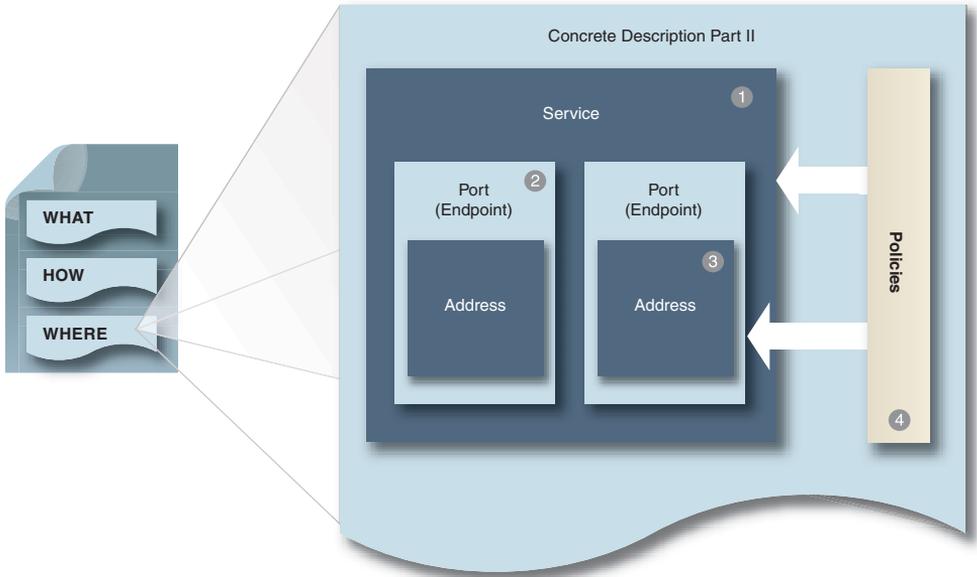


Figure 4.6

The primary parts of the service portion of the concrete description. (Note that this represents both logical and physical views.)

The following sections are numbered to correspond with the numbers on Figure 4.6.

1. Service Definition

The *service definition* simply groups related port (endpoint) definitions together. The port definition is explained shortly.

2. Port (Endpoint) Definition

Each binding definition needs to be associated with a port definition (also referred to as an endpoint definition), which simply acts as a container for the address definition (explained shortly). The same port definition can be used by different port type, operation, or message binding definitions.

3. Address Definition

An *address definition* establishes a physical network address. Depending on the communication technology (transport protocol) defined by the port type, operation, or message binding definition that is referencing the parent port definition for this address definition, the actual address value might be a Web URL, an email address, or some other type of transport-specific address.

4. Policy Definition

Policies can be created for address-related definitions in the concrete description. This may be necessary when certain requirements or characteristics specific to the network address need to be expressed.

How Parts of a Contract Relate to Each Other

As you may have already gathered, Web service contracts are highly modular. Let's take a minute to describe some of the relationships these parts can have:

- On a high level, one abstract description can have one or more associated concrete descriptions, each specifying a different messaging and/or transport protocol.
- One port type (interface) definition can have multiple operation definitions.
- Each operation definition can reference multiple message definitions.
- Each message definition can reference multiple type definitions.
- One port type (interface) definition can be associated with multiple binding definitions.
- Each binding definition can have multiple port (endpoint) definitions.
- A service definition can group multiple port (endpoint) definitions.
- Policies can be applied to most of these definitions.

Figure 4.7 illustrates these relationships by providing an example of one port type definition that is associated with two port type binding definitions (relationship A), each with its own port definition (relationship B).

Based on the numbers used in this diagram, we can further assume that the port type definition (1), the operation definition (2), and a message definition (3) each has its own corresponding binding definition. Also, the port type binding definition is associated with a port (4) that establishes its address.

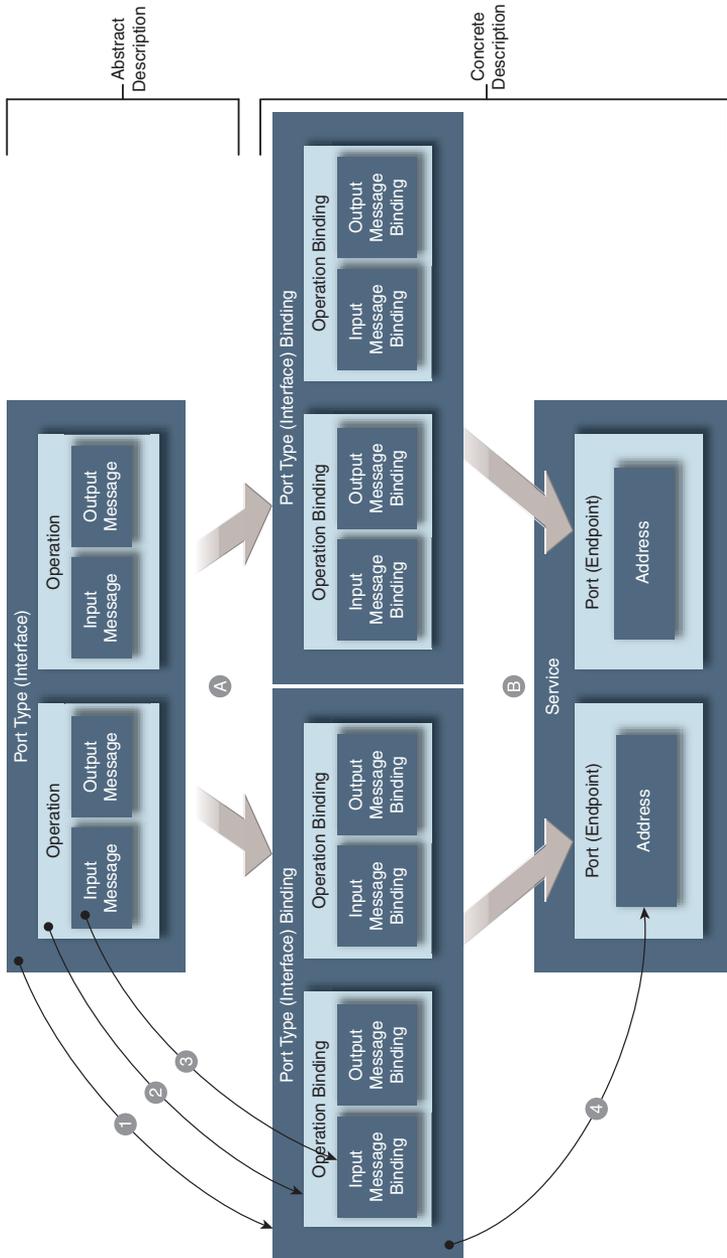


Figure 4.7

An example of how abstract and concrete descriptions can relate to each other. (Note that fault messages have been omitted from this diagram.)

These relationships are discussed in detail in subsequent chapters. At this point it's just helpful for us to understand that various types of relationships can exist.

The Primary Parts of a Message

It's important to remember that building Web service contracts is more than just creating an interface to underlying service logic. The contract defines, in a very specific manner, the actual interaction requirements of the Web service. And, as you may have gathered from the previous two sections, both abstract and concrete descriptions express these interaction requirements by defining input and output messages and the technologies used to support the processing of these messages.

Let's therefore take the time to familiarize ourselves with the basic parts of a message. As shown in Figure 4.8, messages are wrapped in envelopes that have a basic structure of their own.

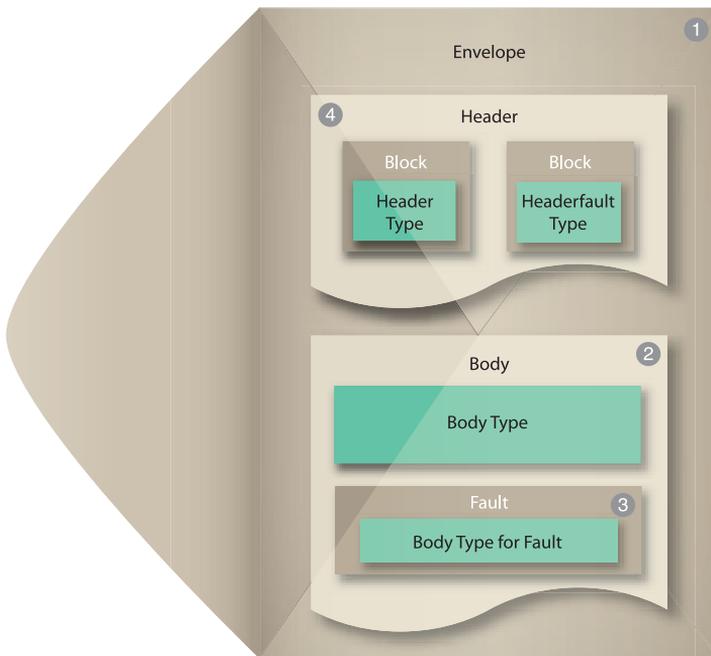


Figure 4.8
The fundamental structure of a message envelope.

Note that the following parts are not referred to as “definitions” because they represent an actual manifestation of what was established in the Web service contract definitions. In other words, runtime processors use the Web service contract definitions to generate these parts of the message (at runtime).

The following sections are numbered to correspond with the numbers on Figure 4.8.

1. *Envelope*

The scope of a message is defined by its *envelope*. This name is appropriate because it describes the message as a container of information. When discussing Web services, the terms “envelope” and “message” are frequently used interchangeably.

2. *Body*

The *body* represents a part of the message reserved for the data or the document being transported. Therefore, the type of information that usually resides within the message body is persistent in that its lifespan is greater than the time it takes for the message to be transmitted.

You’ll notice the body part containing the *body type*. This relates back to the body type definition established in the abstract description. That type definition determined the structure for this portion of the message envelope.

Because a message exists as an actual body of data that is being transmitted somewhere, the body type in the message envelope would typically be populated with real data (like an invoice document) organized as per the body type definition from the message definition of the abstract description.

3. *Fault*

A message can be solely intended as a notification from the Web service to a consumer program that an error has occurred. The details of the error are housed in the fault part of the message envelope, which resides within the body section.

The fault part also has a type associated with it. This type corresponds to the fault type definition that was part of the abstract description’s message definition.

4. *Header*

The *header* part of a message provides a location in which supplementary data about the message (metadata) can be stored. This data is usually temporary or non-persistent in nature in that its lifespan is generally equivalent to or shorter than the duration of the message transmission. A header allows metadata to be organized into sub-divided sections called *header blocks*.

Headers can contain many different types of header blocks, each dedicated to providing a distinct piece of supplementary data. This information is used by different programs to either help route a message or process its contents.

The first header block in Figure 4.8 is structured as per the header type definition that was established in the message definition of the abstract description. Similarly, the second header block is based on the headerfault type definition from the abstract description.

NOTE

Of all these parts, only the message envelope and the body are required. The header, header blocks, and fault parts are optional. Typically, only a body type or fault type is present within a given message (not both). Similarly, a message will either contain header types or headerfault types. As previously mentioned, whether the fault part is included in a message is based on which message definitions are present in the abstract description for a given operation.

With regards to header and headerfault parts, these do not always need to be defined as part of the Web service contract. In fact, in practice it has become more common for headers to be added, modified, and removed during the runtime processing of the message. This is explained in detail in Chapter 15.

SUMMARY OF KEY POINTS

- The primary parts of an abstract description are the port type, operation, message, type, and policy definitions.
 - The primary parts of the concrete description are the port type binding, operation binding, message binding, service, port, and address definitions, as well as associated policy definitions.
 - Definitions from abstract and concrete descriptions can have a variety of relationships.
 - Abstract and concrete descriptions jointly define how real-life input and output messages are created.
-

4.3 Technologies Used to Create Web Service Contracts

Now that we've described the primary parts of a Web service contract and also how they can relate to each other, let's identify the development technologies suitable for creating these parts.

Each of these technologies shares the following characteristics:

- *Language* – Every part of a Web service contract is defined by writing code that conforms to a markup language. This is different from a traditional programming language, in that we do not compile our final results into a software program. We simply use the languages to create definitions that exist as textual documents; easy to access and read, but also reliant upon supporting platform programs for run-time processing.
- *XML-Based* – The markup grammar used to write these languages is expressed in XML. Therefore, you end up writing a Web service contract as a set of XML documents for the purpose of exchanging messages that also exist as XML documents and themselves carry data that is further represented with XML.
- *Vendor-Neutral* – Web service contract technologies were all developed by the same industry standards organization (the W3C) through rigorous, multi-year processes. Each was owned by a technical committee comprised of members from different vendor and practitioner organizations, and the final design of most of these language features required a consensus across this committee. Therefore, each of these technologies is also considered a legitimate industry standard.

More information regarding standards development processes, procedures, and stages is provided in Appendix B. Also note that these next sections only briefly introduce each technology; more in-depth coverage is provided in subsequent chapters.

“Element” vs. “Construct” vs. “Definition”

Before we introduce the technologies, let's first establish some additional terminology. If you are familiar with XML, you know that all of these XML-based languages are comprised of vocabularies of pre-defined elements, similar to those used in HTML (another Web language that was ratified into a standard through the W3C).

An additional term you'll see used in these next sections and throughout the remaining chapters is *construct* or *element construct*. A “construct” is a term used to refer to an element that contains nested child elements (in which case the element is implemented with an opening and closing tag). Think of a construct as an “element container.”

For example, an HTML table exists as a table construct comprised of various nested elements (and nested constructs).

There are no rules as to when an element is referred to as a construct. It is just helpful at times to communicate that a particular part of a contract is represented as a container of elements.

The individual parts and definitions of a Web service contract that we've been explaining all exist as element constructs that contain further details. We make a distinction between a definition and a construct (or element) when we need to discuss how a part of a contract is implemented (coded) as opposed to designed.

So a definition, a construct, and an element can all refer to the same thing, just from different perspectives.

As we get into the details of these technology languages, we'll use a style convention whereby specific language element names are displayed in a different font. For example, we might state that the port type definition is created using the `portType` construct which is comprised of opening and closing `portType` tags.

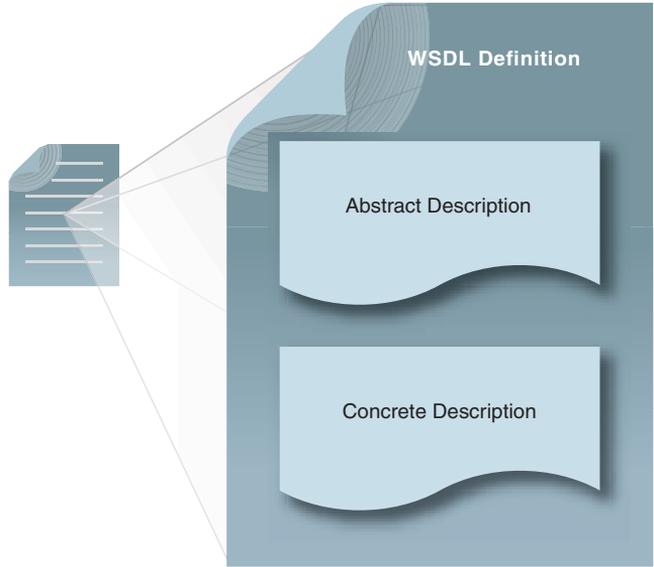
Web Services Description Language (WSDL)

WSDL is considered to be the most fundamental and important technology for Web services development because it has become the de facto language for writing Web service contracts. Using WSDL, contract structure is defined and then further detailed and extended with the other technologies covered in this section.

As you may have gathered, we've already been discussing WSDL in this chapter. The basic Web service contract structure illustrated in previous diagrams has represented a WSDL definition, and the abstract and concrete descriptions are the fundamental parts of a WSDL structure (Figure 4.9).

Figure 4.9

The basic WSDL definition structure that we have been exploring throughout this chapter.



