

A DETAILED INVESTIGATION OF INTEROPERABILITY FOR WEB SERVICES

Thesis submitted in fulfilment of the requirements for the degree of
MASTER OF SCIENCE
of
RHODES UNIVERSITY

by

MADELEINE WRIGHT

December, 2005



RHODES UNIVERSITY
Grahamstown • 6100 • South Africa

ABSTRACT

The thesis presents a qualitative survey of web services' interoperability, offering a snapshot of development and trends at the end of 2005. It starts by examining the beginnings of web services in earlier distributed computing and middleware technologies, determining the distance from these approaches evident in current web-services architectures. It establishes a working definition of web services, examining the protocols that now seek to define it and the extent to which they contribute to its most crucial feature, interoperability.

The thesis then considers the REST approach to web services as being in a class of its own, concluding that this approach to interoperable distributed computing is not only the simplest but also the most interoperable. It looks briefly at interoperability issues raised by technologies in the wider arena of Service Oriented Architecture. The chapter on protocols is complemented by a chapter that validates the qualitative findings by examining web services in practice. These have been implemented by a variety of toolkits and on different platforms. Included in the study is a preliminary examination of JAX-WS, the replacement for JAX-RPC, which is still under development. Although the main language of implementation is Java, the study includes services in C# and PHP and one implementation of a client using a *Firefox* extension.

The study concludes that different forms of web service may co-exist with earlier middleware technologies. While remaining aware that there are still pitfalls that might yet derail the movement towards greater interoperability, the conclusion sounds an optimistic note that recent cooperation between different vendors may yet result in a solution that achieves interoperability through core web-service standards.

ACM Categories and Subject Descriptors

Categories

D.2.12 [Interoperability]: **Data Mapping, Distributed Objects, Interface Definition Languages**

I.7.2 [Document Preparation]: Markup languages

H.3.5 [Information Storage and Retrieval]: On-line Information Services – Web-based Services

D.2.11 [Software Engineering]: Software Architectures

H.5.3 [Web-based Interaction] Group and Organization Interfaces

H.5.4 [Hypertext/Hypermedia]: Architectures

General Terms

Design, Standardization, Languages

Additional Key Words and Phrases

Web Service, Representational State Transfer, World Wide Web

ACKNOWLEDGEMENTS

The writer wishes to acknowledge with much gratitude the generous help and support she has received from the Computer Science Department at Rhodes University and from those in the Department of Information Systems who had expertise in areas where she had none, and were willing to share it with her. She would particularly like to single out her supervisors, George Wells and Peter Clayton, for their wise words, expert guidance and attempts to keep her focused when she was awash in a sea of information. George Wells is owed an extra debt of gratitude for making timetable spaces in which this work could be brought to conclusion. Through the Centre for Excellence in the department, her supervisors also provided the resources that made much of this research possible and she is therefore indebted also to the sponsors of the Centre for Excellence:

- Business Connexion
- Comverse
- Telkom
- Thrip: Technology and Human Resources Industry Program
- Verso Technologies

The writer is indebted to the following companies for the use of their commercial software:

- to Parasoft (especially James Sun), for allowing her to use evaluation versions of *SOA Test* well beyond the normal evaluation period and then negotiating an affordable one-year licence.
- to Mindreef (especially Cheryl Bosquet), for also allowing her to extend evaluations of *SOAPScope* and then generously giving her a year's complementary licence.
- to Snowtide Informatics (especially Chas Emerick), for allowing her a free, extended licence for their *PDFTextStream* API, used in the indexing service in Chapter 7.
- to Altova for their education partner program which enabled her to use freely tools such as *XMLSpy* and *MapForce*.

Thanks are also due to contributors to the various forums on which she posted queries (perhaps especially the Apache Axis forums), and to the following expert developers who were patient and kind enough to give of their time to answering questions and solving problems, specifically:

- Russel Butek for correspondence following on from his article on the interpretation of arrays in XML, referred to in section 3.4.2 ;
- Steve Loughran for his insight into JAX-RPC and for allowing the author to read his paper on that subject prior to publication in 2005, referred to in section 4.2.4.3;
- Dmitri Stogov (one of the creators of the SOAP extensions for PHP 5) for his helpful articles on the Zend site and for responding to a query made by the author concerning the creation of a server for the document/wrapped literal SayHello service, mentioned in section 7.2.9;
- Conrad O'Dea for his help with Celtix, referred to in section 3.4.4;
- Jerome Dochez (now the project leader for GlassFish) for his help with GlassFish, mentioned in section 7.2.6.
- Christian Weyer of ThinkTecture for his help with the jWSCF plugin for Eclipse, mentioned on page 93;
- Jeff Barr of Amazon for replying to a query about the relative speed of REST versus SOAP, mentioned in section 5.3.

TABLE OF CONTENTS

A DETAILED INVESTIGATION OF INTEROPERABILITY FOR WEB SERVICES	i
ABSTRACT	ii
ACM Categories and Subject Descriptors	iii
Categories	iii
General Terms	iii
Additional Key Words and Phrases	iii
ACKNOWLEDGEMENTS	iv
TABLE OF CONTENTS	vi
LIST OF FIGURES	x
LIST OF GRAPHS AND TABLES	xi
NOTES	xii
Note 1: References Used	xii
Note 2: Acronyms	xiii
Note 3: Capitalization	xiii
Note 4: US versus British Spelling Conventions	xiv
CHAPTER 1: INTRODUCTION	1
1.1 Introduction: the Issues	1
1.2 Interoperability as the Distinguishing Feature of Web Services	2
1.3 Research Methods	5
1.4 The Design	5
CHAPTER 2: FOUNDATIONS OF WEB SERVICES	8
2.1 Introduction	8
2.2 Similarities and Differences	8
2.3 Client/Server: the Beginnings of Distributed Computing	9
2.4 Middleware Communication Architectures	10
2.4.1 The Remote Procedure Call (RPC)	10
2.4.2 The Distributed Object Model	12
2.4.2.1 CORBA	14
2.4.2.2 Component Object Models	14
2.4.2.3 Transactional Component Middleware	15
2.4.2.4 Integrating Legacy Systems	15
2.4.2.5 Web Services versus Distributed Objects	17
2.4.2.5.1 Comparisons Between Web Services, CORBA and RMI	18
2.4.2.6 Recognizing the Remoteness of Distributed Objects	22
2.4.3 Message-Oriented Middleware	23
2.4.4 Summary	26
CHAPTER 3: DEFINING WEB SERVICES AND THEIR CURRENT STATUS	28
3.1 Introduction: Definition Problems	28
3.1.1 The Web in Web Services	34
3.1.2 The Distinction between Web Services and Service-Oriented Architecture	35
3.1.3 A Working Definition	37
3.2 Advantages of Web Services	38
3.3 Adoption of Web Services	40
3.4 Barriers to Interoperability	43

3.4.1 Transport Issues	43
3.4.2 User-Defined Data Types	46
3.4.3 Exception-Handling: <code>java.rmi.RemoteExceptions</code> and SOAP Faults	49
3.4.4 Toolkits	52
3.5 Summary	56
CHAPTER 4: A QUALITATIVE ANALYSIS OF CORE WEB-SERVICE COMPONENTS AND THEIR CONTRIBUTION TO INTEROPERABILITY FOR WEB SERVICES	58
4.1 Introduction	58
4.1.1 Interoperability-Testing Frameworks	60
4.2 SOAP: no longer an acronym	61
4.2.1 Defining SOAP: Versioning Issues	61
4.2.2 SOAP as a Requirement for Web Services	63
4.2.3 Problems with SOAP	64
4.2.4 The SOAP Message Structure	68
4.2.4.1 Service Styles and their Encodings	71
4.2.4.1.1 RPC	73
4.2.4.1.2 Document	74
4.2.4.1.3 Wrapped	75
4.2.4.2 SOAP <i>Programming</i> Models	77
4.2.4.2.1 RPC	78
4.2.4.2.2 Document	79
4.2.4.3 Advantages and Disadvantages of the RPC Processing Model	80
4.2.4.4 Advantages and Disadvantages of the Document-Style Programming Model	83
4.2.4.5 Different Types of Soap Client	84
4.2.4.5.1 Static Stub Client	84
4.2.4.5.2 Dynamic proxy client	85
4.2.4.5.3 Dynamic invocation client	85
4.2.4.5.4 Application Client	85
4.2.4.5.5 Summary of SOAP Client Approaches	86
4.3 WSDL: Web Services Description Language	87
4.3.1 WSDL: a simple description of a web service	89
4.3.1.1 The Abstract Section	90
4.3.1.2 The Concrete Section	92
4.3.2 WSDL Data-Typing	93
4.3.3 WSDL First	95
4.3.4 Conclusions Regarding WSDL	99
4.4 UDDI – Dead in its Tracks?	100
4.5 Summary	101
CHAPTER 5: REST – AN ALTERNATIVE	104
5.1 Introduction: the Revolt against Complexity	104
5.2 Representational State Transfer	105
5.3 Advantages of REST over Conventional Web Services	109
5.4 Disadvantages of REST over Conventional Web Services	110
5.5 Summary	111
CHAPTER 6: WEB-SERVICES COORDINATION	113
6.1 Introduction	113

6.2 Orchestration and Coordination.....	113
6.2.1 Web Services Choreography Interface (WSCI).....	115
6.2.2 BPEL.....	116
6.3 The Enterprise Service Bus.....	119
6.4 A First Look at WCF.....	120
6.5 Summary.....	122
CHAPTER 7: VALIDATION OF QUALITATIVE FINDINGS.....	123
7.1 Introduction.....	123
7.1.1 The Platforms.....	124
7.1.1.1 Apache Axis 1.2 (Java).....	124
7.1.1.2 GlassFish (Java).....	125
7.1.1.3 Sun Application Server 8.1 (Java).....	125
7.1.1.4 BEA WebLogic 9.0 (Java).....	126
7.1.1.5 Web Matrix (C#).....	127
7.1.1.6 <i>Visual Studio 2005</i> (C#).....	127
7.1.1.7 Apache 2 (PHP).....	127
7.1.1.8 Mozilla <i>Firefox</i> (JavaScript).....	128
7.2 Web Services.....	129
7.2.1 The Simple Calculator Service.....	129
7.2.2 An Even Simpler Date Service.....	130
7.2.3 The Indexing Application: Java.....	132
7.2.3.1 A <i>Firefox</i> Front End.....	134
7.2.4 The Dictionary Service: Java.....	135
7.2.4.1 Using Exceptions in the Dictionary Service.....	137
7.2.4.2 Testing on Other Servers.....	139
7.2.4.3 C# Clients for the Dictionary Service.....	139
7.2.4.4 Metadata and WSDL Generation.....	143
7.2.5 The Enumeration Service.....	146
7.2.6 A Simple Hello Service.....	148
7.2.7 A Different Programming Style with JAX-WS Provider and Dispatch.....	150
7.2.8 REST-Style Messaging.....	152
7.2.9 A Simple HelloWorld Service: PHP.....	155
7.2.10 A CheckNumbers Service.....	159
7.3 Implementations of SOAP 1.2.....	161
7.4 Different Types of Java Client.....	163
7.4.1 Static Stub Client.....	163
7.4.2 Dynamic Proxy Client.....	163
7.4.3 Dynamic Invocation Client.....	164
7.5 Summary: a Synthesis of Findings.....	165
CHAPTER 8: CONCLUSION.....	168
8.1 Introduction: the Changing Scene.....	168
8.2 No Single Solution.....	169
8.3 Problems Still Needing Solutions.....	172
8.4 Changing Requirements.....	173
8.5 Summary of Achievements in this Study.....	173
8.6 Future Work.....	174

APPENDIX A: List Of Acronyms Used In The Text.....	176
APPENDIX B: Creating A <i>Firefox</i> Extension.....	179
APPENDIX C: The Complexity Of Web-Service Specifications [Jeon, c2005]	181
APPENDIX D: The Inner Workings of the Indexing Service.....	182
APPENDIX E: An Example WSDL File.....	184
REFERENCES	188
INDEX	207

LIST OF FIGURES

Figure 2-1: A Web-Centric View of Distributed Computing	9
Figure 2-2: The Remote Procedure Call	12
Figure 2-3: The Main Components of the ORB Architecture and their Interconnections [Keahey, 1998]	13
Figure 2-4: Generation of Client and Server Components from Interface for Web Services, CORBA and Java-RMI [adapted from Gray, 2004]	19
Figure 2-5: The Point-to-Point MOM Option	24
Figure 2-6: The Publish/Subscribe MOM Option	25
Figure 3-1: Web-Service Protocols [adapted from Wilkes, 2004]	30
Figure 3-2: Accessing and Composing Services in a SOA [adapted from Lomow and Newcomer, 2004]	36
Figure 3-3: Redrawn from Web Service Projects, Forrester Report, 2004	41
Figure 3-4: The Architecture of JAXB [redrawn from the J2EE 1.4 Tutorial]	46
Figure 4-1: Overall Design of a Web Service	58
Figure 4-2: The SOAP Process [adapted from Microsoft, date unknown]	62
Figure 4-3: A representation of the basic SOAP Message Structure	69
Figure 4-4: An RPC <i>Programming</i> Model [adapted from Tyagi, 2004a]	78
Figure 4-5: Document-Style Interaction [Adapted from Tyagi, 2004a]	79
Figure 4-6: JAX-RPC Client Invocation Models [redrawn from Joseph, 2003]	84
Figure 4-7: XML Infoset Diagram of WSDL 2.0 [Booth and Liu, 2005]	87
Figure 5-1: REST Web-Service Structure [Hinchcliffe, 2005]	106
Figure 6-1: The relationship between WSDL and WSCI [adapted from Arkin et al, 2002]	115
Figure 6-2: Schematic of a BPEL Process for Handling a Purchase Order [Andrews <i>et al.</i> , 2003]	117
Figure 6-3: The Role of the Service Bus in SOA [adapted from Weerawarana et al, 2005]	119
Figure 7-1: UML Diagram of the Index and Query Application	132
Figure 7-2: A Graphical Rendering of the Web-Service WSDL	134
Figure 7-3: Graphical Depiction of the Schema for the Dictionary Service	136
Figure 7-4: <i>XMLSpy</i> Graphical Representation of the WSDL for the Dictionary Service	137

LIST OF GRAPHS AND TABLES

Table 2-1: Gray's results for a single (simple) exchange.....	20
Table 2-2: Packet Numbers for the Different Technologies [Jeckle <i>et al.</i> , 2004]	21
Table 2-3: Package sizes with a payload first of 0 bytes and then with a payload of 10,000 bytes [Jeckle <i>et al.</i> , 2004].....	22
Table 3-1: Page-Count of Web-Services Specifications Available in Late 2004 [Bray, 2004b] .	33
Table 3-2: Economic Advantages of Paperless Trade [redrawn from Australian Government, 2001]	39
Table 3-3: Loose and Tight Coupling [redrawn from Slama <i>et al.</i> , 2004]	39
Table 3-4: JAXB Mapping of XML Schema Built-in Data Types [Sun, 2005a]	47
Table 4-1: WSDL-to-Java Mappings, with WSDL 2.0 names in blue	88
Table 5-1: Differences between REST and SOAP in terms of process integration.....	110
Table 7-1: Findings from the Experimental Services	167
Table 8-1: Performance Differences Between SOAP and RMI [Adapted from IBM Software Group, 2003].....	171

NOTES

Note 1: References Used

Extensive use has been made of *Safari Online*, and several other Computer Science publisher sites where the writer has membership and where the content is not only refereed and edited but also sometimes consists of chapters from recently published books. One such site is *InformIT*, an offshoot of Pearson Educational Publishers who are also partners with other major Computer Science publishing companies. O'Reilly publishers, who run *Safari Online*, also manage *XML.com* and *On Java*. Weblogs (or blogs) are usually considered too informal for inclusion in a serious academic study. References drawn from the very few blogs that have been included are by writers (such as Anne Thomas Manes) who have already made a significant contribution to some aspect of web-services technology and whose opinions are founded in experience. A brief description, indicating the experience of less well-known authors, has been appended to references other than those for published books and specification documents.

The writer has included this note because it might be argued that equal weight should not be given to unpublished or informal writing and other unrefereed material, alongside refereed papers such as conference proceedings and contributions to, for example, ACM and IEEE journals. The writer has found nothing in refereed journals that would insubstantiate the findings in the unrefereed material and has found, further, some of the refereed material referencing the same unrefereed sources¹ that she has used. She has also been extremely careful in that the unrefereed material used has been authored by practitioners in the field of web services who have either (a) written the standards and specifications, (b) been major driving forces in the implementation of successful and widely-used web services, or (c) contributed to the approved literature on the subject through books published by well-known publishers in the field.

¹ See Kumar, Das and Padmanabhuni, *WS-I Basic Profile: A Practitioner's View*, Proceedings of the IEEE Conference on Web Services, 0-7695-2167-3/04, which cites two papers published on the IBM Developer Works site. See also Vinoski, *WS-Nonexistent Standards*, IEEE Internet Computing, November/December, 2004 which references an article the MSDN site and even mentions blogs. See also Goth, *Critics Say Web Services Need a REST*, IEEE Distributed Systems Online, December, 2004, which refers to several papers published on xml.com.

Copies of all the online references cited in the work are included on the accompanying CD under their short reference title, and are hyperlinked within the text.

Note 2: Acronyms

Web services suffer not only from a proliferation of standards but also from a proliferation of acronyms. Except when it appears in a citation, the first time an acronym is used in the text it is accompanied by its full form in italics. For ease of reading, a list of acronyms and their meanings has been included at the end of the text. Terms such as XML, HTTP, HTML and J2EE are considered to have passed into common use and have not therefore been listed. After its first use the WS-I (*Web Services Interoperability Organization*) Basic Profile is referred to by its shorter name of Basic Profile. Although Sun has recently officially removed the "2" from J2EE, it is still common practice to include it in the acronym, and it has therefore been retained.

Note 3: Capitalization

Although the term "web service" is frequently capitalized as "Web Service" or "Web service" in normal use, the less intrusive non-capitalized "web service" form has been used throughout this thesis. This decision was based not only on personal preference but also on the argument put forward in the June 2004 draft addition to the current Oxford English Dictionary: "Originally written with a capital initial, web compounds are now increasingly written with a lower-case w. Since it is difficult to make an objective judgement about the dominant capitalization in particular cases and the evidence is changing too rapidly for such a judgement to be of any lasting value to the reader, the compounds below have been routinely presented with a lower case w irrespective of the quotation evidence" [[OED, 2004](#)].

RPC as the name of a technology is, like RMI, usually given in upper-case, although the norm for its use as a style value inside WSDL is lower-case. Because it is difficult in some cases to

separate the two, the upper-case form has been used throughout the text, except in quotation style (*Courier New*) referring to the WSDL value, where it appears in small capitals.

Note 4: US versus British Spelling Conventions

Where words such as *marshalling* and *unmarshalling* were concerned, British lexicographical practice has been followed and the double "-ll-" version has been used.

CHAPTER 1: INTRODUCTION

1.1 Introduction: the Issues

The focus of this thesis is a survey of interoperability, the crucial and necessary feature of web services if they are to succeed where earlier attempts at interoperable distributed systems have floundered. Focus in business enterprise is currently shifting onto wider issues of web-services integration and choreography, as discussed in Chapter 6, but this scaffolding is built on top of the core standards and cannot escape any difficulties still inherent in them. This thesis examines these core standards in the light of the issue of interoperability, considers features of selected key web-service implementations that might be a barrier to interoperability and attempts to answer the question: does genuine interoperability make web services succeed in the arena of distributed computing where middleware systems have failed?

The thesis does not take into account features that might otherwise be associated with web-service performance such as transaction management, security, load-balancing, concurrency and speed. As is mentioned in the conclusion to this thesis, it would not, for instance, be sensible, to use a web service to handle large numbers of small SOAP messages where the XML overheads would be disproportionately large and would adversely affect performance. Such a point needs to be made in the context of the usefulness of web services but does not impinge on their interoperability.

Writing a thesis about web services is like chasing a rapidly moving target. Web services constitute a computing landscape that is constantly being modified, reshaped and extended, not only through the proliferation of standards and specifications produced by the various working groups, but also by the marketing pitches of major web services tool vendors, each of which wants to reap the gains of developing the definitive product. An editorial published in the October 2004 issue of the *Web Services Journal* states the problem very clearly: "There are too many standards, at too granular a level, and too much competition" [[Rhody, 2004a](#)].

In addition to this, there are even major divisions in approach between the web-services marketing and the web-services technology arms of the same vendor, as pointed out by Birman [2004]. Also in the *Web Services Journal*, but in a different issue, the significant point was raised that there are virtually no human interfaces into web services: "Even though a large number of Web services are designed solely for computer to computer communication the continuing reality is that more Web services are designed to interact directly or indirectly with human beings" [Rhody, 2004b]. (Examples cited in texts on web services bear this out in their references to holiday booking applications, banking applications or customer orders. The Amazon and Google web services, possibly the most well-known, widely used and truly global web services, are intended for human interaction.) Representative definitions of web services discussed in Chapter 3 are, however, unconditional in their requirement of machine-to-machine communication as a defining feature.

1.2 Interoperability as the Distinguishing Feature of Web Services

The term interoperability is itself capable of several definitions. Guest derives his formal definition from his study of the ISO Information Technology Vocabulary:

Interoperability enables communication, data exchange, or program execution among various systems in a way that requires the user to have little or no awareness of the underlying operations of those systems... this is the ultimate goal of building a solution today [2003].

Here Guest distinguishes *interoperability* (enabling communication and data exchange between different platforms) from the concepts of *migration* (rewriting a component to run on another platform) and *portability* (moving a component to a different vendor's implementation but using the same platform). Guest also cites two advantages of interoperability: it enables the reuse of existing systems, and uses to best advantage the technical merit of the platform on which the component has been developed.

Senior (Sun) Java Architect Tyagi also points out that there are several possible meanings for the term *interoperability* but the one he favours is this:

... the functional characteristics of the service should remain immutable across differing application platforms, programming languages, hardware, operating systems and application data models. Web services by definition should be interoperable and the service consumer should not be tied to the service implementation [[Tyagi, 2004b](#)].

Tyagi makes clear recommendations concerning requirements for interoperability in web services, which may be summarized as follows:

- web services must comply with existing agreed standards, and their compliance with these standards must be tested against the *Web Services-Interoperability* (WS-I) Basic Profile (the Basic Profile is discussed in Chapter 4 and used as a testing agent in Chapter 7);
- toolkits used to develop web services must not:
 - conform only to a subset of the standards and depend on the implementation either of optional specifications or of ambiguities in the specifications;
 - have proprietary extensions;
 - have any customization of SOAP messages (e.g. for compression or security reasons);
- services must be tested against multiple client types;
- application integration services must be open to the need to use an intermediate data model to resolve the differences between systems;
- the creator of a web service must be aware that any complex, application-defined data types may not be fully interoperable, even though they can be represented as schema data types, because they may need custom handlers for serialization and deserialization²;
- extensions to the *Web Services Description Language* (WSDL) should be avoided.

The features listed by Tyagi are discussed in later chapters, where factors that detract from web-services interoperability are examined. Some of these features, such as proprietary extensions for web services and the customization of SOAP messages, would fall foul of the Basic Profile, to which most vendors are anxious to show conformity (*Visual Studio 2005* even has a new web-

² *serialization* is sometimes defined as the process of converting from a language-specific data type into XML, with *deserialization* being the reverse process. It is also used to refer to the conversion of, for example, a SOAP message into a TCP/IP buffer stream and its transportation between the client and the server.

service header to display conformity), and proprietary extensions are becoming increasingly unlikely as a result, although, in recent contradiction of that, Sun are now planning proprietary extensions to WSDL with JAX-WS (see page 152).

Chapter 4 mentions the limitations of the Basic Profile in its lack of support for recent standards but also discusses current web-service implementations that use only a subset of W3C Schema. Chapters 3 and 4 include some discussion of the increasingly *less* proprietary *wrapped* extension to WSDL message styles. Chapter 7 looks at testing against multiple client types and the problems of user-defined data types. Of all these concerns, the most significant is data-typing, a language issue (discussed in Chapter 3, 4 and 7). As Tyagi points out, too often it is assumed that because a data type may be expressed in a schema, it may also be adequately converted to another language on another system. Chapter 2 also discusses semantic problems arising from the need to map the types from one schema onto another that may be very different.

A further feature, central to interoperability but not raised by Tyagi, became significant from the beginning of 2005, with the publication of the *SOAP Message Transmission Optimization Mechanism* (MTOM) by the *World Wide Web Consortium* (W3C), which defines binary representations in XML. This issue is discussed in Chapter 4.

Java was chosen as the main language of implementation for this thesis and, wherever possible, either open-source, non-commercial or freely-available APIs and software were used for trial and development because it was felt that such products were more likely to be standards-compliant, and less driven by the need to create proprietary features. The other languages of implementation were C#, against which Java programs have been tested both as clients and servers, and PHP. A free editor (*SharpDevelop*) was used to create C# programs and they were run on a free C# web server, *Web Matrix*. Both these applications were chosen for their convenience, small footprint, ease of use in development and because their simplicity was closer to the tools chosen for Java development. Some testing was also done using *Visual Studio 2005* because of its very recent release and its support for version 2.0 of the .NET Framework. The PHP development was carried out both with a simple text editor and with an evaluation version of the latest *Zend Studio* because of its new support for WSDL generation. The latest version (5)

of PHP was used and tested on the latest version (2) of the open-source Apache server. PHP is a fairly recent arrival on the web-services scene but its different status as an interpreted scripting language widened the scope of the testing.

1.3 Research Methods

This thesis presents a comparative survey of the features and functions of web services from the viewpoint of *interoperability*. Its aim is a synthesis and assessment of the issues surrounding these features and functions and therefore it omits consideration of such matters as security and load-testing which might otherwise have been included. To arrive at a systematic assessment, Chapter 4 sets out a qualitative analysis of the basic components of web services and some different approaches towards building web services. Statements on web services from leaders in the field, whose often contradictory positions are validated by their considerable practical experience, are included to illustrate the contested state of the field. The contradictions inherent in the subject matter are the substance of the problem: to what extent are web services really interoperable? The conclusions arrived at are the result of a combination of analyzing web-service features and functions qualitatively and of validating the findings through practical experiments with web services in Chapter 7. A summary of the main problem areas and the experimental findings may be found in Table 7-1.

1.4 The Design

Chapter 2 of this thesis aims to place web services in the context not only of the evolution of distributed computing but also of contemporary and future distributed computing requirements. It outlines the major middleware systems that might be considered forerunners of web services, and attempts to show the extent to which web services might or might not be considered a development of these systems. Some of these systems still coexist with web services and the chapter includes a discussion of one system which attempts to combine middleware systems with web services.

Chapter 3 formulates a working definition of web services. As the arguments demonstrate, such a definition is itself a moving target. There are different interpretations and definitions of nearly all the components of a web service. We are told by Vogels that, "One of the key architects in the W3C's Web Services Architecture working group stated quite bluntly that they did not have the luxury of describing Web services in a simple manner because none of the participating vendors could agree on a single definition" [[Vogels, 2003](#)]. Chapter 3 also examines the extent to which web services have already been adopted as a business strategy, or are likely to be in the near future, and discusses the advantages offered to businesses by web-service implementations. Such an examination requires consideration of the advantages and drawbacks of some currently available web-services toolkits. Lastly, a definition of web services would not be complete without placing it in the context of *Service-Oriented Architecture* (SOA), a necessary companion to web services in the enterprise, enabling the reuse of service components and the orchestration of complex interactions between them.

As indicated above, Chapter 4 moves towards creating a taxonomy of web services by making a qualitative analysis of its constituent parts. It examines in some detail the core accepted components of web services, SOAP, WSDL and the standard for *Universal Description, Discovery, and Integration* of web services (UDDI). Each of these standards is discussed in the light of their contribution to web services' interoperability. With regard to SOAP, the chapter makes a clear distinction between messaging formats and programming models, which are sometimes confused through their identical naming. Included in this chapter is a survey of the more controversial elements of SOAP implementations and also of the implications of some of the SOAP client models. The chapter presents a fresh approach to issues and problems associated with emerging versions of several key technologies.

Chapter 5 is concerned with *Representational State Transfer* (REST), an alternative to the commonly accepted web-services model, which uses XML over HTTP both with, and more especially without, benefit of SOAP and WSDL. Because of the extreme simplicity of its model, REST web services assure interoperability. The chapter demonstrates this and examines its feasibility.

Although web-services orchestration and choreography are outside the major focus of this thesis in that they are systems or architectures for coordinating web services as a larger whole, they involve user and service interactions requiring interoperability. Chapter 6 therefore includes a brief description of the currently competing standards, *Web Services Choreography* (WSC), a recommendation by the W3C, and *Business Process Execution Language for Web Services* (BPEL), standardized by the *Organization for the Advancement of Structured Information Standards* (OASIS). The chapter also includes a brief discussion of the *Enterprise Service Bus* (ESB) and takes a brief look at the *Windows Communication Foundation* (WCF, formerly code-named *Indigo*), the new Microsoft framework for integrating web services.

Chapter 7 presents a validation snapshot of the qualitative findings in the earlier chapters, looking at web services in practice and examining sample services implemented in different ways and with different clients as a means of assessing the comparative interoperability of varying approaches to web services. One service sets out to provide means of indexing and searching web pages that might be required in the course of a research programme. In addition to the more usual types of SOAP client, a Mozilla *Firefox* extension has been implemented, which may be used to access the service. C# and PHP clients were also written for the service.

Further service samples were written as clients and servers in both C# and Java, as well as in PHP, a more recent arrival on the web-services scene. The interactions in these services were much simpler but some of the data types used were more complex to illustrate the issues raised by datotyping and XML serialization and deserialization from the viewpoint of different languages. A further section in this chapter examines the new JAX-WS web-service framework, currently still under development. Also included in Chapter 7 are conclusions regarding interoperability reached with the aid of commercial testing frameworks such as Parasoft's *SOA Test* and Mindreef's *SOAPScope*.