A WSDL document ends up acting as somewhat of a container in which elements from almost all other languages covered in this book converge. Chapters 7 and 8 introduce the WSDL language as it applies to abstract and concrete descriptions respectively. As you explore how these parts of the WSDL definition are assembled, you will be able to see just how much code in the WSDL document does *not* originate from the WSDL language.

Chapter 9 provides further WSDL coverage specific to the WSDL 2.0 standard, and Chapters 14 and 15 drill down into a wide range of advanced topics.

XML Schema Definition Language (XML Schema)

XML Schema provides a formal language for defining the structure and validation constraints of XML documents. Because XML is a native part of the WSDL language and because Web services operate by exchanging XML messages, XML Schema is a highly complementary technology for creating Web service contracts.

Schemas serve as the part of a contract that describes the detailed structure of messages. They define what elements and attributes may appear in the message, in what order, and what type of data they are allowed to contain.

All of the type definitions displayed in the previous figures are assumed to be created using the XML Schema language. The types section displayed earlier in the physical

view of an abstract description (Figure 4.4) is technically part of the WSDL language, but its primary purpose is to house XML Schema elements.

XML Schema code can be embedded directly within WSDL documents or kept as separate documents and then referenced from the WSDL definitions. Each approach has its pros and cons.

When XML schemas exist as independent files, you can establish a many-to-many relationship between the schema documents and the WSDL message definitions. One schema might be used to describe several messages, particularly if they are related and share types. Similarly, each message definition can reference several different XML Schema types originating from different XML Schema documents.

NOTE

In case you're wondering why sometimes the word "schema" is capitalized and other times it isn't, this is explained in the "XML Schema" vs. "XML schema" section at the beginning of Chapter 6.

Most often, you will be designing your own schema from the ground up, based on your specific requirements. At other times, you will be using an industry schema that was developed by an industry organization, a vendor, or even a private corporation. When using an industry standard XML schema, you may wish to reuse only parts of it or extend it to meet your specific needs.

There are multiple ways to structure XML documents, just like there are a number of ways to design a database. Once you have decided on an XML structure (i.e. which elements and attributes you want to use) there is also more than one way to express that structure in an XML Schema definition.

Figure 4.10 provides a preview of common XML Schema constructs that we will be covering in Chapter 6. You will have the freedom to create a variety of different types based on XML Schema features that provide a series of built-in types and allow for the assembly of composite types commonly used to express complex structures and even entire business document structures.

Introductory XML Schema coverage is provided in the next chapter. Chapters 12 and 13 then delve into more advanced topics related to complex message design.

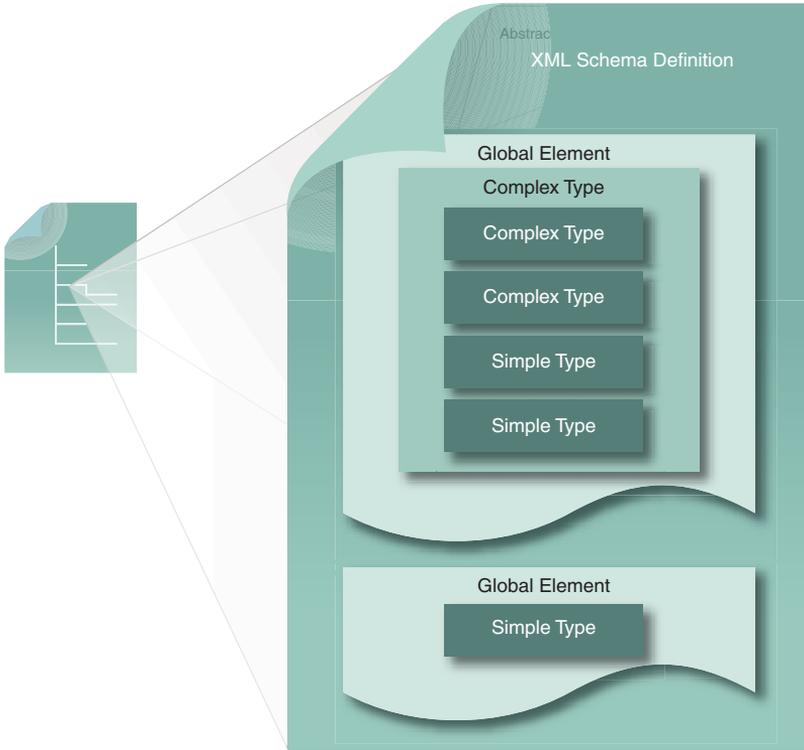


Figure 4.10

A logical view of a sample XML Schema definition structure comprised of different types that can then be used to define the structure of WSDL message definitions. Note the symbol on the left used to represent an XML Schema document.

WS-Policy Language

The WS-Policy standard specifies a policy framework specifically for Web services that enables behavior-related constraints and characteristics to be expressed as *machine-readable* metadata. This means that we can use this language to extend the base Web service contract provided by WSDL and XML Schema in order to add requirements or promote characteristics of the Web service that supplement the definitions in the abstract and concrete descriptions.

Figure 4.11 shows us that policy definition documents are just as modular as WSDL and XML Schema documents. As with XML Schema types, individual policy requirements (also referred to as policy assertions) can be combined or related to each other in order to form more complex composite policies (called policy expressions). You even have the ability to offer consumers a choice between two or more policies via the use of policy alternatives.

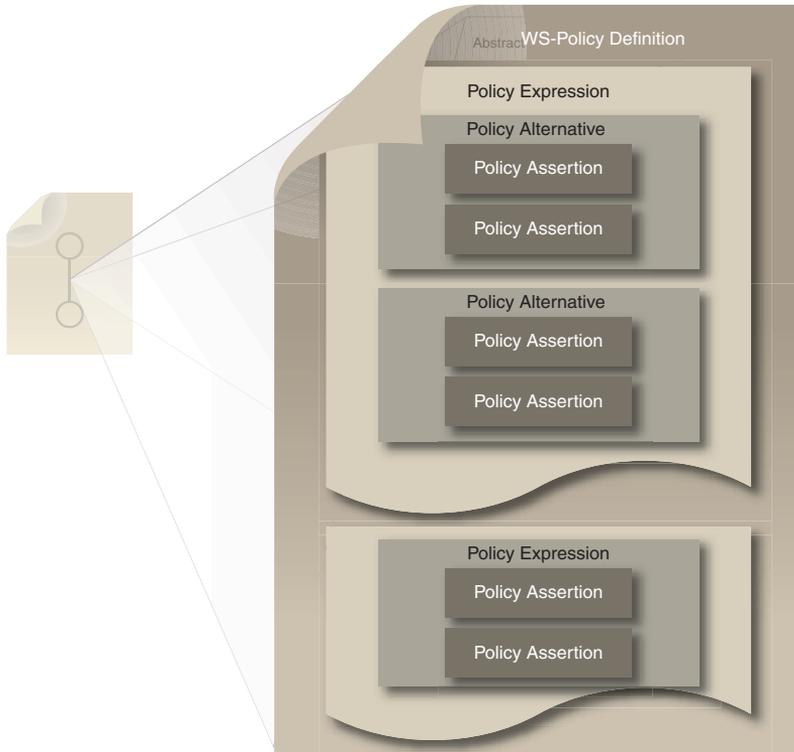


Figure 4.11
A sample WS-Policy definition with two composite policy expressions, each comprised of multiple policy assertions. Note the symbol on the left used to represent a policy document.

Another similarity that WS-Policy shares with XML Schema is how it relates to WSDL. Policies can also reside within or outside of the WSDL document. The related WS-PolicyAttachment language is used to associate (attach) policies to the various definitions in the abstract and concrete descriptions.

The WS-Policy and WS-PolicyAttachment languages are introduced in Chapter 10. More advanced topics are covered later in Chapters 16 and 17. Part of this coverage includes an exploration of how policy assertions are used by other languages to implement their features.

SOAP Language

SOAP provides a messaging format and mechanism that supports the standardized exchange of XML messages between software programs. SOAP has become the most prevalent messaging protocol used in conjunction with WSDL.

The SOAP language is used to express the structure of XML-based envelopes that host message definitions populated with actual data. Although at runtime SOAP language elements are usually auto-generated, SOAP is very much a technology related to the design-time creation of Web service contracts.

SOAP messages and message exchanges are fundamentally defined within the message and operation definitions of the WSDL document, and various SOAP language elements are further added to the concrete description in order to bind them to SOAP as a messaging protocol.

SOAP messages are most commonly:

- transmitted via HTTP
- described using XML Schema types
- associated with WSDL definitions in the concrete description

As shown in Figure 4.12, a standard SOAP message is divided into a set of parts. These parts are individually defined by XML Schema types via WSDL message definitions.

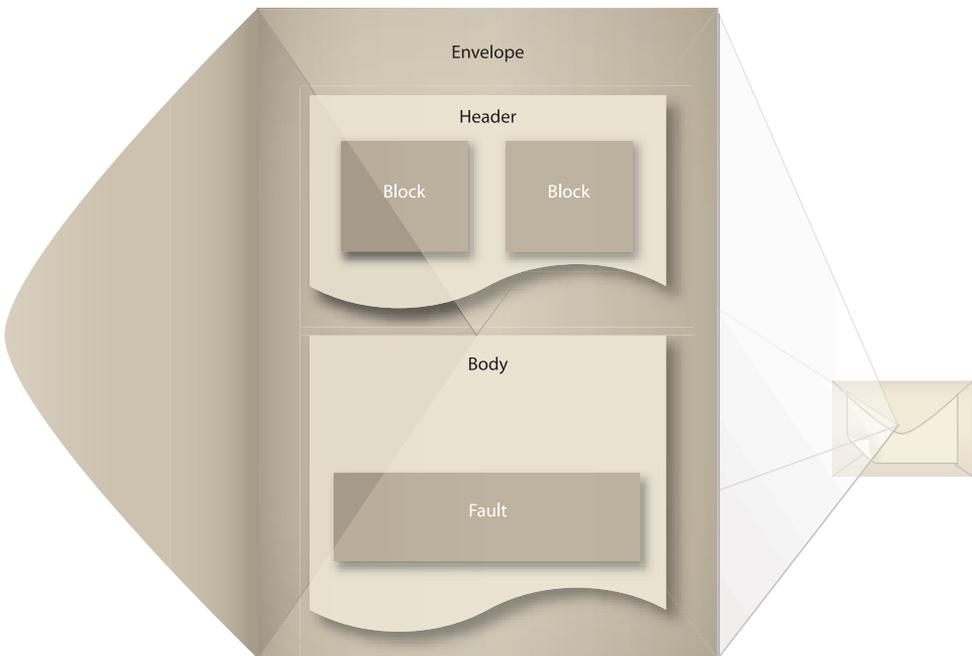


Figure 4.12

The SOAP message structure is defined by the SOAP standard and expressed with the SOAP language. Note the use of the symbol on the right to indicate a SOAP message in this book.

There are many supplemental technologies that extend SOAP messages via the use of SOAP header blocks. Many of these technologies are part of the WS-* specifications (also known as the second-generation Web services platform).

SOAP is introduced in Chapter 11. Advanced message design considerations are covered in Chapters 18 and 19 in relation to the WS-Addressing standard which provides fundamental routing and Web service instance identification header blocks.

Technologies for the Abstract Description

Figure 4.13 shows a high level mapping between the primary abstract description definitions and the related Web service technologies that can be used to create these definitions.

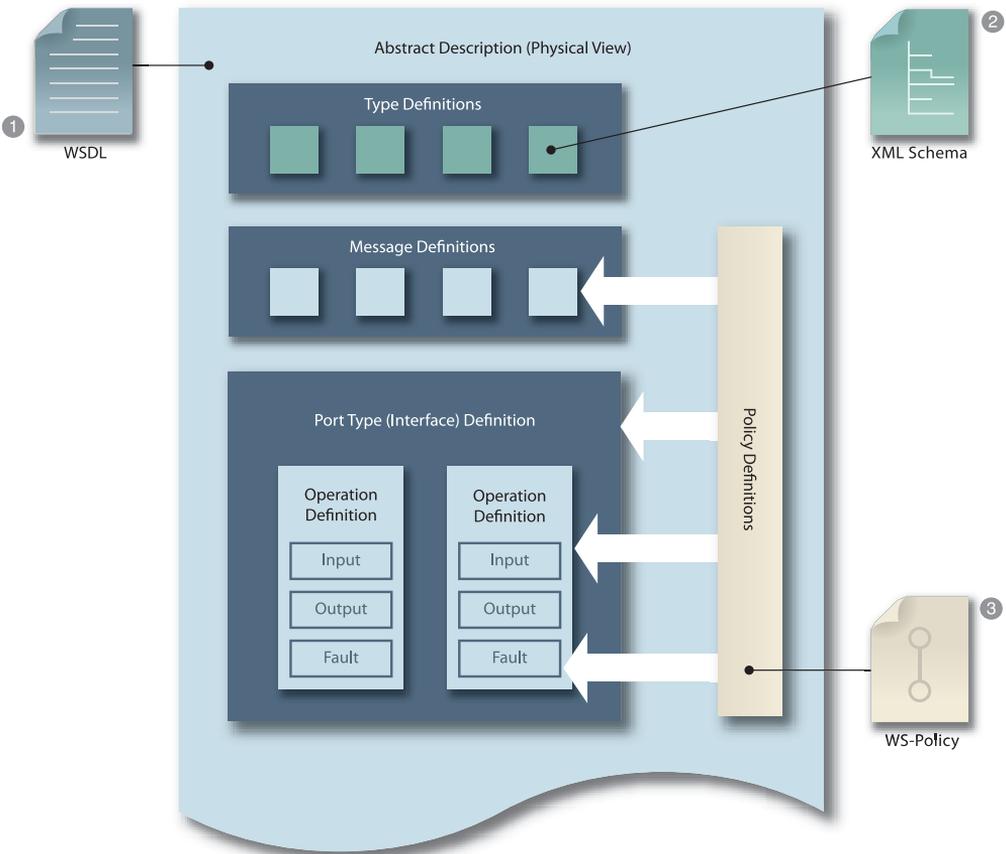


Figure 4.13
A high level mapping of the abstract description parts and related industry technologies.

What this figure tells us is that the overall structure of the abstract description (including the port type, operation, and message definitions) is created using WSDL (1).

The types part of a Web service contract is created collectively with WSDL and XML Schema (2) code in that the WSDL language provides a container construct wherein XML Schema code can be placed to express the various XML Schema types.

Finally, policy definitions can be separately created using the WS-Policy language (3). These policies can then be attached to the different WSDL definitions using the related WS-PolicyAttachment language.

Technologies for Concrete Descriptions

Let's now turn our attention to the technologies used to build a concrete description for a Web service. Figure 4.14 shows us the primary concrete description parts as they relate to three of the technology languages we just introduced.

In this depiction of a concrete description, we again establish how the WSDL (1) language is responsible for creating the overall structure, including the port type, operation and message binding definitions; as well as the service, port, and address definitions.

All of the constructs used to create these definitions can be further supplemented with special SOAP language statements (2) that allow the definitions to be directly associated with the SOAP messaging protocol.

And, as with the abstract description, all parts can be further supplemented with policy statements expressed using the WS-Policy language (3) as well as the related WS-PolicyAttachment language.

The WS-I Basic Profile

The Web Services Interoperability Organization (WS-I) develops specifications dedicated to establishing baseline interoperability between Web services using the technologies we just described.

The primary WS-I specification we'll be making reference to throughout this book is the Basic Profile, a document that essentially consists of a series of requirements and recommendations as to how Web services-related technology languages should be used together. Wherever appropriate, we'll be highlighting WS-I Basic Profile guidelines to supplement and provide further insight into contract design-related content.

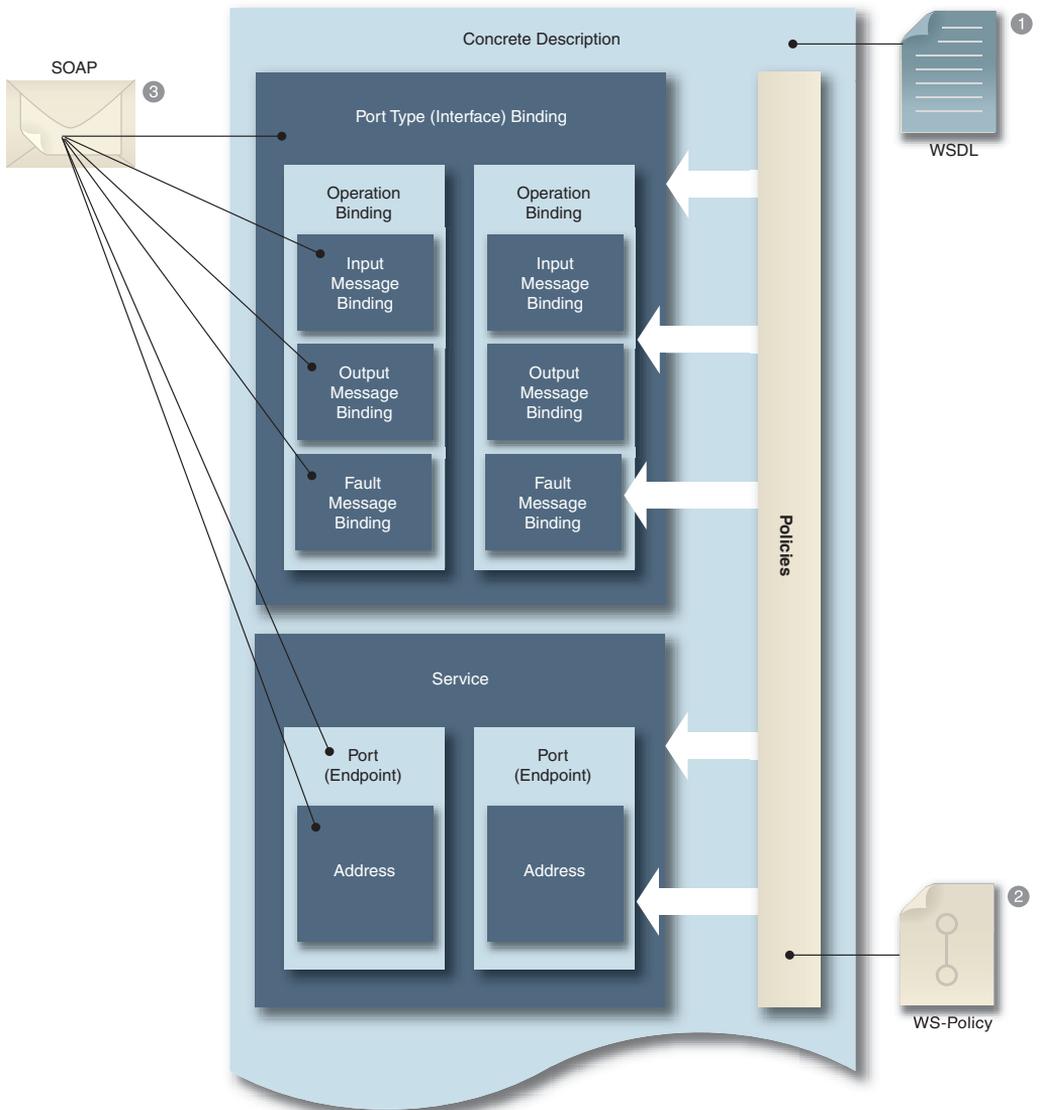


Figure 4.14
A look at the concrete description parts and the technology languages used to create them.

NOTE

In addition to the WS-I Profile, which is light on XML Schema guidelines, the W3C XML Schema Patterns for Databinding Working Group has documented a set of best practices for designing schemas for Web services. The XML Schema patterns are divided into two categories, basic and advanced, and there is a separate specification for each:

- Basic patterns use only the basic features of XML Schema and should already be correctly implemented by all toolkits.
- Advanced patterns are those that may still cause some tool interoperability problems today, but should be considered recommended patterns for the future.

To view the actual specifications for any of the languages covered in this book, visit www.soaspecs.com.

SUMMARY OF KEY POINTS

- The primary technology languages used to create the abstract description are WSDL, XML Schema, and WS-Policy.
- The primary technology languages used to create the concrete description are WSDL, SOAP, and WS-Policy.
- A WSDL document will be comprised of code originating from all of these languages.
- XML Schema and WS-Policy definitions can optionally be placed into stand-alone documents that are then referenced from within the WSDL definition.

4.4 Guidelines for Using Web Service Contract Technologies

It is not difficult to learn the mechanics of building Web service contracts. Once you get to know the technologies we've been discussing in this chapter, you will be able to author custom contracts with relative ease.

As we already established, every primary technology we cover in this book exists as an XML-based, industry-standard, markup language. Even though each serves a different purpose, all share a common syntax and many common conventions.

While learning these languages is a fundamental requirement to customizing your Web service contracts, it is not the most important skill, nor is it the most challenging part of creating effective contracts

Writing a Web service contract and *designing* an effective Web service contract for a given service as part of a federated service inventory are two different things. This is where technology features, design philosophies, and business requirements converge. This book primarily attempts to help you with the first two items.

Keep in mind that these technology languages are merely tools. You can just as easily use them to build a bad Web service contract as you could a good one. It is therefore helpful to remember that the decisions you make when designing the different parts of a contract will eventually impact the usability, flexibility, and governance burden of your services.

Before we proceed to the series of chapters that explore these technologies in detail, here are a few guidelines to keep in mind.

Auto-Generation Tools

Many development tools allow you to generate XML code for Web service contract languages. While this is useful for learning purposes, it may not be suitable for designing standardized Web service contracts in support of service-orientation.

The Standardized Service Contract design principle requires that service contracts be designed in accordance with existing design standards. The Service Loose Coupling principle further explains that contracts generated from underlying parts of a service's implementation can inherit negative forms of coupling that will make the service burdensome to own and govern.

These considerations also tie directly into the Canonical Schema and Canonical Expression design patterns that promote the use of design standards within the boundary of a service inventory.

Flexibility vs. Restrictiveness

One common dilemma in XML-based design, and indeed in software engineering in general, is the tension between flexibility and restrictiveness. At one end of the spectrum, a contract could be highly flexible in that it might allow just about any consumer to send it any type of content. The advantage of this approach is that you will never have to change the service contract when the message structure changes.

However, this technique also has its weaknesses. If you don't define a message structure, you can't pre-define validation logic in the contract layer, and you have no formal documentation of what your service can do. Applications that process the messages will have no idea of what to expect.

On the other hand, you could write a very rigid schema for your contract that must be modified every time there is the slightest change or addition to the message definition. Such contracts are brittle and have a ripple effect on the solutions that consume and form dependencies on the service. You may want this kind of rigidity if you are implementing a service that processes messages requiring strict precision (such as bank transactions), but such rigidity often does more harm than good.

With service-orientation there is the tendency to be less restrictive with technical contracts than with traditional object and component interfaces. The priorities advocated by the Service Abstraction principle and the Validation Abstraction design pattern support this tendency. However, in the end, the ideal usually lies somewhere in the middle. The key is to identify the areas of the contract that require flexibility and to then design them so that they allow the necessary variation.

Modularity and Reuse

Reuse of software saves time and money, and programs that are reused are often more robust and better designed all around. The individual parts that comprise Web service contracts are no exception.

As you will learn in the upcoming chapters, there is plenty of opportunity for reuse within Web service contract languages. Leveraging language features that allow for the creation of modules and provide include mechanisms can significantly streamline all parts of the service contract (and can further end up saving a lot of time that would be spent writing redundant code).

Then, of course, there is the notion of reusing parts of a contract across different contracts. This is where standardization-related patterns, such as Canonical Schema, Schema Centralization, and Policy Centralization, prove extremely valuable in not only getting more ROI out of the service contract code, but—more importantly—ensuring that these key parts are consistent across all the service contracts that establish a federated endpoint layer within a service inventory.

As with anything, though, overuse of something can lead to the opposite of the intended effect. XML language modules and centralization techniques need to be carefully chosen. Over-modularizing code within or across service contracts will inevitably lead to an inhibitive and complex architecture.

NOTE

Ultimately, one of the biggest challenges to achieving reuse is not related to technology, but to organizational governance. This is an on-going area that is addressed by all levels of service-orientation and SOA and will be covered in the upcoming book *SOA Governance*.

Clarity and Ease of Processing

With their hierarchical structure, larger-sized Web service contracts can quickly become unwieldy. Although they are intended to be read by machines, it does not mean that it doesn't matter if they are difficult to interpret by humans (especially considering that you may not want to rely on auto-generation tools to create and maintain them).

People from both the service provider and consumer sides will need to discover, understand, and work with these contracts, and any misunderstandings will introduce risk. Furthermore, overly complex contracts lead to increased governance effort and a higher cost of ownership.

Of course contracts have a job to do and sometimes this job requires that they be designed to a degree of complexity and sophistication. However, never lose sight of the fact that others will end up having to use (and perhaps evolve) what you are building today. Good contract design requires that you strive for clarity and avoid complexity or ambiguity wherever possible.

Here are some guidelines:

- *Consistent Design* – Use the same name for parts of the contract that mean the same thing, and different names for parts that are conceptually different. Developing a naming standard (as per the Canonical Expression pattern) may be the only way to guarantee that contracts are consistent in how they express themselves. This includes choosing names that are human-readable and, ideally, provide a business context. Also, when possible, use a consistent order and structure for elements that tend to be reused.
- *Limited Depth* – Some structural elements can be very useful, but you should not introduce unnecessary constraints or nested levels into the data structures that underlie your contract types. As per the Service Abstraction principle, try to define a balanced level of constraint granularity.

- *Good Documentation* – Simply providing a name and a structure for elements is not enough for the contract to be communicative to a wide range of project team members. Use annotations to further document elements and types, as per the Service Discoverability principle. (There are tools that will generate human-readable documentation from schema annotations.)

SUMMARY OF KEY POINTS

- The use of auto-generation tools can make it challenging to author standardized Web service contracts.
 - Granularity, reusability, and clarity are among the common Web service contract design considerations.
-

Index

A

- abstract attribute (complexType element), 329
- abstract descriptions (of service contracts), 168
 - in case study, 192-195
 - defined, 52
 - definitions element, 169-172
 - documentation element, 172-173
 - importing (case study), 400-403
 - message definitions, 181-186
 - operation definitions, 186-190
 - physical view of, 56-57
 - port type definitions, 190-191
 - primary parts of, 53-55, 61-63
 - structure of, 175-176
 - technologies for, 73-74
 - type definitions, 176-181
 - WSDL 2.0 changes, 226-235, 238
 - WSDL definition example, 218, 224, 235, 238
- abstract types (xsd:extension element), 329-331
- abstract WSDL descriptions, versioning and, 602
- acknowledgement messages, 342-343
- ACORD (Association for Cooperative Operations Research and Development), standardized namespaces from, 97
- Action element (WS-Addressing) pseudo schema, 730
- Action setting (SOAP) for message dispatch logic, 455-458
- Action values (WS-Addressing), 587
 - HTTP and, 589-590
 - reference table for, 590-591
 - as required, 587
 - WSDL and, 588-589
- ActionCon case study. *See* case study
- active intermediaries (SOAP), 294-296
- actors. *See* roles (SOAP)
- Add operation messages, 343-354
- address attribute (endpoint element), 235, 481
- address definitions, 61
- Address element (WS-Addressing) pseudo schema, 730
- agnostic, defined, 340
- agnostic logic, defined, 29
- aliases. *See* property aliases
- All element (WS-Policy) pseudo schema, 730
- all element (XML Schema) pseudo schema, 731
- allowed sets, 659
- alternative compatibility, 545-547
- anonymous address value, 557-558
- anonymous types, named types versus, 129-130
- any element (XML Schema) pseudo schema, 731
- #any value (element attribute), 446-448, 653-656
 - validation and, 449-450
 - XML Schema and, 450
- anyAttribute element (XML Schema) pseudo schema, 731

application/xml form (MIME serialization), 482

assertion compatibility, 544

assertion descriptions, 532-533

assertion profiles, 532-533

assertions. *See* policy assertions

associative rule (WS-Policy), 253-254

asynchronous data exchange

- bindings, 430-432
- guidelines for usage, 437
- polling, 429-430
- WS-Addressing, 432-436
- WS-BPEL 440-441

asynchrony, synchrony versus, 427-429

attaching policies to WSDL definitions, 257

- case study, 266-269
- embedded attachments, 263-265
- policy attachment points, 258
- policy subjects, 258-262
- wsp:PolicyReference element, 263

attachment points. *See* policy attachment points

attribute declarations. *See also* declarations

- code example, 125
- content models and, 141
- global versus local, 126-128

attribute element, child of complexType (XML Schema) pseudo schema, 731

attribute element, child of schema (XML Schema) pseudo schema, 731

attribute wildcards, 313

attributeFormDefault attribute (xsd:schema element), 149

attributes. *See also* elements; types

- custom attributes, extending policy assertions with, 528-529
- described, 83
- elements versus, 124-126
- namespaces and, 89

authentication parameters for HTTP

- binding, 484

auto-generation tools for service contracts, 77

B

backwards compatibility

- in Flexible versioning strategy, 614
- in Loose versioning strategy, 614-615
- versioning and, 603-605

base types for user-defined types, 133

binding

- to HTTP, 472
 - with WSDL 1.1, 474-480
 - with WSDL 2.0, 480-484
- WSDL and WS-Addressing, 580-586

binding definitions, 198-215

- defined, 59
- extensibility elements, 201-202
- inheritance, 202-203
- literal versus encoded message types, 209-212
- relationships, 61
 - for SOAP 1.2, 212-215
- SOAP extensibility elements, 208-209
- soapAction attribute (soap11:operation element), 203-204
- style attribute (soap11:binding element), 204-207

binding element (WSDL 1.1), 198-200

- reusability, 200
- transport attribute, 202

binding element (WSDL 1.1) pseudo schema, 731

binding element (WSDL 2.0) pseudo schema, 731

binding extensions (SOAP), 418-420

binding portion of concrete descriptions, primary parts of, 58-59

bindings (asynchronous data exchange), 430-432

body (of message), defined, 64

Body element (SOAP 1.1) pseudo schema, 732

Body element (SOAP 1.2) pseudo schema, 732

body element (SOAP), 208

body type definitions

- in body (of message), 64
- defined, 55

books, related, 5-6. *See also* Web sites

buckets, extension, 316-318
 built-in simple types, 130-132
 business document messages, 341-342
 business-centric services. *See* services,
 business-centric