The Conclusion draws the thesis together, offering an assessment of the extent to which web services may accurately be seen as interoperable, suggesting credible directions and solutions, and listing possible future extensions to the work.

CHAPTER 2: FOUNDATIONS OF WEB SERVICES

2.1 Introduction

While the focus of this chapter is on the differences between web services and the technologies that preceded them, it aims to keep in view the fact that web services can complement earlier methods of distributed computing. The chapter examines different client/server and middleware architectures including CORBA, RPC, RMI and Message-Oriented Middleware, comparing them with features of web services that offer similar functionality.

2.2 Similarities and Differences

Past and present solutions to the problems of distributed computing necessarily have much in common. Slama *et al.* point out that a successful web services architecture will need to "embrace existing and upcoming technologies instead of replacing or excluding them" [2004]. Indeed, as Slama *et al.* go on to demonstrate, the modes of communication adopted by all types of distributed computing are variants on either synchronous or asynchronous mechanisms and, whatever the approach, additional runtime features are required such as "security support, fault tolerance, load balancing, transaction handling, logging, usage metering, and auditing".

Slama *et al.* also discuss a major difference between older and newer approaches. While the earliest attempts at distributed computing were very network-dependent in that they necessarily took a low-level approach to details such as socket programming, later attempts benefited from the availability of higher-level protocols, such as TCP/IP, which abstracted that dependence away until it is today possible to use a toolkit to "write" distributed programs without any knowledge of how the network actually works or of what is taking place when the code is executed. The problem with this situation is, of course, that when something goes wrong, it can be very difficult even to locate the problem, much less fix it. Slama *et al.* warn: "...it is the experience of the recent two decades that the developer's awareness of the distribution is still crucial for the efficient implementation of distributed software architecture."

2.3 Client/Server: the Beginnings of Distributed Computing

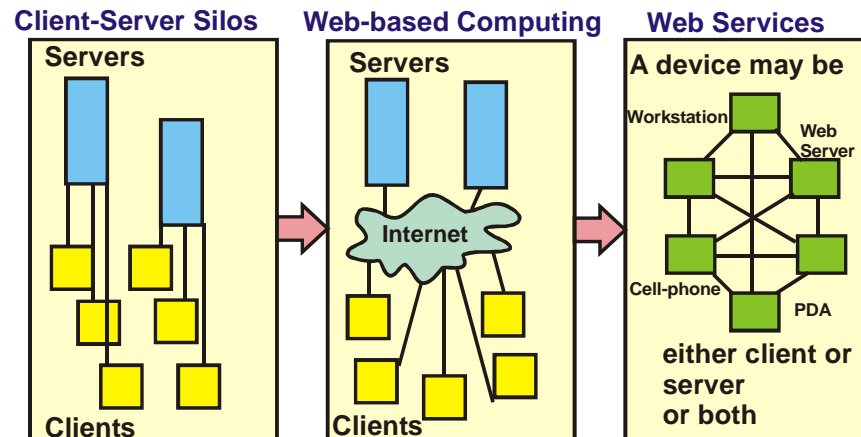


Figure 2-1: A Web-Centric View of Distributed Computing

Distributed computing began with the client/server model, with a local machine as a client which requested some kind of processing from another local machine as a server on an intranet. Later, client browsers requested processing from web servers across the internet, using the universally accepted standards of HTML, HTTP and TCP/IP. Distributed applications, distinct from browsers, also requested processing from servers across the internet. Such requests could not always be fulfilled by the computing power of one machine alone and multiple servers worked (and still work), processing in parallel, reaching solutions through distributing the processing.

The development of a web-service model can partly be illustrated in the much simplified diagram in Figure 2-1, redrawn from Kleijnen and Raju [2003], where the earliest silo pattern (a tightly-coupled system with dedicated hardware and software) is shown as the beginning of a movement to web-based computing. The arrow linking the second and third parts of this diagram suggests the same kind of evolutionary pattern as that evident between client-server silos and web-based computing, but hides a greater distance between these later systems, which differ from one another to a greater extent.

Although this diagram is drawn from a rather narrow, web-centric view of distributed computing, which includes neither the significant features of middleware, to be discussed later in this chapter, nor the development of parallel computing, it does illustrate the movement away

from the tightly-coupled, client-server model towards a peer-to-peer model, in which any network node may be a server and any network node may be a client; in which a network node does not have to be a desktop computer to assume either role but may be a cell-phone or a PDA or, in the future, perhaps a fridge, a car or even a house; and in which multiple nodes may be involved in the same service.

2.4 Middleware Communication Architectures

2.4.1 The Remote Procedure Call (RPC)

At its most basic level, RPC can loosely be defined as the exchange of network packets between two physically remote systems which have no shared memory but appear to operate transparently as one "local" system by which the programmer is shielded from the mechanics of the message passing.

A tighter definition of RPC is provided by the Open Group [[The Open Group, 1997](#)] which provides two complementary aspects of the RPC model:

1. the client/server paradigm in which the meeting-place for both client and server is the programming interface, and
2. the "program/stub/run-time system", which spreads the responsibility for implementing an RPC call between application code and a number of RPC components, many of them invisible to the programmer. For example, stubs (or proxies) on both the client and the server carry out processes which are not actually generated by the programmer, but are controlled by the specifications of an interface definition language, to handle the interface between code and the run-time system. This approach conceals the communication process from the programmer and is closer to the CORBA (*Common Object Request Broker Architecture*) or RMI (*Remote Method Invocation*) model.

Because there must be a run-time binding between client and server, RPC by its nature is not loosely-coupled. Such a binding involves the knowledge by a client, also known as a "requester agent" [[Booth and Liu, 2005](#)], of the service endpoint or network address. The Open Group

explains: "In order for an RPC to occur, a relationship must be established that ties a specific procedure call on the client side with the manager code that it invokes on the server side" [[The Open Group, 1997](#)].

At the start, many incompatible RPC systems were developed with equally incompatible protocols. RFC³ 1057 [[Sun Microsystems, 1988](#)] was originally drawn up in June 1988 by Sun Microsystems, with input from IBM and BEA, partly as a response to the need for platform-independent communication structures that might access file systems across the Internet. An Open Standards version (or Version 2) was later produced by Sun, in 1995, as RFC 1831. Of particular interest is section 7.4 of this second version which describes "Other Uses of the RPC Protocol" and states:

The intended use of this protocol is for calling remote procedures. Normally, each call message is matched with a reply message. *However, the protocol itself is a message-passing protocol with which other (non-procedure call) protocols can be implemented* [my italics] [[Srinivasan, 1995](#)].

At an equivalent point in the earlier version, RFC 1057, this feature had been presented apologetically as a possibly erroneous use of the protocol. The movement away from the original meaning of RPC was further evidenced in a technical article on document-based web services, on the Sun Developer website, which explained:

Although JAX-RPC [*Java API for XML-based RPC*] and its name are based on the RPC model, it offers features that go beyond basic RPC. It is possible to develop web services that pass complete documents and also document fragments [[Tyagi, 2004a](#)].

An even more recent article explains almost apologetically: "JAX-RPC is somewhat of a misnomer, since it supports both RPC-style and document-style web services" [[Panda, 2005](#)]. And in May 2005 the final apparent repudiation of RPC came in the announcement on an official Java site that "JAX-RPC 2.0 has been renamed to JAX-WS 2.0 (Java API for XML-Based Web Services)" [[Kohlert et al, 2005](#)].

³ RFC is incorporated in the names of a series of documents about network computing standards and is an acronym for *Request for Comments*.

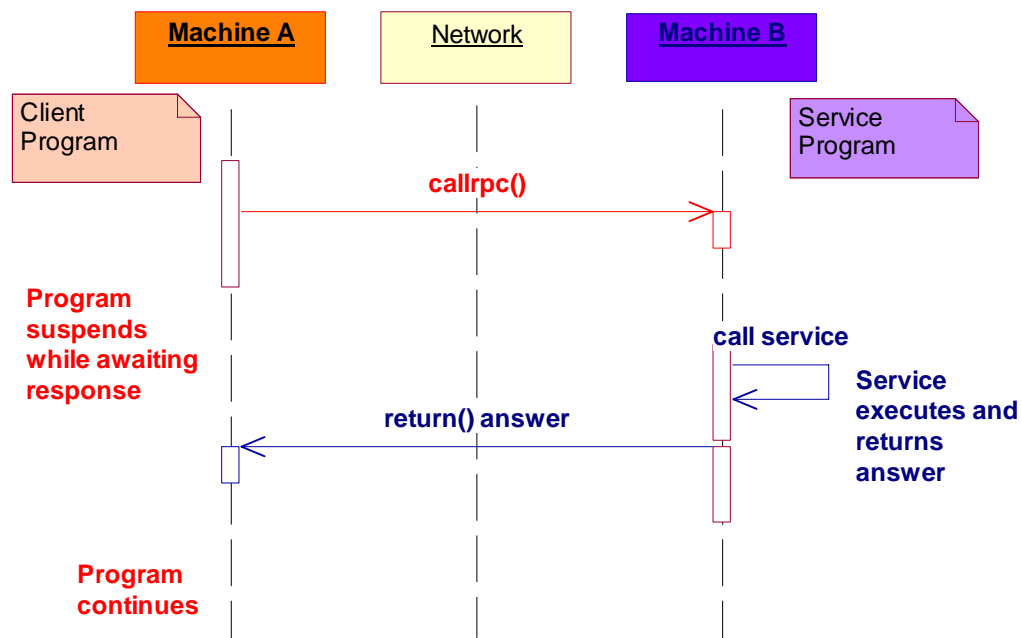


Figure 2-2: The Remote Procedure Call

The processes involved in a Remote Procedure Call can best be illustrated by the sequence diagram in Figure 2-2, redrawn from Thomas *et al.* [2003]. Here a program on a distant machine may be invoked by a local program through a process in which the network appears transparent to the caller. RPC calls are usually synchronous but the Java API for XML-RPC 1.1 standard specifically allows for asynchronous messaging as well. The diagram illustrates the synchronous pattern of RPC calls, in which the calling program blocks until a reply is received. RPC represents tight-coupling in that it needs to be implemented on both ends of a service call and because the client blocks until a response is received from the service, leaving wide open the possibility of delay occasioned by network problems.

2.4.2 The Distributed Object Model

RPC was widely adopted in the 1980s. A similarly prominent position in the 1990s, in terms of programming style, was occupied by object orientation, which created a need to find the means of exchanging distributed objects across the Internet. Raj makes the connection between object orientation and distributed-object systems in an OMG (*Object Management Group*) paper: "Distributed object computing extends an object-oriented programming system by allowing

objects to be distributed across a heterogeneous network, so that... these distributed object components interoperate as a unified whole" [1998].

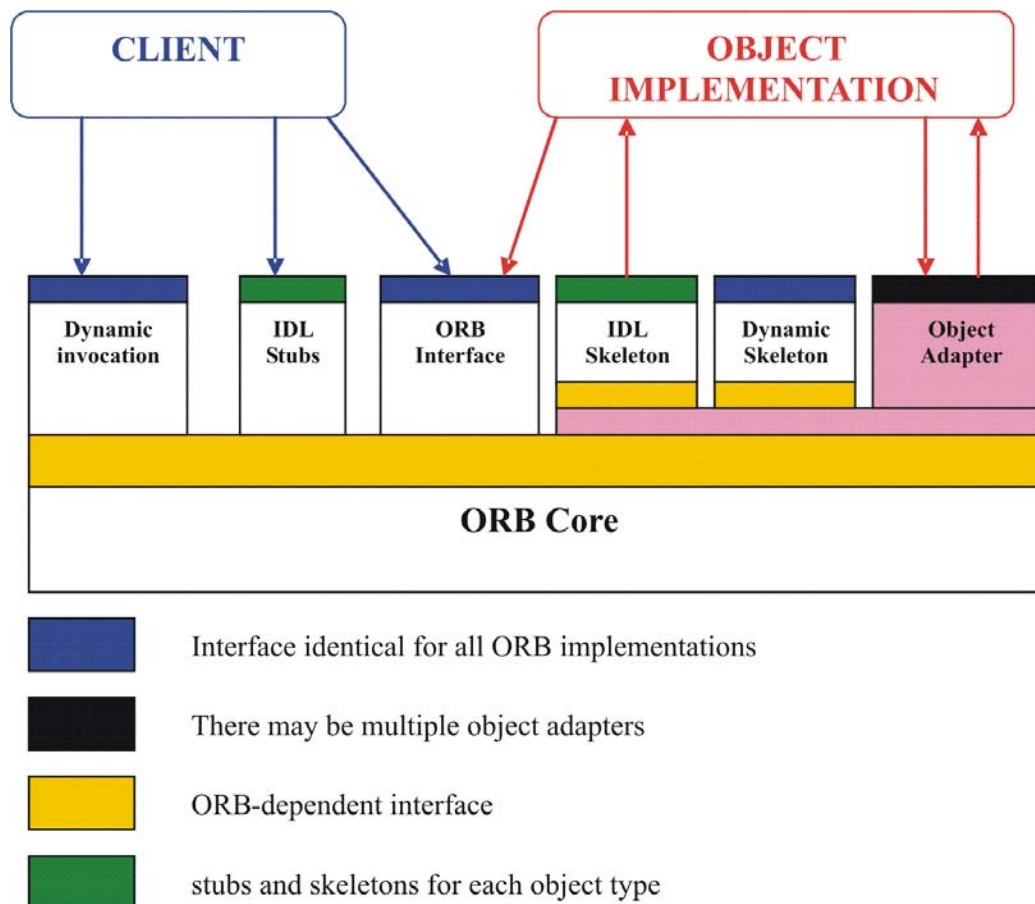


Figure 2-3: The Main Components of the ORB Architecture and their Interconnections [Keahey, 1998]

Britton and Bye define the difference between earlier middleware systems and the distributed object model: "Instead of client and server, there are client and object" [2004]. The need for change was also fuelled by the movement away from 2-tier client/server systems to 3-tier and eventually n-tier systems, in which typically the application logic is divided across several layers, including a database layer, and in which the system is composed of reusable, discrete components. Systems developed to enable the exchange of distributed objects included the *Common Object Request Broker Architecture (CORBA)* developed by the *Object Management Group (OMG)*, a non-profit consortium. Implemented differently (and expensively) by different vendors, it experienced major interoperability problems and its history is surely an object lesson (in the more general sense of the word) for web-service tool vendors.

2.4.2.1 CORBA

A CORBA application includes an *Object Request Broker* (ORB), a client in one location and a service implementation in another. Figure 2-3 shows the interconnections of the ORB architecture. The ORB is crucial to interoperability because it matches the client to the server and provides the communication mechanisms needed by each in order for the other to be understood. Interoperability problems arose initially in the lack of a defined protocol to be used between ORBs, but the development of first the *Internet Inter-ORB Protocol* (IIOP) and then of an interface in terms of the *Portable Object Adapter* (POA) went some way towards solving these problems. The development process, however, took too long and was bedevilled by the addition of vendor extensions to the basic standards, by the lack of implementation by vendors of later OMG specifications, and possibly also by the interaction complexity that arises from the use of multiple interfaces.

2.4.2.2 Component Object Models

At the same time that the OMG was presenting CORBA as the solution for distributed systems, Microsoft was developing the *Component Object Model* (COM), with interoperable, reusable components which, in the case of object-oriented programming, could create objects and make them available to external clients. After COM came the *Distributed Component Object Model* (DCOM), which allowed for objects or components to be accessed across a network. These components were proprietary in that their use was limited to Microsoft Windows operating systems and they cannot therefore be considered interoperable in a wider sense.

Microsoft was not alone in developing a component model: Sun's simpler component model was the *JavaBean* which runs inside a *Java Virtual Machine* (JVM) and has therefore the added advantage of platform independence. JavaBeans are still a crucial part of the interoperability of Java-based web services in that they provide the means for serializing and deserializing complex objects to and from XML. Sun also developed the Java-specific (and therefore not interoperable across some platforms) *Remote Method Invocation* (RMI), which established mechanisms

whereby an object inside a JVM running on a distant machine might be manipulated. RMI was preferred by Sun to RPC, because "RPC... does not translate well into distributed object systems, where communication between program-level objects residing in different address spaces is needed" [[Sun-RMI, 2004](#)].

2.4.2.3 Transactional Component Middleware

Both Microsoft and Sun went on to develop what Britton and Bye first termed *transactional component middleware* [2004], a term describing the use of a container for the components, which provides extra services such as transaction management and resource pooling. Neither Microsoft nor Sun, however, solved the interoperability problem that had troubled distributed computing almost from its outset. It might even be said that object-orientation exacerbated the problem by requiring tighter coupling between the systems. For the object on a distant machine to be accessed, the program on the local machine must first have a reference to the object, in the same way that a program must have a pointer to the address of a local object it wishes to use. Such a system does not work well across language and platform boundaries, and web services have replaced object references with endpoints⁴ as the definition of a service.

2.4.2.4 Integrating Legacy Systems

A major focus for current web services, as well as a significant reason for their attractiveness to the corporate mind, is the legacy system, usually an earlier form of middleware, which needs to be made more widely accessible through integration into a more modern distributed system. Integration problems can arise because of the differences between the old and the new systems. As has been seen, legacy middleware frequently suffers from a tight coupling of client and server. Vinoski argues that the process of integrating such systems into web services is

fraught with problems. One of the worst is that it causes inappropriate details (about protocols, type systems, interaction models, and so on) to show through the Web services

⁴ An endpoint is the combination of a network address and a port at which a service may be accessed.

level from the underlying systems, destroying the service encapsulation and isolation that Web services are supposed to provide [[Vinoski, 2002](#)].

He argues that while a web service, such as one that might handle purchase-order documents, is coarse-grained, stateless and loosely-coupled, legacy systems that might be integrated within it will more likely be fine-grained, stateful and closely coupled. His example of an integrated legacy system is a credit-card authorization process, which would run naturally over RPC because further purchase-order procedures could not continue without receiving that response.

Medicke and Pack argue on similar lines to Vinoski, using for their illustrations of coarse-grained web services the processes of ordering a meal in a restaurant, and using a High Street ATM: " How would the restaurant-goer's experience be if there [were] a different process for ordering each part of the meal, or when withdrawing cash at an ATM, there were a hundred different menu options. Service interfaces are expected to be simple and intuitive" [[Medicke and Packe, 2003](#)]. It was the complexity of fine-grained systems that was partly responsible for preventing their wide adoption: fine-grained systems lay themselves open to proprietary techniques for dealing with the details.

A related issue for interoperability with legacy systems is raised by Halevey [[2005](#)] in his discussion of the problems of semantic heterogeneity for database schema. Because legacy systems were built sometimes haphazardly, and some of the data required to be used may not even be structured, data views may have to be assembled from multiple sources which require mappings to preserve the semantics. These mappings are difficult to create and, according to Halevey, prone to errors. One example Halevey cites is that of Amazon. Companies whose products are exposed on the Amazon website are required to conform to a particular schema which may bear little correspondence to any schemas used internally by the companies. In such a situation, multiple mappings may be possible and there is room for confusion. Halevey points out that:

Resolving schema heterogeneity is inherently a heuristic, human-assisted process. Unless there are very strong constraints on how the two schemas you are reconciling are

different from each other, one should not hope for a completely automated solution [[Halevey, 2005](#)].

Because such processes are difficult to automate, simply wrapping such a system as a web service may not achieve interoperability. There is the added difficulty that the complexity of such a system may not lend itself to exposure through a web-service registry, when method and parameter names "do not capture the underlying semantics of the Web service". Halevey cites an attempt to resolve that problem in the form of a search engine, Woogle⁵, which:

is based on analyzing a large collection of Web services and clustering parameter names into semantically meaningful concepts. These concepts are used to predict when two Web service operations have similar functionality [[Halevey, 2005](#)].

Halevey does not consider it ultimately impossible for the complicated mappings to be automated but he does argue that the tools necessary to accomplish this are not yet available and that it will take time for them to be developed. Like the examples cited by Vinoski and Medicke and Pack, the problem for web services lies in the fine-grained nature of legacy systems and the data they expose.

2.4.2.5 Web Services versus Distributed Objects

A further difference between web services and traditional distributed object middleware lies in the design approach of the two types of system. Web services are best thought of as "top-down", aiming first to identify the business process to be modelled, while distributed model architectures can be thought of as "bottom-up" in their emphasis on technical details at a low level. Legacy system integration can also be considered a "bottom up" approach in that it can start from an existing low-level application. In a pure "top-down" approach, WSDL is supposed to be the starting point of a web service, although many web-service toolkit vendors, anxious to pre-empt the learning curve that WSDL requires, ignore this requirement and attempt to generate the WSDL for each service from the code written for it. The problems caused by this approach are discussed in Chapters 4 and 7.

⁵ See www.cs.washington.edu/woogle.

With object-oriented languages such as Java, C# and, more recently, PHP, objects form the base of the coding and creation of a web service, but web-service programming objects are not passed between the client and the service. Only public methods are exposed in a service interface and only a data representation of the public fields may be exchanged. An object has state, but web-service messaging is stateless (there is no way of defining state in WSDL). Objects have traditionally been serialized in a binary encoding, while web services serialize data in XML. Recent technology may make both these approaches available to web services, although whether such methods are counter to the interoperability objectives of web services remains to be seen. The original developers of XML certainly thought so [[Bray, 2001](#)], at least in terms of message transmission between PCs, though it would be contrary to the principles of interoperability to think only in terms of one platform, particularly when there is currently a great expansion of possible platforms across multiple devices.

Vogels is in no doubt about the differences between web services and distributed object systems, itemizing his reasons as follows:

Web services share none of the distributed object systems' characteristics. They include no notion of objects, object references, factories, or life cycles. Web services do not feature interfaces with methods, data-structure serialization, or reference garbage collection. They deal solely with XML documents and document encapsulation [[Vogels, 2003](#)].

Of interest in this context is that JAX-WS has removed the need for web services to create an interface, relying now wholly on the implementation class and thereby possibly acknowledging that WSDL is a sufficient interface.

2.4.2.5.1 Comparisons Between Web Services, CORBA and RMI

In contrast to Vogels, Gray's comparative study of web services, CORBA and Java-RMI [[2004](#)] makes a convincing argument that there are more than superficial similarities between distributed object systems and the static stub client model for web services (discussed in Chapter 4). His diagram in Figure 2-4 illustrates the similarities by showing how a client-side stub can be

generated from an interface definition in web services, in CORBA and in Java-RMI. The similarity between Java-RMI and web services is particularly significant in that the J2EE 1.4 implementation of web services is built upon the earlier technology of servlets and RMI. All three approaches here are shown by Gray to start with a type of interface from which a client stub may be produced – although the interface type differs between Java-RMI, where it is an interface in *code*, and web services which ideally should start not with code but with a WSDL definition, the focus of section 4.3.3 in Chapter 4.

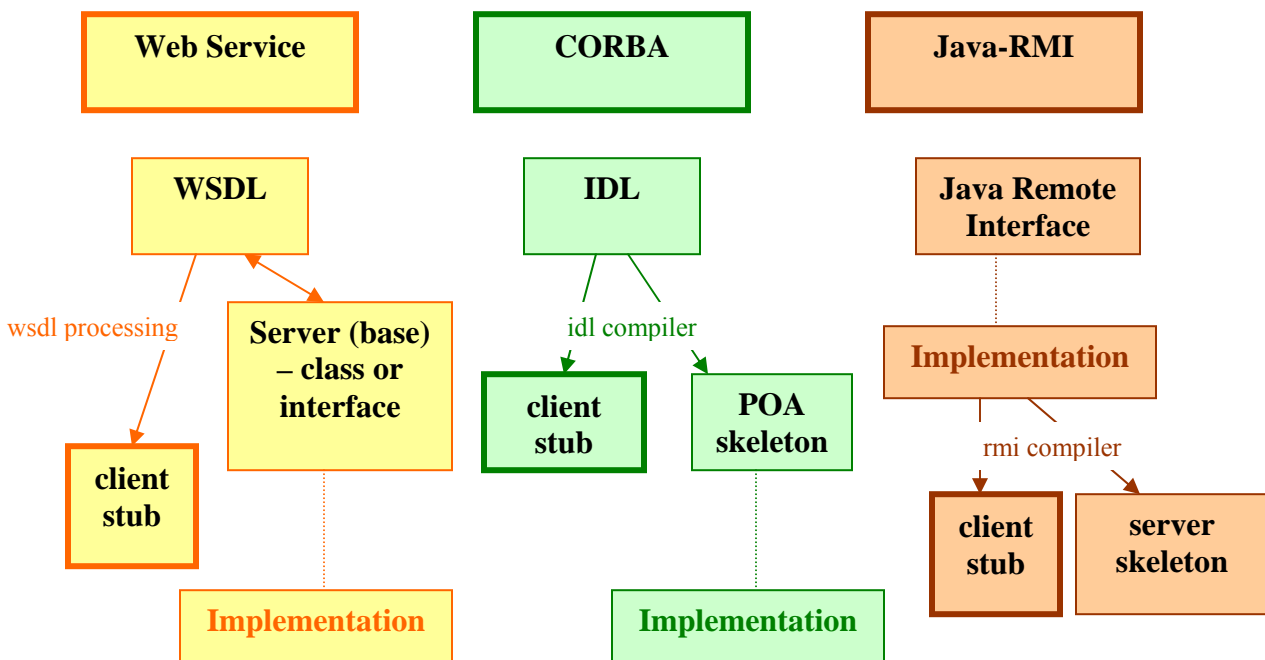


Figure 2-4: Generation of Client and Server Components from Interface for Web Services, CORBA and Java-RMI [adapted from [Gray, 2004](#)]

Burner emphasizes a similar difference between web services and CORBA when he explains: "Making the messages and the service contracts the design centre of web services is the fundamental difference between the web-services architecture and CORBA" [[Burner, 2003](#)]. Gray offloads the blame for ignoring this architectural ideal onto the JAX-RPC and .NET development environments, which not only permit but also encourage the model to be developed in reverse, and adds: "This bottom-up approach is easier for most developers" [[2004](#)]. He also dismisses the differences between a WSDL file and a remote interface definition with the comment that: "The inclusion of the service end point's URI⁶ is really the only major semantic

⁶ Uniform Resource Indicator

difference from an IDL or `java.rmi.Remote` interface declaration", making light of the significance of the encoding and the focus on message passing crucial to web services.

Technology	Total Latency	Total Packets	Total data transferred in bytes
WS	0.11s	16	3338
CORBA	0.48s	8	1111
CORBA & name server	0.86s	24	3340
Java RMI	0.32s	48	7670

Table 2-1: Gray's results for a single (simple) exchange

Further differences between the three approaches begin to appear in the communication process. While all three employ connection-oriented protocols, CORBA and Java-RMI use tightly-coupled methods that require state to be maintained, while web services need statelessness in order to be loosely coupled.

Gray's comparative findings on speed and efficiency vary, depending on the complexity of the data transferred in the examples he uses. For a single exchange with *simple* data, the web-services technology was both faster and more efficient, unless CORBA was run without a name server, as Table 2-1 above illustrates.

Gray pinpointed a significant difference in the payload size for his web-services example as opposed to that for CORBA and Java-RMI. Web services can be and frequently are used for the transmission of much more data than is usual in the other two technologies, and the transmission of text with XML tags further increases the size of the data to be transmitted. He found that the number of packets transmitted was much greater for web services in all cases, as can be seen from his statistics in Table 2-2.

Gray's simple "Calculator" example, with four arithmetic methods accepting and returning integers, transferred only numerical data. His "iterator" example repeatedly invoked the service inside a loop. The "data" service returned a data structure, while the "large data" example modeled the retrieval of potentially much more complex data from a database.

Even with the use of a *Serial API for XML* (SAX) parser, which does not have the considerable memory overhead of a *Document Object Model* (DOM) parser, there was still significantly greater system-resource usage for the web-services "data" and "large data" examples. Gray found the average memory usage for the "large data" example were (for web services) 4.75MB, (for Java-RMI) 2.6MB and (for CORBA) 2.0MB. With the "iterator example", however, the web-service system performed at a level closer to CORBA (~2.1MB), when it was adapted to become a hybrid of web services and CORBA. This greater economy was achieved by a sacrifice of statelessness and loose coupling, both necessary features in web services.

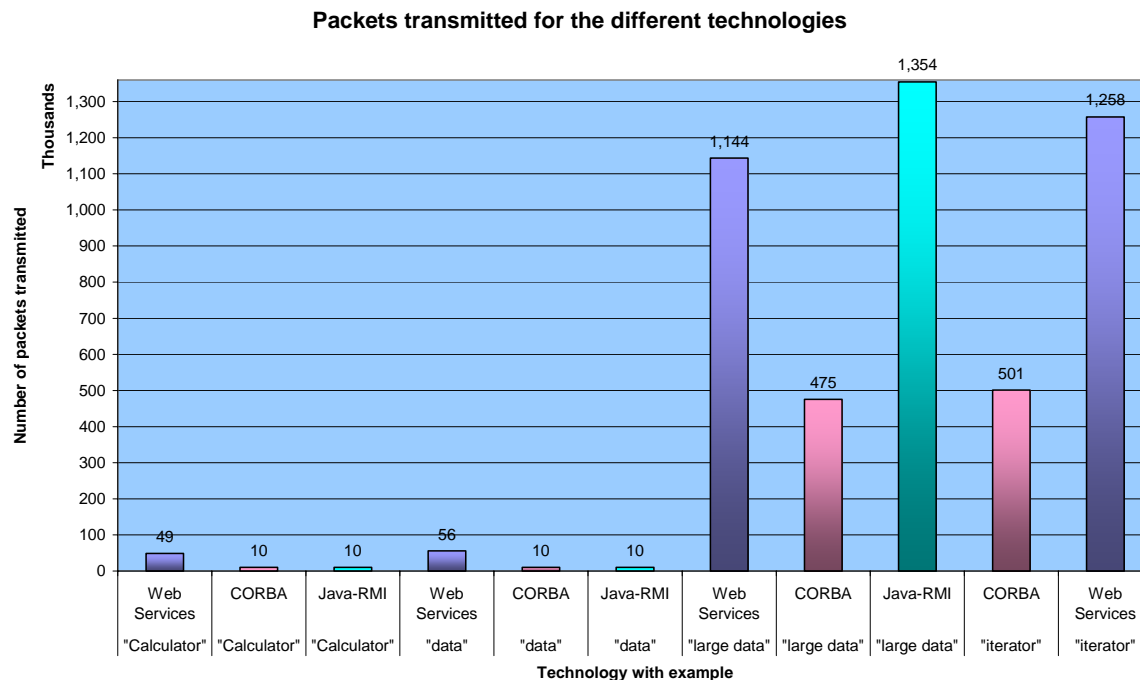


Table 2-2: Packet Numbers for the Different Technologies [Jeckle et al., 2004]

With a package size of nil, similar results were initially demonstrated in a DaimlerChrysler presentation, illustrated in Table 2-3, which aimed to compare Java-RMI, CORBA and SOAP [Jeckle et al., 2004]. Jeckle et al., however, found that, with large payload sizes, the package size differences virtually disappeared, with CORBA ultimately displaying the heaviest package size. They came to the conclusion that "package size scales linear with payload size", but that response times grew exponentially with package size, almost evening out the higher cost of the HTTP overhead for SOAP. Larger payload sizes produced a better performance for SOAP sent

directly over TCP rather than over HTTP. They reached the conclusions that the SPEC 2000⁷ value of the server correlated well with web-service performance; that HTTP, rather than SOAP, was the performance bottleneck; and that there was plenty of room for optimizing web services to increase their performance further.

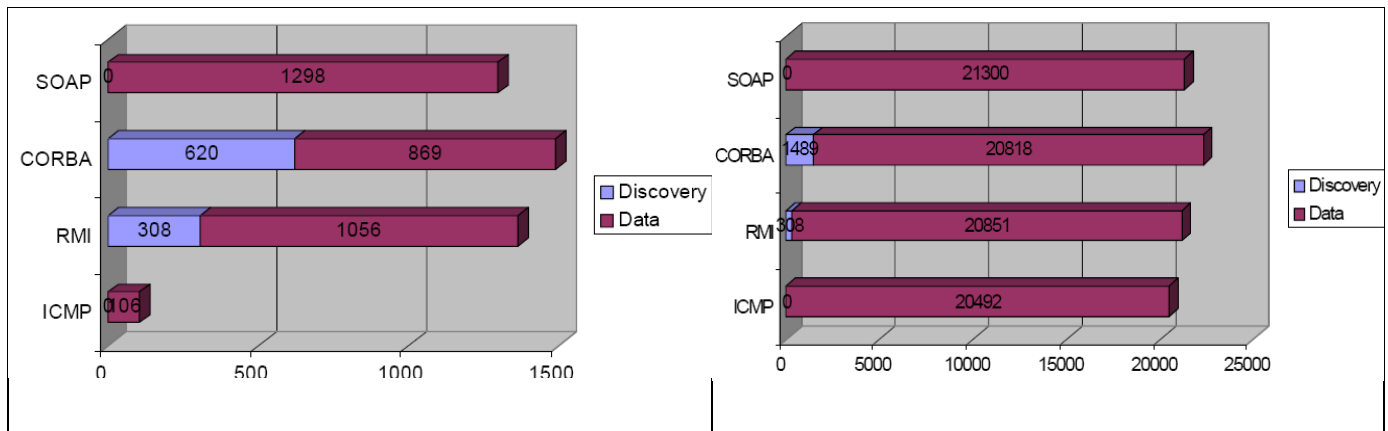


Table 2-3: Package sizes with a payload first of 0 bytes and then with a payload of 10,000 bytes [Jeckle *et al.*, 2004]

In other sections of his investigation [2004], Gray tossed aside the central question of data representation in web services with a casual: "Tiresome practical details like common data representations are avoided through the use of textual representations." His major concerns were comparative message transmission times, CPU and memory usage, and his testing was all local and Java-based. His study raised interesting questions about the usefulness of session state, a feature of both CORBA and Java-RMI, and he produced helpful data on both latency and system resource consumption, but his omission of the interoperability and data representation issues made his study significantly incomplete. RMI works properly only when there is a JVM on both the web server client and the server machines. Because the central issue of web services is interoperability, RMI is an inappropriate tool for making web-service calls.

2.4.2.6 Recognizing the Remoteness of Distributed Objects

Half a decade before the term "web service" was first used, Waldo *et al.* explained the error of considering it possible to treat remote objects as if they were local objects:

⁷ SPEC 2000: SPEC is an acronym for the *Standard Performance Evaluation Corporation*, which aims to produce fair computer benchmarking. See <http://www.spec.org/>.

There are fundamental differences between the interactions of distributed objects and the interactions of non-distributed objects. Further, work in distributed object-oriented systems that is based on a model that ignores or denies these differences is doomed to failure, and could easily lead to an industry-wide rejection of the notion of distributed object-based systems [[Waldo et al., 1994](#)].

Waldo *et al.* argue that it is impossible to separate the interface of an object from the context in which it is used. They also give an all too familiar description of a ten-year cycle in which a new distributed computing paradigm is proclaimed, alongside a new programming model, without any real change in the numbers of distributed applications because the real problems are left unsolved: "partial failure and the lack of a central resource manager". Interestingly, from the viewpoint of web services, they suggest that part of the solution might lie in greater awareness of communication patterns and argue for transparency of boundaries between local and remote objects. This might be represented in web service terms as the boundary between client and service.

It is, however, the issue of interoperability, together with the related issue of data representation, which pose the most significant problems for distributed computing. The distributed object model failed to find effective solutions for either. The vendors of products implementing the model defeated themselves with proprietary systems that became too costly to implement on a large scale and did not interoperate with each other. It can even be argued that the vendors did not actually want interoperability because they each wanted their product to be the only one to succeed.

2.4.3 Message-Oriented Middleware

With the precursor to IBM's WebSphere in the mid 1990s came the beginnings of *Message-Oriented Middleware* (MOM). While the focus of RPC is on the method call, the focus of messaging software is on the *message* and its delivery. The messaging mode adopted by MOM is loosely coupled and therefore has the advantage of being usually, but not exclusively, asynchronous. It includes not only messages (each consisting of a header, used for network

routing, and a payload of business information to be exchanged, sometimes formatted in XML) but also message queues which may be described as a generalized type of mailbox. The function of the queue is as a decoupling mechanism, in that it is a repository for the messages – a truly *middleware* system – that makes it possible for sender and recipient to be disjointed and for the recipients to be individual or multiple, as in email. (The distinction between modern MOM systems and email is that, while email may be used either for person-to-person communication or for application-to-application communication, MOM is used purely for the latter.) MOM also moves away from the client-server paradigm into a peer-to-peer model in which one message may be sent to any number of recipients by multiplexing.

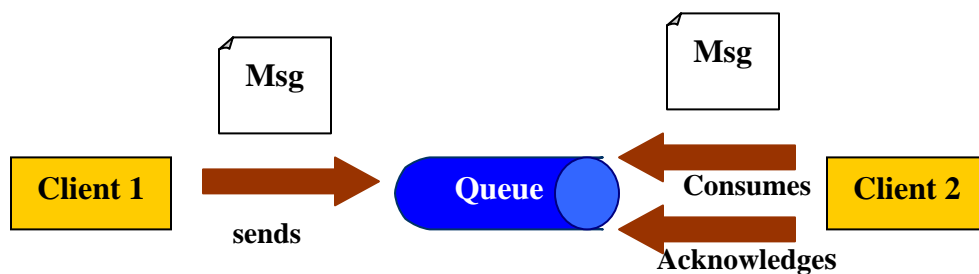


Figure 2-5: The Point-to-Point MOM Option

There are two different messaging subsystems: *point-to-point*, where one queue is used for one sender and one recipient, as illustrated in Figure 2-5, redrawn from the J2EE (1.4) Tutorial [[Armstrong et al., 2005](#)], and *publish-subscribe*, where one *publisher* can send data to those who have *subscribed* to the topic and who will probably be unknown to the *publisher*. This system is illustrated in Figure 2-6, also redrawn from the J2EE Tutorial.

The queues can be chained as they are in an email system, where queues feed into each other, and different quality of service levels may be activated depending on the services supported by a particular system.

MOM provides a bridge to web services, not only because of its loosely-coupled and asynchronous mechanisms and its focus on the exchange of messages or documents, but also because of the promise it offers of reliable message delivery. It differs considerably from distributed object technologies in its loose-coupling and asynchronous mode. As the emphasis in the web services world moves away from traditional RPC and closer to the idea of a messaging contract, far from being only a precursor to web services, MOM technologies are increasingly

being coupled with the web-services protocol stack, particularly as a way of handling legacy systems and applications. A July, 2005 newsletter from O'Reilly Network's ONJava.com questioned:

Is the era of the Remote Procedure Call (RPC) fading?.. the nature of network activity with its latency, unreliability, and potential for asynchronicity, make it ideally suited for a different approach. This explains the growing interest in messaging-oriented systems [O'Reilly, 2005].

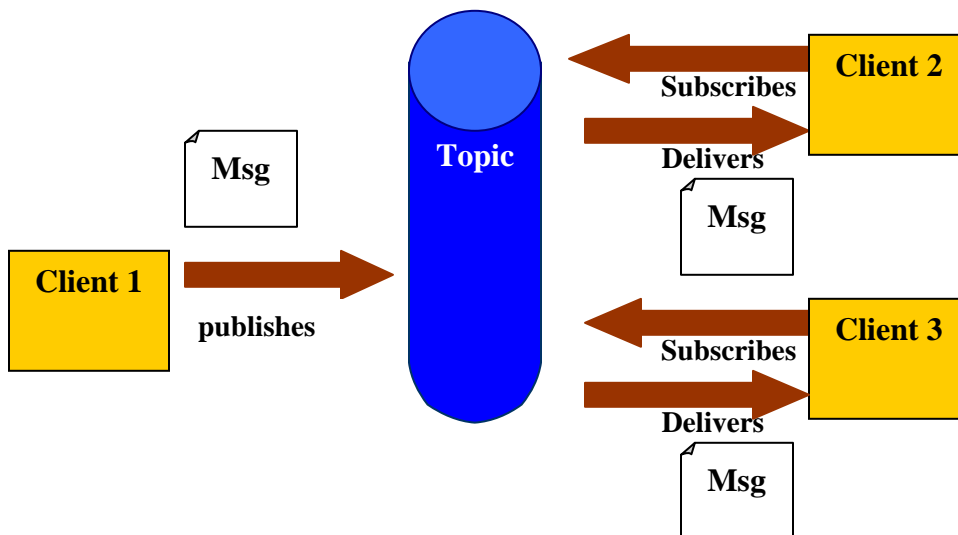


Figure 2-6: The Publish/Subscribe MOM Option

The current Java Messaging technology is *Java Message Service (JMS)*, first made available in 1998. It differs from earlier MOM systems in that it supports both the point-to-point and the publish/subscribe messaging subsystems, making it possible for them to be combined within one application. Apache Axis incorporates the use of JMS as a transport protocol. Although the asynchrony of message-oriented web services looks very attractive, it is more tightly coupled than it appears, in that the client needs to use the same implementation as the service provider.

Vinoski [2002] focuses on the similarity between messaging systems and web services in his statement that: "Messaging-based systems essentially let data travel from one service to another, allowing each to process the data as necessary without tightly coupling the services". He argues, however, that web services, far from reinventing the wheel of message passing, have a

significant difference in their implementation of open standards which promote scalability and interoperability, not present in the earlier, proprietary messaging systems.

2.4.4 Summary

The discussion so far has attempted to sketch the background for the emergence of web-services technologies over the last few years. The situation is confirmed by Manes, who summarizes her version of the reasons that have driven the search for an improvement upon traditional middleware frameworks [Manes, 2003]:

- Traditional middleware doesn't support heterogeneity.
- Traditional middleware doesn't work across the Internet.
- Traditional middleware isn't pervasive.
- Traditional middleware is hard to use.
- Traditional middleware is expensive.
- Traditional middleware maintenance costs are outrageous.
- Traditional middleware connections are hard to reuse.
- Traditional middleware connections are fragile.

Manes argues that web services potentially solve all of these problems in that they are not only language-, device- and platform-neutral but they are also more efficient and reusable through their loosely-coupled architecture. Web services are also cheaper and faster to implement than traditional middleware solutions. Manes' approach suggests that web services are indeed the next step. Vogels, however, defines web services as distinct from what has gone before in that, far from being the next step, they are messaging technologies, or "distributed *systems* technologies...now sometimes deployed in areas where distributed *object* applications have failed in the past [author's italics]" [[Vogels, 2003](#)].

Vogels' refutation of the idea that web services developed out of distributed object systems is supported by the naming change for the SOAP protocol, one of the key web-services protocols, discussed in Chapter 4. SOAP began its life as an acronym for *Simple Object Access Protocol* but this was changed in the most recent 1.2 specification document, which declared somewhat

tersely: "In previous versions of this specification the SOAP name was an acronym. This is no longer the case" [[Gudgin et al., 2003](#)]. Vogels is thinking specifically of distributed object technologies rather than the message-oriented middleware mentioned in the previous section, so in a sense he is both right and wrong in his claim. Such arguments naturally lead on to the question discussed in the next chapter: how are web services to be defined?

CHAPTER 3: DEFINING WEB SERVICES AND THEIR CURRENT STATUS

3.1 Introduction: Definition Problems

It might seem surprising that, nearly six years into web-service development, finding a definition is still a problem. That this is the case is confirmed in the latest draft release (October, 2005) of JSR 109, *Implementing Enterprise Web Services v. 1.2*, in which it is stated that "There is no commonly accepted definition for a Web service" [[Pandey, 2005](#)]. This chapter aims to reach a working definition of web services and to explain the various problems encountered on the way, some of which are semantic but others of which are founded in different approaches to distributed computing. A good place to start is with the W3C Web Architecture Group's definition of web services:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [[Booth et al., 2004](#)].

There is nothing controversial about this definition except, perhaps, its inclusion of "Web-related standards", not surprising in a document from the W3C, but open to challenge from those who see the "web" in web services as more of a starting point for less web-focused, service-oriented architectures. The definition is wedded to SOAP and WSDL (both standards also released as recommendations by the W3C), as well as (more conditionally) to HTTP, but it includes no mention of UDDI, the third member of the usual web-services triumvirate, suggested reasons for the absence of which will be discussed at the end of Chapter 4. It is unconditional in its requirement of interoperability and that the interaction will be between machines, not the human interface sought by Rhody and mentioned at the start of Chapter 1. One of the members of the working group that produced it later described the definition as, "vaguely interesting but not

really that useful (at least not as the independent, industry-wide definition of a Web service and its associated architecture I was hoping we'd produce)" [Newcomer, 2005]. Vinoski, who was a charter member of the group, states:

I knew early on that the group was doomed when it took two iterations, each approximately three months long, to reach agreement on a basic definition for a web service [2004].

The W3C definition given above was actually a revised version of an earlier one which read:

A Web service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols [Booth *et al.*, 2002].

It is interesting to see how much more loose and open this earlier definition is, not tied to specific protocols (something lost in the current version) – but also without the requirement for interoperability or for machine-to-machine interaction (something now gained), although it does specify that its definition is machine-discoverable.

A more concrete but paradoxically more open definition is that given by Lomow and Newcomer:

A service is a location on the network that has a machine-readable description of the messages it receives and optionally returns. A service is therefore defined in terms of the message exchange patterns it supports [2005].

Here a service is not defined by the protocols it uses but through a location and the messages it can exchange. It is not even confined in this definition to the "web" but more broadly placed on the "network". Although its message *description* must be "machine-readable", the messaging sources are not so limited.

When seeking a definition of web services, it is important to remember that the term is used in two different senses, embodied in the singular and plural nouns *web service* and *web services*. On the one hand a web service, also known as a "provider agent" [Booth and Liu, 2005], is a piece of software which offers business logic that may be useful to other processes. Most

developers agree that a web service is a "resource... designed to be consumed by software rather than by humans" [Manes, 2003]. On the other hand, as mentioned in Chapter 2 (page 26), Vogels argues that web services are a messaging technology. This puts the focus on the message exchanges between the applications, not on the business logic that the messages contain or on what is done with the messages before they are sent and after they are received. Shah and Apte take this view one step further in their contention that web services are a "packaging strategy" for business logic [2004] (although the idea that web services are a "wrapping technology" is challenged by Provost, for whom their strength is that they offer "component interoperability based on progressively more vertical standards" [2003]). Shah and Apte support their argument with the claim that the core of web services is the business module which must be seen as independent from the technologies used to package it because these will change.

The technology and the business module can be seen as distinct entities. It is the technology encapsulating a business process that is designed for application interoperability and that must be platform- and language-independent. While this study uses individual services as examples, its focus is on the technology. Business processes can be as diverse as the businesses they represent and interoperability is no more a feature of a business process than a mail system is a feature of a letter, or an aeroplane of an individual air passenger. The one exists for a while in the context of the other but they are distinct.

MAINSTREAM	EARLY ADOPTION	EXPERIMENTATION	SPECIFICATION
SOAP WSDL UDDI	WS-SECURITY WSRP	ASAP BPEL WS-Coordination WS-Policy	SOAP MTOM WS-Addressing WS-CAF WS-Choreography WSDM WS-Eventing WS-Federation WS-IL WS-Provisioning WS-Reliable Messaging
[W3C + OASIS recommendations]	[OASIS specifications]	[OASIS specifications]	[W3C recommendations + OASIS specifications + proprietary 'standards']

Figure 3-1: Web-Service Protocols [adapted from [Wilkes, 2004](#)]

Lest the impression be given that web services are a technology that has been carefully crafted, it is important to remember Loughran's statement that "Web Services grew by accretion, not planning" [2003]. The truth that web services "just grew" can be seen in any timeline of web services and web-service standards and recommendations. It can be argued, for example, that the initial WSDL standard was released in September 2000 by Microsoft and IBM because the W3C's Schema Language, which might well have pre-empted it, was not ready for official recommendation status until the following year. Such ad hoc development does make for potentially richer web-services standards, in that it allows for more than one approach to a particular aspect of the technology, but it also makes for duplication, confusion and interoperability problems. That the accretion continues is evidenced in the 2004 diagram of web-service protocols [Wilkes, 2004] contained in Figure 3-1 and even more in the larger diagram of specifications contained in Appendix C [Jeon, c2005].

Wilkes used his diagram to illustrate how only the "Mainstream" protocols had been fully implemented and adopted. At his time of publishing, those in the "Specification" column existed only in draft format, while the columns in-between listed those protocols that had implementations with greater or lesser degrees of stability. This diagram illustrates only the tip of the iceberg of the related specifications that are now available. Indications of the sources of the specifications have been added to it. It is interesting to note that, while the earliest and most stable standards are those that come from the W3C, OASIS has become the standardizing body of choice for the major vendors to push specifications through at a faster pace without the loss of rights to potential royalties. There have been recent rumblings at OASIS about the development of specifications such as WS-Federation that compete with existing standards.⁸ That Microsoft and Sun have more recently come to an agreement about the joint use of the competing specification does nothing to detract from the fact that Microsoft is currently retaining possession of it.⁹

In late 2004 Bray counted the pages of the then existing specifications and came up with the totals listed in Table 3-1. It is interesting to note the high proportion (almost 70%) of

⁸ See: http://searchwebservives.techtargget.com/originalContent/0,289142,sid26_gci993124,00.html [Mimoso, 2004].

⁹ See: <http://channels.microsoft.com/presspass/press/2005/may05/0513MSSunFS.asp> [Microsoft, 2005].

specifications for which Microsoft (listed as "M": Oasis is "O" and the W3C is "W") is responsible.

Group	Specification	Page Count
Security	Web Services Security (O)	56
	UsernameToken Profile (O)	15
	X.509 Certificate Token Profile (O)	16
	Policy Language (M)	13
	Trust Language (M)	41
	Secure Conversation Language (M)	17
	Web Services Federation Language (M)	28
	WS-Federation: Active Requestor Profile (M)	14
	WS-Federation: Passive Requestor Profile (M)	13
	Kerberos Binding (M)	17
	Reliable Messaging	Reliable Messaging (M)
Transactions	Coordination (M)	16
	Atomic Transaction (M)	10
	Business Activity Framework (M)	13
Metadata	WSDL 1.1 (W)	32
	Policy Framework (M)	15
	Policy Attachment (M)	10
	Policy Assertions Language (M)	9
	Dynamic Discovery (M)	22
	Metadata Exchange (M)	23
Messaging	SOAP 1.2 Primer (W)	39
	SOAP 1.2 Messaging Framework (W)	47
	SOAP 1.2 Adjuncts (W)	25
	Addressing (M)	15
	MTOM (W)	13
	Enumeration (M)	27
	Eventing (M)	21
	Transfer (M)	17
	SOAP-over-UDP (M)	7
Management	Web Services for Management (M)	23

Business Process	BPEL4WS (M)	74
Specification Profiles	Devices Profile (O)	24
	WS-I Basic Profile (O)	50
	TOTAL PAGES:	783

Table 3-1: Page-Count of Web-Services Specifications Available in Late 2004 [[Bray, 2004b](#)]

Morgenthal believes that a generic web service can be described in terms of five simple requirements. A web service, he says, must be [[2003](#)]:

- atomic
- self-describing
- accessible
- declarative
- composite.

It should be *atomic* in that no part of a service must interfere with any other part and in that it must be complete in itself; *self-describing* in terms of the interface definition language used to define each service which, as XML, contains everything needed to understand and process it¹⁰; *accessible* in terms of the transport protocols that might be used, such as HTTP or SMTP; *declarative* in that related standards state what is required in a general sense without prescribing implementation details; and *composite* because it will be made up of several parts.

So far, with the exception of those provided by Lamow and Newcomer, the definitions have been rather abstract. Expanding on Morgenthal's requirements brought in concrete aspects in terms of transport protocols and XML. It is now time to examine a more concrete definition of a web service to see if it conforms to the conditions and notions so far described. For this we can turn again to Vogels, whose definition of a web service embraces four key constituents [[2003](#)]:

1. The service is software that can process an XML document it receives through some combination of transport and application protocols;
2. The XML document is the Web service's keystone because it contains all the application-specific information that a service consumer sends to the service for processing;
3. The address, also called a port reference, is a protocol binding combined with a network address that a requester can use to access the service;

¹⁰ See [de hÓra, 2002](#).

4. The envelope is a message-encapsulation protocol that ensures that the XML document to be processed is clearly separated from other information the two communicating processes might want to exchange.

Vogels muddies the waters by confusing the service definition not only with the messaging technology but also with the application handling the message. His first point does, however, fulfil Morgenthal's requirement that a web service should be accessible and declarative, in its inclusion of transport protocols and XML. The second point also concerns the declarative aspect of a web service, giving pre-eminence to the data contained in the XML document that is exchanged and therefore to the business logic that it encapsulates. Vogels' third point is a definition of a web-service endpoint, which is an integral part of an individual service. In his fourth point, hinting at SOAP in his use of the term "envelope", Vogels deals with the atomic and composite aspects of web services which are embodied in the messaging. Here he is more concerned with the technology than with the business logic.

3.1.1 The Web in Web Services

It is important to distinguish web services (increasingly embraced by the term *service-oriented architecture* (SOA)) from the web of pages and hyperlinks. This distinction is nicely made by Apte and Mehta, who argue that the main differences are encapsulated in the form of service interactions. A web service, they maintain, is an "active component... an active program or a software component in a given environment that provides and manages access to a *resource* [my italics] that is essential for the function of other entities in the environment... whereas a Web page is a static, one-time representation of some information" [Apte and Mehta, 2002]. While serving as a pointer to a useful distinction – that web services, unlike conventional web pages, are software components – this definition is limited. Web pages are increasingly thought of as dynamic and can also function as part of a larger application, particularly, perhaps, involving user input to a larger service. The default starting point for Microsoft web services in *Visual Studio 2005* is a web page.

Many of the arguments that broke out over the nature of web services, before the release of SOAP 1.2 in 2003, had their basis in the concept of the *web* in web services. Those supporting the web philosophy propounded by Fielding (discussed in Chapter 5), and with a proprietary interest in the HTTP standards (Fielding was one of the developers of HTTP 1.1), took great objection to SOAP because they felt it violated long-held principles which, they believed, underpinned the success and freedom of the Internet. Over the last two years there has been a movement away from the term "*web* services" towards another term, "Service-Oriented architecture", examined in the next section, which arouses less controversy and does not limit the transmission of SOAP messages to HTTP.

By contrast, Neward argues for keeping the two terms together to ensure interoperability. He makes a distinction between "web" and "services" that he thinks is fundamental to defining a term that he states causes great confusion even in 2005:

in the term "Web services", there are two basic concepts we keep mixing up and confusing. "Web", meaning interoperability across languages, tools and platforms, and "services", meaning a design philosophy seeking to correct for the flaws we've discovered with distributed objects and components. These two ideas, while definitely complementary, stand alone, and a quick examination of each reveals this... By combining interoperability with services, we create "things" that can effectively stand alone for the foreseeable future. [2005].

3.1.2 The Distinction between Web Services and Service-Oriented Architecture

Service-oriented architecture is a design methodology for the organization and reuse of services viewed as components within a larger system. While in current usage service-oriented architecture and web services are often linked, the major distinction between them is that SOA is an *architecture*, while web services are a *technology*, an implementation of an architecture. Manes considers a web service, as distinct from SOA, to be a resource that may be a business process, accessed through an API, over loosely-coupled connections, located by means of a registry and based on XML technologies [Manes, 2003]. Implementations of a service-oriented

architecture may use earlier middleware components such as MOM, and are not necessarily confined to web services. Service-oriented architecture is a necessary companion to web services in the enterprise where complex interactions between service units may need to be carefully orchestrated. Chapter 6 discusses the technologies created to enable the choreography of such interactions.

Service-oriented architecture works at a higher and more abstract level than web services, as an infrastructure which focuses, not on the technologies which implement a web-services system, but on the ways in which the units of which such a system is composed may interact with each other across a variety of environments. The significance of SOA is in its emphasis upon service as a *means of interaction*. Lomow and Newcomer give a diagram (Figure 3-2) illustrating a service-oriented architecture applied to the requirements customers have of a banking service. (Notice how this is an example involving *user* interaction.) In the diagram the organizing

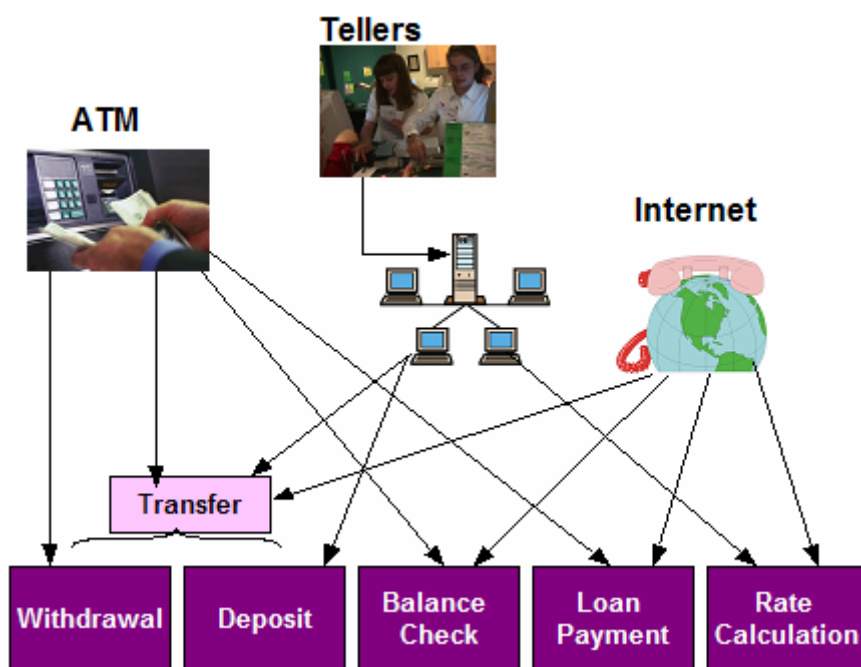


Figure 3-2: Accessing and Composing Services in a SOA [adapted from Lomow and Newcomer, 2004]

principle is expressed through the *interactions* with and between services, not on the means or technology of any single interaction. Message exchanges are the heart of SOA.

Newcomer asserts that the major difference between current and previous IT systems is the movement away from

specific implementation environments such as object orientation, procedure orientation, and message orientation to solve these business problems, resulting in systems that were often tied to the features and functions of a particular execution environment technology such as CICS [*Customer Information Control System*], IMS [*IP Multimedia Subsystems*], CORBA, J2EE, and COM/DCOM [Lomow and Newcomer, 2004].

A service-oriented system is not confined in this way.

It is possible that there is currently a greater acceptance of SOA than of web services per se. While some might see this as the next stage for web services, it might also be seen as a restriction of the original vision of the universal applicability of web services, which, under SOA, are seen more as the means of exposing and integrating existing applications within a corporate intranet.

3.1.3 A Working Definition

Of all the available definitions of web services, probably the clearest and simplest is that given by Newcomer in mid-2005:

It's the action of sending an XML document to a receiving system that understands how to parse the XML and map it to an underlying execution environment, and optionally receiving a reply, also in the form of an XML document... A Web service must exist independently of any programming language's view of it. If it didn't, we would not achieve the benefit of universal interoperability... The whole thing really has to start and end with the XML, not the Java or the C# or the Perl or the Python or the COBOL, SQL, or whatever [[Newcomer, 2005](#)].

This definition places XML solidly at the heart of web services, regardless of language, platform or protocol, stressing their interoperability, and it also broadens the scope in that web services are not tied to any set of specifications. The writer does not consider that she can improve on this definition and she is in complete agreement with it.