C

Candidate Recommendation (standards
 development process), 727
 Canonical Expression design pattern, 77, 79,
 133, 753
 Canonical Policy Vocabulary design pattern,
 531-532
 Canonical Schema design pattern, 44, 77-78,
 370, 531, 753
 Canonical Transport Protocol design
 pattern, 420
 Canonical Versioning design pattern, 44,
 612, 754
 capability. *See* service capability
 capability granularity, 35
 case study
 abstract description versioning, 621
 abstract descriptions, 192-195
 background, 18-20, 119-121, 273-274
 binding definition for SOAP 1.2,
 214-215
 common type libraries, 362-363
 complete WSDL definition,
 218-235, 238
 conclusion of, 722-723
 custom header blocks, 469-471
 custom MEPs, 425-426
 custom policy assertions, 523-527
 definitions element (WSDL 1.1), 174
 document-literal message type,
 210-212
 documentation element
 (WSDL 1.1), 174
 embedded XML schema types, 178-180
 external XML schema types, 180-181
 generic versus specific elements,
 321-323
 HTTP binding with WSDL 1.1, 477-480
 HTTP binding with WSDL 2.0, 483
 ignorable policy assertions, 509-510

importing abstract descriptions,
 400-403
 importing schemas into WSDL
 documents, 409-415
 industry schemas
 reusing types, 381-384
 UBL structure, 387-392
 MAP headers, 567-568
 message definitions in abstract
 descriptions, 183-186
 message dispatch logic, 451-452
 namespace prefixes in, 748
 nested policy assertions, 501-505
 operation definitions in abstract
 descriptions, 190
 parameterized policy assertions,
 501-505
 policy attachments, 266-269
 policy definitions, 494-496
 schema definition, 152-154, 163-166
 schema reusability (xsd:include
 element), 356-360
 self-descriptive message design,
 585-586
 service and port elements, 217-218
 SOAP intermediaries, 280-281
 soap:role attribute, 286-287
 structural elements, 155-161
 style, 18
 WS-Addressing messaging rules,
 576-580
 WS-Addressing policy assertions, 595
 WS-BPEL extensibility elements,
 443-444
 xsd:import element, 363-369
 centralized policies. *See* policy definitions
 chapters, described, 6-14
 child elements, benefits of, 125-126
 choice element (XML Schema) pseudo
 schema, 732
 chorded circle symbol, explained, 28, 32
 class inheritance, emulating, 327-333
 coarse-grained constraints, 602-603
 Code element (SOAP 1.2) pseudo
 schema, 732

code examples

- abstract attribute (complexType element), 329
- abstract description versioning in case study, 621
- abstract descriptions, 195
- acknowledgement message, 343
- active intermediaries (SOAP), 295
- adding fault messages to operations (versioning), 641-642
- adding new operations (versioning), 624
- address attribute (endpoint element), 481
- anonymous address value, 558
- #any attribute value (WSDL 2.0), 653-656
- #any value (element attribute), 446-447
- associative rule (WS-Policy), 253-254
- asynchronous data exchange with WS-BPEL 441
- attribute declaration, 125
- attributes for SOAP extensibility elements, 209
- backwards compatibility, 604-605
- binding construct for header blocks, 469
- binding definition for SOAP 1.2, 213-215
- binding element (WSDL 1.1), 199
- changing MEP of operations (versioning), 637, 639
- child elements for
 - wsp:PolicyAttachment element, 490
- common type libraries, 363
- commutative rule (WS-Policy), 253
- compact form, 535
- compatible changes, 607
- compatible policy assertions, 547
- complete WSDL definition, 224, 238
- complex types, 138, 330
- concrete WSDL description, 582
- consumer policy expression, 543
- container elements, 162
- custom header blocks in case study, 470-471
- custom MEPs in case study, 426
- custom policy assertion attributes, 528-529
- custom policy assertions, 247-248
- custom:LogMessage policy assertion, 522-524
- decimal precision and scale in user-defined simple types, 135
- default namespace, 146
- defined sets versus unknown sets (versioning), 660
- definitions element (WSDL 1.1), 169, 171, 174
- desc element, 377
- distributive rule (WS-Policy), 254-255
- document binding style, 206
- document instance based on complex types, 336
- document-literal message type, 211-212
- documentation element (WSDL 1.1), 173-174, 702
- domain policy definition in case study, 494
- effective policy, 541
- element attribute (WSDL part element), 186
- element declaration, 124
- element-only content complex type, 140
- elementFormDefault attribute, 148
- embedded attachments, 264
- embedded XML schema types, 180
- embedded, ignorable policy assertions, 514
- embedded, required and ignorable policy assertions, 515
- embedding EPRs in WSDL documents, 584
- empty content complex type, 141
- empty policy operators, 256
- endpoint policy subject, 260
- EndpointName attribute, 582
- enumerated list of values, 134
- extends attribute (WSDL interface element), 228
- extensibility elements of binding definitions, 202

- extension buckets, 317
- external XML schema types, 181
- externally referenced elements (versioning), 671
- fault messages, 232-233, 579
- fine-grained and coarse-grained constraints, 602
- form attribute (element element), 150-151
- forwards compatibility, 606
- generic elements, 320, 322
- global element declaration, 127
- global element declarations for header-related types, 467-468
- global policy definition in case study, 496
- GS1 ElectronicGamePlayer-Information.xsd schema, 383
- header-related global elements, assigning to element attributes, 468
- HTTP binding with WSDL 1.1 in case study, 479
- HTTP binding with WSDL 2.0 in case study, 483
- HTTP request message, 479
- HTTP response message, 480
- HTTP-specific attribute values, 481
- httpbind:address extensibility element, 475
- http:binding and httpbind:operation extensibility elements, 474
- idempotent rule (WS-Policy), 252
- identity constraints, 374
- ignorable policy assertions, 505
 - in case study*, 510
 - failed attempt*, 511
 - schema definition for*, 521
 - with target namespace reference*, 521
 - targeted at consumers*, 509
- ignorable termination policy assertions, 702-703
- import element (WSDL 1.1) with high visibility, 407
- import element (WSDL 2.0) with constrained visibility, 407
- import element (WSDL 2.0) with high visibility, 408
- importing abstract descriptions in case study, 401-403
- importing multiple schemas, 412
- importing schemas into WSDL documents in case study, 415
- In-Out MEPs, 234
- include element (WSDL 2.0) with high visibility, 408
- include element (WSDL), 403
- incompatible changes, 609
- industry schemas, 381-384
- interface element (WSDL 2.0), 227
- interface element with operation element, 228
- interface inheritance, 415
- intersection algorithm, 546
- length restrictions on user-defined simple types, 135
- lineItem complex type, 312-313
- mandatory termination policy assertions, 703-704
- many-to-many relationships, 372-374
- MAP headers in case study, 568
- message dispatch logic in case study, 452
- message element (WSDL 1.1), 181
- message element with part element, 183-184
- message policy subject, 261
- messages from endpoints in case study, 577
- MIME serialization in HTTP bindings, 476
- mixed content types, 378-379
- mustRetain attribute, 710
- mustUnderstand attribute, 696-697
- namespace attribute (xsd:any wildcard), 685
- namespace prefixes and global element declarations, 145-146
- nested policy assertions, 499, 503
- none address value, 558
- normal form, 534, 537
- normalization of merged policies, 541
- operation element (WSDL 1.1), 186, 190
- operation policy subject, 261
- optional attributes, 347

- optional elements (versioning), 669
- optional policy assertions, 250
- parameter-centric Update operation message, 350
- parameterized policy assertions, 500, 505, 511
- partial validation, 711
- policy alternative provided by consumer program designer, 546
- policy alternative provided by Web service contract, 545
- policy alternatives at attachment points, 539-540
- policy assertions, 244-246
- policy attachments in case study, 269
- policy expressions, 246, 502
- policy versioning, 690-694
- poNumber XML Schema element declaration, 210
- port element with address extensibility element, 216
- port types
 - adding version identifiers to*, 643
 - multiple port types*, 646
 - version numbers in namespace prefixes*, 650
- portType element (WSDL 1.1), 191
- processContents attribute (xsd:any wildcard), 687
- property aliases, 443
- property declaration, 442
- range of values for user-defined simple types, 135
- reducing granularity levels, 653
- referencing elements (versioning), 670
- regular expression patterns for user-defined simple types, 136
- relationship elements, 376
- removing operations (versioning), 632-634
- renaming imported schemas (versioning), 627
- renaming operations (versioning), 626, 629-630
- reply messages in case study, 578
- rep:Representation header block, 290
- response SOAP message, 573
- retaining unknown elements, 708
- reusing complex message types, 453-454
- Robust In-Only MEP 424
- schema components (versioning)
 - adding*, 669-670
 - adding new*, 665-666, 676-677
 - modifying constraint of*, 668, 673-675
 - removing*, 666-667, 672, 680-682
- schema defining XML document structure, 123
- schema reusability in case study, 366-368
- schemas for replacement elements, 679
- service and port elements, 217-218
- service element hosting endpoint element, 235
- service element with port element, 216
- service policy subject, 259
- simple content complex type, 139
- SOAP Action setting, 455-457
- SOAP message document, 206
- SOAP messages with/without well-formed XML, 447-448
- SOAP over HTTP anonymous value, 594
- SOAP over SMTP binding, 432
- soapAction attribute (soap11:operation element), 204
- soap:Body element, 276
- soap:DataEncodingUnknown value (soap:Code element), 302
- soap:Detail element, 303
- soap:Envelope element, 275
- soap:Fault element, 278
- soap:Header element, 277
- soap:mustUnderstand attribute, 288, 698
- soap:MustUnderstand value (soap:Code element), 300
- soap:Receiver value (soap:Code element), 302
- soap:role attribute, 284-285, 287
- soap:Sender value (soap:Code element), 301

- soap:VersionMismatch value (soap:Code element), 299
- specific elements, 319
- sp:TransportBinding and sp:SymmetricBinding policy assertions, 526
- structural elements, 157-161
- style attribute (soap11:binding element), 205
- styleDefault attribute (WSDL interface element), 229
- targetNamespace attribute, 144
- termination policy assertions for individual operations, 705-706
- tns: namespace prefix, 145
- transport attribute (WSDL binding element), 202
- type composition, 336
- types element (WSDL 1.1), 176-177
- UBL 2.0 schema, 386
 - case study*, 389
 - customized elements in*, 391-392
- UBL ExtensionContentType complex type and ExtensionContent global element, 386
- UPA rule violation (versioning), 661
- Update operation messages, 347
- user-defined simple type with length restriction, 133
- validation by projection, 713
- versioning concrete descriptions, 651
- versioning schemas, base example for, 662-663
- wrapper elements for wildcards, 678
- wrapping policy definitions in WSDL definitions, 492-493
- WS-Addressing and asynchronous data exchange, 436
- WS-Addressing policy assertions in case study, 595
- wsam:Addressing policy assertion, 459, 518, 520, 593
- wsam:AnonymousResponse policy assertion, 593
- wsam:NonAnonymousResponse policy assertion, 594
- wsaw:Action attribute, 459-460
- wsa:Action element, WSDL and, 588-589
- wsa:Action element, 565
- wsa:Action header, 460
- wsa:Address element, 557
- wsa:EndpointReference element, 557
- wsa:FaultTo element, 564
- wsa:From element, 563
- wsa:MessageId element, 565
- wsa:Metadata element, 560, 581
- wsa:ReferenceParameters element, 559, 570-571
- wsa:RelatesTo element, 566
- wsa:ReplyTo element, 563
- wsa:To element, 562
- wscor:CoordinationContext header block, 292
- WSDL definition version identifiers, 619
- wsdli:wsdLocation attribute (wsa:Metadata element), 583
- wsp:All element, 249
- wsp:ExactlyOne element, 249
- wsp:Optional and wsp:Ignorable attributes, 507-508
- wsp:Optional attribute, 506
- wsp:Policy element, 491
- wsp:PolicyAttachment element, 489
- wsp:PolicyReference element, 263, 265, 492
- wsp:PolicyURIs attribute, 491
- wsrn:Sequence header block, 291
- XML document instance, 122
- XML Schema for complete document, 342
- XML schema for derived console type, 331
- xsd:all element, 325
- xsd:any element, 311
- xsd:anyAttribute element, 313
- xsd:anyType type, 318
- xsd:choice element, 324
- xsd:extension element, 330
- xsd:group element, 334
- xsd:include element, 355, 358-360
- xsd:schema element, 185
- xsd:sequence element, 326

- color, in figures, 14
- comment notation in HTML, 173. *See also*
 - documentation
- common type libraries, 361-363
- commutative rule (WS-Policy), 252-253
- compact form, defined, 534
- compatibility
 - backwards compatibility, 614-615
 - forwards compatibility
 - in Loose versioning strategy*, 614-615
 - WSDL definitions and*, 652-656
 - lack of (Strict versioning strategy), 613-614
 - version numbers and, 610-611
 - versioning and, 603
 - backwards compatibility*, 603-605
 - compatible changes*, 607-608
 - forwards compatibility*, 605-606
 - incompatible changes*, 608-610
- compatibility guarantee, defined, 611
- Compatible Change design pattern, 608, 754
- compatible changes, versioning and, 607-608
- complex service compositions, 30
- complex types, 137-141
 - as abstract, 329
 - defined, 129
 - element-only content complex types, 139-141
 - empty content complex types, 141
 - reusing, 453-454
 - simple content complex types, 139
- complexContent element (XML Schema)
 - pseudo schema, 732
- complexity in versioning strategy
 - comparison, 616
- complexType element
 - abstract attribute, 329
 - child of element (XML Schema)
 - pseudo schema, 733
 - child of schema (XML Schema) pseudo schema, 732
- components
 - adding new
 - Flexible versioning strategy*, 668-671
 - Loose versioning strategy*, 676-679
 - Strict versioning strategy*, 665-666
 - defined, 658
 - modifying constraint of
 - Flexible versioning strategy*, 673-657
 - Loose versioning strategy*, 683-687
 - Strict versioning strategy*, 667-668
 - removing
 - Flexible versioning strategy*, 671
 - Loose versioning strategy*, 680-682
 - Strict versioning strategy*, 666
 - renaming
 - Flexible versioning strategy*, 671-672
 - Loose versioning strategy*, 683
 - Strict versioning strategy*, 666-667
- composite policies, 248
 - wsp:All element, 249
 - wsp:ExactlyOne element, 248-249
 - wsp:optional attribute, 250-251
- compositions. *See* service compositions; type composition
- concrete descriptions (of service contracts)
 - binding definitions, 198-215
 - defined, 52
 - definitions element, 169-172
 - documentation element, 172-173
 - importing abstract descriptions (case study), 400-403
 - port definitions, 215-218
 - primary parts of
 - binding portion*, 58-59
 - relationships among*, 61-63
 - service portion*, 60-61
 - service definitions, 215-218
 - structure of, 198-199
 - technologies for, 74-75
 - versioning, 650-652
 - WSDL 2.0 changes, 226-235, 238
 - WSDL definition example, 218-235, 238
- Concurrent Contracts design pattern, 44, 128, 191, 339, 417, 512-516, 755

- constraint granularity, 310
 - balancing, 79
 - content model groups, 323-326
 - defined, 36
 - extension buckets, 316-318
 - fine-grained versus coarse-grained constraints, 602-603
 - generic versus specific elements, 319-323
 - wildcard elements, 310-316
- constraints of schema components,
 - modifying
 - Flexible versioning strategy, 673-675
 - Loose versioning strategy, 683-687
 - Strict versioning strategy, 667-668
- constructs, defined, 66
- consumer-service policy compatibility, 542-543
 - alternative compatibility, 545-547
 - assertion compatibility, 544
 - levels of, 543-544
- consumers
 - partial validation, 711-713
 - targeting with ignorable policy assertions, 508-509
- container elements, 161-163
- containment relationships, 371
- content model groups, 323-326
- content models
 - attribute declarations and, 141
 - defined, 129
 - structural elements and, 160
- content sets
 - explained, 659-660
 - types of, 707
- content types, 137-141
- Contract Centralization design pattern, 44, 755
- Contract Denormalization design pattern, 44, 128, 339, 350, 756
- contract-level validation, bypassing, 449-450
- contract-to-functional coupling, 41
- contract-to-implementation coupling, 41, 511
- contract-to-logic coupling, 41
- contract-to-technology coupling, 42
- contracts. *See* service contracts
- cookies, HTTP binding and, 484
- correlation (WS-BPEL), 441-443
- coupling. *See also* Decoupled Contract design pattern; Service Loose Coupling design principle
 - contract-to-functional coupling, 41
 - contract-to-implementation coupling, 41, 511
 - contract-to-logic coupling, 41
 - contract-to-technology coupling, 42
 - logic-to-contract coupling, 42
 - types of, 41-43
- CRUD message types, 337-338
 - acknowledgement messages, 342-343
 - Add operation messages, 343-345
 - as agnostic, 339-340
 - business document messages, 341-342
 - chattiness of, 338-339
 - Delete operation messages, 350-351
 - document-centric Update operation messages, 346-348
 - Get operation messages, 345-346
 - parameter-centric Update operation messages, 348-350
- custom attributes, extending policy assertions with, 528-529
- custom extensibility elements, 420-421
- custom header blocks, 466-469
 - case study, 469-471
 - in WSDL 2.0, 471
- custom header faults, 466-469
- custom MEPs, 425-426
- custom namespaces. *See* target namespaces
- custom policy assertions, 247-248, 518
 - Canonical Policy Vocabulary design pattern, 531-532
 - case study, 523-527
 - checklist, 529-530
 - description documents, 532-533
 - processing logic for, 521-523
 - schema design for, 518-521
- custom transports for SOAP bindings, 419-420
- custom XML schema namespace, 114-115

D

- data granularity, defined, 36
- data types. *See* types
- decimal precision and scale for
 - user-defined simple types, 135
- declarations
 - attribute declarations (code example), 125
 - element declarations (code example), 124
 - global, 123, 126-128
 - local, 123, 126-128
- declaring namespace prefixes, 102-103
- Decomposed Capability design pattern, 45, 756
- Decoupled Contract design pattern, 45-47, 757
- default namespaces, 87
 - code example, 146
 - combined with no namespace, 110
 - defined, 92
 - explained, 107-109
 - namespace prefixes combined with, 108
- defined content set, 707
- defined sets, defined, 659
- definitions, defined, 67
- definitions element (WSDL 1.1), 169-172, 406, 733
- Definitive XML Schema*, 5
- DELETE method (HTTP), 473
- Delete operation messages, 350-351
- dependencies. *See* compatibility
- description element (WSDL 2.0), 170, 733
- deserialization, defined, 476
- design patterns. *See also* www.soapatterns.org
 - Canonical Expression, 77, 79, 133, 753
 - Canonical Policy Vocabulary, 531-532
 - Canonical Schema, 44, 77-78, 370, 531, 753
 - Canonical Transport Protocol, 420
 - Canonical Versioning, 44, 612, 754
 - Compatible Change, 608, 754
 - Concurrent Contract, 512-516
 - Concurrent Contracts, 44, 128, 191, 339, 417, 755
 - Contract Centralization, 44, 755
 - Contract Denormalization, 44, 128, 339, 350, 756
 - Decomposed Capability, 45, 756
 - Decoupled Contract, 45-47, 757
 - Distributed Capability, 45, 757
 - Dual Protocols, 420
 - Legacy Wrapper, 337
 - Logic Centralization, 337
 - Messaging Metadata, 45, 758
 - Partial Validation, 45, 711-712, 758
 - Policy Centralization, 45, 78, 487-488, 694, 759. *See also* policy definitions
 - Proxy Capability, 45, 405, 759
 - Redundant Implementation, 405
 - Rules Centralization, 344
 - Schema Centralization, 45, 78, 354, 664, 699-700, 760
 - service contracts and, 44-47
 - Service Instance Routing, 552
 - Service Messaging, 45, 760
 - State Messaging, 553
 - Termination Notification, 45, 701, 761
 - Validation Abstraction, 45, 78, 132, 344, 761
 - Version Identification, 45, 610, 762
- design principles
 - Service Abstraction, 39, 43, 78-79, 132, 230, 511
 - Service Autonomy, 39
 - Service Composability, 30, 40, 44
 - service contracts and, 39-44
 - Service Discoverability, 40, 43, 80, 173, 230
 - Service Loose Coupling, 39-43, 77, 371, 510-511
 - Service Reusability, 29, 39, 43, 340
 - Service Statelessness, 40, 463
 - Standardized Service Contract, 31, 35, 39-41, 77, 130, 133
- designing messages. *See also* schemas; WS-Addressing
 - CRUD message types, 337-338
 - acknowledgement messages*, 342-343
 - Add operation messages*, 343-345
 - as agnostic*, 339-340

- business document messages,*
 - 341-342
- chattiness of,* 338-339
- Delete operation messages,* 350-351
- document-centric Update operation messages,* 346-348
- Get operation messages,* 345-346
- parameter-centric Update operation messages,* 348-350
- for task services, 351
- designing namespaces, 90
- detail element (SOAP 1.1) pseudo schema, 733
- Detail element (SOAP 1.2) pseudo schema, 733
- development tools, auto-generating service contracts, 77
- diagrams. *See* figures
- directionality (of SOAP messages), 570-573
- discoverability. *See* Service Discoverability design principle
- Distributed Capability design pattern, 45, 757
- distributive rule (WS-Policy), 254-255
- document binding style, 205
- document-centric Update operation messages, 346-348
- document-encoded message type, 210
- document-literal message type, 209-212
- documentation
 - of custom policy assertions, 532-533
 - importance of, 80
- documentation element (WSDL), 172-173, 618, 701
- domain policy definitions, 487
- domain service inventories, 31
- Dual Protocols design pattern, 420

E

- effective policies, 262, 541
- element attribute
 - #any value, 446-450
 - WSDL part element, 182, 185
- element declarations (code example), 124. *See also* declarations

- element element
 - child of schema (XML Schema) pseudo schema, 733-734
 - child of sequence/choice/all (XML Schema) pseudo schema, 734
 - form attribute, 149-151
- element wildcards, 311
- element-only content complex types, 139-141
- elementFormDefault attribute (xsd:schema element), 148
- elements. *See also* attributes; types
 - attributes versus, 124-126
 - child elements, benefits of, 125-126
 - defined, 67
 - explained, 82-83, 122-123
 - externally referenced, 671
 - generic versus specific elements, 319-323
 - global versus local declarations, 126-128
 - granularity, 128
 - namespace prefixes, 102-106
 - namespaces and, 84-88, 109-111
 - referencing, 670
 - replacement elements, 676-679
 - structural elements, 154-163
 - types versus, 124
 - wildcard elements, 310-316
 - wrapper elements for wildcards, 677-678
- elements declarations, global versus local, 126-128
- embedded attachments, 263-265
- embedded XML schema types, 177-180
- embedding
 - EPRs in WSDL documents, 583-584
 - WSDL references in EPRs, 581-583
- empty content complex types, 141
- empty policy operators, 256
- encoded message types, literal message types versus, 209-212
- encoding, 98-99
- encoding types, 178

- endpoint definitions
 - defined, 60
 - relationships, 61
 - in WSDL 2.0, 235
 - endpoint element (WSDL 2.0), 217, 734
 - address attribute, 235, 481
 - endpoint policy subject, 259-260
 - endpoint references. *See* EPRs
 - EndpointReference element (WS-Addressing) pseudo schema, 734
 - endpoints, 216
 - multiple policy endpoints, versioning and, 693-694
 - replying to messages from, 575
 - entity relationships, 370
 - entity service model, 28
 - enumerated lists of values for user-defined simple types, 134
 - enumeration element (XML Schema) pseudo schema, 734
 - Envelope element (SOAP 1.1) pseudo schema, 734-735
 - envelopes, messages versus, 275. *See also* message envelopes
 - EPRs (endpoint references), 551-553
 - MAPs and, 554-555
 - wsa:Address element, 557-558
 - wsa:EndpointReference element, 556-557
 - wsa:Metadata element, 560
 - wsa:ReferenceParameters element, 559-560
 - WSDL binding, 580
 - case study*, 585-586
 - embedding EPRs in WSDL documents*, 583-584
 - embedding WSDL references in EPRs*, 581-583
 - equivalence of policy operators, 256
 - error codes for HTTP faults, 484
 - error messages. *See* fault messages
 - ESB (Enterprise Service Bus), policy centralization and, 497
 - ESB Architecture for SOA*, 6
 - ExactlyOne element (WS-Policy) pseudo schema, 735
 - examples. *See* case study; code examples
 - expanded names, defined, 91
 - extending policy assertions with custom attributes, 528-529
 - extends attribute (interface element), 227-228, 415-416
 - extensibility
 - container elements and, 162
 - with interface inheritance, 415-417
 - structural elements and, 160
 - of WSDL
 - custom extensibility elements*, 420-421
 - SOAP binding extensions*, 418-420
 - wSDL:required attribute*, 420-421
 - extensibility attributes (WSDL 2.0), wsdli and wsdli, 463-465
 - extensibility elements
 - of binding definitions, 201-202
 - custom elements, 420-421
 - SOAP, 208-209
 - WS-BPEL, 439-440, 443-444
 - extension buckets, 316-318
 - extension element, child of complexContent (XML Schema) pseudo schema, 735
 - extension element, child of simpleContent (XML Schema) pseudo schema, 735
 - extension points in industry schemas, 384-392
 - external policy definitions. *See* policy definitions
 - external XML schema types, 177, 180-181
 - externally referenced elements, 671
 - extreme loose coupling in WSDL 2.0 446
 - #any value (element attribute), 446-450
 - validation, 449-450
- ## F
- facets, 133
 - fault (of message), defined, 64
 - fault element (SOAP), 208, 735-736
 - fault element (WSDL 2.0), 231-233
 - fault element of binding element (WSDL 2.0) pseudo schema, 736
 - fault element of interface element (WSDL 2.0) pseudo schema, 736
 - fault element, child of operation (WSDL 1.1), 187

fault message definitions, 55
 fault messages (SOAP), 297

- adding to operations, 639-642
- SOAP 1.1 and, 1.2 compared, 304
- soap:Code element, 298-302
- soap:Detail element, 303
- soap:Fault element, 278
- soap:Node element, 297
- soap:Reason element, 297
- soap:Role element, 298
- soap:Text element, 298
- soap:Value element, 298
- WSDL fault messages versus, 303-304

 fault messages (WSDL), SOAP fault messages versus, 303-304
 faultcode element (SOAP 1.1) pseudo schema, 736
 faults

- header faults, 466-469
- HTTP, error codes, 484
- SOAP faults, WS-Addressing and, 592
- WS-BPEL restrictions on, 439

 faultstring element (SOAP 1.1) pseudo schema, 736
 FaultTo element (WS-Addressing) pseudo schema, 736
 federation, 38
 field element (XML Schema) pseudo schema, 737
 figures

- color. *See* color, in figures
- poster. *See* poster Web site
- symbols. *See* symbols
- Visio Stencil. *See* Visio Stencil

 fine-grained constraints, 602-603
 flexibility. *See* message flexibility
 Flexible versioning strategy

- explained, 612, 614
- multiple port types, 643-646
- operations
 - adding fault messages to, 642
 - adding new, 622-624
 - changing MEP of, 637-639
 - removing, 630-635
 - renaming, 627-629

schema components

- adding new, 668-671
- modifying constraint of, 673-675
- removing, 671
- renaming, 671-672

 form attribute (element element), 149-151
 format conventions for target namespaces, 147
 formatting patterns for user-defined simple types, 136-137
 forwarding intermediaries (SOAP), 294
 forwards compatibility

- in Loose versioning strategy, 614-615
- versioning and, 605-606
- WSDL definitions and, 652-656

 fractionDigits element (XML Schema) pseudo schema, 737
 From element (WS-Addressing) pseudo schema, 737
G
 generic elements, specific elements versus, 319-323
 GET method (HTTP), 473
 Get operation messages, 345-346
 global declarations, 123, 126-128, 145-146
 global policy definitions, 487
 global types. *See* named types
 glossary Web site, 5, 15, 37, 339
 governance

- in reusability, 79
- in versioning strategy comparison, 616

 granularity

- constraint granularity, 310
 - balancing, 79
 - content model groups, 323-326
 - extension buckets, 316-318
 - generic versus specific elements, 319-323
 - wildcard elements, 310-316
- element granularity, 128
- levels of, 35-36
- of operations, reducing, 652-653

 group element, child of schema (XML Schema) pseudo schema, 737

group element, descendant of complexType (XML Schema) pseudo schema, 737
 groups. *See* content model groups
 GS1 industry schema, 381-384

H

header (of message), defined, 64-65
 header blocks (SOAP), 208, 283, 570-573
 custom header blocks, 466-471
 defined, 64, 275
 examples of, 289-292
 soap:Header element, 276-277
 soap:mustUnderstand attribute, 287-289
 soap:relay attribute, 289
 soap:role attribute, 283-287
 Header element (SOAP 1.1) pseudo schema, 737
 Header element (SOAP 1.2) pseudo schema, 737
 Header element (SOAP), 208
 header faults, custom, 466-469
 header type definitions, 55-57
 headerfault element (SOAP), 208
 headerfault type definitions, 55-57
 headers (HTTP). *See also* header blocks (SOAP)
 declaring, 484
 with SOAP Action setting, 458
 HTML, 66
 comment notation, 173
 tags in narrative content, 377-379
 HTTP, 203. *See also* headers (HTTP); SOAP over HTTP binding
 binding to, 472
 with WSDL 1.1 474-480
 with WSDL 2.0 480-484
 as messaging protocol, 472-473
 methods, list of, 473
 SOAP messages and, 204
 wsa:Action element and, 589-590
 HTTP intermediaries, defined, 280
 httpbind:address extensibility element, 475-476
 httpbind:binding extensibility element, 474-475
 httpbind:operation extensibility element, 474-476

I-J

idempotent rule (WS-Policy), 252
 identity constraints (relationship validation), 374
 Ignorable attribute (WS-Policy) pseudo schema, 737
 ignorable policy assertions, 505-506
 case study, 509-510
 considerations for usage, 510-511
 targeting consumers with, 508-509
 versioning and, 692-693
 wsp:Ignorable attribute versus
 wsp:Optional attribute, 506-508
 ignorable termination policy assertions, 702-703
 import element (WSDL), 397-399
 case study, 400-403
 importing XML Schema
 definitions, 405
 include element (WSDL) versus, 404-405
 merging WSDL documents,
 explained, 406
 transitive limitations of, 397-399
 import element (WSDL 1.1), 406-407, 737
 import element (WSDL 2.0), 407-408, 737
 import element (XML Schema) pseudo schema, 738
 imported XML schema types. *See* external XML schema types
 importing schemas into WSDL documents, 406-412. *See also* xsd:import element
 case study, 409-415
 import element (WSDL 1.1) with high visibility, 406-407
 import element (WSDL 2.0) with constrained visibility, 407
 import element (WSDL 2.0) with high visibility, 407-408
 include element (WSDL 2.0) with high visibility, 408
 multiple schemas, importing, 411-412
 transitive limitations, 409-411
 In MEPs, 234

In Plain English sections

- described, 82
- namespaces, 89
 - lack of*, 111
 - naming conventions*, 96-97
 - terminology of*, 93

In-Out MEPs, 234

inbound MEPs, 188

include element (WSDL), 403-408

include element (XML Schema) pseudo schema, 738

incompatible changes, versioning and, 608-610

industry schemas, 379-380

- adding wildcards, 384-392
- reusing types, 380-384

industry standard namespaces, 97-98

industry standards development process, 726-728

infaul element (WSDL 2.0), 231-233

inheritance

- in binding definitions, 202-203
- class inheritance, emulating, 327-333
- interface inheritance, 415-417
- service-orientation and, 416-417
- types and, 132

initial senders (SOAP), defined, 279

inline policy assertions. *See* embedded attachments

input element, child of operation (WSDL 1.1), 187

input message definitions, defined, 55

inter-policy compatibility (merged policies), 541

inter-service inheritance, 416-417

interface definitions, 227

- in abstract descriptions, 53
- associating with binding definitions, 59
- fault, infaul, outfaul elements, 231-233
- interface element, 227-231
- relationships, 61

interface element (WSDL 2.0), 227, 738

- extends attribute, 227-228, 415-416
- styleDefault attribute, 230
- wsdl:safe attribute, 230-231

interface inheritance, 415-417

interfaces, multiple, 191

intermediaries (HTTP), defined, 280

intermediaries (SOAP), 293

- active intermediaries, 294-296
- case study, 280-281
- defined, 279
- forwarding intermediaries, 294