The current accumulation of complex specifications not only narrows the general acceptability of web services but also forms a barrier to interoperability. If the accumulation continues at the present rate, web services will either become the province of the few who can afford the expensive tools to implement them, because mastering the Babel of specifications will be beyond the scope of any normal developer – or the simmering revolution already evident in the ranks of those who work with XML on a daily basis¹¹ will take hold and another, simpler solution will be found. An apt illustration of the complexity of the specifications may be seen in the image incorporated into this thesis as Appendix C because of its size. It displays all too evidently the "spaghettification" of specifications, in which organizations contend with each other by producing different ways of solving the same problem and in which there is continual overlap and redundancy. It is hard to see how anyone can view such a proliferation of specifications without incredulity.

3.2 Advantages of Web Services

A major advantage of web services lies in the promise of integration. Earlier distributed systems used binary encodings for their data, which made them platform-, application-, and language-dependent. Web-services technologies such as XML offer platform-independence. Regardless of how the web service XML "message" is sent or of how it is handled at its destination, the message itself is not only comprehensible to any system but also has a life expectancy unlimited by the currency of a particular binary encoding. A web service may expose the functionality of legacy code without having to make any alterations to the code in order to make it reusable, thereby enabling interactions with it that could not have been foreseen when the code was written.

¹¹ An example of this approach is a column written by Bray, one of the authors of the XML Specification, in which he writes: "No matter how hard I try, I still think the WS-* stack is bloated, opaque, and insanely complex. I think it's going to be hard to understand, hard to implement, hard to interoperate, and hard to secure." (<http://www.tbray.org/ongoing/When/200x/2004/09/18/WS-Oppo>.) [Bray, 2004] This view has also been reflected in the trade press – see the article at http://news.com.com/Trying+to+make+Web+services+make+sense/2100-7345_3-5242747.html?tag=nefd.lede [LaMonica, 2004], which issues a warning note: "Without clear direction on standards, the payoff of the massive industry bet on Web services could be delayed – or derailed – because customers are sitting on the sidelines of a politicized and contentious standards process." Sun's President and COO recently admitted: "[Web services have] either got to be simplified, or radically rethought... today's web services initiatives are in danger of vastly overcomplicating a very simple (really simple) solution." [Schwartz, 2005].

There is no question of the financial advantage of web services. Bosak reproduces a table (Table 3-2) of the savings estimated from the use of paperless systems [[Bosak, 2004](#)].

Savings by product (in U.S. dollars) estimated from use of paperless systems.			
Item	Volume	Value (CIF)	Saving Estimate
Coal — bulk by sea	10,000 tons	\$520,000	\$7,800 or 1.5 percent
Rice — bulk by sea	1,500 tons	\$810,000	\$17,820 or 2.2 percent
Machine parts — by sea	20-foot container	\$175,000	\$5,425 or 3.1 percent
Sugar — bagged by sea	1,500 tons	\$273,000	\$12,012 or 4.4 percent
Fresh asparagus — by air	45 kg	\$1,370	\$206 or 15 percent

Table 3-2: Economic Advantages of Paperless Trade [redrawn from [Australian Government, 2001](#)]

A major benefit of web services and a major contributor to web services' interoperability is that of loose coupling. When different types of program running on different platforms, beyond an intranet, are required to interact, loose coupling is essential. Slama *et al.* list several requirements for loose coupling, among them dynamic binding and a weak type system [2004].

Level	Tight Coupling	Loose Coupling
Physical coupling	Direct physical link required	Physical intermediary
Communication style	Synchronous	Asynchronous
Type system	Strong type system (e.g., interface semantics)	Weak type system (e.g., payload semantics)
Interaction pattern	OO-style navigation of complex object trees	Data-centric, self-contained messages
Control of process logic	Central control of process logic	Distributed logic components
Service discovery and binding	Statically bound services	Dynamically bound services
Platform dependencies	Strong OS and programming language dependencies	OS- and programming language independent

Table 3-3: Loose and Tight Coupling [redrawn from Slama *et al.*, 2004]

They suggest that the price to be paid for loose-coupling, which might include a more complex and even more expensive system overall, plus "[a]dditional efforts for development and higher skills ...to apply the more sophisticated concepts of loosely coupled systems", will balance out in the long term if the flexibility of such a system is fully utilized. Their table, shown in Table 3-3, illustrates the suitability of web services to meet the challenges of loose coupling: every item in the "Loose Coupling" column can be expressed as a feature of web-services technology, which is at its best asynchronous, message-oriented (with the messaging helping to perform the function of an intermediary), distributed, dynamic and platform and language-independent and, ideally, independent, too, of strong data typing.

In addition to the more general benefits offered by web services, the SOAP protocol in particular offers the benefits of simplicity of use, extensibility, flexibility, loose coupling and a foundation in XML [[Loughran and Smith, 2005](#)]. SOAP, its benefits and drawbacks, will be discussed in more detail in Chapter 4.

3.3 Adoption of Web Services

There is little doubt that the competing aims of the major vendors, and more particularly of the major standardization bodies, have been a major stumbling-block to the acceptance of web services, as has been the proliferation of different, and not always interoperable, web-services tool implementations. As the key standards have solidified, however, the situation has begun to change. It was significant that in August 2004 five of the major competing organizations, BEA Systems, IBM, Microsoft, SAP and Sun Microsystems, *jointly* submitted the latest version of WS-Addressing to the W3C and not to OASIS.

Microsoft and IBM in particular have in the recent past turned to OASIS for the ratification of standards they wished to push through without either the lengthy discussion process sometimes involved in ratification by the W3C, or the royalty-free stipulation enjoined by the W3C. Significantly, towards the end of February, 2005, a group of signatories wrote an open letter to

OASIS, protesting about the current, supposedly "reasonable and non-discriminatory" (RAND) licensing policy adopted by the organization. This policy, they claimed, would actually

"..discriminate against open source and free software to the point of prohibiting them entirely... This is not a new issue for us. We fought hard for a royalty-free patent policy in W3C and encouraged that standards organization to commit its members to open standards. But some W3C member companies, steadfast opponents of software freedom, moved their efforts to OASIS. Without consulting the free software/open source community, they produced a patent policy designed so that we cannot live with it."

[[Rosen et al., 2005](#)].

Whether the companies so stigmatized will react positively to such a complaint remains to be seen.

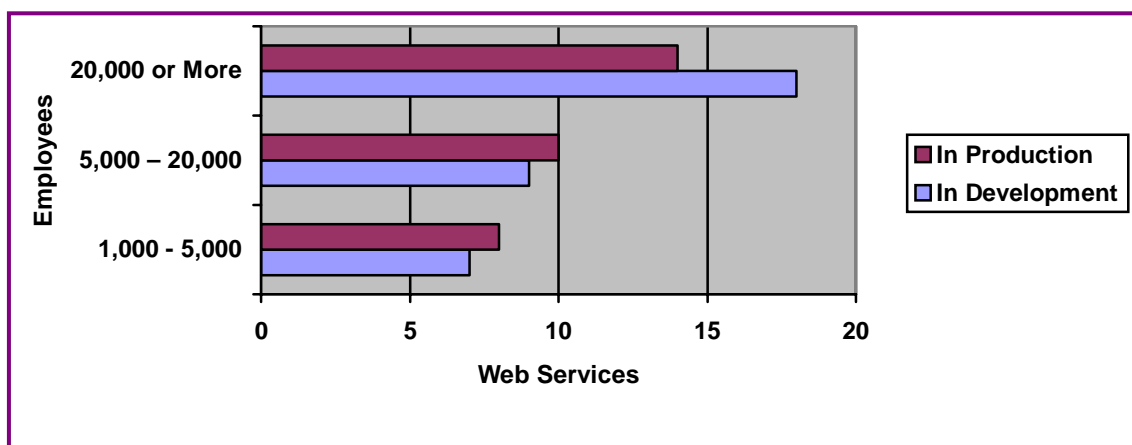


Figure 3-3: Redrawn from Web Service Projects, Forrester Report, 2004

In spite of the acrimony around the standards for web-service technologies, Leavitt was in no doubt about the future growth of web services:

"IDC estimates that worldwide spending on Web services-based software projects will reach \$11 billion by 2008, compared to \$1.1 billion in 2003. A Gartner survey of 110 companies also indicated that 54 percent are already working on Web services projects this year or have plans to begin soon. [Figure 3-3] shows results from a 2004 Forrester Research survey of about 280 large North American firms. Survey respondents identified a total of 66 Web services that are either in production or in development... An Evans Data survey indicated that one out of every 10 companies is investing in Web services

development and integration this year. About 13 percent of the respondents said that a majority of their development funds are going to Web services, and IBM is investing more than \$1 billion a year" [[Leavitt, 2004](#)].

Another Gartner study [[Cantera, 2004](#)] reported the web-services software market was \$6.2 billion in 2003, and predicted that it would grow to \$44.5 billion by 2007 when web-services software would comprise 41% of the overall market.

Smaller examples of the fulfilment of this growth potential, eBay and PayPal in 2004 joined Amazon and Google in exposing SOAP web services using WSDL. According to Iverson, "eBay processes over a billion web-service requests a month" [Iverson, 2004]. While not using SOAP and WSDL, both Yahoo and O'Reilly's *Safari OnLine* in mid 2005 also went public with web-service APIs.

Late in 2005, an e-commerce analyst sounded a cautious note, however, when he depicted current usage and development of web services as follows:

- 75% of all web-services projects are purpose-built for database and data warehouse integration projects.
- 65% are developed to support integration to legacy or mainframe applications.
- 60% are developed for internal portal integration efforts with legacy systems and typically three or fewer databases internal to the company.
- One manufacturer of complex pumps and valves focused on headcount reductions equaling \$500,000 the first year, only to find that the two engineers who would be let go were actually needed for integration work for the web service aimed at streamlining complex order capture. Net result: The web service worked and another application server engineer needed to be added to the project.
- One manufacturer of computer equipment uses web services to integrate order capture and pricing systems that are part of their SAP ERP instance—and the result is the ability to publish price updates to worldwide channels within 48 hours. The web service will also add Oracle pricing integration—yet that will be over a year away and require months of internal development [[Columbus, 2005](#)].

All of this points to the costs of the development of web services and to their current deployment on the intranet rather than on the internet, which suggests that they are not currently being seen widely as a solution to real, interoperable distributed computing. According to Columbus, web services have contributed largely in the area of application integration, particularly database integration, where companies have used web services to design solutions for the automation of order processing and transaction tracking, sometimes without the companies themselves even being aware that a Service-Oriented Architecture is what has actually been implemented.

3.4 Barriers to Interoperability

So what are the barriers to interoperability inherent in web services, that go beyond or, more accurately, underlie problems with toolkits and vendor lock-in, and that contribute to the cost of development mentioned in the previous section? They can be divided into three main categories: transport issues, data-typing and exception handling, although a further category could be labelled "mistakes" for problems such as misunderstandings arising from the complications of WSDL namespace scoping or schema versioning.

3.4.1 Transport Issues

Several transport issues may be found in the current standards. In the SOAP standard, one difficulty is caused by the generally deprecated `SOAPAction` header, which is required under version 1.1 but discouraged under some circumstances by the WS-I Basic Profile, and includes no specification for how its value should be represented in the WSDL file. Kumar, Das and Padmanabhuni report that:

Practitioners have found out that a valid SOAP request message to a sample web service may result in different responses, ranging from a fully valid correct response to a SOAP fault response, depending on the value of the SOAPAction header [2004].

In the past, some toolkits have allowed it to be ignored, assuming a default empty string value but others (specifically those based on .NET code) require more detail. Some XML tools (e.g.

Stylus Studio version 4.5) will even refuse to validate a WSDL file that does not contain it. Chapter 7 takes a closer look at the current implementations of SOAP, examining the benefits that will arise from changing from SOAP 1.1 to SOAP 1.2.

An earlier transport issue, still alive in those SOAP toolkits which do not yet support SOAP 1.2, concerns the use of HTTP `POST`, given in earlier versions as the definitive method for sending SOAP requests and responses¹². SOAP 1.2, more logically in the context of information retrieval, allows the use of the idempotent `GET` method through the use of *Message Exchange Patterns* (MEPs), the SOAP Web Method, and the HTTP Accept header, in which a SOAP response may be the result of a non-SOAP request. For information retrieval, the default action in SOAP 1.2 is in fact a simple HTTP request. The 1.2 specification chastises as "counter to the spirit of the Web" SOAP/RPC implementations which conceal the identity of a web resource behind "some intermediate entity" and states:

The SOAP response message exchange pattern with the HTTP `GET` method is used when an application is assured that the message exchange is for the purposes of information retrieval... [T]he HTTP SOAP `GET` usage does not allow for a SOAP message in the request [specifically where all the arguments can be represented in the URI]... [T]he response to [an] HTTP `GET` request from a requesting SOAP node is a SOAP message in the HTTP response [[Mitra, 2003](#)].

Such a message exchange embraces both the SOAP technology and the more simple approach embodied in the REST philosophy for web services. However, even in late 2005, implementations of SOAP 1.2 are tentative. Microsoft's *Visual Studio 2005*, which offers a default implementation, is only just emerging from its beta version, and JAX-WS, which also supports it, is still under development and therefore not officially part of a J2EE platform. The SOAP 1.2 documentation for JAX-WS admits: "Currently the support is limited and not tested extensively" [[Sun, 2005b](#)]. Although there was documented support for SOAP 1.2 in WebLogic 8.1, it came with a health warning that it was suitable only for development environments and it is not included in the documentation for WebLogic 9.0 where the Ant tasks have been completely reformulated. Systinet Server 6 for Java does include an implementation of it, but

¹² See [Box et al., 2000](#), section 6.1. The document nowhere mentions the `GET` method.

the Basic Profile does not yet "permit" the use of SOAP 1.2, and section 7.3 in Chapter 7 shows what happens in interoperability testing with web services that do use it.

A further transport issue involves the use of cookies to maintain state during a prolonged exchange – although SOAP is intended to be *stateless*. JAX-RPC, which allows state in web services, recommends the use of cookies in this way, but of course cookies may be turned off inside a communicating browser, rendering state maintenance unusable, except through visible URL-rewriting, and web services are not confined to browsers. The disadvantages of maintaining state in web services include problems of recovery after failure, memory consumption and scalability.

An issue which can become a problem is the supremacy of the declared HTTP encoding over any declared XML encoding when XML is sent over HTTP. The creators of the XML Specification were aware of this, as may be seen from (the non-normative) Appendix F of the Recommendation [[Bray et al., 2004](#)]. RFC 3023 was written to handle this problem [[Murata, 2001](#)], as is explained by Pilgrim [[2004](#)]:

According to RFC 3023, if the media type given in the Content-Type HTTP header is application/xml, application/xml-dtd, application/xml-external-parsed-entity, or any one of the subtypes of application/xml such as application/atom+xml or application/rss+xml or even application/rdf+xml, then the character encoding is determined in this order:

1. the encoding given in the charset parameter of the Content-Type HTTP header, or
2. the encoding given in the encoding attribute of the XML declaration within the document, or
3. utf-8.

On the other hand, if the media type given in the Content-Type HTTP header is text/xml, text/xml-external-parsed-entity, or a subtype like text/AnythingAtAll+xml, then the encoding attribute of the XML declaration within the document is ignored completely, and the character encoding is:

1. the encoding given in the charset parameter of the Content-Type HTTP header, or
2. us-ascii.

The upshot of all this is that in some cases the XML-encoding will be ignored, with the potential for severe interoperability problems for those needing non-ASCII character sets. According to Pilgrim [2004], although Apache now recognizes XML encoding accurately, IIS does not. He also points out that, unnervingly, none of the popular XML parsers at that point (mid-to-late 2004) supported RFC 3023. Pilgrim concludes that the only truly workable encoding is US-ASCII and he has a parting shot directed at both clients and servers who ignore Postel's law¹³ and accept virtually anything: "The entire world of syndication only works because everyone happens to ignore the rules in the same way. So much for ensuring interoperability."

Web-service testing in Chapter 7 reveals that while "text/xml" is still usually given as the mime type in the HTTP headers generated by various toolkits using SOAP 1.1, the charset parameter usually contain the correct encoding for the XML. Those toolkits with support for SOAP 1.2 provide a more accurate mime type.

3.4.2 User-Defined Data Types

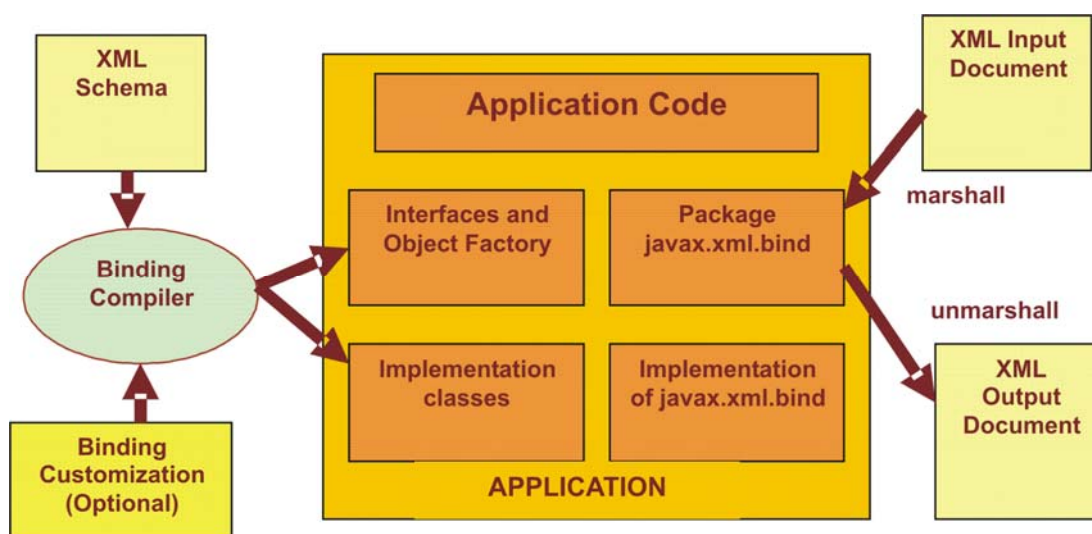


Figure 3-4: The Architecture of JAXB [redrawn from the J2EE 1.4 Tutorial]

Still a major challenge to web-service developers is the issue of complex data typing. The serialization of complex data types can be managed with varying degrees of efficiency by

¹³ See RFC 793: "be conservative in what you do, be liberal in what you accept from others."

toolkits. A hand-coded approach with Java would first define the complex data type in schema notation and then create a JavaBean-type class corresponding to the schema. A serialization class might then be written to perform the type conversion between Java and XML.

Aware of the seriousness of this issue, Sun has focused its current Web Service Developer Tutorial (1.6) on data binding [[Sun, 2005a](#)]. Some impression of the complexity of the issue may be conveyed by Figure 3-4, which is a diagrammatic representation of the *Java Architecture for XML Binding* (JAXB), Sun's proposed solution to this problem, which includes a special compiler and a flexible API enabling the developer to "tweak" results to solve particular problems. The diagram reveals the level of processing necessary for the marshalling and unmarshalling of XML.

The mapping in JAXB between Java and XML is illustrated in the following table of data types that may be automatically serialized into XML:

XML Schema Type	Java Data Type
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	java.util.Calendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:time	java.util.Calendar
xsd:date	java.util.Calendar
xsd:anySimpleType	java.lang.String

Table 3-4:JAXB Mapping of XML Schema Built-in Data Types [[Sun, 2005a](#)]

It is easy to see from this that only a limited number of Java data types have made it into this default list, although customized bindings for other data types may be built. JAXB does not currently support the whole schema specification, omitting less-used features such as the redefinition of a declaration, the notion of `keys` and `keyrefs`, the `AnyAttribute` wildcard and

substitution groups. While there are cogent reasons for these omissions (for example, the lack of an available data binding and complexity [see [Fialli and Vajjhala, 2003](#)]), the omissions theoretically pose an interoperability problem for implementations in other programming languages. Schema constructs other than data types, such as `choice` and `sequence` do not map well into programming languages. Kumar, Das and Padmanabhuni point out that different W3C Schema versions produce unpredictable results and are a source of interoperability which is addressed by the Basic Profile's recommendation that only the later (2001) version be used [2004]. Kumar, Das and Padmanabhuni also discuss problems still not addressed by the Basic Profile, involving the lack of precision, not only in the representation of infinity but also in the representation of time, where some vendors employ nanoseconds while others support only milliseconds. They state:

These issues can be of utmost importance in transactions like credit card validations and single sign on situations across heterogeneous security mechanisms [2004].

Although issues of data typing are discussed in a more detailed context in Chapters 4 and 7, general interoperability problems including data typing were outlined in a 2005 video featuring a Microsoft web-services self-styled "evangelist". Guest argues that the exchange of complex data types causes problems in situations where the following apply [[2005](#)]:

1. The returning of "empty" arrays.
2. The use and comparison of e.g. `Date` data types which are handled differently in Java and .NET. The incompatibility here will not always be realized until something goes wrong. If the developer starts with a `java.util.Date` object, it will be converted into a schema `dateTime` data type, but any future mapping back again to Java will produce a `java.util.Calendar` object, causing a `NumberFormatException` to be thrown. Nearly all of the methods of `java.util.Date` are deprecated, however, so it would be unlikely to be used by a developer with any knowledge of Java. The simple `Date` service described in section 7.2.2 of Chapter 7 shows some experimentation with these issues.
3. The generation by a toolkit of schema types from programming types, when these should preferably be designed by the programmer before the coding takes place

(Guest suggests using a wizard or toolkit to generate the code from the schema to ensure interoperability). This issue is the subject of section 4.3.3 in Chapter 4.

Point (1.) above is of particular interest as a source of much confusion. The reason for this is an inaccurate mapping between an XML version of an "array", which is usually represented by an element which may have a minimum occurrence of zero and a maximum occurrence of "unbounded", and a Java array, which can be both "null" – uninitialized – and "empty", initialized to contain elements but where each element is without content. Butek and Scheuerle point out, that " an empty instance of a Java array and a null instance of a Java array map to the same XML instance" [2004]. Because an XML "array" is really a number of elements without a "container", the solution argued by Butek and Scheuerle is the provision of a "wrapper" element. Butek and Scheuerle make the further comment that, while an XML Schema will provide an appropriate structure, that structure does not conform to any currently recognized standard and is likely to be misunderstood by web-service toolkits, although BEA's WebSphere has made provision for its use. Section 7.2.4.3 of Chapter 7 takes this issue further in examining problems arising from the use of array types.

The concept of notations (or annotations) with which web services may be configured (used for some time in C# and a more recent arrival in Java) might be said to tie the WSDL more tightly to the code implementation of a service and therefore contradict the essential function of a web service, which is to be language-independent. Vinoski points out that it is almost impossible to avoid including language-specific styles and idioms when the starting point for a service is the code [Vinoski, 2005]. Section 7.2.4.4 of Chapter 7 examines the use of annotations as a programming model for web services that can provide a middle way between the Scylla of a language-restricted, code-first approach and the Charybdis of the complexities of WSDL First.

3.4.3 Exception-Handling: `java.rmi.RemoteExceptions` and SOAP Faults

Major problems arise when the web service fails. If the eight fallacies of distributed computing [Deutsch, c1991] are correct, the service will fail at some point. What sort of a response should the client receive when an error occurs? In Java, errors may theoretically be handled by

exceptions. Not all languages, however, are as well equipped as Java or C# to handle exceptions¹⁴. Exceptions were introduced into PHP, for example, only this year with PHP 5.

Possible exceptions that may be thrown in Java are:

- `java.lang.RuntimeException`
- `java.rmi.RemoteException`
- `javax.xml.rpc.soap.SOAPFaultException`
- `javax.xml.rpc.JAXRPCException`
- a user defined exception.

According to the Axis User Guide [2005], (and this applies to any other JAX-RPC 1.1 conformant server) only server methods that throw `java.rmi.RemoteException` will have their exceptions encoded within SOAP responses as SOAP faults. Exceptions that are not `RemoteException` or descendents of `RemoteException` can alternatively be mapped to WSDL faults. Automatic generation of a WSDL document within, for example, Axis, produced a document that could not cope adequately with faults and omitted them entirely. The only ways to have them included in the WSDL was either to hand-code the WSDL or to amend it after generation.

Services monitored with the Axis SOAP Monitor appeared to produce no SOAP response if the service failed, even when the server method did throw a `RemoteException`. HTTP monitoring, however, did reveal that a SOAP message had been returned, containing a SOAP fault, even though this had not been displayed in the SOAP Monitor that Axis provides. If problems like this are encountered with one language on one platform inside one toolkit, it is not surprising that there are problems across toolkits. Changes coming in the new JAX-WS, however, include the removal of the mandate for methods to throw `RemoteException` (as well as the need for the service method to implement `java.rmi.Remote`) which should help.

Wang and Butek specifically recommend against throwing a `RemoteException` from the server because of the difficulties of interoperability even within different JAX-RPC runtimes, much less across different language barriers [Wang and Butek, 2004]. They recommend instead the use of

¹⁴ Difficulties are more likely for older languages such as C or Pascal.

a specific `RuntimeException`, a `javax.xml.rpc.soap.SOAPFaultException`, because of the features of this exception that enable it to return more information to the client. They do not recommend, however, that the remote method should itself throw this exception: rather they believe it is more usefully thrown by handler classes which will map SOAP faults to whichever exception is most appropriate.

Despite the warning against the use of `RemoteException`, it was found that classes extending `RemoteException` were in fact correctly interpreted by Axis as SOAP Faults – but the version of Axis used was Java-based and the language of implantation was also Java. In general `RemoteException` is not considered to be portable. Instances of `RuntimeException` are not easily caught by a non-Java client and their mapping to other types of exception depends upon the client runtime, creating potential problems for interoperability.

A SOAP fault returned from a server method is converted to an exception that should be able to be understood by a SOAP client for a web service. The nature of that exception appears ambiguous and its handling appears to depend on the SOAP toolkit that is being employed. Much information is available detailing the steps that must be taken with particular toolkits but such information does not necessarily translate meaningfully across toolkit boundaries. Seely, for example, refers to toolkit settings that may be tuned, and to proprietary code [[Seely, 2002](#)].

Many writers agree that the best kind of exception that may be thrown is a user-defined exception which is then mapped in the WSDL to a `wsdl:fault`. The major advantage of a user-defined exception is that, because it is visible inside the WSDL, the client may prepare for it and it is therefore more interoperable.

There seems little that a client may do to rectify a network fault. Recovery from partial failure on a network is beyond the scope of this thesis but is a very significant problem for all distributed interactions. With application failures, the need is not so much for the client to be able to rectify a fault that occasioned an exception but for the client to understand that it did occur and, if at all possible, why, so that the error may not be repeated during a future exchange. Because the indexing service (to be described in section 7.2.3 of Chapter 7) returns strings to the

client, it was easy to develop a way of dealing with `JAXRPCException` instances that returned a short message to the client, explaining what problem had been encountered, depending on the exception that had been thrown. This would be more difficult to implement for service methods that did not return a string.

3.4.4 Toolkits

Many current web-service toolkits promote distributed *object* technologies, particularly on the RPC-style of web-service exchanges (discussed in Chapter 4), at the expense of a data- or document-centric approach, perhaps because it is felt that developers will feel more comfortable in a familiar environment than in dealing with raw XML. Provost draws an only partly humorous picture of such vendors:





"Web services are a cinch," they'll say. "Just write the same code you always do, and then press this button; presto, it's now a web service, deployed to the application server, with SOAP serializers, and a WSDL descriptor all written out."...Sales and marketing folks want to be able to demonstrate that little or no coding and a lot of code generation add up to a complete web service. [[Provost, 2003](#)].

Problems arise not only from this thin disguise of the very different distributed-object technologies sometimes used to implement web services, but also from different implementations of standards in different web-service toolkits. These are pinpointed in a serious comment made by the company MindReef, the developers of SOAPscope (a Java-based tool for testing SOAP services) and winners of the InfoWorld 2005 Technology of the Year Award:

Microsoft broke rank from the other toolkit vendors when they introduced the notion of a Wrapped Document/Literal message having the name of the operation as the root element of the message content. They also stepped outside of the schema specification for serializing their DataSet type. Other toolkits have had to reverse engineer the techniques to remain compatible. With the behavior of toolkits diverging in various ways, interoperability has become a major concern [[MindReef, 2004](#)].

The significance of the different message types, including the wrapped and document/literal styles, will be discussed in greater detail in Chapter 4, but it is worth emphasizing at this point that the behaviour of major companies responsible for web-service toolkits has in the past been to try to force the web-service community to adopt their standards rather than to conform to those ratified by the standardizing bodies.

Apart from the servers and toolkits used in this study, which will be discussed at greater length in Chapter 7, other web-services toolkits available in late 2005 and freely available for non-commercial development include:

-  SUN's *Java Web Service Developer Pack*¹⁵ (in late 2005 at version 1.6), which is meant to be installed within either a (Sun-configured, non-current) version of Tomcat or one of the available Sun web servers. Reasons for not including this toolkit in the study are given in section 7.1.1 of Chapter 7.
-  MICROSOFT's *Office XP Web Services Toolkit*¹⁶, which may be used from an Office 2003 program and uses the inbuilt VB Editor to create a proxy object.
-  KAPOW's *RoboSuite*¹⁷, linked to BEA WebLogic, which uses a point and click wizard to send and return objects and their attributes, stored in a database-type format, using existing web sites as a handle for the web-service creation. The demo does not even mention the use of a WSDL file, although it displays the SOAP messages that it constructs and sends.
-  CAPE CLEAR's free *SOA Editor*¹⁸, renamed from their earlier WSDL Editor (the renaming indicating the shift of emphasis towards SOA, which is more clearly




¹⁵ See <http://java.sun.com/webservices/jwsdp/index.jsp> [Sun, 2005a].

¹⁶ See <http://www.microsoft.com/office/previous/xp/webservices/toolkit.asp> [XP Toolkit, 2005].

¹⁷ See http://kdc.kapowtech.com/presentations/IntegrationBEA_viewlet_swf.html for an interactive demo [Kapow, 2005].