Internationalized Resource Identifiers (IRIs), 169, 484

interoperability, 38

intersection, defined, 543. *See also*

consumer-service policy compatibility

intersection algorithm

- alternative compatibility and, 545-547
- assertion compatibility and, 544
- defined, 543

intra-service inheritance, 416-417

intrinsic interoperability, 38

inventories. *See* service inventories

IRIs (Internationalized Resource Identifiers), 169, 484

K—L

key element (XML Schema) pseudo schema, 738

keyref element (XML Schema) pseudo schema, 738

languages, xml:lang attribute, 298

Last Call Working Draft (standards development process), 727

lax compatibility, 546

Legacy Wrapper design pattern, 337

length element (XML Schema) pseudo schema, 738

length restrictions for user-defined simple types, 134-135

literal message types, encoded message types versus, 209-212

local declarations, 123, 126-128

local elements, associating with target namespaces, 147-151

local names, defined, 91

local types. *See* anonymous types

location attribute (httpbind:operation and httpbind:address elements), 475-476

locking, types of, 714
 Logic Centralization design pattern, 337
 logic-to-contract coupling, 42
 logical view of abstract descriptions, 53-55
 loose coupling. *See* Service Loose Coupling design principle
 Loose versioning strategy
 explained, 613-615
 schema components
 adding new, 676-679
 modifying constraint of, 683-687
 removing, 680-682
 renaming, 683
 WSDL definitions and, 652-656

M

mandatory elements
 (soap:mustUnderstand attribute), 287-289
 mandatory unknown elements, defining, 695-699
 many-to-many relationships, 371-372
 MAP elements, MEP requirements for, 575-576
 MAPs (message addressing properties), 553
 case study, 567-568
 EPRs and, 554-555
 transport-independence of, 554
 wsa:Action element, 564-565
 wsa:FaultTo element, 564
 wsa:From element, 562-563
 wsa:MessageId element, 565
 wsa:RelatesTo element, 565-566
 wsa:ReplyTo element, 563-564
 wsa:To element, 561-562
 maxExclusive element (XML Schema) pseudo schema, 738
 maxInclusive element (XML Schema) pseudo schema, 738
 maxLength element (XML Schema) pseudo schema, 738
 maxOccurs attribute (xsd:any element), 311
 Member Submission (standards development process), 728
 MEP requirements for MAP elements, 575-576

MEPs (message exchange patterns), 187-188, 421
 custom MEPs, 425-426
 of operations, changing, 635-639
 outbound MEPs, 422-423
 Robust In-Only MEP 423-424
 in WSDL 2.0 234
 merging policies, 538-542
 merging WSDL documents, 406
 message addressing properties. *See* MAPs
 message binding definitions, defined, 59
 message correlation (WS-BPEL), 441-443
 message definitions
 in abstract descriptions, 54, 181-186
 physical structure of, 57
 relationships, 61
 message design. *See also* schemas; WS-Addressing; XML
 case study, 163-166
 CRUD message types, 337-338
 acknowledgement messages, 342-343
 Add operation messages, 343-345
 as agnostic, 339-340
 business document messages, 341-342
 chattiness of, 338-339
 Delete operation messages, 350-351
 document-centric Update operation messages, 346-348
 Get operation messages, 345-346
 parameter-centric Update operation messages, 348-350
 structural elements, 154
 case study, 155-161
 container elements, 161-163
 for task services, 351
 tenets of, 118
 message dispatch logic, 450-451
 case study, 451-452
 SOAP Action setting, 455-458
 unique message types, creating, 453-454
 WS-Addressing wsa:Action header, 458-461
 message element (WSDL 1.1), 181-182, 227, 739
 message envelope, defined, 64

- message exchange patterns. *See* MEPs
 - message flexibility, constraint granularity and, 310
 - content model groups, 323-326
 - extension buckets, 316-318
 - generic versus specific elements, 319-323
 - wildcard elements, 310-316
 - message information (MI) headers. *See* MAPs
 - message instances, versioning, 714-716
 - message paths (SOAP), defined, 279
 - message policy subject, 261
 - message schemas. *See* schemas
 - message types, literal versus encoded, 209-212
 - message-level security, 296
 - message-specific metadata, 55
 - MessageID element (WS-Addressing)
 - pseudo schema, 739
 - messageLabel attribute (WSDL 2.0), 234
 - messages. *See also* fault messages (SOAP); SOAP
 - envelopes versus, 275
 - fault messages (WSDL), SOAP fault messages versus, 303-304
 - namespaces in, 114-115
 - replying from endpoints, 575
 - required parts of, 65
 - structure of, 63-65
 - Messaging Metadata design pattern, 45, 758
 - messaging protocol binding, 59
 - messaging protocols, HTTP as, 472-473
 - messaging rules (WS-Addressing), 570
 - case study, 576-580
 - directionality, 570-573
 - MEP requirements for MAP elements, 575-576
 - replying to messages from endpoints, 575
 - metadata
 - message-specific, 55
 - in messages, 64
 - Metadata element (WS-Addressing)
 - pseudo schema, 739
 - Metadata specification, 551
 - MI (message information) headers. *See* MAPs
 - MIME serialization in HTTP bindings, 476-477, 482
 - minExclusive element (XML Schema)
 - pseudo schema, 739
 - minInclusive element (XML Schema)
 - pseudo schema, 739
 - minLength element (XML Schema) pseudo schema, 739
 - minOccurs attribute (xsd:any element), 311
 - mixed content types, 377-379
 - modularization of WSDL, 396
 - benefits of, 396
 - case study, 400-403, 412-415
 - import element, 397-399
 - importing schemas, 406-412
 - include element, 403-406
 - interface inheritance, 415-417
 - merging WSDL documents, explained, 406
 - multiple interfaces, 191
 - multiple policies. *See* effective policies
 - multiple policy endpoints, versioning and, 693-694
 - multiple port types (Flexible versioning strategy), 643-646
 - multiple schemas, importing into WSDL documents, 411-412
 - mustRetain attribute, 710
 - mustUnderstand attribute, defining (code example), 696-697
 - MustUnderstand value (soap:Code element), 289
- N**
- Name attribute (wsp:Policy element), 491-492
 - named model groups, 334-337
 - named types, anonymous types versus, 129-130
 - namespace attribute (xsd:any element), 312, 683-685
 - namespace declarations, 92
 - namespace prefixes
 - alphabetical reference tables, 748-750
 - declaring, 102-103

- default namespaces combined
 - with, 108
 - defined, 86, 92
 - global element declarations and, 145-146
 - naming conventions for, 107
 - overriding, 103-106
 - for policies, 243-244
 - runtime XML document generation, 147
 - with version identifiers, 647-650
 - WSDL, lack of, 168
 - xmlns attribute (WSDL definitions element), 170-172
 - namespaces, 82, 143. *See also* default namespaces; target namespaces
 - attributes and, 89
 - case-sensitivity of, 99
 - custom XML schema namespace, 114-115
 - designing, 90
 - elements and, 84-88
 - industry standard namespaces, 97-98
 - lack of, 109-111
 - library analogy, 89, 93, 96-97, 111
 - in messages, 114-115
 - naming conventions for, 94-96
 - schema reuse and, 151
 - service contracts and, 112-114
 - SOAP namespace, 113-115
 - terminology for, 91-93
 - URL encoding and, 98-99
 - version numbers in, 96
 - versioning and, 613
 - as working URLs, 100-101
 - WS-Addressing, 115
 - WS-Policy namespace, 114
 - for WSDL definitions, 170
 - WSDL namespace, 113
 - XML Schema namespace, 113
 - Namespaces in XML specification, 85
 - naming conventions
 - container elements, 161
 - importance of, 79
 - namespace prefixes, 107
 - namespaces, 94-96
 - user-defined simple types, 133
 - XML, 125
 - narrative content, 377-379
 - nested policy assertions, 437, 498-505, 544
 - nesting
 - policy operators, 252
 - structural elements, avoiding overuse of, 161
 - Next (SOAP role), 282
 - Node element (SOAP 1.2) pseudo schema, 739
 - nodes (SOAP)
 - defined, 280
 - roles and, 282
 - non-agnostic logic, defined, 29
 - non-ignorable unknown elements. *See* mandatory unknown elements
 - None (SOAP role), 282
 - none address value, 558
 - normal form, defined, 534
 - normalization
 - defined, 534
 - of merged policies, 541
 - of policies, 534-538
 - wsp:Optional and wsp:Ignorable attributes, 508
 - Notification MEP, 188. *See also* outbound MEPs
 - notification service for this book series, 16
 - NotUnderstood element (SOAP 1.2) pseudo schema, 739
- O**
- object-orientation, XML Schema versus, 327
 - class inheritance, 327-333
 - named model groups, 334-337
 - one-to-many relationships, 371
 - One-Way MEP, 187
 - online resources. *See* Web sites
 - operation binding definitions, defined, 59
 - operation definitions
 - in abstract descriptions, 54, 186-190
 - physical structure of, 57
 - relationships, 61
 - versioning, 622-642
 - operation element (WSDL 1.1), 186-187
 - operation element of binding element (WSDL 2.0) pseudo schema, 740

- operation element of interface element
 - (WSDL 2.0) pseudo schema, 740
 - operation of binding element (WSDL 1.1)
 - pseudo schema, 739
 - operation of portType element (WSDL 1.1)
 - pseudo schema, 740
 - operation overloading, 189, 229
 - operation policy subject, 260-261
 - operation styles, 484
 - operations. *See also* service operations
 - adding fault message to (versioning), 639-642
 - adding new (versioning), 622-624
 - changing MEP of (versioning), 635-639
 - granularity levels, reducing, 652-653
 - overloading, 416
 - removing (versioning), 630-635
 - renaming (versioning), 624
 - Flexible versioning strategy*, 627-629
 - Strict versioning strategy*, 625-627
 - termination dates, including*, 630
 - operators. *See* policy operators
 - optimistic locking by versioning message
 - instances, 714-716
 - Optional element (WS-Policy) pseudo
 - schema, 741
 - optional policy assertions, 250-251, 691-692
 - organizational roles, 601
 - OTA (Open Travel Alliance), standardized
 - namespaces from, 97
 - outbound MEPs, 188, 422-423
 - outfault element (WSDL 2.0), 231-233
 - output element, child of operation
 - (WSDL 1.1), 187
 - output message definitions, defined, 55
 - overloaded operations, 189, 229, 416
 - overriding namespace prefixes, 103-106
- P**
- parameter-centric Update operation
 - messages, 348-350
 - parameterized policy assertions, 500-505
 - parameterOrder attribute (WSDL operation
 - element), 188-189
 - parametric policy assertions, defined, 523
 - part element (WSDL 1.1), 182-183
 - partial validation, 711-713
 - Partial Validation design pattern, 45,
 - 711-712, 758
 - partners, defined, 440
 - passing data through with wildcards,
 - 314-315
 - pattern attribute (WSDL 2.0), 234
 - pattern element (XML Schema) pseudo
 - schema, 741
 - percent-encoding, defined, 98
 - pessimistic locking, 714
 - physical view of abstract descriptions, 56-57
 - policies, 242-243. *See also* WS-Policy
 - attaching to WSDL definitions, 257
 - case study*, 266-269
 - embedded attachments*, 263-265
 - policy attachment points*, 258
 - policy subjects*, 258-262
 - wsp:PolicyReference element*, 263
 - composite policies, 248
 - wsp:All element*, 249
 - wsp:ExactlyOne element*, 248-249
 - wsp:optional attribute*, 250-251
 - consumer-service policy compatibility,
 - 542-543
 - alternative compatibility*, 545-547
 - assertion compatibility*, 544
 - levels of*, 543-544
 - effective policies, defined, 262
 - namespace prefixes for, 243-244
 - reusability, 486
 - runtime representation, 533
 - merging policies*, 538-542
 - normalization of policies*, 534-538
 - structure of, 243
 - termination policy assertions, 701-702
 - ignorable termination assertions*,
 - 702-703
 - for individual operations*, 704-706
 - mandatory termination assertions*,
 - 703-704
 - versioning, 602, 690-691
 - endpoints, multiple*, 693-694
 - ignorable policy assertions*, 692-693
 - optional policy assertions*, 691-692
 - policy alternatives*, 691
 - propagating changes*, 694-695

- policy alternatives
 - defined, 248
 - in normal form, 535
 - versioning and, 691
- policy assertions
 - custom policy assertions, 247-248, 518
 - Canonical Policy Vocabulary design pattern*, 531-532
 - case study*, 523-527
 - checklist*, 529-530
 - description documents*, 532-533
 - processing logic for*, 521-523
 - schema design for*, 518-521
 - defined, 243
 - explained, 244-246
 - extending with custom attributes, 528-529
 - ignorable, 505-506
 - case study*, 509-510
 - considerations for usage*, 510-511
 - targeting consumers with*, 508-509
 - versioning and*, 692-693
 - wsp:Ignorable attribute versus wsp:Optional attribute*, 506-508
 - nested, 437, 498-505
 - optional policy assertions, 250-251, 691-692
 - parameterized, 500-505
 - termination policy assertions, 701-702
 - ignorable termination assertions*, 702-703
 - for individual operations*, 704-706
 - mandatory termination assertions*, 703-704
 - WS-Addressing, 592
 - wsam:Addressing policy assertion*, 593
 - wsam:AnonymousResponse policy assertion*, 593-594
 - case study*, 595
 - wsam:NonAnonymousResponse policy assertion*, 594
- policy attachment points, 258. *See also*
 - policy subjects
 - determining, 497
 - merging policies, 538-542
- Policy Centralization design pattern, 45, 78, 487-488, 694, 759. *See also*
 - policy definitions
- policy definitions, 486-487. *See also*
 - WS-Policy
 - in abstract descriptions, 55
 - for address-related definitions, 61
 - case study, 494-496
 - challenges to implementation, 496-497
 - in concrete descriptions, 59
 - defined, 243, 486
 - designing, 488-493
 - domain policy definitions, 487
 - global policy definitions, 487
 - physical structure of, 57
 - Policy Centralization design pattern and, 488
 - relationships, 61
- Policy element (WS-Policy) pseudo schema, 741
- policy expressions, 243, 246, 512-516
- policy operators, 248
 - composition rules, 251-252
 - associative rule*, 253-254
 - commutative rule*, 252-253
 - distributive rule*, 254-255
 - empty operators*, 256
 - equivalence*, 256
 - idempotent rule*, 252
 - defined, 243
 - wsp:All element, 249
 - wsp:ExactlyOne element, 248-249
 - wsp:optional attribute, 250-251
- policy processors, 489, 500
- policy subjects, 258-262
 - endpoint policy subject, 259-260
 - message policy subject, 261
 - operation policy subject, 260-261
 - service policy subject, 258-259
 - in WSDL 2.0, 262
- policy type, defined, 245
- PolicyAttachment element (WS-Policy) pseudo schema, 741
- PolicyReference element (WS-Policy) pseudo schema, 742
- polling, 429-430
- port definitions, 60-61, 215-218

port element (WSDL 1.1), 216-217, 742

port type definitions

- in abstract descriptions, 53, 190-191
- associating with binding definitions, 59
- physical structure of, 57
- relationships, 61
- versioning, 643-650

port types

- adding version identifiers to, 643
- multiple (Flexible versioning strategy), 643-646
- namespace prefixes with version identifiers, 647-650
- transport protocols within, 203

portType element (WSDL 1.1), 191, 742

POST method (HTTP), 473

poster Web site, 15

prefixes. *See* namespace prefixes

Prentice Hall Service-Oriented Computing Series from Thomas Erl, 5, 14, 16, 601
(see ad in back of book)

prerequisite reading, 4-5

primary schemas, defined, 699

primitive service compositions, 30

processContents attribute (xsd:any element), 312, 685-687

processing logic for custom policy assertions, 521-523

propagating policy version changes, 694-695

properties, defined, 442

property aliases, defined, 442

Proposed Recommendation (standards development process), 727

protocols for SOAP bindings, 419

Proxy Capability design pattern, 45, 405, 759

pseudo schemas, alphabetical reference, 730-745

publish-and-subscribe model, 423

PUT method (HTTP), 473

Q—R

qualified elements, namespaces and, 147-151

qualified names, defined, 86, 93

range of values for user-defined simple types, 135

Reason element (SOAP 1.2) pseudo schema, 742

receivers (SOAP), defined, 279

recognized content set, 707

recognized sets, defined, 659

Recommendation (standards development process), 727

redefinition, 392

Redundant Implementation design pattern, 405

ReferenceParameters element (WS-Addressing) pseudo schema, 742

referencing elements, 670

regular expression patterns for user-defined simple types, 136-137

RelatesTo element (WS-Addressing) pseudo schema, 742

relationship elements, 375-376

relationships

- in abstract descriptions (of service contracts), 176

- in schemas, 369-371

- many-to-many relationships*, 371-372

- one-to-many relationships*, 371

- separate relationship elements*, 375-376

- xsd:key element*, 373-375

- xsd:keyref element*, 373-375

- among service contract parts, 61-63

- of types in service contracts, 141-142
- validation, 374

RelationshipType attribute (wsa:RelatesTo element), 565-566

relative URLs, namespaces and, 99

removing (versioning)

- operations, 630-635

- schema components

- Flexible versioning strategy*, 671

- Loose versioning strategy*, 680-682

- Strict versioning strategy*, 666

renaming (versioning)

- operations, 624

- Flexible versioning strategy*, 627-629

- Strict versioning strategy*, 625-627

- termination dates, including*, 630

- schema components
 - Flexible versioning strategy*, 671-672
 - Loose versioning strategy*, 683
 - Strict versioning strategy*, 666-667
 - replacement elements
 - defined, 311, 676
 - schemas for, 678-679
 - replying to messages from endpoints, 575
 - ReplyTo element (WS-Addressing) pseudo schema, 742
 - Request-Response MEP, 187
 - resource representation SOAP header block (RRSHB), 290
 - restriction element (XML Schema) pseudo schema, 743
 - retaining unknown elements, 707-710
 - reusability. *See also* Service Reusability design principle
 - of agnostic logic, 29
 - of binding elements, 200
 - of policies, 486
 - policy definitions and, 486-487
 - case study*, 494-496
 - challenges to implementation*, 496-497
 - designing*, 488-493
 - domain policy definitions*, 487
 - global policy definitions*, 487
 - Policy Centralization design pattern*, 488
 - of schemas, 354. *See also* industry schemas
 - namespaces and*, 151
 - xsd:import element*, 360-369
 - xsd:include element*, 355-360
 - structural elements and, 158-159
 - of types, 380-384
 - Robust In-Only MEP, 423-424
 - Role element (SOAP 1.2) pseudo schema, 743
 - roles (SOAP)
 - defined, 281-282
 - nodes and, 282
 - soap:role attribute, 283-287
 - ultimateReceiver, 283
 - rpc binding style, 207
 - RPC data exchange, 188
 - rpc-encoded message type, 209
 - rpc-literal message type, 210
 - RRSHB (resource representation SOAP header block), 290
 - Rules Centralization design pattern, 344
 - runtime policy representation, 533
 - merging policies, 538-542
 - normalization of policies, 534-538
 - runtime XML document generation, namespace prefixes and, 147
- S**
- Schema Centralization design pattern, 45, 78, 354, 664, 699-700, 760
 - schema components. *See* components
 - schema element (XML Schema) pseudo schema, 743
 - schema modules, defined, 356
 - schemas. *See also* pseudo schemas; XML Schema
 - in case study, 152-154, 163-166
 - content sets, explained, 659-660
 - for custom policy assertions, 518-527
 - defined, 68
 - elements. *See* elements
 - embedded XML schema types, 177-180
 - external XML schema types, 177, 180-181
 - importing into WSDL documents, 406-412
 - case study*, 409-415
 - import element (WSDL 1.1) with high visibility*, 406-407
 - import element (WSDL 2.0) with constrained visibility*, 407
 - import element (WSDL 2.0) with high visibility*, 407-408
 - include element (WSDL 2.0) with high visibility*, 408
 - multiple schemas, importing*, 411-412
 - transitive limitations*, 409-411
 - industry schemas, 379-380
 - adding wildcards*, 384-392
 - reusing types*, 380-384
 - narrative content, 377-379
 - primary schemas, defined, 699

- relationships in, 369-371
 - many-to-many relationships*, 371-372
 - one-to-many relationships*, 371
 - separate relationship elements*, 375-376
 - xsd:key element*, 373-375
 - xsd:keyref element*, 373-375
- for replacement elements, 678-679
- reusability, 354
 - namespaces and*, 151
 - xsd:import element*, 360-369
 - xsd:include element*, 355-360
- secondary schemas, defined, 699
- structure of, 121-122
- types. *See* types
- versioning, 602
 - base example for*, 662-663
 - custom strategies for*, 717
 - Flexible versioning strategy*, 668-675
 - Loose versioning strategy*, 676-687
 - mandatory unknown elements*, 695-699
 - retaining unknown elements*, 707-710
 - Schema Centralization design pattern*, 699-700
 - Strict versioning strategy*, 665-668
 - target namespaces and*, 664
 - UPA rule*, 660-661
- XML Schema compared, 118
- scope of versioning, 601-602
- secondary schemas, defined, 699
- security, message-level, 296
- selector element (XML Schema) pseudo schema, 743
- senders (SOAP), defined, 279
- sequence element (XML Schema) pseudo schema, 743
- serialization, MIME serialization in HTTP bindings, 476-477, 482
- Service Abstraction design principle, 39, 43, 78-79, 132, 230, 511
- Service Autonomy design principle, 39
- service candidates, defined, 32
- Service Composability design principle, 30, 40, 44
- service compositions, defined, 29-31
- service contracts
 - abstract descriptions. *See* abstract descriptions (of service contracts)
 - analogy, 2-3
 - concrete descriptions. *See* concrete descriptions (of service contracts)
 - consumer-service policy compatibility, 542-547
 - defined, 34-35, 50
 - design patterns and, 44-47
 - design principles and, 39-44
 - goals and benefits of service-oriented computing, 37-39
 - namespaces and, 112-114
 - structure of, 51
 - technologies for creating. *See* technologies for service contracts
 - type usage by, 141-142
 - versioning. *See* versioning
- service definitions, 60-61, 215-218, 235
- Service Discoverability design principle, 40, 43, 80, 173, 230
- service element (WSDL 1.1), 216-217, 743
- service element (WSDL 2.0) pseudo schema, 744
- service granularity, defined, 35
- service instances
 - applications for, 465-466
 - defined, 462
 - services versus, 462-463
 - wsdlx and wsdl: extensibility attributes, 463-465
- service inventories, defined, 31
- service inventory blueprints, 31
- Service Loose Coupling design principle, 39-43, 77, 371, 510-511
- Service Messaging design pattern, 45, 760
- service models, defined, 28-29
- service operations, defined, 28
- service policy subject, 258-259
- service portion of concrete descriptions, primary parts of, 60-61
- Service Reusability design principle, 29, 39, 43, 340

Service Statelessness design principle,
40, 463

service-orientation

defined, 25-26

design principles. *See* design principles

inheritance and, 416-417

object-orientation versus, 327

class inheritance, 327-333

named model groups, 334-337

service-oriented analysis, defined, 31-32

service-oriented architecture. *See* SOA

Service-Oriented Architecture: Concepts,

Technology, and Design, 5, 438

service-oriented computing

defined, 25

goals and benefits of, 37-39

service-oriented design, defined, 33

service-related granularity, defined, 35-36

services

business-centric, 38

candidates. *See* service candidates

defined, 2, 28

passing data through with wildcards,
314-315

service instances versus, 462-463

service models. *See* service models

simple content complex types, 139

simple types

built-in simple types, 130-132

defined, 129

user-defined simple types, 132-137

simpleContent element (XML Schema)

pseudo schema, 744

simpleType element, child of

element/attribute (XML Schema) pseudo
schema, 744

simpleType element, child of schema (XML

Schema) pseudo schema, 744

SMTP. *See* SOAP over SMTP binding

SOA (service-oriented architecture), 26

defined, 26-27

terminology, 24-37

types, 27

SOA *Design Patterns*, 5, 47

SOA *Governance*, 6, 79, 601

SOA *Magazine*, *The Web site*, 16

SOA *Principles of Service Design*, 5

SOA with .NET, 6

SOA with Java, 6

SOAP, 71-73. *See also* messages;

WS-Addressing

Action setting for message dispatch
logic, 455-458

binding extensions, 418-420

bindings, asynchronous data

exchange, 430-432

envelopes versus messages, 275

extensibility elements, 208-209

fault messages. *See* fault messages

(SOAP)

faults, WS-Addressing and, 592

header blocks, 208, 283

examples of, 289-292

soap:mustUnderstand attribute,
287-289

soap:relay attribute, 289

soap:role attribute, 283-287

structure of, 570-573

HTTP binding alternatives, 472

intermediaries, 293-296

messages

capitalization of element names, 208

HTTP and, 204

structure of, 274-278

namespace, 113-115

namespace reference, 748-750

nodes, 280-282

roles, 281-283

standardized namespaces for, 97

terminology, 278-279

what is not included, 272-273

SOAP 1.1

Body element pseudo schema, 732

detail element pseudo schema, 733

Envelope element pseudo schema, 734

Fault element pseudo schema, 735

faultcode element pseudo schema, 736

faultstring element pseudo
schema, 736

Header element pseudo schema, 737

trailer elements, 276

SOAP 1.2

binding definitions for, 212-215

Body element pseudo schema, 732

- Code element pseudo schema, 732
- Detail element pseudo schema, 733
- Envelope element pseudo schema, 735
- Fault element pseudo schema, 736
- Header element pseudo schema, 737
- Node element pseudo schema, 739
- NotUnderstood element pseudo schema, 739
- Reason element pseudo schema, 742
- Role element pseudo schema, 743
- soap:Upgrade element, 299
- Subcode element pseudo schema, 744
- Text element pseudo schema, 745
- Upgrade element pseudo schema, 745
- SOAP Binding specification, 551
- SOAP over SMTP binding, 431-432
 - WS-Addressing and, 432-436
- soap11: namespace prefix, 202
- soap11:binding element (style attribute), 204-207
- soap11:operation element (soapAction attribute), 203-204
- soapAction attribute (soap11:operation element), 203-204
- SoapAction element (WS-Addressing) pseudo schema, 744
- soap:Body element, 276
- soap:Code element, 298-302
- soap:Detail element, 303
- soap:Envelope element, 275
- soap:Fault element, 278
- soap:Header element, 276-277
- soap:mustUnderstand attribute, 287-289, 697-699
- soap:Node element, 297
- soap:Reason element, 297
- soap:relay attribute, 289
- soap:role attribute, 283-287
- soap:Role element, 298
- soap:SupportedEnvelope element, 300
- soap:Text element, 298
- soap:Upgrade element, 299
- soap:Value element, 298
- Solicit-Response MEP, 188. *See also*
 - outbound MEPs
- specific elements, generic elements versus, 319-323
- standardized namespaces, 97-98
- standardized schemas. *See* industry schemas
- Standardized Service Contract design principle, 31, 35, 39-41, 77, 130, 133
- standards. *See* design standards; industry standards
- state data, defined, 462
- stateful services. *See* service instances
- strategic benefits of service-oriented computing, 39
- strict compatibility, 545
- Strict versioning strategy
 - defined, 612
 - explained, 613-614
 - operations
 - adding fault messages to*, 639-642
 - adding new*, 624
 - changing MEP of*, 635-637
 - removing*, 630-635
 - renaming*, 625-627
 - schema components
 - adding new*, 665-666
 - modifying constraint of*, 667-668
 - removing*, 666
 - renaming*, 666-667
- strictness in versioning strategy
 - comparison, 615
- structural elements, 154
 - case study, 155-161
 - container elements, 161-163
- style attribute (soap11:binding element), 204-207
- styleDefault attribute (WSDL interface element), 230
- Subcode element (SOAP 1.2) pseudo schema, 744
- substitution groups, 333, 392
- symbols
 - chorded circle symbol, 28, 32
 - color in, 14
 - cookie-shaped symbol, 34
 - legend, 14
 - Visio Stencil, 15
- synchrony, asynchrony versus, 427-429

T

tags. *See* elements

target namespaces, 143-147

- associating local elements/attributes with, 147-151

- defined, 93

- format conventions, 147

- import element (WSDL) and, 397

- schema versioning and, 664

- targetNamespace attribute, 144-146

- tns: prefix, 147

targetNamespace attribute (WSDL definitions element), 170

targetNamespace attribute (xsd:schema element), 144-146

task service model, defined, 28

task services, message types, 351

technologies for service contracts, 66-75

- for abstract descriptions, 73-75

- characteristics of, 66

- guidelines for usage, 76-80

- SOAP, 71-73

- WS-I Basic Profile, 74

- WS-Policy, 70-71

- WSDL, 67-68

- XML Schema, 68-70

termination dates, including when renaming operations, 630

Termination Notification design pattern, 45, 701, 761

termination policy assertions, 701-702

- ignorable termination assertions, 702-703

- for individual operations, 704-706

- mandatory termination assertions, 703-704

Text element (SOAP 1.2) pseudo schema, 745

tns: namespace prefix, 147

To element (WS-Addressing) pseudo schema, 745

totalDigits element (XML Schema) pseudo schema, 745

trailer elements (SOAP 1.1), 276

transitive limitations

- of import element (WSDL), 397-399

- importing schemas into WSDL documents, 409-411

transport attribute (WSDL binding element), 202

transport protocol binding, 59

transport protocols

- MAPs and, 554

- within port types, 203

transports for SOAP bindings, 418-420

type attribute (WSDL part element), 182, 185

type composition, 334-337

type definitions, type element

- in abstract descriptions, 55, 176-181

- physical structure of, 57

- relationships, 61

type derivation, 327

type substitution, 333

types. *See also* attributes; elements

- abstract types (xsd:extension element), 329-331

- built-in simple types, 130-132

- common type libraries, 361-363

- complex types, 137-141, 329

- content models, 129

- content types, 137-141

- defined, 129

- elements versus, 124

- inheritance and, 132

- named versus anonymous types, 129-130

- reusing, 380-384

- service contracts and, 141-142

- user-defined simple types, 132-137

types element (WSDL 1.1), 176-181, 745

types element (WSDL 2.0) pseudo schema, 745

U

UBL (Universal Business Language) industry schema, 384-392

ultimate receivers (SOAP), defined, 279

ultimateReceiver (SOAP role), 282-283

Uniform Resource Identifiers. *See* URIs

Uniform Resource Locators. *See* URIs

Uniform Resource Names. *See* URNs

unique message types, creating, 453-454

Unique Particle Attribution. *See* UPA rule

Universal Business Language. *See* UBL
industry schema

unknown content set, 707

unknown elements

mandatory unknown elements,
defining, 695-699
retaining, 707-710

unknown sets, defined, 659

unqualified elements, namespaces and,
147-151

UPA (Unique Particle Attribution) rule, 314,
660-661

Update operation messages

document-centric, 346-348
parameter-centric, 348-350

Upgrade element (SOAP 1.2) pseudo
schema, 745

URI attribute (wsp:PolicyReference
element), 491-492

URI element (WS-Policy) pseudo
schema, 745

URIs (Uniform Resource Identifiers), 94, 168

URL encoding, namespaces and, 98-99

URLs (Uniform Resource Locators), 94, 168
namespaces as, 94-96, 100-101

URNs (Uniform Resource Names), 94, 168

use attribute (SOAP extensibility elements),
208-212

user-defined simple types, 132-137

decimal precision and scale, 135
enumerated lists of values, 134
facets, 133
length restrictions, 134-135
regular expression patterns, 136-137

utility service model, defined, 29

V

validation

bypassing, 449-450
partial validation, 711-713
of relationships, 374

Validation Abstraction design pattern, 45,
78, 132, 344, 761

validation by projection, 712-713

vendor diversification, 38

Version Identification design pattern, 45,
610, 762

version identifiers

adding to port types, 643
conventions for, 718
in namespace prefixes, 647-650
for WSDL definitions, 618-621

version numbers, 96, 610-612

versioning, 600

compatibility and, 603

backwards compatibility, 603-605
compatible changes, 607-608
forwards compatibility, 605-606
incompatible changes, 608-610

constraint granularity levels in, 602-603

message instances, 714-716

partial validation, 711-713

policies, 690-691

endpoints, multiple, 693-694
ignorable policy assertions, 692-693
optional policy assertions, 691-692
policy alternatives, 691
propagating changes, 694-695

schemas

base example for, 662-663
custom strategies for, 717
Flexible versioning strategy, 668-675
Loose versioning strategy, 676-687
mandatory unknown elements,
695-699
retaining unknown elements, 707-710
Schema Centralization design
pattern, 699-700
Strict versioning strategy, 665-668
target namespaces and, 664
UPA rule, 660-661

scope of, 601-602

strategies for, 612-616

termination policy assertions, 701-706

terminology, 601-603

version numbers, 610-612

WSDL definitions, 618

concrete descriptions, 650-652
forwards compatibility, 652-656
operation definitions, 622-642
port type definitions, 643-650
version identifiers for, 618-621