¹⁸ See <http://www.capescience.com/soa/index.shtml> [Cape Clear, 2004].

embodied in their commercial offering, Cape Clear 6, advertised as the latest version of their Enterprise Service Bus). The editor enables graphical WSDL editing, still biased towards the SOAP encoding of SOAP version 1.1, towards RPC-style, and with a customized `SOAPAction` element. It has not been updated since 2004 and it produces indecipherably complex error messages, sometimes the length of a page. The errors it lists do not appear as errors in more recent validating XML editors such as *XMLSpy* and the Java-based *Oxygen*, but it does offer a basic starting point for WSDL creation, the files created through its graphical interface are fully editable elsewhere and its help files are excellent. It also observes the requirement (discussed in section 4.3.1.1) that the WSDL `targetNamespace` be dereferencible to a WSDL document.

-  ORACLE offers several free and commercial products which include web-service toolkits, such as its *Application Server 10g*¹⁹.
-  IONA has a free product, *Celtix*²⁰ which, like Cape Clear 6, is a version of an Enterprise Service Bus. It is a companion to their commercial offering, Artix, which replaced their original *XMLBus*, a toolkit specifically for web services. It is a work in progress, requiring Java 5, Apache Ant and the latest available release of JAX-WS. It is therefore one of the most standards-compliant toolkits, providing implementations of SOAP 1.2 and MTOM. The developers have produced an Eclipse plugin and are collaborating with the Apache Synapse project (still in its very early stages) which aims to provide a framework for web services.
-  The Apache Beehive Project, contributed by BEA, which uses the metadata facilities offered by Java 5 to construct web services and offers a simpler approach to enterprise applications. It uses metadata to turn any Java class into a web service, but does not yet implement JAX-WS.

¹⁹ See http://www.oracle.com/appserver/java_edition.html [Oracle, 2005]

²⁰ See <http://celtix.objectweb.org/> [Celtix, 2005]. The CTO of Celtix is Eric Newcomer, a prolific writer on SOA and referenced elsewhere in this thesis.

-  IBM is a major contributor to web-services development. Although normally a commercial product, WebSphere is freely available to educational institutions.
-  Systinet is also a major contributor to web-service development and its Server for Java version 6 includes an implementation of SOAP 1.2. It has also produced a version of Eclipse bound to this server. It does not yet implement JAX-WS.

As may be seen from the products listed above, many of the toolkits form part of a complete web-services development environment, that often comes linked to or including a server.

A general problem with toolkits is that some of them rely on versions of classes which, as versions change, may differ from those used in the web server, resulting, in Java, in an `IncompatibleClassChangeError`, which was experienced when switching, in May, 2005, from Axis *1.2 RC3* to version *1.2 Final* inside Tomcat. The solution was to upgrade Tomcat to the latest version. The error showed itself only when calls were made by different client programs and not when the application was actually deployed on Tomcat.

Are toolkits to be dismissed out of hand because some of them flout the standards and others (probably all of them) have versioning issues? Not necessarily. Interoperability is not the only feature that should be considered in this context. The present state of web-services technology means that it is highly desirable that, in industrial use, the time to development be not protracted while developers come up to speed with current specifications, not to mention the APIs and type systems. Standards-compliant toolkits are one way of achieving this. Although they are aware that those who believe a web service should be created from code will not agree with them, the authors of the Overview to the Cape Clear SOA Editor emphasize the need for service designers to be able to "define the service interface without reference to existing technical APIs" [[Cape Clear, 2005a](#)]. They consider the advantages of starting from a schema, over a code-first approach, to be a more strongly-typed WSDL end-product and significantly greater chances of

interoperability²¹. The advantages of a WSDL-first approach are discussed in more detail in section 4.3.3. Chapter 4 considers the significance of starting from a schema, which is also a natural approach when creating a REST web service, described in more detail in Chapter 5.

A balance between machine-generation of code and hand-coding is, of course, desirable, in which fine-tuning may be enabled without too steep a learning curve. Shah and Apte suggest a workable compromise:

Auto-generating a key artifact such as the interface should not be an option for systems in production because this technique forces the enterprise to depend on the convenience tools for service interfaces. On the contrary, a process must be put in place by the enterprise to manage interface definitions and their future enhancements. The auto-generated WSDL merely serves as a starting point to build the WSDL manually for production-ready services [[Shah and Apte, 2004a](#)].

3.5 Summary

Consideration of the preceding issues reveals that interoperability is not lightly achieved, and that there is a significant distinction between the ability to communicate and the meaningfulness of the communication. Meaning depends greatly on context. Shirky contended in 2001 that most of the web services available then already had a shared context and that the instances of web services given as examples in the specifications either depended upon a context that was already widely known or were too trivial to require one. He gave a whimsical but telling depiction of this problem with the following account:

Two old men were walking down the street one day, when the first remarked to the second, "Windy, ain't it?"

"No," the second man replied, "It's Thursday."

"Come to think of it," the first man replied, "I am too. Let's get a coke" [[Shirky, 2001](#)].

²¹ Formerly known as the WSDL Editor and built for the Java platform, the renamed SOA Editor offers no support yet for WSDL 2.0

CHAPTER 4: A QUALITATIVE ANALYSIS OF CORE WEB-SERVICE COMPONENTS AND THEIR CONTRIBUTION TO INTEROPERABILITY FOR WEB SERVICES

4.1 Introduction

This chapter examines the core web-service components: SOAP, WSDL and (to a lesser degree, for reasons which will be explained in section 4.4) UDDI. It explores the different aspects and implementation features of each standard in order to determine how much each contributes towards web-services interoperability. These findings are validated in Chapter 7 by practical examples of web services.

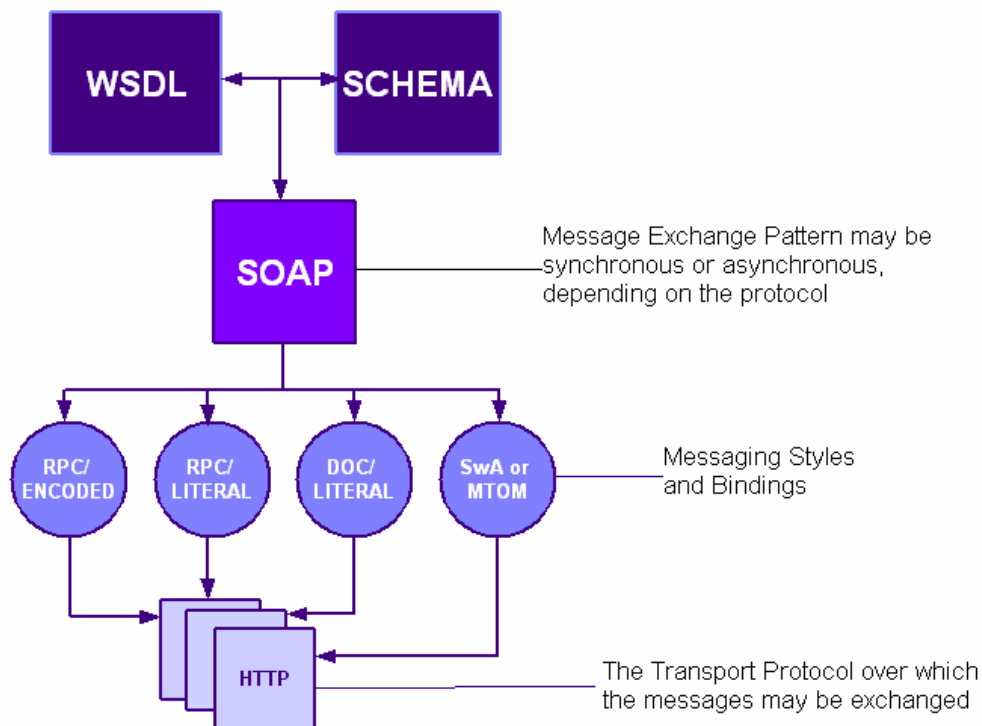


Figure 4-1: Overall Design of a Web Service

Figure 4-1 illustrates the wide view that this chapter encompasses. The image is also a graphical representation of the "WSDL-First" philosophy (explored in section 4.3.3) in that it depicts the starting point for a web service seen as business interface semantics embodied in the combination of a schema and a WSDL document. What it does not display is the service programming interface and implementation which are behind the WSDL and the schema (or may be generated from them) and with which the client ultimately communicates. The rest of this chapter examines the details captured in this image, but it may be helpful at this point to give a brief overview of the different elements here represented.

The natural beginning for a SOAP-based approach to interoperable web services is the combination of an XML schema and a WSDL document which defines the interface for the service. Although earlier versions of JAX-RPC required a service endpoint interface, the latest version, JAX-WS, omits this requirement, possibly in recognition of the fact that WSDL is a sufficient interface. The schema is necessary in method-based services because it provides XML definitions for data types used in the service. The reason for this as the starting point for a service, rather than a coded implementation, is discussed later in section 4.3.3. Choices in the WSDL determine the nature of the SOAP messages that are sent back and forth between the client and the service, and may also determine the transport protocol used to convey the message exchanges. One of these choices, the message exchange patterns (MEPs) or service styles (document or RPC), is discussed in section 4.2.4.1. Whether the MEPs are synchronous or asynchronous has depended in the past on the choice of protocol, although the latest toolkits such as BEA's WebLogic 9.0 and *Visual Studio 2005* offer both styles by default, regardless of protocol.

The WSDL 2.0 specification is not expected to become a recommendation until 2006 (WSDL 2.0 differed so markedly from its earlier draft, WSDL 1.2, that it was given a new version number). Toolkit and web-server implementations do not yet incorporate the new version but it forms part of the discussion because of the indications it gives of trends and developments in web services. SOAP appears the more controversial specification and has therefore been handled at greater length.

4.1.1 Interoperability-Testing Frameworks

While not in itself an integral web-services specification, the Basic Profile should be considered at this point. The vast number of competing web-service specifications have made it necessary for some kind of arbitration to preserve interoperability. Even the accepted standards do not provide implementation details. For these reasons, in 2002, the Web-Services Interoperability Organization was founded by Microsoft, IBM and a number of other industry partners (not for the first eight months including Sun Microsystems).

In 2003 this consortium produced the Basic Profile 1.0, which consisted of a series of recommendations for combining standards and specifications so that interoperability would not be compromised and vendors could produce toolkits that would not conflict with each other. In particular it laid down mechanisms for handling web-services messaging, discovery, description and security, making HTTP 1.1 the only transport protocol and making mandatory both the HTTP POST method and the HTTP SOAP binding. In 2004, the Basic Profile 1.0 was superseded by the Basic Profile 1.1, which was based on the earlier version but resolved technical issues that had arisen from it, such as MIME bindings, and the need to include attachments in SOAP messages, as outlined, for example, in the specification for *SOAP Messages with Attachments* (SwA). This new version also made possible the later inclusion of binary attachments as specified in the *SOAP Message Transmission Optimization Mechanism* (MTOM).

The WS-I organization have produced an interoperability testing toolkit that may be used by vendors and implementors of web services alike. The testing mechanisms have been incorporated into most of the latest web-services toolkits and alert the user to violations that have been detected so that they may be corrected before the service is employed.

In this context, it should also be mentioned that the Basic Profile is not the only available interoperability-testing apparatus. Bertolino and Polino have put forward a mechanism for testing that they name the Audition Framework, which takes the form of a Protocol State Machine [2005]. It relies heavily, however, upon the presence of a UDDI Service Broker. (UDDI is discussed further in section 4.4 of this chapter.) Jiang and Systä suggest a WSDL

validation model, based on UML Profiles, which will enforce Basic Profile recommendations and which they have demonstrated to be more accurate than the Basic Profile alone for detecting errors [2005].

Yu, Huang and Ye recommend an approach different from the Basic Profile and based on an ontology library which creates "reasoning rules for error analysis and communication data control", based on data which has been captured and stored in the library [2005]. There is a distributed, Java-based system which uses a Petri-net to replay the operations as an aid to understanding errors that may have arisen. The system tests interoperability against items such as erroneous method calls, loss of data, data format errors, and semantic errors such as might occur in "the composition process of Web services, such as complex workflow integration" [2005]. Unlike the Basic Profile, their viewpoint is the basic web service on the wire, using the core specifications of SOAP 1.1, WSDL 1.1 and UDDI. They do not attempt to prescribe or proscribe the use of different specifications or versions or to examine the implementation problems inherent in the specifications. There is a work in progress.

4.2 SOAP: no longer an acronym

4.2.1 Defining SOAP: Versioning Issues

As may be inferred from Figure 4-2 **Error! Reference source not found.**, SOAP is an XML message format. The diagram suggests that SOAP merely contains XML, when SOAP is actually itself an XML "language", but the whole SOAP specification describes much more than a message format in that it includes, for example, message exchange patterns and an illustrative binding to the HTTP protocol. As mentioned in Chapter 2, while SOAP version 1.1 used the name as an acronym for *Simple Object Access Protocol*, the inappropriate acronymic meanings were removed in version 1.2 with the laconic sentence "SOAP 1.2 will not spell out the acronym" [[Mitra, 2003](#)].

In the light of this it might be thought curious that, under the glossary heading *Simple Object Access Protocol (SOAP)*, Microsoft, one of the original designers of SOAP, still in 2005 offers

the following definition, in an apparent attempt to emphasize the protocol's interoperability while at the same time confining its use to HTTP:

SOAP defines a message format in XML that travels over the Internet using Hypertext Transfer Protocol (HTTP). By using existing Web protocols (HTTP) and languages (XML), SOAP runs over the existing Internet infrastructure without being tied to any operating system, language, or object model [[Microsoft, 2005a](#)].²²

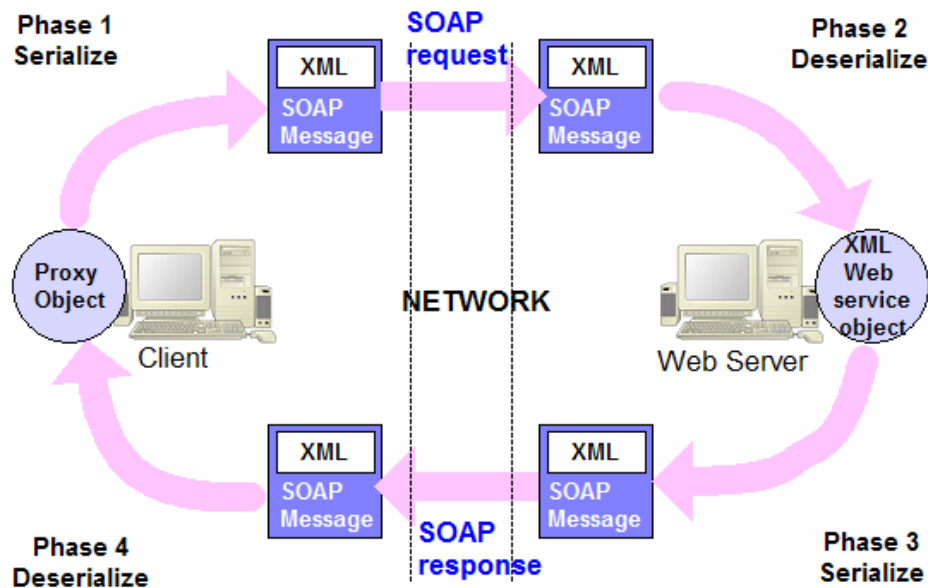


Figure 4-2: The SOAP Process [adapted from [Microsoft, date unknown](#)]

One of the significant changes between versions 1.1 and 1.2 of SOAP that is not taken into account in the glossary definition above is that the concrete binding to HTTP was provided not as a proscription but as an illustration of what was possible. As version 1.2 states:

SOAP enables exchange of SOAP messages using a variety of underlying protocols...

The HTTP binding in SOAP 1.2 Part 2 *illustrates* [my italics] the specification of a binding. Additional bindings can be created by specifications that conform to the binding framework introduced in this chapter [[Gudgin et al., 2003](#)].

²² A surprisingly disorganized Microsoft (2005) webcast on interoperability between .NET and Apache Axis, given by a Microsoft Regional Director, also defined SOAP in terms of its acronymic sense [[Ruebush, 2005](#)]. (Ruebush also consistently interpreted the "I" in "WS-I" as standing for Integration rather than for Interoperability.) Microsoft are not alone, however. The latest Systinet web-services primer also uses the acronymic sense, despite the fact that Systinet's latest server implements SOAP 1.2 [[Systinet, 2005](#)].

Over two years after version 1.2 was ratified, a 2005 webinar on interoperability, jointly produced by Microsoft and Systinet, still uses SOAP 1.1 "for interoperability" [[Guest, 2005a](#)]. An article written later in 2005 for the O'Reilly online publication *On Java*, says much the same: "JAX-RPC 1.1 mandates the use of SOAP 1.1" [[Panda, 2005](#)], and the first sentence on the Sun Developer Network site for JAX-RPC, in May 2005, reads as follows: "You can use the Java API for XML-based RPC (JAX-RPC) to build Web applications and Web services, incorporating XML-based RPC functionality according to the SOAP 1.1 specification" [[Sun, 2005](#)].

In all fairness it should be added that the original version of JAX-RPC 1.1 preceded SOAP 1.2 and is also about to be superseded by version 2.0 (already in late November, 2005, it is in public review release) which fully supports SOAP 1.2. It should further be noted that the *Windows Communication Foundation* (WCF), the new Microsoft web-services framework, which is to be released in the later part of 2005 (discussed in section 6.4 of Chapter 6), is based on SOAP 1.2. The coexistence of these offerings with the statements given above reveals something of the muddle surrounding standards and versions in web services. It will be interesting to see whether the Basic Profile updates its support for SOAP. Currently it supports only SOAP 1.1 .

4.2.2 SOAP as a Requirement for Web Services

Based not only on XML 1.0²³ but also on the W3C specifications for XML Schema and XML Namespaces, SOAP is seen by many as an essential part of web services. Its popularity may be explained by a list of its advantages given by Daniels *et al.* [[2004](#)], who represent it generously as:

- A mechanism for defining the unit of communication...
- A processing model ...
- A mechanism for error handling...
- An extensibility model...
- A flexible mechanism for data representation...
- A convention for representing Remote Procedure Calls (RPCs) and responses as SOAP messages...

²³ SOAP 1.2, like WSDL 2.0, is based on the XML Infoset.

- A protocol binding framework.

The requirement that SOAP should be used as the messaging format for web services cannot easily be divorced from the requirement that WSDL should be employed as its definition language, and not only because WSDL contains named extensibility elements that refer to SOAP. It might be argued (and indeed is – see Chapter 5) that XML schema can by itself replace the functionality of SOAP and WSDL, with messages in the form of schema-conformant XML documents sent across the wire. Daniels *et al.* [2004] have a response to such a suggestion: if you formalize your business logic only in a schema, the schema will have to be extensible to cope with change, and you then run the risk of violating the principle of loose coupling by forcing client business processes to change along with them. Viewed from this standpoint, a schema alone may be seen as too fine-grained. In contrast, SOAP, they argue, has "vertical" extensibility built into its framework through the presence of SOAP headers, and "horizontal" extensibility through the concept of intermediaries or applications that may:

- process parts of the message while it is still in transit;
- help to increase the scalability of a distributed service;
- provide value-added services such as security.

The use of SOAP, they conclude, enables the creation at the outset of processing methods that future changes will not break.

4.2.3 Problems with SOAP

Initially and sometimes still defined as "simple" and "lightweight" [Gudgin *et al.*, 2003], SOAP turns out to be neither. It needs XML parsers on client and server, and the overheads of using it can hardly be described as lightweight: examples of such overheads were illustrated in Section 2.4.2.5.1, and the following SOAP message contains a payload of a mere 24 bytes (in bold below) in a declared message-content total of 756 bytes.

```
POST /axis/services/DictServiceImpl HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.2RC3
Host: 127.0.0.1:1234
Cache-Control: no-cache
```

```

Pragma: no-cache
SOAPAction: ""
Content-Length: 756

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:storeEntryInDB
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="dct">
      <de href="#id0"/>
      </ns1:storeEntryInDB>
      <multiRef id="id0" soapenc:root="0"
        soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/
        encoding/"
        xsi:type="ns2:DictEntry"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns2="urn:dct">
        <headword xsi:type="xsd:string">weirdness</headword>
        <plural xsi:type="xsd:string">weirdnesses</plural>
        <pos xsi:type="xsd:string">noun</pos>
      </multiRef>
    </soapenv:Body>
  </soapenv:Envelope>

```

Although the declared encoding for the XML is UTF-8, the message given above has been sent in that encoding only because the declared HTTP *character set* has been set to `utf-8`: the SOAP 1.1 content-type of `text/xml` causes the XML document encoding to be ignored, for reasons that were explained on page 45. Complexities of character encoding thus further detract from the so-called simplicity of SOAP.

With this example, the source of the problem is that the version of SOAP supported in the release of Axis that was used for the service is still 1.1. Axis 1.3 has more support for SOAP 1.2²⁴, which requires the content-type to be more specifically `application/soap+xml` as a means of avoiding the confusion of a SOAP message with any other type of XML content. Because even Axis 1.3 is still an implementation of JAX-RPC 1.1, and defaults still to SOAP 1.1, it uses HTTP `POST` for nearly all its SOAP requests, despite the inclusion in version 1.2 of the safer, idempotent HTTP `GET`. (Section 7.2.1, however, contains a simple example of the way in which it is possible to use HTTP `GET` with Axis 1.2.)

²⁴ The API in Axis 1.3 offers more opportunities to choose SOAP 1.2, although there is virtually nothing in the documentation, most of which has not been upgraded from Axis 1.2.

The standard language bindings for SOAP 1.2 are for XML Schema data types but, where more complex data types are involved, the situation becomes correspondingly more complicated, even with the use of standard JavaBean representations. There are considerable differences between simple JavaBean classes that a programmer might write to represent a complex type, and those generated from a schema referenced in a WSDL by, for example, Apache Axis. (WebLogic generated JavaBean classes are actually simpler.) The Axis-generated beans contain methods for serialization and deserialization as well as for complex type descriptions using the XML Schema QName data type, which refers to a namespace-qualified name required in WSDL. The following example was generated from a hand-written WSDL for a web service:

```
static {
    typeDesc.setXmlType(new javax.xml.namespace.QName("urn:dct",
        "DictEntry"));
    org.apache.axis.description.ElementDesc elemField = new
        org.apache.axis.description.ElementDesc();
    elemField.setFieldName("headword");
    elemField.setXmlName(new javax.xml.namespace.QName("urn:dct",
        "headword"));
    elemField.setXmlType(new
        javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema",
        "string"));
    elemField.setNillable(true);
    typeDesc.addFieldDesc(elemField);
    elemField = new org.apache.axis.description.ElementDesc();
    elemField.setFieldName("plural");
    elemField.setXmlName(new javax.xml.namespace.QName("urn:dct", "plural"));
    elemField.setXmlType(new
        javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema",
        "string"));
    elemField.setNillable(true);
    typeDesc.addFieldDesc(elemField);
    elemField = new org.apache.axis.description.ElementDesc();
    elemField.setFieldName("pos");
    elemField.setXmlName(new javax.xml.namespace.QName("urn:dct", "pos"));
    elemField.setXmlType(new
        javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema",
        "string"));
    elemField.setNillable(true);
    typeDesc.addFieldDesc(elemField);
}
```

The level of complexity can be appreciated by a comparison between the JavaBean, partially rendered above, and the simpler schema construct below, from which it was generated:

```
<xs:complexType name="DictType">
  <xs:sequence>
    <xs:element name="headword" type="xs:string" nillable="true"/>
  </xs:sequence>
</xs:complexType>
```



```

<xs:element name="plural" type="xs:string" nillable="true"/>
<xs:element name="pos" nillable="true">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="noun"/>
      <xs:enumeration value="verb"/>
      <xs:enumeration value="adjective"/>
      <xs:enumeration value="adverb"/>
      <xs:enumeration value="article"/>
      <xs:enumeration value="preposition"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:schema>

```

Levels of code complexity vary with WSDL code-generators. It could be argued that potential code complexity is one more reason for starting a web service with a WSDL and a schema, rather than with interface and implementation classes. This issue is discussed at greater length in section 4.3.3.

Although SOAP has been widely adopted in industry applications and especially in vendor toolkits, not all SOAP implementations are created equal, not least in which version of SOAP they support, although that problem should be resolved as newer toolkit versions are released and more vendors scramble to comply with the Basic Profile. Even when the versioning is taken care of, there could still be interoperability problems if vendors were to insert proprietary details that would lock customers in to their product. The SOAP header inside the SOAP envelope, for example, is an extensibility element capable of modifying the message in many implementation-dependent ways. The standards compliance of implementations tested (see Chapter 7) suggests that most vendors are currently anxious to conform to the requirements of the Basic Profile and to be seen to be interoperable.

A source of confusion is that rules for SOAP messages are distributed across both WSDL and SOAP specifications. SOAP messaging styles, for instance, rely on protocol bindings which are usually given as SOAP extensibility elements in WSDL documents, for example:

```

<wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="tns.storeEntry">
    <wsdlsoap:operation soapAction=""/>

```

```
<wsdl:input name="tns.storeEntryRequest">
  <wsdlsoap:body use="literal"/>
</wsdl:input>
```

where the `wsdlsoap` elements are the commonly used extensibility elements for SOAP.

A further interoperability problem that neither SOAP nor WSDL addresses is occasioned by the lack of granularity in message sequencing that does not extend beyond single message pairs. Zur Muehlen, Nickerson and Swenson point out that "complex business scenarios that require the sequencing of several message pairs cannot be described sufficiently using SOAP and WSDL, but an additional standard is needed". They suggest that:

an information system needs to keep track not only of the individual request–response or notify–response message pairs, but it also needs to correlate the different message pairs to an overall context, so that it can identify messages that are duplicates or out of sync. In essence, the description of the overall interaction requires a process model [2004].

4.2.4 The SOAP Message Structure

SOAP is both a messaging protocol and a message specification. Although it may seem useful to examine SOAP separately from these two different viewpoints, there are areas in which the two cannot be isolated. The messaging specification is first briefly considered. Figure 4-3 gives a graphical representation of the XML structure of a simple SOAP message.

SOAP 1.2 changes the message structure slightly in that it no longer permits elements inside the `Envelope` but outside the SOAP body. Another version change, which is really more of a clarification, rests on the SOAP messaging style as determined in the WSDL file [[Gudgin et al., 2003a](#)]. If the style is given as `RPC`, only one child element is allowed within the SOAP body apart from the SOAP fault – although that "child" may have any number of "grandchildren" elements. By default and by convention, if the style is given as `document`, any number of parallel "child" elements are allowed within the SOAP body. The word "document", used in apposition to the binding to "RPC", actually appears nowhere in that sense even in SOAP 1.2. The concept of style arose in the WSDL specification and is discussed in Section 4.2.4.1 below.

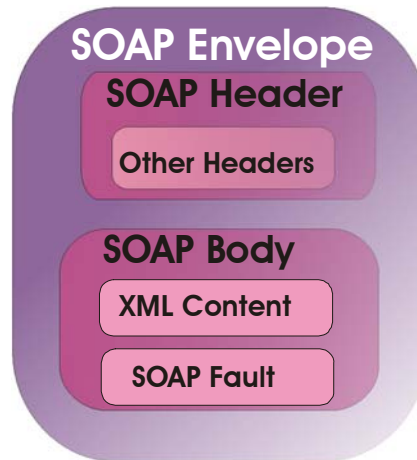


Figure 4-3: A representation of the basic SOAP Message Structure

A more complex SOAP message may also include binary attachments following the specification for *SOAP with Attachments* (SwA) [Barton et al., 2000], originally developed by Microsoft who then rejected it as a failure. The position with regard to binary attachments changed at the beginning of 2005 when the W3C granted recommendation status to the *SOAP Message Transmission Optimization Mechanism* (MTOM) [Gudgin et al., 2005], which looks set to replace both the SwA model and the Microsoft proprietary alternative, *Direct Internet Messaging Extensions* (DIME), also later rejected as a failure [see McMillan, 2003]. While SOAP with Attachments permitted binary files to be carried outside the SOAP message infrastructure, MTOM makes these binary elements part of an XML information set within the SOAP envelope. In doing this, it complies with WS-I security requirements because, as part of the message, the attachment may now be signed.

MTOM goes even further: it not only allows for the binary encoding of XML within a SOAP message but it also permits the encapsulation of binary-encoded XML within an HTTP message, regardless of whether or not SOAP is being used for the message exchange. The recommendation aims to solve the problems of interoperability arising from the use of binary attachments and will help in circumstances which involve, for example, the transmission of a binary image file between different platforms.

MTOM is a solution that will enable developers to make full use of existing binary formats such as GIF, JPEG and PDF within an XML context. MTOM makes this possible by using an

alternative serialization of XML known as XOP or *XML-binary Optimized Packaging*. XOP is explained briefly by one of the XML Protocol Working Group:

..XOP is an alternate serialisation of XML ... with an XML document as the root part.
...This allows you to avoid the bulk and overhead in processing associated with encoding, the only way that you can fit binary data directly into an XML world...XOP can be used for any XML-based format; MTOM is just a description of how XOP is layered into the SOAP HTTP transport [[Nottingham, 2004](#)].

Bray's response to Nottingham's views on MTOM and XOP represents the suspicions concerning interoperability and the possible balkanization of XML shared by much of the XML community:

..it's reasonable to be nervous about MTOM or any other kind-of-XML-but-not-quite, because there's the potential that when someone says "I'll send you XML" it won't be clear any more that you can expect Unicode-with-angle-brackets, and that's a real loss. I'm prepared to be convinced that the way MTOM is presented and packaged will ameliorate that risk enough [to] give it a positive return on investment; but it is reasonable to be worried [[Bray, 2004a](#)]²⁵.

A tangible interoperability issue is whether the vendors will support MTOM or deliver their own proprietary binary formats, as the *Windows Communication Framework* (see section 6.4 in Chapter 6) already does alongside the MTOM standard [[Microsoft, 2005b](#)], but misuse of, or the failure to comply with, a standard can hardly be attributed to the existence of that standard! JAX-WS supports it, as does *Visual Studio 2005*. A matter of interest is whether (or when) the WS-I Basic Profile, which currently supports SwA will switch to supporting MTOM²⁶.