Visio Stencil, 15

vocabularies, described, 83

W

- W3C, 52, 66
 - standards development process, 726-728
 - Web Architecture specification, 230
 - Web Services Glossary, 428
- Web architecture, Web services architecture versus, 472-473
- Web service contracts. *See* service contracts
- Web services. *See also* services
 - asynchronous data exchange
 - bindings*, 430-432
 - guidelines for usage*, 437
 - polling*, 429-430
 - WS-Addressing*, 432-436
 - Decoupled Contract design pattern and, 46-47
 - defined, 33-34
 - synchronous versus asynchronous aspects, 427-429
- Web Services Addressing Core specification, 551
- Web services architecture, Web architecture versus, 472-473
- Web Services Description Language. *See* WSDL
- Web Services Interoperability Organization. *See* WS-I Basic Profile
- “Web Services Policy 1.5 – Guidelines for Policy Assertion Authors” document, 530
- Web sites
 - GS1 industry schema information, 384
 - SOABooks.com, 6, 15-16
 - SOAGlossary.com, 5, 15, 37, 339
 - SOAMag.com, 16
 - SOAMethodology.com, 31
 - SOAP over SMTP binding information, 432
 - SOAPatterns.org, 5, 15, 47
 - SOAPosters.com, 15
 - SOAPinciples.com, 5, 44
 - SOAspecs.com, 15-16, 76, 85, 246, 296, 343, 501, 530, 538, 551, 728
 - WhatIsSOA.com, 5, 37, 39
 - XMLEnterprise.com, 4
- well-formed XML documents, #any value (element attribute), 447-448
- whhttp:location attribute, 480-481
- whhttp:method attribute, 480-481
- whhttp:methodDefault attribute, 480-481
- wildcards, 310-316
 - adding to industry schemas, 384-392
 - disadvantages, 316
 - passing data through to Web services, 314-315
 - schema versioning with, 676-687
 - UPA rule and, 314
 - wrapper elements for, 677-678
- work effort, version numbers and, 611
- Working Draft (standards development process), 727
- Working Group Note (standards development process), 728
- wrapper elements for wildcards, 677-678
- WS-Addressing, 246, 288, 291, 526, 550-551
 - Action element pseudo schema, 730
 - Action values, 587-591
 - Address element pseudo schema, 730
 - asynchronous data exchange, 432-436
 - EndpointReference element pseudo schema, 734
 - EPRs, 551-553, 556-560
 - FaultTo element pseudo schema, 736
 - From element pseudo schema, 737
 - MAPs, 553
 - case study*, 567-568
 - EPRs and*, 554-555
 - transport-independence of*, 554
 - wsa:Action element*, 564-565
 - wsa:FaultTo element*, 564
 - wsa:From element*, 562-563
 - wsa:MessageId element*, 565
 - wsa:RelatesTo element*, 565-566
 - wsa:ReplyTo element*, 563-564
 - wsa:To element*, 561-562
- MessageID element pseudo schema, 739
- messaging rules, 570
 - case study*, 576-580
 - directionality*, 570-573
 - MEP requirements for MAP elements*, 575-576
 - replying to messages from endpoints*, 575

- Metadata element pseudo schema, 739
- namespace, 115
- namespace reference, 748-750
- policy assertions, 592-595
- ReferenceParameters element pseudo schema, 742
- RelatesTo element pseudo schema, 742
- ReplyTo element pseudo schema, 742
- service instances, identifying, 466
- SOAP faults, 592
- SoapAction element pseudo schema, 744
- To element pseudo schema, 745
- wsa:Action header, 458-461
- WSDL binding, 580-586
- WS-BPEL, WSDL and, 438-444
 - asynchronous data exchange, 440-441
 - case study, 443-444
 - correlation of messages, 441-443
 - extensibility elements, 439-440
- WS-Coordination, 291-292, 551
- WS-I Basic Profile, 74, 127, 173, 589
 - binding element (WSDL 1.1), 200
 - document-literal message type, 210
 - importing XML Schema definitions, 405
 - outbound MEPs, 422-423
 - part element (WSDL 1.1), 182-183
 - service and port elements, 217
 - SOAP header blocks, 472
 - specifications in, 551
 - transport protocols, 203
 - types element (WSDL 1.1)
 - requirements, 178
 - WS-BPEL compliance with, 438
- WS-Policy, 70-71. *See also* policies; policy definitions
 - All element pseudo schema, 730
 - ExactlyOne element pseudo schema, 735
 - Ignorable element pseudo schema, 737
 - namespace, 114
 - namespace reference, 748-750
 - Optional element pseudo schema, 741
 - Policy element pseudo schema, 741
 - PolicyAttachment element pseudo schema, 741
 - PolicyReference element pseudo schema, 742
 - URI element pseudo schema, 745
 - versioning and, 602
- WS-PolicyAttachment, 71, 257
- WS-ReliableMessaging, 245-246, 290-291, 343, 551
- WS-SecureConversation, 551
- WS-Security, 296, 501
- WS-SecurityPolicy, 501, 526
- WS-Trust, 551
- wsam:Addressing element, 246
- wsam:Addressing policy assertion, 459, 593
- wsam:AnonymousResponse policy assertion, 593-594
- wsam:AnonymousResponses element, 435
- wsam:NonAnonymousResponse policy assertion, 594
- wsam:NonAnonymousResponses element, 436
- wsaw:Action attribute, 459-460
- wsa:Action element, 564-565, 587
 - HTTP and, 589-590
 - as required, 587
 - WSDL and, 588-589
- wsa:Action header, 458-461
- wsa:Address element, 557-558
- wsa:EndpointReference element, 556-557
- wsa:FaultTo element, 564
- wsa:From element, 562-563
- wsa:isReferenceParameter attribute, 573
- wsa:MessageId element, 565
- wsa:Metadata element, 560
- wsa:ReferenceParameters element, 559-560
- wsa:RelatesTo element, 565-566
- wsa:ReplyTo element, 563-564
- wsa:To element, 561-562
- WSDL (Web Services Description Language), 67-68. *See also* WSDL 1.1; WSDL 2.0; WSDL definitions
 - abstract WSDL descriptions,
 - versioning and, 602
 - custom header blocks and header faults, 466-471
 - extensibility, 418-421
 - fault messages, SOAP fault messages versus, 303-304

- MEPs, 421-426
- message dispatch logic, 450-451
 - case study*, 451-452
 - SOAP Action setting*, 455-458
 - unique message types, creating*, 453-454
 - WS-Addressing wsa:Action header*, 458-461
- modularization. *See* modularization of WSDL
- namespace, 113
 - prefixes, lack of*, 168
 - reference*, 748-750
- standardized namespaces for, 97
- WS-Addressing binding, 580-586
- WS-BPEL and, 438-444
- wsa:Action element and, 588-589
- WSDL 1.1. *See also* WSDL; WSDL 2.0; WSDL definitions
 - abstract description design. *See* abstract descriptions (of service contracts)
 - binding element pseudo schema, 731
 - binding to HTTP with, 474-480
 - complete WSDL definition, 218-224
 - concrete description design. *See* concrete descriptions (of service contracts)
 - converting definitions to WSDL 2.0 239
 - definitions element pseudo schema, 733
 - import element pseudo schema, 737
 - interface inheritance, emulating, 417
 - message element pseudo schema, 739
 - operation of binding element pseudo schema, 739
 - operation of portType element pseudo schema, 740
 - physical view of abstract descriptions, 57
 - port element pseudo schema, 742
 - portType element pseudo schema, 742
 - service element pseudo schema, 743
 - SOAP binding extensions, 418
 - types element pseudo schema, 745
- WSDL 2.0. *See also* WSDL; WSDL 1.1; WSDL definitions
 - #any attribute value, 653-656
 - binding element, 200, 731
 - binding to HTTP with, 480-484
 - complete WSDL definition, 235, 238
 - custom header blocks, 471
 - custom MEPs, 425-426
 - description element pseudo schema, 733
 - document binding style, 207
 - document structure, 226-227
 - endpoint definitions, 235
 - endpoint element, 235, 734
 - extensibility attributes, wsdlx and wsdl, 463-465
 - extreme loose coupling, 446-450
 - fault element of binding element pseudo schema, 736
 - fault element of interface element pseudo schema, 736
 - import element pseudo schema, 737
 - interface definitions, 227-233
 - interface element pseudo schema, 738
 - MEPs, 234
 - message element, lack of, 183
 - operation element of binding element pseudo schema, 740
 - operation element of interface element pseudo schema, 740
 - operation overloading, 189
 - parameterOrder attribute, lack of, 189
 - physical view of abstract descriptions, 57
 - policy subjects in, 262
 - port element, renamed endpoint element, 217
 - portType element, changed to interface element, 191
 - Robust In-Only MEP, 423-424
 - service definitions, 235
 - service element pseudo schema, 744
 - SOAP binding extensions, 419
 - types element pseudo schema, 745

WSDL definitions

- attaching policies to, 257
 - case study*, 266-269
 - embedded attachments*, 263-265
 - policy attachment points*, 258
 - policy subjects*, 258-262
 - wsp:PolicyReference* element, 263
- propagating policy version changes to, 694-695
- version numbers, 611
- versioning, 618
 - concrete descriptions*, 650-652
 - forwards compatibility*, 652-656
 - operation definitions*, 622-642
 - port type definitions*, 643-650
 - version identifiers for*, 618-621

wrapping policy definitions in, 492-493

wsdl:required attribute, 214, 420-421

wsdli extensibility attribute, 463-465

wsdls:safe attribute (WSDL 2.0), 230-231, 484

wsdlx extensibility attribute, 463-465

wsp:All element, 249

wsp:ExactlyOne attribute, 691

wsp:ExactlyOne element, 248-249

wsp:Ignorable attribute, 506-509

wsp:Optional attribute, 250-251, 506-508, 691

wsp:Policy element, 246, 491-492

wsp:PolicyAttachment element, 489-490

wsp:PolicyReference element, 263-265, 491-492

wsp:PolicyURIs attribute, 490-491

wsrmp:RMAssertion element, 245

X—Z**XML**

- attributes, 83, 89, 124-126
- constructs, defined, 66
- definitions, defined, 67
- documents, runtime generation, 147
- elements. *See* elements
- naming conventions, 125
- types. *See* types
- vocabularies, described, 83
- Web site for information, 4

XML Schema, 68-70. *See also* schemas

- all element pseudo schema, 731
- any element pseudo schema, 731
- anyAttribute element pseudo schema, 731
- attribute element, child of
 - complexType pseudo schema, 731
- attribute element, child of schema pseudo schema, 731
- choice element pseudo schema, 732
- complexContent element pseudo schema, 732
- complexType element, child of element pseudo schema, 733
- complexType element, child of schema pseudo schema, 732
- components, defined, 658
- custom XML schema namespace, 114-115
- definitions, version numbers, 611
- element element, child of schema pseudo schema, 733-734
- element element, child of
 - sequence/choice/all pseudo schema, 734
- enumeration element pseudo schema, 734
- extension element, child of
 - complexContent pseudo schema, 735
- extension element, child of
 - simpleContent pseudo schema, 735
- field element pseudo schema, 737
- fractionDigits element pseudo schema, 737
- group element, child of schema pseudo schema, 737
- group element, descendant of
 - complexType pseudo schema, 737
- import element pseudo schema, 738
- importing with import element (WSDL), 405
- include element pseudo schema, 738
- key element pseudo schema, 738
- keyref element pseudo schema, 738
- length element pseudo schema, 738

- maxExclusive element pseudo schema, 738
- maxInclusive element pseudo schema, 738
- maxLength element pseudo schema, 738
- minExclusive element pseudo schema, 739
- minInclusive element pseudo schema, 739
- minLength element pseudo schema, 739
- namespace, 113
- namespace reference, 748-750
- object-orientation versus, 327
 - class inheritance*, 327-333
 - named model groups*, 334-337
- pattern element pseudo schema, 741
- restriction element pseudo schema, 743
- schema element pseudo schema, 743
- schemas compared, 118
- selector element pseudo schema, 743
- sequence element pseudo schema, 743
- simpleContent element pseudo schema, 744
- simpleType element, child of element/attribute pseudo schema, 744
- simpleType element, child of schema pseudo schema, 744
- standardized namespaces for, 97
- totalDigits element pseudo schema, 745
- WSDL #any value (element attribute) and, 450
- XML Schema Definition Language. *See* XML Schema
- xmlns attribute, 86, 92, 107, 170-172
- xml:lang attribute, 298
- XPath, 375
- xsd:all element, 325
- xsd:any element, 311-312, 683-687
- xsd:anyAttribute element, 313
- xsd:anyType type, 318, 683
- xsd:choice element, 323-325, 378, 675
- xsd:date type, 132
- xsd:enumeration facet, 134
- xsd:extension element, 132, 329-331
- xsd:fractionDigits facet, 135
- xsd:group element, 334-335
- xsd:import element, 360-369
- xsd:include element, 355-360
- xsd:integer type, avoiding, 132
- xsd:key element, 373-375
- xsd:keyref element, 373-375
- xsd:length element, 133
- xsd:maxExclusive facet, 135
- xsd:maxInclusive facet, 135
- xsd:maxLength facet, 134
- xsd:minExclusive facet, 135
- xsd:minInclusive facet, 135
- xsd:minLength facet, 134
- xsd:pattern facet, 136
- xsd:restriction type, 132-133
- xsd:schema element, 408
 - attributeFormDefault attribute, 149
 - elementFormDefault attribute, 148
 - targetNamespace attribute, 144-146
- xsd:sequence element, 140, 326
- xsd:simpleType element, 133
- xsd:totalDigits facet, 135
- xsd:unsignedByte type, avoiding, 132
- xsd:unsignedInt type, avoiding, 132
- xsd:unsignedLong type, avoiding, 132
- xsd:unsignedShort type, avoiding, 132