²⁵ See also Bray's comments: "The question is, should there be an official binary encoding that is labeled as 'Binary XML'? The World Wide Web Consortium is looking pretty intensely at that. A lot of us are pretty dubious at that idea. You'd like it to be smaller, easier to parse and be self-describing in the same way that XML is. But it is far from clear that you can have one binary encoding that will meet all these objectives" [[Bray, 2005](#)].

²⁶ The idea of encoding binary data is not new to XML. XML Schema defines a value space for `base64Binary` and `hexBinary` data, representing them as the actual octets [[Biron and Malhotra, 2004](#)], and XML itself allows for non-text items such as image files to be represented by URI values as attributes where the specification defines an unparsed entity as "a resource whose contents may or may not be text, and if text, may be other than XML... Beyond a requirement that an XML processor make the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities" [[Bray et al, 2004](#)].

4.2.4.1 Service Styles and their Encodings

Many Java implementations of SOAP, even those which are not commercial such as Apache Axis, still use a default messaging style of RPC/encoded. Loughran and Smith go so far as to state that: "The bias is such that, for Java development, it is widely seen that JAX-RPC *is* SOAP" [2005a]. The emphasis looks set to change for Java toolkits, however, when JAX-WS is officially released, because its default messaging style is document/wrapped literal. Experimentation in late November, 2005, with the latest Java SOAP toolkits which use this style as their default may be seen in Chapter 7, especially section 7.2.7.

Document/wrapped literal is the default .NET style and Microsoft implementations are explored further also in Chapter 7, sections 7.2.2 and 7.2.4.3.

The WSDL specification defines the difference between an RPC messaging style and a document messaging style as follows:

The style attribute [the `<wsdl:soap:binding>` `style` attribute] indicates whether the operation is RPC-oriented (messages containing parameters and return values) or document-oriented (message containing document(s)). This information may be used to select an appropriate programming model. The value of this attribute also affects the way in which the Body of the SOAP message is constructed... [Christensen *et al.*, 2001].

This definition makes it very clear that the distinction is between SOAP *messaging* styles, which do not necessarily dictate the *programming* style.

RPC and document are the two possible named values in the WSDL specification for the `style` attribute but, through the separate `style` attribute in its descriptor file, Axis follows the mandate in the JAX-RPC 1.1 specification [Chinnici *et al.*, 2003], as well as adopting the proprietary Microsoft format, in introducing a third possibility, `wrapped`, which is a variant of `document`. RPC style was initially the more popular despite the fact that document style was given as the default in the WSDL specification²⁷, but RPC has lost popularity as the move towards document-

²⁷ "If the `soap:binding` element does not specify a style, it is assumed to be 'document'" [Christensen *et al.*, 2001].

style has gained momentum over the last two years, exemplified by the most recent systems examined in Chapter 7. Newcomer says of the document-oriented style:

As the simplest and most abstract form of Web services, the document oriented style has the advantage of preserving more of the characteristics of XML that make it so helpful - namely its independence from any one particular programming language and [its] unique type system. Because XML can be used as an independent data type system, it can be used to solve one of the biggest problems in integration and interoperability, data type compatibility [[Newcomer, 2005](#)].

The second part of the service style is set in the `<wsdl:soap:body>` `use` attribute, which determines the XML encoding that will be used for any data types in the message that cannot be interpreted simply as literal string values. The two possible values for the `use` attribute are `encoded` and `literal`. The choice of `encoded` (still the default for some toolkits, including even Apache Axis 1.3, but now beginning to change, as evidenced in Chapter 7) was the usual combination with RPC, and is now heavily discouraged by the Basic Profile [section 4.7.4, [Ballinger et al., 2004](#)]. [E]ncoded means that data types will be encoded into XML according to the SOAP version 1.1 section 5 encoding, which preceded the use of W3C Schema and includes a very limited set of data types. Because the data types are so limited, the potential for interoperability problems is greater with this encoding, as was recognized by the Basic Profile.

The other choice, `literal`, usually but not exclusively means that the data types will be encoded into XML according to W3C Schema rules and is now the popular choice, supported by the Basic Profile. SOAP version 1.2 is more liberal than its predecessor, allowing for the use of other schema types, including the increasingly popular RELAX NG (*Regular Language Description for XML, Next Generation*). The potential to use a variety of schema types does not detract from interoperability because schema namespaces are clearly indicated in the WSDL namespace sections.

Because the `style` and `use` attributes are generally considered together to form a service model, it may be helpful to consider briefly these pairs of options before going on to look at the messaging styles they indicate:

- `RPC/encoded` is specifically disallowed in the Basic Profile because it is thought to duplicate the function of XML namespaces, as well as provide too limited a set of data types.
- `rpc/literal` means that the encoding rules are specified by a schema and this option is permitted in the Basic Profile but tends to be a rarer choice, perhaps because of the lessening in popularity of the RPC binding. The PHP service in Chapter 7 (section 7.2.9) was implemented with this style, as was the Date service within Axis (section 7.2.2).
- `document/literal` carries overheads for the developer of schema design and validation, on top of WSDL construction, plus any time it might take to negotiate the terms of the schema with service clients. This approach, which usually starts with the Schema and the WSDL, is considered a top-down approach, also referred to as a *WSDL-First* approach, and is discussed later in section 4.3.3 of this chapter as well as throughout Chapter 7.
- `document/encoded` is not an option required to be implemented in JAX-RPC, although examples of it apparently occur in the literature [see [Tyagi, 2004a](#)]. The Basic Profile does not permit the encoded option and so this alternative is unlikely to be implemented in the future. No attempt to implement it was made in this study.
- `document wrapped` is a further style option (not included in either the SOAP or WSDL specifications) that was created originally by Microsoft to deal with identification problems arising from the use of straight document style, details of which are given later in this section.

The rest of this section examines the different style choices, together with their implications for the structure and interoperability of SOAP messages.

4.2.4.1.1 RPC

If the WSDL `style` value appears as `RPC`, defining "a uniform representation of remote procedure calls and responses" [[Box et al., 2000](#)], the first and only "parent" XML element inside the SOAP Body element is named after the method request (or response). Child sub-elements are used to represent the parameters or arguments to this method, usually encoded after the SOAP section 5 encoding, now disallowed by the Basic Profile. Together the parent element

(add) in the example that follows, and its child elements (two integers), are taken to be a rendering in XML of a method call and will produce the following SOAP message:

```

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <oxy:calc.add xmlns:oxy="urn:calc"
      SOAP-
      ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <i1>15</i1>
      <i2>9</i2>
    </oxy:calc.add>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Parent element giving the method name

The advantage of the RPC- style, as may be seen in the SOAP message given above, is that it clearly names the method, useful for identification to a service handling a number of different methods which may take similar parameters. The PHP sayHello service described in section 7.2.9 of Chapter 7 and the Java Calculator service described in section 7.2.1 both illustrate why identification is necessary. When more than one method call is possible, a document-style service cannot easily distinguish between the methods. A disadvantage of RPC style, whether the encoding is *encoded* or *literal*, is that the only parts of a schema that appears in the message are the method parameters, making XML validation more limited. Butek argues against the RPC/*encoded* style with the complaint that "The type encoding info... is usually just overhead which degrades throughput performance" [2005]. (Using *literal* in place of *encoded* removes the need for type encoding information inside a message.)

4.2.4.1.2 Document

If a WSDL *style* value is set to *document*, any valid XML may be contained by the `<SOAP-ENV:Body>` element. A document-style SOAP message in which an actual document is being included as the message content might look like this:

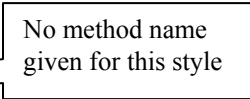
```

< SOAP-ENV:Body>
  <ns:someDocument xmlns:ns="appropriateURI">
    ...[XML document to be passed]
  </ns:someDocument>
</ SOAP-ENV:Body>

```


When document-style services are used for method calls (RPC *programming* style), the WSDL should contain a `<types>` element, defining or referencing any schema data types used, and these types become the *direct child elements of the SOAP body element*, as shown in the example below, which uses the W3C Schema encoding to send integer parameters to the calculator service and method that were used in the first illustration:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <i1 xmlns="urn:calc">15</i1>
    <i2 xmlns="urn:calc">9</i2>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



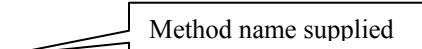
This style is less appropriate for method calls because it loses the method name and is therefore less interoperable. Sections 7.2.1 and 7.2.9 in Chapter 7 examines what actually happens with services when this messaging style is chosen. An example such as the one given above also contravenes the WS-I prohibition against having more than one top-level element inside the SOAP body, excluding a fault element [see section 4.7.10 in [Ballinger et al., 2004](#)]. For XML documents, however, this style is ideal because everything inside the SOAP `body` element may be validated against a schema.

4.2.4.1.3 *Wrapped*

The "wrapped" style is meant for use with method calls, was developed initially for .NET web services and is supported by JAX-RPC version 1.1, which describes it as "wrapping" the parameters to a method call inside "an `xsd:sequence` [or XML Schema `complexType`] element named after the operation [element in the WSDL]" [[Chinnici et al., 2003](#)].

Manes explains that the wrapped style "produces document/literal services, and yet it supports an RPC-style programming interface" [[Manes, 2005a](#)]. A calculator service wrapped example illustrates how the wrapped style is more useful for method calls in its inclusion of a method name:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
```



```

    <add xmlns="urn:calc">
      <i1>15</i1>
      <i2>9</i2>
    </add>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Where a complex data type is involved, the wrapped style goes even further and "unwraps" a complex type into its constituent parts, which are then supplied as parameters inside the SOAP message, as the example below illustrates:

```

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <storeEntry
      xmlns="http://localhost:8080/axis/services/DictServiceImpl">
      <de>
        <headword>weirdness</headword>
        <plural>weirdnesses</plural>
        <pos>noun</pos>
      </de>
    </storeEntry>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Method name supplied

Name of the complex type

Here the complex data type `de` is decomposed into its parts, which are `headword`, `plural` and `pos`. These parts are enclosed inside an element bearing the name given to the data type in the `types` section of the WSDL, and that element is itself enclosed inside an element with the method name `storeEntry`. As with the calculator service example, it can be seen that this style ensures that the method name is not lost, a clear advantage over the straight `document` style for a method invocation.

At a first glance, the wrapped style looks very similar to the RPC style. Butek explains the difference:

In the RPC/literal SOAP message, the [element named for the method] child of `<soap:body>` was the name of the operation. In the document/literal wrapped SOAP message, the [element named for the method] clause is the name of the wrapper element which the single input message's part refers to. It just so happens that one of the characteristics of the wrapped pattern is that the name of the input element is the same as

the name of the operation. This pattern is a sly way of putting the operation name back into the SOAP message [[Butek, 2005](#)].

Method overloading would not work with a wrapped-style service. Method overloading is in any case not permitted by the Basic Profile stipulation that operation names must be unique [section 4.5.3, [Ballinger et al., 2004](#)].

In addition to automatic construction of SOAP messages from the WSDL `style` attribute, Apache Axis provides another approach, based on the *SOAP with Attachments API for Java* (SAAJ) and sent over JMS, in which the user programmatically constructs a SOAP message. JMS is a Sun API, not supported by Microsoft, but bridges linking the two are available and implementers of message queues such as IBM have created .NET-specific implementations that will interoperate with JMS.

4.2.4.2 SOAP Programming Models

It is important to note that, although service styles dictate the format of SOAP messages that will be sent over the wire and are *intended* to correlate with the programming approaches or message exchange patterns that handle the actual sending and receiving of the messages, messaging and programming styles should be seen as *distinct*. It is, for instance, possible to format a message as document style and then send it over a protocol that uses RPC.

Haas and Brown define the required binding of a SOAP message to a protocol as a "formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange" [[2004](#)]. It is usually the choice of a protocol that determines the MEP, although the latest toolkits such as BEA's WebLogic and *Visual Studio 2005* have mechanisms for both synchronous and asynchronous messaging which are protocol-neutral and conform to the *Web Services Addressing* (WS-Addressing) specifications, two of which are currently at the W3C "Candidate Recommendation" stage [see [Gudgin and Hadley, 2005](#), [Gudgin and Hadley, 2005a](#), [Gudgin and Hadley, 2005b](#)]²⁸.

²⁸ Although not one of the core specifications such as SOAP and WSDL, the WS-Addressing specification needs to be mentioned, not only because it is one of the areas in which Sun and Microsoft have recently (2005) joined forces but also because of the significant effect that its implementation will have on the SOAP protocol bindings. WS-

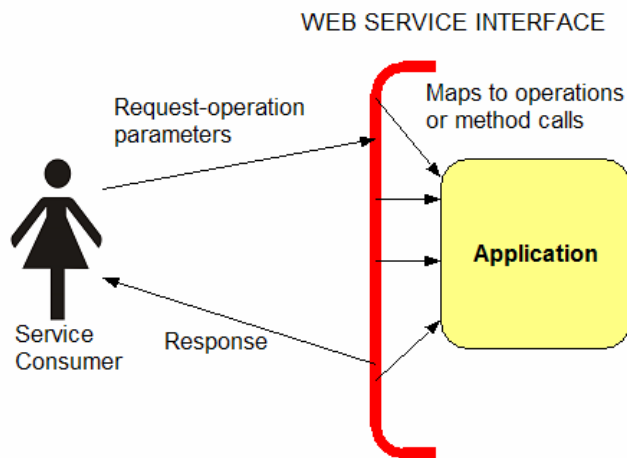


Figure 4-4: An RPC Programming Model [adapted from [Tyagi, 2004a](#)]

4.2.4.2.1 RPC

The choice of protocol is usually determined by the toolkit. RPC is still the protocol most widely used for SOAP messages and in many RPC-based toolkits the MEPs are hidden from the user. A toolkit such as Axis will, for example, interpret an RPC messaging style choice as a call to create a service proxy stub on the client side which looks like the real procedure, but prepares and transports data across the interface. It actually marshals or gathers procedure-call parameters into a SOAP message resembling the one given on page 74. The serialization of the message into XML (determined by the WSDL `use` attribute) will be handled by the underlying protocol, which is usually HTTP in the case of RPC.

The reverse process happens on the server, where a listener process, such as a servlet or JSP page, deserializes the transported buffer stream and calls on another stub to unmarshal and decode the parameters and then bind them to internal variables and data structures, after which the called procedure is invoked. (Section 7 in SOAP 1.1 (or Adjuncts in SOAP 1.2) defines rules

Addressing aims to include the service endpoint inside a specially designed SOAP header, rather than at the transport level, so that protocols other than HTTP may carry it efficiently, particularly when a message is designed to be carried over several "hops" by more than one protocol. The downside of the specification is that it does not sit well beside those sections of SOAP 1.2 (quoted on page 44) intended to assure transparency of URIs, and that it contradicts the earlier, more generalized definition of web services given by the W3C (quoted on page 29).

for marshalling and encoding SOAP messages but gives no details containing serialization [Box *et al.*, 2000].) Figure 4-4 illustrates the interactions of a service with an RPC programming model.

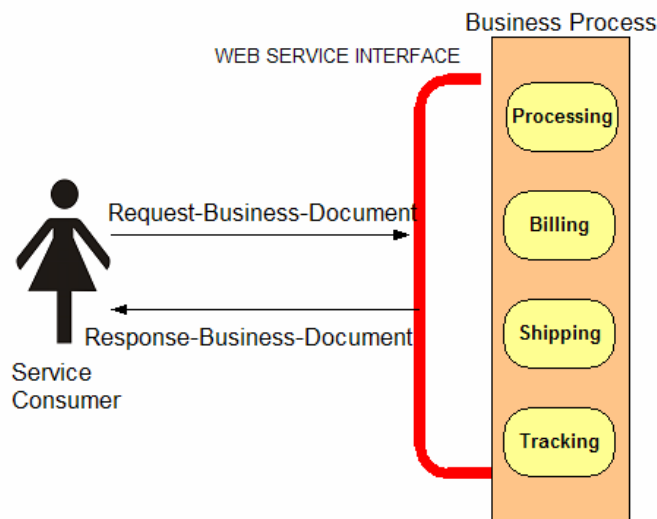


Figure 4-5: Document-Style Interaction [Adapted from [Tvagi, 2004a](#)]

4.2.4.2.2 Document

With a document style *programming model*, the assembling or marshalling of the data is left to the programmer's choice, because a document-style message is a wrapper for any plain XML content that is sent over the wire using any transport protocol of choice. The message may use namespaces either on a SOAP element or, in the case of an embedded document, on the document root element, to indicate schemas that will aid the recipient to interpret it, as may be seen from the document-style examples in the previous section. If no namespaces are provided, the client and server must agree on the procedures to be followed, which makes for closer coupling in that the client has to know the service in advance. (WSDL 2.0 requires namespaces, to preempt such a situation.) The handling of the message may be done in a variety of ways, with parsers, for example, or XSLT processors. It is possible that the client and server will use different technologies to serialize/deserialize and marshal/unmarshal the message.

The interactions of a service using a document style programming model should normally follow those illustrated in Figure 4-5, although many RPC-based toolkits conceal from their users that, even for a document-style message, they are still employing RPC for the message transmission.

4.2.4.3 Advantages and Disadvantages of the RPC Processing Model

SOAP over RPC was widely adopted at the outset of web services, probably because RPC had been conventionally used in distributed-object applications, looked more like CORBA and DCOM, and was therefore familiar. RPC is identified with object- and component-oriented technologies and, as was seen in Chapter 2, suffers from the problems of tight coupling and synchronicity (blocking on a remote call can be disastrous for an application).

The combination of RPC with XML has become increasingly controversial – hence the name change for the next JAX-RPC specification to JAX-WS. If web services are a *messaging* technology, a remote procedure call is more of a proscriptive *command*. Various other arguments put forward against sending SOAP messages over RPC are its tight coupling, its reliance on synchronous technology, its slowness, verbosity and requirement of heavy processing on both client and server. Possibly the strongest argument is given by Shah and Apte, who explain:

RPC [programming] -style services attempt to exchange technology-specific data objects and APIs by serializing the data objects into XML and de-serializing again at the other end. Since web services are technology- and platform-neutral, de-serialization on the client side (based on the web services engine) to convert back to the data objects may not re-create the exact signature of the objects as defined by the provider [2004a].

Shah and Apte argue that even using the same technology on the server and the client (for example, the Java2WSDL and WSDL2Java tools within Axis) will produce different versions of an API, while different technologies will produce an even greater variance, resulting in a possibly damaging loss of control over the business interface. (Experimental results testing this assertion may be found throughout Chapter 7 and especially in section 7.2.4.4.) Shah and Apte concede, however, that the RPC-style may still be the best choice for converting legacy systems

into web services [2004a]. Butek and Scheuerle stress that the success of "round-tripping"²⁹ depends on proper adherence to Java coding conventions in the first place [Butek and Scheuerle, 2004a], although findings in section 7.2.2 might indicate some engineering taking place behind the scenes to make incompatible types appear compatible. Increasingly tools such as Altova *MapForce* are easing data type conversions. An example of the use of *MapForce* as an aid to code generation is given in section 7.2.5 of Chapter 7.

Loughran and Smith offer some similar arguments against using RPC with SOAP (in this case referring to the use of XML for method calls), when they illustrate the mismatch between XML and Java objects, contending that the semantic mappings underlying the terms "serialization" and "deserialization" are at the base of the problem. They term it the O/X (Object to XML) mapping, along the lines of the infamous O/R (Object to Relational Database) mapping, to emphasize the intractable nature of the problem:

Undermining it all is a fundamental difference between the type systems of XML (especially that of XML Schema) and that of Java, making any mapping both complex and brittle [2005a].

Among their examples of mainly data-typing incompatibilities not listed elsewhere are:

- the way that XML Schema can define new types by restriction while Java classes use inheritance to define new types only by extension (for example, an *ISBN-type* might be seen as a restriction of a schema `string` type, whereas an ISBN Java class would inherit from `String` and could not limit the use of normal `String` functionality).
- The related problem of representing XML schema enumerations, even with Java 1.5: "The enumeration names in the Java source no longer contain any informative value at all, other than a position number in the set. Any change to the enumeration could reorder the values, without this change being detected by code that used the enumeration. The defect would only show up in interoperability testing" [Loughran and Smith, 2005a]. Schema enumerations are restrictions on any kind of values other than boolean, and, unlike Java `enum` values, have no underlying order. The problem described here could

²⁹ Defined as "the process of mapping from one representation to another and back again" [Butek and Scheuerle, 2004b], it is concerned mainly with the accuracy of server-side code.

occur, but only as the result of gross negligence on the part of those altering the service code, without maintaining some kind of versioning indication.

- The non-trivial difficulty in mapping the much broader set of XML-permissible names to the narrower set of Java identifiers, along with the problem that newer versions of Java may break existing structures by including as keywords names that have been used (`enum` being one example of this, already seen as a problem in earlier versions of Axis after Java 5 was released).
- The difficulty of representing XML namespaces as package names, for the same reasons that bedevil identifier-mapping, mentioned above, although Loughran and Smith admit that Java 1.5 annotations will help to solve this by enabling the association of metadata with generated types.
- Problems arising from the common non-validation of the serialized XML message that is received, that have the potential to prevent interoperability (there is no requirement that messages or their content be validated).
- The inability (at least of JAX-RPC 1.1) to handle non-serializable data, despite the ability that SOAP gives to include it. MTOM offers one possible solution to this problem.

Many of the above data-typing issues might be perceived as extreme cases to be avoided in web-service implementation. In section 7.2.4.4 of Chapter 7, round-tripping is explored as one solution to data-type problems. However, it is difficult to consider SOAP data-typing issues in isolation from WSDL. While the source of the problem is the RPC-style programming model which requires the exchange of data types between programming languages and XML, it should be seen in the context of the WSDL document in which the data types are declared. This issue is therefore taken up from a slightly different angle later in this chapter in section 4.3.2.

JAX-RPC 1.1 has other interoperability problems, among them the difficulties of using the mandatory `java.rmi.Remote` and `java.rmi.RemoteException` classes that were discussed in Chapter 3. Method parameters must also be JAX-RPC-supported Java language types. Tyagi phrases carefully, if ambiguously, the relationship between the Basic Profile and JAX-RPC: J2EE containers "must ... satisfy all the interoperability requirements from WS-I Basic Profile

outlined in JAX-RPC [my italics]"[2004a]. Loughran and Smith go even further by suggesting that JAX-RPC tries to make SOAP look like RMI [2005a].

RPC is, however, acknowledged to be the better approach for incorporating non-XML legacy systems into web services and might therefore be said to perform better in a hybrid environment. It works well on an intranet, where the problems of treating remote objects as if they were local ones are not so obvious or serious. It also has the advantage of following well-known and standardized procedures and offers optimizations for the overheads of marshalling and serialization. Neither client nor server is left in the dark about what to do with an RPC call.

4.2.4.4 Advantages and Disadvantages of the Document-Style Programming Model

If the RPC-style programming model has the disadvantages outlined in the previous section, what are the advantages of the document-style programming model? One obvious advantage is that the document-style programming model is using XML as it was originally intended: it exchanges data not object-oriented code and, because the messaging format can be seen as a packaging strategy for actual documents, the interface can be much more coarse-grained than is needed for a method call. The document-style programming model is more suited to being asynchronous and may therefore be much more loosely-coupled. Because a document-style messaging format is also mapped onto a schema, it can be easier to process and validate.

The document-style programming model is excellent for passing complex business *documents* such as purchase orders, receipts and invoices – the kinds of documents for which schema-processing is essential. It is also an excellent choice when a service is broken into a subset of services, each of which handles a particular type of processing. Unlike the RPC programming model, it tolerates change better because of the type of processing that can be done by, for example, XSLT, which can accept details it wants from a document, and ignore the rest. The document-style programming model also works equally well for synchronous and asynchronous messaging.

Services following the document-style programming model are, however, often more complex to create and process. Because there is no standard way of handling a document-style message, some negotiation regarding the schema to be used is required in advance, which does not make for loose coupling. Document-style programming carries the same, if not greater overheads for marshalling and serialization, because more data may be transferred.

4.2.4.5 Different Types of Soap Client

The subject matter of this thesis is *web services*. To get a fuller picture of the issues surrounding interoperability it is helpful also to examine typical implementations of SOAP *clients* and how they interoperate with the services they target. The clients need to be able both to address the service in the way it expects and also to handle the details of any response that is returned. JAX-RPC 1.1 defines three different types of client, as may be seen from Figure 4-6: static stub, dynamic proxy and dynamic invocation interface. There is also the possibility of creating a SOAP client through the programmatic assembly of a SOAP message in an application client or a REST-style client through the programmatic exchange of XML messages directly over HTTP. Each of the three former client types can be represented as method calls, tied to RPC.

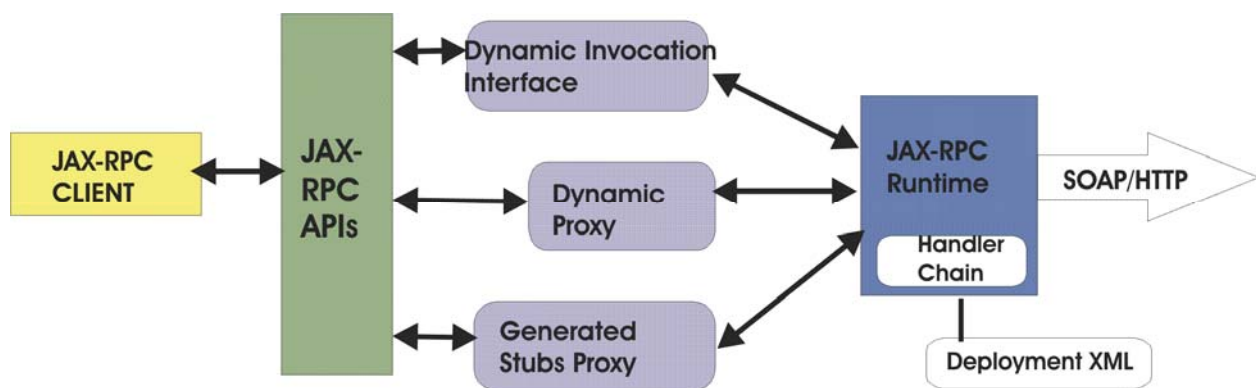


Figure 4-6: JAX-RPC Client Invocation Models [redrawn from [Joseph, 2003](#)]

4.2.4.5.1 Static Stub Client

A static stub client is an RPC version of RMI, a local object that acts as a proxy for the remote service, created at *development* time, not at runtime. It relies on an implementation-specific

class and needs to know either the WSDL or the interface in advance. Toolkits usually enable static stub client generation from the WSDL file. With Axis, one example of static stub client code generation produced approximately 170 lines of *supporting* code (excluding the static client itself, which had to be hand-coded) and generated four files. Programmatically, through a service *locator* object, a service object was retrieved, by which calls could be made on the remote service. An example of a static stub client may be seen in section 7.4.1 of Chapter 7.

4.2.4.5.2 Dynamic proxy client

Dynamic proxy clients are classes created at *runtime*, usually from the service interface, not from an implementation-specific class. They need to know either the WSDL or the interface in advance. They are not easy to autogenerate because their methods are hard-coded. Dynamic clients were created in files with approximately 44 lines of code, as opposed to the 170 lines of their static stub counterparts. An example of a dynamic proxy client may be seen in section 7.4.2 of Chapter 7.

4.2.4.5.3 Dynamic invocation client

With a dynamic invocation interface, a client can call a remote procedure even if the signature of the remote procedure or the name of the service is unknown until runtime, when the details may be discovered through some kind of broker (an original function of UDDI). An example of this style of client that attempts to call a service written in another language is discussed in Chapter 7, in section 7.4.3.

4.2.4.5.4 Application Client

As outlined briefly at the end of section 4.2.4.5, it is also possible simply to write, for example, a SAAJ client, in which it is the programmer who constructs the SOAP message. This method is more programming-intensive, relies on prior knowledge of the service and hard-codes the interface details, which could be a source of difficulties, should the client not be aware of later changes to the interface. Such an approach does not usually rely on a toolkit. No SAAJ clients

were written for this study. The only application client that was created was for a REST service and appears in section 7.2.8 of Chapter 7.

4.2.4.5.5 Summary of SOAP Client Approaches

Most of these different approaches to creating SOAP clients are predicated on the same object-oriented, method-call foundation and suffer from the same drawbacks as services built using the older technologies in that they are not truly *loosely*-coupled. Their focus is less on a service contract, with message-exchange patterns, and more on the transmission of object types. In SOA the boundaries between service and client should be distinct, with each side handling its own part of the contract. The RPC approach tends to hide those boundaries, making it seem, as did distributed-object technologies in the past, that the service sits in the same process as the client, the pitfalls of which were examined in section 2.4.1 in Chapter 2.

Welcome changes, however, look set to appear with the new JAX-WS, which introduces two new interfaces, `javax.xml.ws.Dispatch` and `javax.xml.ws.Provider`, with the explanation that "in some cases operating at the message level is desirable" [[Hadley and Chinnici, 2005](#)].

This will provide an alternative to other JAX-RPC APIs which "are designed to hide the details of converting between Java method invocations and the corresponding XML messages".

`Dispatch` will also provide support for asynchronous and for one-way messages. The results of some experimentation with the `Provider` interface is discussed in section 7.2.7.

4.3 WSDL: Web Services Description Language

The concept of a description (or definition) language did not begin with WSDL. CORBA and DCOM both used interface definition languages (IDLs), which the OMG had borrowed from *Common Business-Oriented Language* (COBOL). COBOL had used the term to define a declarative language that made it possible to describe a programming interface. WSDL, however, goes further than its predecessors. Like a SOAP message, each WSDL is an XML instance document that must be capable of being validated against its W3C schema by an XML processor, and from there it may become the basis for automatic code generation on both the client and the server.

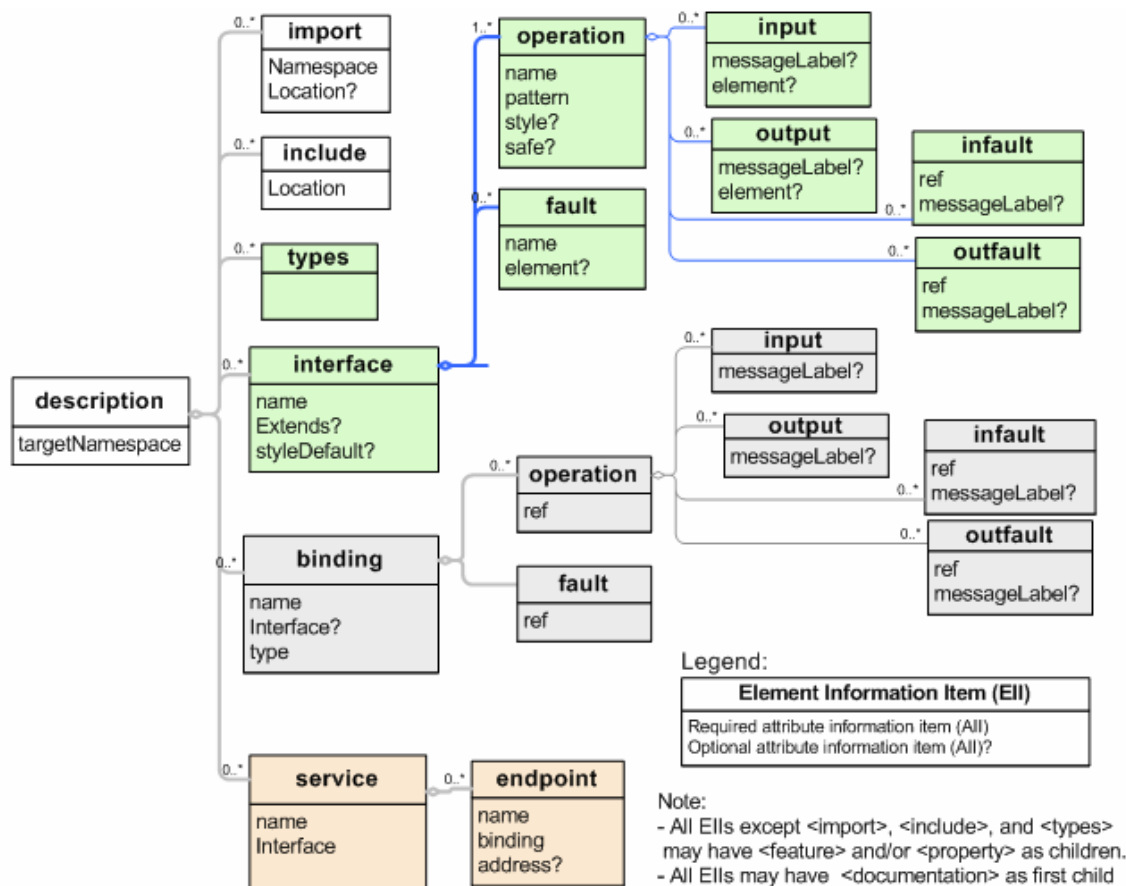


Figure 4-7: XML Infoset Diagram of WSDL 2.0 [Booth and Liu, 2005]

The primer to WSDL 2.0 describes the purpose of the specification in terms of language theory as defining the "sentences" available to the language (of web services) and also the meaning of each "sentence". Instead of using literal symbols and characters in that definition, however, WSDL 2.0 uses the XML Infoset (see Figure 4-7), which gives an abstract model of the *data*, as

distinct from the syntax, that may be found in a well-formed XML document [[Cowan and Tobin, 2004](#)]. (SOAP 1.2 also uses the Infoset.) Some dissenting voices in the XML community believe, however, that the interoperability of XML lies rather in its standardized syntax than in its data model:

...describing data structures in a straightforward, interoperable way is really hard to get right and very often fails. At the end of the day, if you really want to interoperate, you have to describe the bits on the wire. That's what XML does [[Bray, 2000](#)].

Table 4-1 gives a generalized graphical representation of the WSDL-to-Java mappings for a service that may be reached through one or more `ports` (service endpoints, or network addresses, renamed as `endpoints` in WSDL 2.0), defined by a `portType` (an interface, appropriately renamed as `interface` in WSDL 2.0). The `portType` includes the `operations` (the class methods) which link the message exchange patterns to the `messages` that will represent the operations, along with the message `fault`, input and output parameters (the `parts`) for each method. Although the top-level `message` constructs were part of WSDL 1.1, these have been dropped in version 2.0 in favour of using the `types` section to define the messages. This has implications for details in other specifications that refer to them, but is no more of a versioning problem than for any other specification.

WSDL ELEMENT	JAVA CORRESPONDENCE
(Definitions [Description]) <i>targetNamespace attribute</i>	Package
Types	Data Types
PortType [Interface]	Interface
(Operation) name attribute	Method
Input Message: Part (only version 1.1)	Method Input Parameter
Output Message: Part (only version 1.1)	Method Output Return Value
Fault: Part (only version 1.1)	Exception
Binding	
Port [Endpoint]	Network Address
Service	Interface

Table 4-1: WSDL-to-Java Mappings, with WSDL 2.0 names in blue

Like SOAP, WSDL is not the simplest of specifications. As evidence that its complexity was confusing even to its own developers, some of the examples in the WSDL 1.1 specification are

inaccurate [Pasley, 2001]. `[W]sdl:import` statements are used incorrectly in place of `xsd:import` statements, as the Basic Profile points out [section 4.2.2, Ballinger *et al.*, 2004]. On top of that, Peeters noted that the WS-I-suggested corrections for these statements were incorrect in terms of the W3C Schema specification [2003]!

Despite its own complexity, WSDL greatly simplifies the automated creation of SOAP-based clients in terms both of time and of code. WSDL enables programmers to use their own data types (within certain restrictions – see section 4.3.2 below) and have them serialized into XML, from which they may be deserialized on the client side into the data types of other programming languages – although this approach (Code First, as opposed to WSDL First) is not considered interoperability-friendly. Through WSDL, the service interface needs to be defined only once, not for each programming language that might access the service [see Ruby, 2002]. This is a great advance on the old distributed models such as CORBA and DCOM, which required binary protocols linked to object-specific models, and required proprietary interface definition languages.

4.3.1 WSDL: a simple description of a web service

It may be helpful at this point to mention that a full WSDL file has been included in APPENDIX E: An Example WSDL File, which may be used as a reference to the explanations in this section. To avoid confusion, in the text that follows the changed WSDL 2.0 element names have been highlighted and placed in square brackets after the current WSDL 1.1 names. Exceptions to this occur in newer quotations containing WSDL 2.0 names. In these exceptional cases, while the WSDL 2.0 name is still highlighted, it is the WSDL 1.1 name which is given inside the square brackets. WSDL 2.0 might be seen as a simplification of the earlier version in that element names correspond more directly to their functions and there are overall fewer elements (the message elements are omitted and the binding elements have been made shorter).

WSDL documents can be divided into an abstract and a concrete section, with the concrete section coming last and making explicit reference to structures in the abstract section. The abstract section comprises the definitions of a web service and includes the `types`, the top-level

message elements, the top-level operation and the portType [interface] elements. The concrete section is bound to the abstract section through the binding element and comprises the port [endpoint] and service elements. In what follows, the focus is on the Java implementations of WSDL.

4.3.1.1 The Abstract Section

The latest working draft of the WSDL 2.0 Primer gives clear general definitions of the main elements that compose a WSDL document.

According to this Primer, "The types element describes the kinds of messages that the service will send and receive" and "[t]he interface [portType] element describes what abstract functionality the Web service provides" [Booth and Liu, 2005]. The grouping of elements that make up this section is often defined as abstract in that they are not tied to a protocol binding. According to JAX-WS, there should be a direct mapping between the overarching WSDL <definitions> [description] element, and especially the targetNamespace attribute, to the Java package that contains the web service:

A wsdl:definitions [description] element and its associated targetNamespace attribute is mapped to a Java package... Implementations MUST provide a means for the user to specify the Java package name corresponding to the value of a wsdl:definitions [description] element's targetNamespace attribute when mapping a WSDL definitions [description] element to a Java package" [Hadley and Chinnici, 2005].

According to the WSDL 2.0 Core Language Draft,

The value of the targetNamespace attribute information item SHOULD be a dereferencible IRI [Internationalized Resource Identifier]... It SHOULD resolve to a human or machine processable document that directly or indirectly defines the intended semantics of those components. It MAY resolve to a WSDL 2.0 document which provides service description information for that namespace [Chinnici et al., 2005].

types
[messages]
operations
portType

(Although this was not actually spelt out in WSDL 1.1, all the examples of target namespaces given there do resolve to WSDL documents.) The significance of the `targetNamespace` attribute is made even clearer in the primer for version 2.0, which describes it as comparable to an XML schema target namespace and states:

The value of the WSDL target namespace MUST be an absolute URI. Furthermore, it SHOULD be dereferenceable to a WSDL 2.0 document that describes the Web service that the WSDL target namespace is used to describe [[Booth and Liu, 2005](#)].

The `targetNamespace` is now considered so important that it has been made a required attribute in WSDL 2.0.

There is a direct mapping between a WSDL `portType` `[interface]` element and a Java interface and the two are tied more closely together in WSDL 2.0 by the name change from `portType` to `interface`. The request and (optional) response messages to the service are mapped in both WSDL versions as `input` and `output` sub-elements on a parent `operation` element inside the `portType` `[interface]` element.

There is also a direct mapping from the `name` attribute of an `operation` element, which must be unique, to a programmed service method and, within the WSDL, the `name` attribute cross-references a schema definition element within the `types` section. The `operation` elements each specify a message exchange pattern, as well as the type of message each may send or receive. While these are URI-identified in WSDL 2.0 as "in-out", "out-in", "in-only" and "out-only", WSDL 2.0 does not exclude the future possibility of other patterns but cautions against the interoperability problems the indiscriminate use of these might create. Although the `messages` in the `operation` `input` and `output` attributes are essentially the same as the elements defined in the `types` section, they are distinguished from these by a different namespace that identifies them as components inside message exchange patterns.

To maintain naming uniqueness, the Basic Profile specifies that there must be no method overloading in the interface methods, as was mentioned in section 4.2.4.1 above. JAX-WS prevents method overloading that might arise from a code-first approach by providing a `RequestWrapper` annotation to resolve naming conflicts. Similarly, while the .NET SDK allows

method overloading in the base class, it insists on the presence of a differentiating `MessageName` attribute which ultimately translates overloaded methods into differently named operations. WSDL 2.0 also supports the Basic Profile prohibition on method overloading.

4.3.1.2 The Concrete Section

In the more concrete section of a WSDL document, the `binding` element combines the abstract description of a `portType` [interface] with a specific transmission protocol. Where WSDL 1.1 defined extensibility *elements* (e.g. `wsdlssoap`), WSDL 2.0 defines namespace-specified protocol *attributes* for `binding`, `operation` and `fault` elements. Booth and Liu define the functions of the binding and service element respectively as describing "how to access the service" and "where to access the service" [2005]. WSDL 2.0 provides explicit bindings to SOAP 1.2 as a message format and to HTTP 1.1 as a protocol, with a further attribute (e.g. `http:method="GET"`) determining whether POST or GET is used with HTTP. It also provides for alternative bindings to SOAP 1.1. As with the message exchange patterns, WSDL 2.0 allows for the possibility of other formats, and bindings to other transmission protocols, but again cautions against their indiscriminate use if interoperability is required.



The `service` element contains the web-service name and, depending on the binding, may also contain the service endpoint, thus giving the service a specific network address and binding. The `port` [endpoint] element describes the binding with the `portType` [interface].

WSDL must be capable of being read by a remote machine (see the web-service definition on page 28 and the quotation from Christensen in the following paragraph), which may then construct from it a client for the service it describes, in whatever programming language the client determines. The generation of many service clients was carried out in this study. Client code generation appears particularly in section 7.2.4.4, which talks of general conversions between WSDL and code and vice versa. Much of the generated client code was in the form of supporting classes or libraries, which aided in the creation of a client, rather than directly generating an actual client.

Tools provided by Parasoft's *SOA Test*, BEA's *Test Client*, and *Visual Studio 2005* all have the ability to generate SOAP messages (as opposed to programmed clients) directly from a WSDL, a feature that is also present in specialist XML editors such as *Oxygen* and *XMLSpy*. Axis has a *WSDL2Java* tool (as well as a *Java2WSDL* tool), which generates Java classes from a WSDL file, similar to .NET's `wsdlc` tool. BEA also provides a means of generating a service from a WSDL file. ThinkTecture starts the generation process at an earlier stage by providing a generic tool for generating a WSDL file from a schema. The tool may be used either with .NET languages as an addition to *Visual Studio* (but not yet the 2005 version) or with Java as a plugin for Eclipse. These tools are discussed further, with supporting examples, in Chapter 7.

Cerami highlights the significance of WSDL when he describes it as a platform- and language-neutral contract between those providing and those requesting services, representing “a cornerstone of the web-service architecture, because it provides a common language for describing services and *a platform for automatically integrating those services* [my italics]" [2004]. WSDL was designed to be used by tools that might automate web-service creation, as is confirmed by the following statement from the WSDL 1.1 Specification: "WSDL service definitions... serve as a recipe for automating the details involved in applications communication" [[Christensen et al, 2001](#)].

4.3.2 WSDL Data-Typing

Issues of data-typing are usually first encountered in the creation of a WSDL document which requires definitions of data types so that a client may be able to provide them as input parameters to a service with an RPC programming style, or receive them from it as returned values. This is particularly true of the document/wrapped messaging style that is becoming the norm, as discussed in section 4.2.4.1. Those writing services in object-oriented languages cannot safely use features such as method overloading (as discussed on page 91), polymorphism and inheritance (not yet tested for in the Basic Profile testing kit and so potentially dangerous) which might not be understood by clients written in non-object-oriented languages [see [Manes, 2005](#)].

Kumar, Das and Padmanabhuni point out that, while the W3 Schema mandates a *minimum* number of digits (18) to represent the decimal datatype, it imposes no maximum value, which can result in a lack of precision, and also that "not mentioning the datatype on [the] wire leads to interoperability issues across applications" [2004]

In Java, collection types need to be converted to arrays (the new Axis 1.3 automates such conversions) and, curiously, the primitive `char` type is not supported [see [Chiesa, 2005](#)]. In C#, while a `DataSet` can be serialized as XML, `DataTable` and `DataRow` cannot [see [MacDonald, 2002](#)]. Manes makes the salient point that, with web services, as opposed to distributed object technologies, "...the client's object may be different from the server's object...When the client communicates with the server, it simply passes data, not behavior. It's much more loosely coupled" [2005].

Although in Java arrays should be used instead of other collection types, arrays are not without their own problems and are widely considered to be the most complex aspect of WSDL data-typing. Problems with arrays were outlined in section 3.4.2 and are further discussed in section 7.2.4.3. WSDL 1.1 recommends the use of the deprecated SOAP encoding for array types, depending on whether the WSDL style is `RPC (ArrayOfXXX)` or `document (maxOccurs=unbounded)`, but in its use of SOAP section 5 encoding the first approach is forbidden by the Basic Profile [section 4.3.3, [Ballinger et al., 2004](#)] and is not touched on at all in WSDL 2.0, leaving a vacuum in which developers use arrays at their own risk. (WSDL 1.1 did state that the suggested array style should be eschewed in favour of any schema revision that might offer a better solution [section 2.2, [Christensen et al., 2001](#)].) Pasley points out that the example given in the WSDL 1.1 specification for the WSDL `arrayType` element is invalid [[Pasley, 2001](#)] – the reason, he speculates, is an unnoticed change in the schema specification that occurred between WSDL version 1.0 and version 1.1.

Where Axis data typing is concerned, Gibbs *et al.* [2003] argue for a mix of Axis and Castor, an XML binding tool that can convert between schema and Java more consistently than the Axis `WSDL2Java` tool. The effective use of this tool requires a working knowledge of XML Schema and, while the creation of a simple schema is not difficult, the learning curve necessary to write

the kind of schema that will underlie complex data types is fairly steep. Because Castor is Java-specific, there remains the problem of language interoperability: Castor can guarantee a conversion between XML and Java but that is all.

There are various other open-source data-binding frameworks for Java and XML, including Sun's JAXB, and XMLBeans, created by BEA and donated in 2004 to the Apache Foundation. All of these frameworks will generate JavaBean interfaces from a well-formed W3C XML Schema. For C#, the .NET Framework SDK includes a W3C XML Schema Definition Tool (*xsd*), which generates C# code from a schema. C# does not have an exact equivalent to the JavaBean with its getter and setter methods³⁰, and C# objects generated from the schema are named after the schema types – including in their names a "type" suffix.

4.3.3 WSDL First

It has not been possible to ascertain the exact origin of the phrase "WSDL First" but its first appearance is considered to be as the title of an O'Reilly XML.com article [see [Provost, 2003](#)]. It is the driving force for the tool, WSCF or *Web Services Contract First*, developed by ThinkTecture and mentioned in section 4.3.1.2 above. Many, if not most, web-service developers now argue that the WSDL for a potential service should be created first and only after that should the programming code and configuration files be written or generated. The Axis-users mailing list, for example, is full of exhortations to novices to start the web-service cycle by designing the WSDL. Ewald puts the reason for this very clearly: "Anyone in the trenches actually building systems knows that deriving the details of your contract from your implementation is a sure fire way to cause interop issues" [[Ewald, 2005](#)].

This is echoed by Loughran and Smith who explain precisely why the price of not starting with Schema and WSDL is a diminution of interoperability:

..the act of writing an IDL description forces the author to define the system in terms of the portable datatypes and operations available in the restricted language of the IDL. This can effectively guarantee portability, and is a significant improvement over interfaces

³⁰ C# does have a "property" feature which includes getters and setters.

defined in the implementation languages themselves, which invariably contain constructs which are not portable [2005a].

Interfaces derived from code may need to be changed every time the code changes. Conclusions, drawn from experiments of starting from different vantage points, are detailed throughout Chapter 7, especially in section 7.2.4.4.

Ponnekanti and Fox discuss the interoperability problems of client applications adapting themselves to services offered by different, competing service-providers who employ different WSDL documents [Ponnekanti and Fox, 2004]. Ponnekanti and Fox approach the problem of interoperability from the viewpoint of a static application, which will not be able to adapt itself to change, unless the service providers agree amongst themselves to adopt a single WSDL from which each may derive required operations. This is a much more limited take on interoperability and relies on a programming-first standpoint. If, however, the client application may be at least in part *dynamically* generated from the WSDL of whichever provider is chosen, interoperability problems of the kind detailed in the paper need not arise.

Although WSDL is crucial in that it represents the service contract between a web service and its clients, and aims to incorporate an interoperable type system through the use of XML Schema, toolkits offer somewhat different methods of arriving at a web service, including most commonly generating the WSDL from previously written program code. Provost gives these reasons for opposing such a position:

a WSDL descriptor should be the source document for your web-service build process, for a number of reasons, including anticipating industry standardization, maintaining fidelity in transmitting service semantics, and achieving the best interoperability through strong typing and WXS [*W3C Schema*] [2003].

It might be argued that the WSDL-First approach derives from CORBA, for example, where code was generated from the IDL which was written first. Certainly, both systems need the neutrality of an initial independence from coding paradigms to achieve interoperability. WSDL-First goes a long way towards removing language-specific data-typing problems. Provost also argues:

Under the WS-I Basic Profile, and in all typical practice, web services rely on WXS as the fundamental type model. This is a potent choice. WXS offers a great range of primitive types, simple-type derivation techniques such as enumerations and regular expressions, lists, unions, extension and restriction of complex types, and many other advanced features [2003].

A dissenting voice is provided by a Microsoft web-services developer, Obasanjo, who claims that the WSDL-First approach is actually the source of some interoperability problems because of the incomplete implementations of XML Schema in many SOAP toolkits, including the one incorporated in the .NET framework, which does not support schema features such as substitution groups and namespace-based wildcards:

A core fact of building XML Web services that use WSDL/XSD as the contract is that most people will use object \leftrightarrow XML mapping technologies to either create or consume the web services. However there are fundamental impedance mismatches between the W3C XML Schema Definition (XSD) Language and objects in a traditional object oriented programming language that ensure that these mappings will be problematic [Obasanjo, 2005].

If WSDL First is the choice, Obasanjo suggests writing the WSDL based on a "minimal subset of XSD", although Skonnard prefers the alternative of collaboration between the parties involved as a means of solving interoperability problems [Skonnard, 2005], not exactly a feature of loose-coupling. Obasanjo promotes the alternative of code-first design because starting from an object-oriented viewpoint makes for the creation of less-complicated contracts – not necessarily the case with schema-object mappings for complex types, not to mention the overheads and complexity of converting data into objects to send it over the wire, only to have to convert it from objects again at the receiving end.

The writer found it initially more natural to start by creating the programming interface, and then move on to schema construction for the user-defined types, before beginning the construction of the WSDL. Only at that point, did she feel there would there be enough concrete detail to begin creating the WSDL file. WSDL First, however, offers an abstract conceptualization of a web

service that may then be fleshed out in programming details, in much the same way that a UML-First approach encourages a programmer to begin with an abstract notation for application classes and their interactions. Both of these approaches might be considered extremes and neither of them is necessarily always the best strategy. WSDL is a definition of a *contract* between a service and its clients. The writer began from the assumption that it would be difficult to have a contract between entities that did not pre-exist in code.

In the course of examining these issues, however, her conviction solidified, for all the reasons given above, that the only approach to creating web services that will confer interoperability is WSDL-First. The objective stated in the first chapter of the Java Web Services Tutorial that schema should be bound to "Java representations, making it easy for Java developers to incorporate XML data and processing functions in Java applications" [Sun, 2005a] must be, on reflection, considered to be back-to-front.

Loughran and Smith object in part to the WSDL-First policy, but for a reason that differs importantly from that put forward by Obasanjo. While still supporting the notion of WSDL First, Loughran and Smith claim that the length and complexity of WSDL and schema make web-service development very difficult, having recently (2005) themselves completed a project in which:

the XSD file ... [was] approximately 2000 lines, including all the comments and annotations needed to make it comprehensible. That it takes so many lines to describe a relatively simple service is clearly one reason why this approach is so unpopular [2005a].

The WSDL-First approach raises a further issue. It is expensive in terms of time for a web-service developer to become familiar with a web-service toolkit (even one as apparently transparent as Apache Axis) – time that might instead be used to become familiar with the essential constructs of WSDL. It is not entirely an either/or situation. Ideally the web-service developer should be familiar with both the standards and whatever toolkit(s) he or she may have chosen: WSDL is intended to be machine- and toolkit-readable. To have any control over the generated WSDL constructs is impossible without at least a basic understanding of WSDL and

XML. The creation of web services with the use of metadata annotations, described in Chapter 7, offers a possible compromise.

A further issue is the need for accurate interpretation and validation of WSDL by the available tools. Those used for WSDL validation included Altova *XMLSpy*, Parasoft *SOA Test*, Mindreef *SOAPscope*, *Oxygen*, *CapeClear WSDL Editor* and *Stylus Studio*. One of the WSDL examples contained in the WSDL specification was copied into each of these editors in turn, with the discovery that, while most validated it correctly, one considered the document invalid. The same editor removed the default values for the `SOAPAction` header, required for interaction with .NET services and by the current JAX-RPC specification as well as by the Basic Profile. Namespace scoping in WSDL, while entirely logical when understood, is not intuitive and a surprising number of supposedly accurate WSDL representations, used in online tutorials, were marked as invalid by many of the editors for namespace reasons. At least one program declared that a WSDL was valid before presenting a "run-time" problem when attempting to generate a SOAP message from it. WSDL 1.1 and the Basic Profile do not agree about whether an RPC/literal message part should reference a `type` or an `element`.

4.3.4 Conclusions Regarding WSDL

WSDL has been criticized for being too complex but is perhaps the best available solution for the difficult task of defining a web service in a way that is independent of platform and language. Although very closely linked to SOAP, WSDL can exist without it. WSDL 2.0 provides an HTTP binding which does not use SOAP [see section 2.5.6, [Booth and Liu, 2005](#)] and section 5.3.3 of the specification gives an illustration of a REST style service. REST is the subject of Chapter 5 and is illustrated in section 7.2.8 of Chapter 7.

In the last section of this chapter, focus turns to UDDI, the third leg of the web-services triumvirate.

4.4 UDDI – Dead in its Tracks?

UDDI is an acronym for *Universal Description, Discovery, and Integration*. As an implementation, a UDDI registry service is itself a web service which is intended for the use of other web services. As the specification for a discovery mechanism, its development was spearheaded early in 2000 by Ariba, Microsoft and later IBM, who saw in it a means of providing a business repository or registry for their web services. Microsoft may also have seen it as an extension to its Active Directory technology, which is its current use inside the organization. The MSDN defines a Microsoft-specific implementation under the heading of *Active Directory*:

Microsoft.Uddi provides a Microsoft-specific extension that applies only to Microsoft Active Directory environments. This extension enables applications to discover Microsoft Enterprise UDDI Services servers and their entry points in Active Directory servers on an intranet [[Microsoft, 2005c](#)].

UDDI versions 2 and 3 make use of both SOAP and WSDL. UDDI defines SOAP messages that may be used by web-service providers to advertise their services, and by consumers to make queries about them, and it uses WSDL to define its own interfaces. Unlike SOAP and WSDL, UDDI was not brought to the W3C for standardization but was instead submitted to OASIS, the first significant web-service specification to go that route. Although UDDI 3 was made available in 2003, it was approved as an OASIS Standard only in January, 2005. UDDI 2 has the blessing of the Web Services Interoperability Organization [section 5, [Ballinger et al., 2004](#)], but has not been mandated by the Basic Profile.

It is significant that the W3C Web Services Glossary [[Haas and Brown, 2004](#)] does not once include a reference to UDDI, though references to SOAP and WSDL abound. Lomow and Newcomer comment: "It's safe to say that the original vision of UDDI has not been realized" [[Lomow and Newcomer, 2004](#)]. They give one reason as the unwillingness of companies to enter into transactions with unknown business entities that have not been approved as trading partners. Lamow and Newcomer also argue that UDDI was intended to function within the intranet, where indeed it does operate but in so proprietary a manner that companies are unlikely to want to use other companies' systems for their own.

The reason for the unwillingness of companies to enter into transactions with business entities unknown to them is a lack of trust in the anonymity of machine-only communication.

Wainwright supports Bray in his criticism of the origins of UDDI, which is that standards organizations exist to standardize what is already known to work, rather than to be in the business of inventing technology [Wainwright, 2004]. A further criticism of UDDI is that it works against the loosely-coupled features of web services by being too centralized.

A recent Gartner report describes the current landscape: "most UDDI implementations are locked into a proprietary database management system (DBMS) — for example, IBM uses DB2, Microsoft uses SQL Server or Microsoft Data Engine, and Oracle uses DBMS" [Plummer *et al.*, 2004]. Wainwright also believes that UDDI has been sidelined into a proprietary registry with a home on the intranet rather than on the internet, and that it "has stopped attempting to be a blueprint for a universal services registry. It is now targeted for internal use by enterprises that have large numbers of services and want to track them" [2004].

In an attempt to resolve the issue of trust, the more lightweight WSIL (*Web Services Inspection Language*) was developed (again by Microsoft and IBM) to provide a technology to complement UDDI in the form of an intermediary mechanism through which companies might advertise their own services. This specification, however, seems to have met with no more success than UDDI. No attempt has been made to implement UDDI for the purposes of this study.

4.5 Summary

It is not surprising, in the light of the last section, that the web-service standards that were put together, however awkwardly, as an attempt to solve a real problem – interoperability – have developed to a point where they cannot be ignored even if they are criticized. NetKernel, a REST web server³¹, even has a SOAP interface and methods to verify WSDL documents, but makes no mention of UDDI. It may not be a coincidence that UDDI 1 was pushed through

³¹ See <http://www.1060.org/>.

OASIS, while SOAP and WSDL took much longer to achieve W3C Recommendation status and with a wider consensus.

As a concept, however, UDDI is no less significant to web-service interoperability than SOAP or WSDL. By providing a means of discovery that theoretically anyone could use, it may be seen as a means of enabling companies to interoperate at a higher level and may become significant to SOA. It does not, however, take into proper consideration the issue of trust.

Perhaps it is asking too much of the specifications that they be perfect. Skonnard lays the blame for non-interoperability at their door:

If interoperability is the main promise of Web services, why is it that so many developers and organizations have a difficult time achieving it in practice? With all due respect to our hard-working standards bodies, the primary culprits are the imperfect specifications guiding today's implementations. Ambiguities and too many choices often lead to differing interpretations, resulting in incompatible implementations [[Skonnard, 2004](#)].

With the specifications delivering imperfect standards, Skonnard believes it is the responsibility of the developer not only to identify the problem areas *and take steps to avoid them* but also to follow the guidelines of the Basic Profile, which attempt to clarify ambiguities in the specifications. His advice does not take into account the problems of specification versioning and he specifically recommends, for example, following the Basic Profile advice always to use HTTP `POST` rather than `GET` [section 3.4.2, [Ballinger et al., 2004](#)] despite the fact that SOAP 1.2 allows it. Skonnard further states, "Using cookies to implement stateful Web services interactions is another area that wasn't explicitly defined in the original SOAP messaging specifications", ignoring the requirement that, if they are to be *loosely coupled*, web services need to be stateless.

Despite imperfections and ambiguities, SOAP and WSDL do provide a means of achieving interoperability, especially for complex services which cannot be reliably delivered in any other way and, as the most mature web-service specifications, they provide the groundwork for much that has followed them. There is no doubt that other versions and other specifications will

continue to be developed. As long as the versions are indicated in the service details (usually through the namespaces), interoperability problems should not arise from version co-existence. It is very encouraging to see the gradual improvement in interoperability of the two main specifications discussed in this chapter.

CHAPTER 5: REST – AN ALTERNATIVE

5.1 Introduction: the Revolt against Complexity

This study has shown that, although there is some consensus that the conventional web-services "stack" based on SOAP and WSDL can be made to succeed, there is a matching degree of confusion and uncertainty about its future. Bosworth, former Chief Architect at BEA, the original architect of XML and MS Access at Microsoft, a major contributor to the HTML basis of Internet Explorer, now employed by Google, said recently: " I'm trying, right now to figure out if there is any real justification for the WS-* standards and even SOAP in the face of the complexity when XML over HTTP works so well... So, I'm kind of a skeptic of the value apart from the toolkits. They do deliver some value, (get a WSDL, instant code to talk to service), but what I'm really thinking about is whether there can't be a much simpler [kinder] way to do this" [[Bosworth, 2004](#)].

Many of the developers who were in at the birth of XML deplore the complexity of the current realization of web services. They designed XML so that message exchange over the internet would be both simple and capable of encapsulating complexity where needed, and they find the specification proliferation irksome at the very least. Bosworth is well positioned to be the spokesman for simplicity. He cites it as the major benefit of XML over HTTP: "You don't have to worry about any of the complexity of WSDL or WS-TX [*Web Services Transactions Project*] or WS-CO [*Web Services Coordination*]. Since most users of SOAP today don't actually use SOAP standards for reliability (too fragmented) or asynchrony (even more so) or even security (too complex), what are they getting from all this complex overhead[?]. ...How do you keep it really simple, really lightweight, and really fast[?]. Sure, you can still support the more complex things, but the really useful things may turn out to be simplest ones" [[Bosworth, 2004a](#)].

The history of software systems to date teaches that rigid, over-elaborate systems do not survive, not only because they do not have the flexibility to adapt to change but also because people do not like to be constrained by them. Web services in their current state seem to be at the mercy of

those who want to control and regulate for every possible eventuality, to such an extent that the whole becomes unmanageable, even incomprehensible – worlds away from the creative, innovative spirit that produced the Internet, HTTP and HTML, and later simplified SGML into XML. A theme that echoes frequently in the writings of those who decry conventional web services is the need to wrest control of web services away from IT departments and give it back to those concerned with business logic. This balance of control is a significant issue. Whereas IT departments might focus on control and regulation of the processes, businesses (and people in general) are more concerned with content, with why the software is *used* in the first place.

This also touches on interoperability. There is no point in building a system so technically "perfect" that no one can interoperate with it, because no one knows how.

5.2 Representational State Transfer

One of the alternatives to the conventional web-services stack is REST, an acronym for *Representational State Transfer*, a term coined by Roy Fielding in his PhD dissertation to describe the architectural style of the World Wide Web [[Fielding, 2000](#)]. Fielding, one of the principal writers of the HTTP specification and a co-founder of the Apache Group, defines the term as follows:

Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use [[2000](#)].

Fielding and Taylor explain the advantages of such a system:

"REST is a coordinated set of architectural constraints that attempts to minimize latency and network communication, while at the same time maximizing the independence and scalability of component implementations. This is achieved by placing constraints on

connector semantics, where other styles have focused on component semantics" [Fielding and Taylor, 2002].

This approach uses mechanisms for message transmission that pre-date web services: XML, HTTP and URIs (*Uniform Resource Indicators*), as may be seen in Figure 5-1. One of the early problems with SOAP in the minds of many XML developers, particularly at the W3C, lay in the fact that version 1.1 did not permit the use of HTTP GET, choosing POST instead, regardless of the fact that GET is both less dangerous in its side-effects, and cacheable. This has been remedied in SOAP 1.2, as has the exposure of the URIs in the HTTP headers (they had been concealed in SOAP 1.1). SOAP 1.2 recommends that, where practical, particularly when using the HTTP binding, separate resources are identified by separate URIs, so that SOAP endpoints fit into the web architecture in the same way as other web accessible resources. This has the added advantage that SOAP resources are now suitable for use with the HTTP GET method instead of being tied to HTTP POST (see Section 4.1 of Mitra, 2003).

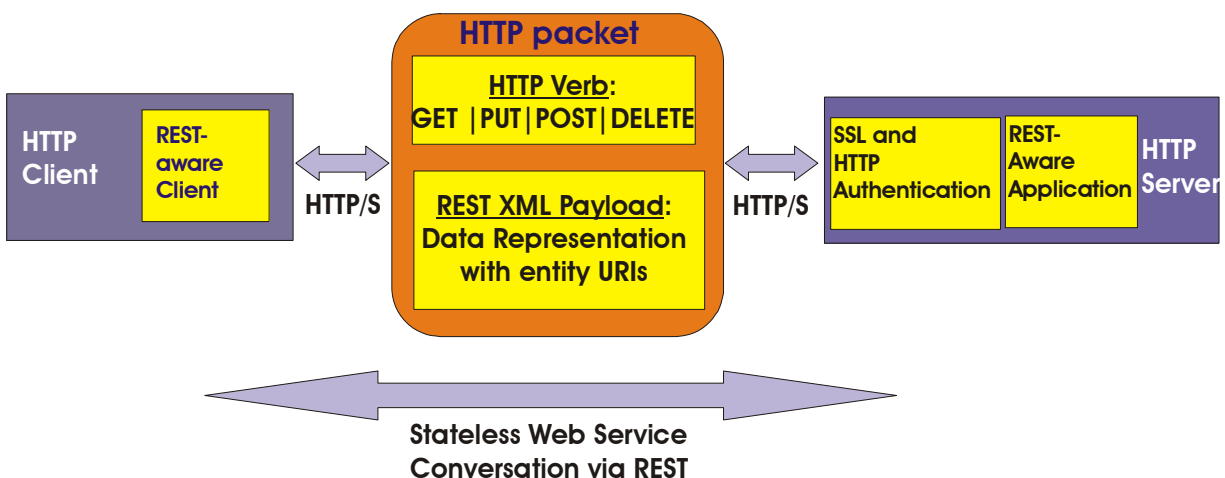


Figure 5-1: REST Web-Service Structure [Hinchcliffe, 2005]

SOAP 1.2 narrowed the gap, at least between SOAP and REST. Zur Muehlen, Nickerson and Swenson take the changes in SOAP 1.2 to mean that "SOAP can be used, but used in such a way that it does not violate REST principles" [2004]. The escalation both in complexity and in the number of specification releases since SOAP 1.2 has, however, prompted individuals such as Bosworth (cited on page 104) to question the current nature of web services. Companies such as

Amazon have actively developed REST-based interfaces for their clients as alternatives to those based on SOAP.

Barr, of Amazon.com, explained in an interview that: "we put up both the SOAP and the REST or XML-over-HTTP interface and the experience so far has been that the REST interface is definitely in the lead. Probably close to 80% of the calls we process are REST-style calls" [Barr, 2003]. Its popularity he attributed to its simplicity: "...people go towards the simplest solution." He did share his belief, however, that SOAP interfaces might become more popular as web-service transactions became more complex.

REST with its basis in the URI is particularly well suited for operations requiring browser access. REST is excellent for simpler operations but where features such as transaction processing and authentication are required, SOAP may still have an edge. Fortunately, there does not have to be a situation in which only SOAP or only REST is chosen. Each may be seen to have complementary uses. Zur Muehlen, Nickerson and Swenson characterize the difference between them in terms not of technology but of style: "REST.. represent[s] a navigational style of design, and .. SOAP.. represent[s] a procedural style [2004].

Another enthusiast for REST is Butterfield, the CEO of Flickr³², a new, different and very popular tag-based, photo-sharing program exposed as a web service. When asked in an interview published on the O'Reilly Network whether Flickr was a next-generation web service, Butterfield replied:

On the strictly practical side, I think we had one person inquire about using the SOAP version of the API. I don't know if any apps were actually built. There is at least one application built on XML-RPC. But all the others – I don't even know how many there are – are built on the REST API. It's just so easy to develop that way; I think it's foolish to do anything else [Butterfield, 2005].

As with Amazon, the user interface is web-based and therefore particularly suited to a REST style.

³² See <http://www.flickr.com/>.

In March of 2005, Yahoo decided to expose its services through a REST interface, explaining that "REST based services are easy to understand and accessible from most modern programming languages... We believe REST has a lower barrier to entry, is easier to use than SOAP, and is entirely sufficient for these services" [[Yahoo, 2005](#)]. In July of 2005, Safari Books Online, a venture shared by O'Reilly Media Inc. and The Pearson Technology Group, joined Yahoo, Amazon and Flickr by publishing their web service also through a REST interface, which now enables users to browse not only bibliographic details but also a book's contents, giving limited access to the text in much the same way that Google Print now offers.

What companies as successful as Amazon and Flickr have to say about the popularity of their REST interfaces is particularly interesting in the light of the conflict that arose in the W3C TAG [*Technical Architecture Group*] when supporters of SOAP-based web services like Manes ridiculed REST as a purely academic pursuit:

W3C is, at heart, an academic organization. And its perfectly reasonable for W3C to pursue its academic goals (REST and the Semantic Web). But if W3C wants to play a major role in business systems, and if W3C wants to continue receiving funding from the big software vendors, then the W3C TAG must be willing to [accommodate] the requirements of big business. If the REST faction continues to try to undermine the existing Web services architecture, it will alienate big business [[Manes, 2002](#)].

Fielding's response to this posting is as pointed as it is obviously angry, not only in its rebuttal of the idea of REST as a purely academic pursuit but in its placing of SOAP in the context of failed distributed-object architectures:

The only reason SOAP remains in the W3C for standardization is because all of the other forums either rejected the concept out of hand or refused to rubber-stamp a poor implementation of a bad idea. If this thing is going to be called Web Services, then I insist that it actually have something to do with the Web. If not, I'd rather have the WS-I group responsible for abusing the marketplace with yet another CORBA/DCOM than have the W3C waste its effort pandering to the whims of marketing consultants [[Fielding, 2002](#)].

5.3 Advantages of REST over Conventional Web Services

It need hardly be said that the main advantage of REST-based services is that they are completely interoperable. All that is required for the client is that it be able to send information to the web server hosting the service and receive information from it in the simplest language of all – text-based XML. There is no problem with data binding and serialization in terms of the message transmission because there are no objects to be transmitted. There are no language or platform issues, no complex specifications to incorporate, no Basic Profile to satisfy. Significantly, of course, no toolkits are required to translate innumerable complex specifications into terms a lay person can understand. An example of a REST service, illustrating its simplicity and interoperability, may be found in section 7.2.8 of Chapter 7.

Most of these are negative advantages. What are the positive ones?

A major advantage lies in the expressive power of XML itself, which goes beyond current programming languages in enabling, for example, the conveyance of precise meaning through inheritance by *restriction*, as well as by extension. A schema data type for an ISBN, for example, is a restriction on the normal string schema data type. All of the normal expressiveness of strings is stripped away and confined to a regular expression representing the numerals, alphabetic characters and hyphens which compose an ISBN.

A second major advantage of working directly with XML is that the processes are seen to be data-centric, rather than object-centric, concerned with data rather than with processes. Thirdly, it is no insignificant advantage for REST-based web-service styles that they are also seen to conform to the principles described in the W3C's recommendation, Architecture of the World Wide Web [[Jacobs and Walsh, 2004](#)]. Fourthly, there is some evidence that, depending on client implementations, REST has some advantage of speed over SOAP-based services in that it does not carry the data type conversion and greater textual overheads that SOAP usually necessitates [[Barr, 2005](#)].

5.4 Disadvantages of REST over Conventional Web Services

The simplicity of REST-based styles means that they may not directly offer features such as authentication and transaction management which more complex web services may require. Smaller REST-based web services may be chained as part of a larger "application" through the inclusion of their URIs within XML messages which enable the client to move dynamically between services as, for example, in booking an airline ticket as part of a holiday package. But REST-based services do not offer the same potential for enabling legacy applications to be incorporated into more modern systems that SOAP-based services do, precisely because REST eschews objects in favour of data, while SOAP may be at home with either or both (legacy systems tend to be distributed-object systems). It can also be seen as a limitation that REST seems to apply exclusively to the web "world" of HTTP or HTTP-similar protocols, and not to other means of transmission.

Zur Muehlen, Nickerson and Swenson give a useful summary of some advantages and disadvantages of REST versus SOAP as they apply to process integration, but the comparisons can be seen to have wider application as well [2004]:

	REST	SOAP
Characteristics	<ul style="list-style-type: none"> • Operations are defined in the messages • Unique address for every process instance • Each object supports the defined (standard [HTTP]) operations • Loose coupling of components 	<ul style="list-style-type: none"> • Operations are defined as WSDL ports • Unique address for every operation • Multiple process instances share the same operation • Tight coupling of components
Self-declared advantages	<ul style="list-style-type: none"> • Late binding is possible • Process instances are created explicitly • Client needs no routing information beyond the initial process factory URI • Client can have one generic listener interface for notifications 	<ul style="list-style-type: none"> • Debugging is possible • Complex operations can be hidden behind façade • Wrapping existing APIs is straightforward • Increased privacy
Possible disadvantages	<ul style="list-style-type: none"> • Large number of objects • Managing the URI namespace can be cumbersome 	<ul style="list-style-type: none"> • Client needs to know operations and their semantics beforehand • Client need dedicated ports for different types of notification.

Table 5-1: Differences between REST and SOAP in terms of process integration

Zur Muehlen, Nickerson and Swenson continue with the fascinating suggestion that the lack of absolute testing of the advantages of one technology (or design) over the other suggests that the conflict between the supporters of each is not a purely technical issue. They conclude that the differences are cultural but leave to a further study an examination of this phenomenon [2004].

5.5 Summary

Inasmuch as REST defines the structures by which the web might be said to have become the most popular technology ever, REST can lay claim to interoperability through its set of simple principles. Fielding's claim is that REST "scales well with large numbers of clients, enables transfer of data in streams of unlimited size and type, supports intermediaries (proxies and gateways) as data transformation and caching components, and concentrates the application state within the user agent components" [[Fielding, 1998](#)], all of which support the notion of interoperability.

Web services call for multiple solutions. There will probably not emerge only one victorious scheme of web services to rule all the rest, despite the ambitions of tools vendors to make this so. In a situation where flexibility obtains, it is possible for REST-based services to coexist with SOAP-based services, as indeed they do today on sites such as Amazon and Flickr. One of the architects of SOAP has even argued that the best combination for web services may be the use of REST to describe how objects might be *accessed* along with the use of SOAP to describe how both state and objects might be *represented* [[Ruby, 2003](#)], although such an approach moves back into the world of distributed objects, as opposed to that of distributed data, where XML excels.

Perhaps the strongest argument in favour of REST is the visible movement away from RPC-style web services to more document-oriented, message-passing styles within a conventional web-services approach, as evidenced in all the latest versions of the main toolkits, whether C#- or Java-based. This represents some convergence of opinion. It is unfortunate that the convergence is still accompanied by a specification bloat (see the diagram in APPENDIX C: The Complexity

Of Web-Service Specifications [Jeon, c2005]) which works against simplicity and, ultimately, against interoperability, especially where competing specifications are concerned.

CHAPTER 6: WEB-SERVICES COORDINATION

6.1 Introduction

Although concepts and issues belonging to Service-Oriented Architectures work at a meta-level above web services, and are in the main beyond the scope of this project, they raise some of the same issues, which therefore deserve consideration. This chapter will briefly examine some SOA developments where they encounter similar problems to web services, or have a bearing on web-services interoperability. The topics discussed in this chapter are a selection of the main approaches towards coordinating web services so that they work together into the bigger whole that SOA comprises. The selection does not claim to be exhaustive. The technologies described here are intended to be distinct from traditional middleware solutions because, according to the Web Services Choreography Interface, (which will be described in section 6.2.1) traditional middleware assumes a less dynamic participation and "a different, more tightly linked and controllable environment, which is not the nature of the Web" [[Arkin et al., 2002](#)]. Traditional middleware systems "normally call for centralized engines, while the nature of the Web is decentralized" [[Arkin et al., 2002](#)]. As the remainder of this chapter will show, not all the solutions proposed maintain this clear distinction.

6.2 Orchestration and Coordination

Web-services orchestration and coordination work at the level of SOA, creating a context for services and their cooperation with each other. They aim to provide a "conversational model" [[Pelz 2003](#)] between different web services, which will therefore be loosely coupled. Pelz considers that the current models are not sufficiently distanced from the business process or individual company and that for them to be really useful, a more objective peer-to-peer approach should be achieved. He cites an analogy offered in an IBM research paper

to illustrate the difference between the business process standards and the conversational model for web services. The current web-services model is analogous to a vending

machine. There are a set number of buttons that can be pressed in a pre-defined order. A conversational model is more analogous to a telephone call, involving a series of exchanges between the parties at each end in a more flexible, dynamic fashion [[Pelz 2003](#)].

The intention of such an approach, he argues, is to give control back to business management and wrest it away from IT management.

Pelz makes a distinction between the terms "orchestration" and "coordination". *Orchestration*, he defines as describing "an executable business process that may interact with both internal and external web services. For orchestration, the process is always controlled from the perspective of one of the business parties". A commonly used analogy for orchestration is of a conductor in charge of an orchestra, which suggests the more centralized control of traditional middleware. Orchestration might also be said to be closer to the imperative nature of object-oriented middleware. *Choreography*, however, is more descriptive, following the paradigm of messaging technologies, and "is more collaborative in nature, in which each party involved in the process describes the part they play in the interaction" [[Pelz 2003](#)]. An analogy for choreography might be from dance, where everyone follows the rules which are individual for each dancer and there is no "conductor" apart from the design. Here choreography is similar to a protocol.

Pelz describes ideals for the execution of web-service processes, which aim to prevent lengthy sequential processing. Asynchrony comes to the fore not only at the service level but also at this meta level. Familiar paradigms are seen to be no longer workable. Long-term transaction processing, for example, cannot rely upon the traditional principles of ACID (database systems being *a*tomic, *c*onsistent, *i*solated and *d*urable), when, for instance, locking a resource across the internet might cause more problems than it is intended to prevent.

Improved exception handling is no less an issue at this level than it has been seen to be for individual web services and Pelz cites a Hurwitz study which claims that it consumes 80% of the time used to create business processes.

6.2.1 Web Services Choreography Interface (WSCI)

Arkin *et al.* position the web-services choreography efforts in relation to the notion of interoperability as follows:

A "stack" of layered standards is emerging that aims to ensure semantic and technical interoperability of Web Services. This stack, developed by the W3C, is still in its early stages and is currently being built from the ground up; several additional layers are needed in order to enable true Web Service collaborations [2002].

In mentioning the significance of the W3C in developing this "stack", Arkin *et al.* do not omit mention of other efforts which are "building semantics and interoperability for business processes and collaborations in a top-down approach" [Arkin *et al.*, 2002], an allusion to orchestration technologies. Whether these efforts include non-W3C initiatives is not clear, but the focus on interoperability is central, if slightly different from the notion of interoperability for individual web services. Collaborative interoperability relates to the meaningful interactions of services with other services, not just that between client and service.

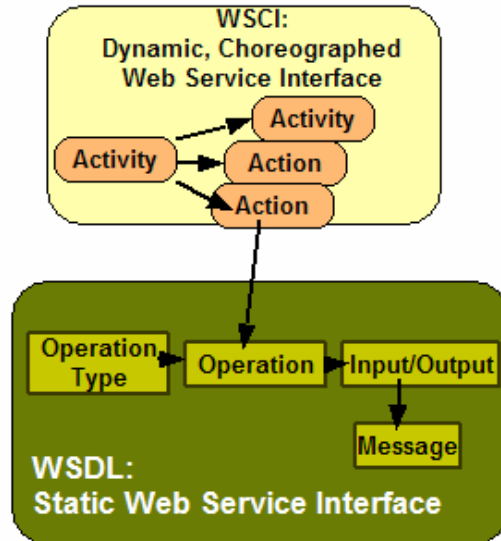


Figure 6-1: The relationship between WSDL and WSCI [adapted from Arkin *et al.*, 2002]

Although WSCI considers the message exchange from the viewpoint of only one of the collaborating partners at a time, it does, however, also offer a global model, which describes "a multi-participant view of the overall message exchange" [Arkin *et al.*, 2002]. WSCI takes over

where WSDL leaves off, extending the interactions beyond client and service to inter-service collaboration. Created initially by Sun, SAP, and BEA, and then submitted to the W3C in 2002, the WSCI choreographs the interactions between collaborating web services (and even the interactions between different parts of the same web service) by defining the series of XML messages that will be exchanged between them, their order, sequencing, relationships and behaviour. Each series of messages WSCI terms a *process*. As may be seen in Figure 6-1, it re-uses the `operation` elements defined in the WSDL for a service.

6.2.2 BPEL

Originally termed BPEL4WS (*Business Process Execution Language for Web Services*), BPEL was created in 2002 by IBM, Microsoft and BEA and taken to OASIS for standardization at approximately the same time that the WSCI was going through the W3C. The vendors' choice of OASIS was probably a reaction to the demand by the W3C that all patents should be royalty-free. BPEL extends programming languages by providing ways in which they may work together but its primary concern is with business processes. It has inherited some features from its forerunner, BPML (*Business Process Management Language*). Verner [2004] describes both systems as being built on the π calculus³³. Version 2.0, renamed as WS-BPEL but not significantly different from its earlier version, was released in 2004 by OASIS.

The BPEL specification argues compellingly for the necessity of a meta-level integration standard above web services which will enable the exciting and dynamic service interactions so far available only in science-fiction:

Systems integration requires more than the ability to conduct simple interactions by using standard protocols. The full potential of Web Services as an integration platform will be achieved only when applications and business processes are able to integrate their complex interactions by using a standard process integration model [[Andrews et al., 2003](#)].

³³ A process algebra, developed by Hoare and Milner. See [Pucella, 2001](#).

One of the differences between this kind of model and web services is that the former needs to be stateful in order to keep track of message sequences and process interactions. BPEL is proposed partly as a solution to the problem of maintaining state – in fact state maintenance is mentioned throughout as crucial. BPEL aims to specify the "visible message exchange behavior of each of the parties involved in the protocol, without revealing their internal implementation" [Andrews *et al.*, 2003]. BPEL offers itself both as a "model" for business interactions and as a "grammar" to define them. It seeks to create relationships, but in doing so it may be returning to an earlier model and obscuring the loosely-coupled nature of web-service interactions in which it is the *messages*, not the participants, which are defined.

While WSCI uses WSDL, the role of BPEL is to attempt to extend WSDL. Like WSCI, BPEL builds upon the WSDL definition of operations, but BPEL attempts to organize their sequence, without making any assumptions about the messaging technology (e.g. SOAP) that will be used. Its focus is on the abstract sections of WSDL, not the concrete ones in which the bindings are made, although it also, separately, describes executable business processes.

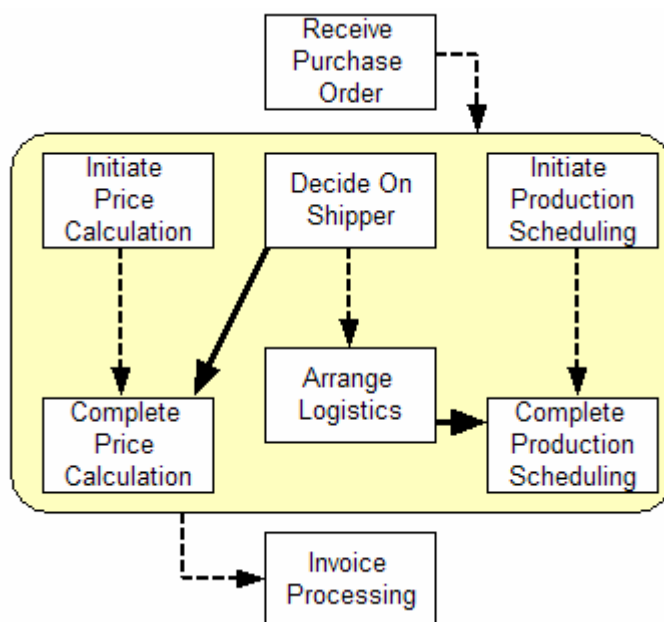


Figure 6-2: Schematic of a BPEL Process for Handling a Purchase Order [Andrews *et al.*, 2003]

The BPEL specification gives an example of a BPEL process in terms of the handling of a purchase order, as illustrated in Figure 6-2, in which the dotted lines "represent sequencing", the grouping (shown here in the yellow rectangle) represents a concurrent sequence, and the two

solid arrows represent data dependencies which arise from the necessity of determining both the shipping price, before a final price calculation may be achieved, and shipping data, before the production schedule can be finalized. In the WSDL document used by this service, these operations are embraced in the abstract by a number of port types or interfaces. BPEL then extends WSDL by introducing a number of extensible elements, the most significant of which for this discussion are `PartnerLinks` (defining the role and the functionality that must be provided by whoever assumes the role) and `variables` elements (for maintaining state).

Defining roles and maintaining state is all too reminiscent of object-oriented middleware. As if that were not enough, the specification continues: "Finally, it is important to observe how an assignment activity is used to transfer information between data variables. The simple assignments shown in this example transfer a message part from a source variable to a message part in a target variable" [[Andrews et al., 2003](#)] and goes on to describe "switch" and "while" elements. The language appears to concern messages but the concepts are closer to programmatic code sequences.

BPEL's focus on the functionality of the role is a key difference between it and WSCI. Where BPEL sees roles, WSCI sees messages. Where BPEL sees hierarchy, employing a central process "engine" controlled by one of the players, WSCI sees equals. BPEL is concerned with constructing a "fractal-like" [[Verner, 2004](#)] meta-service out of smaller services, whereas WSCI is more about services collaborating with each other. BPEL and WSCI are not necessarily competing technologies in that they each deal with different aspects of web-service coordination, but of the two, WSCI seems closer in spirit to the aims of web services.

The weaknesses of BPEL according to Verner, are that it "addresses only processes composed exclusively of Web services" (one of the strengths of web services is that it provides the possibility of incorporating legacy systems), that its processes cannot be rendered graphically, that its design cannot be performed in a top-down fashion, that a BPEL process cannot be analyzed, and, most tellingly, that it is "a vendor-driven process definitions language that has not yet been reflected in a royalty-free standard published by a recognized standards group" [[2004](#)].

6.3 The Enterprise Service Bus

Originally defined by analysts at Gartner (see [PolarLake, 2003]³⁴), the *Enterprise Service Bus* (ESB) is a broker infrastructure which offers an alternative integration of legacy business systems by converting them all to services. The proprietary nature of its existing implementations suggest that it may encounter the same problems as CORBA and a recent Gartner survey pointed to the fact that industry had greeted its arrival with a certain amount of suspicion, a state of mind not alleviated by the common release of some so-called ESB products which are really legacy applications under the wraps (Cape Clear have even created a tool for the detection of such forgeries [see Cape Clear, 2005]).

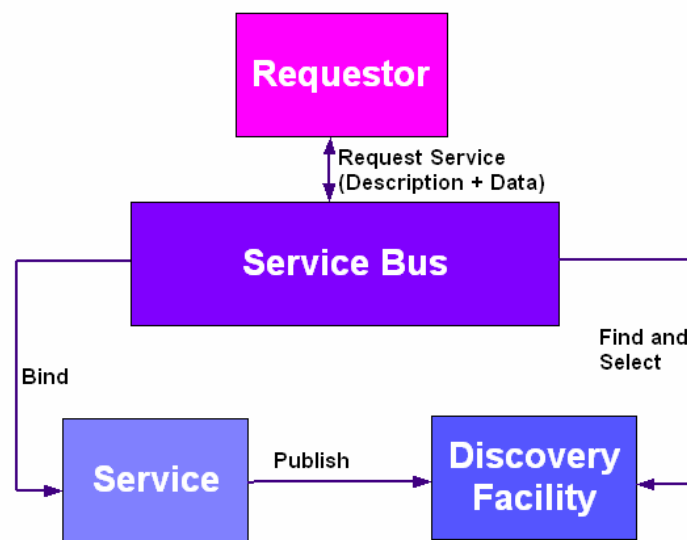


Figure 6-3: The Role of the Service Bus in SOA [adapted from Weerawarana et al, 2005]

An ESB implementation is composed of a network of servers. As a system ESB is message-based and it claims to replace message-queuing middleware. As an architecture it is a model that is closely associated with web services and service-oriented architectures, and its role in the latter is illustrated in Figure 6-3. There is still a certain amount of confusion around its definition, about whether it is a product or an architecture and it seems that the term may apply

³⁴ See e.g. <http://www.webservices.org/index.php/ws/content/view/full/39605> [PolarLake, 2003].

to both equally. Its concerns are with infrastructures and control and its aim is to integrate distributed business processes in a way that makes them loosely coupled. Although it is not confined to web services, it shares design features of coarse granularity, asynchronicity and loose coupling in a distributed environment. Its purpose is to achieve at a meta-level what web services seek to accomplish at a lower level.

6.4 A First Look at WCF

The *Windows Communication Foundation* (now known as WCF, until July 2005 code-named Indigo), unlike the other SOA components featured in this chapter is a product or a series of technologies for service integration, rather than a specification or a protocol. It is included in this chapter because of what it reveals about current trends in distributed computing and SOA. It demonstrates a serious commitment to the principles and values embodied in web services but it is a vendor-specific product, integrated into the new Microsoft *Visual Studio 2005*. Despite its assurances of interoperability with web services generated on other platforms, it is built for Windows and its use locks the developer into .NET technologies.

A challenge to this issue of vendor lock-in might come in the form of the JBI (*Java Business Integration*) Java Specification Request (JSR) 208, which seeks to build a vendor-neutral framework for service integration. Both IBM and BEA decided to withdraw their support for it during 2004 and it is not clear at this stage what other support it will find beyond the Java community³⁵. It is significant that, of the ten technologies listed as being related to JBI, seven are W3C specifications [[JCP, 2005](#)]. Integrated into WCF, on the other hand, are many specifications and technologies that have been processed by OASIS.

³⁵ At the vote taken on June 20th, 2005, to ratify JBI, both IBM and BEA abstained for similar reasons. IBM said: "Many technologies and open specifications are available to the Java programmer today with more compelling interoperability and better mechanisms for component composition." They also thought the specification too complex. BEA were briefer: "BEA believes that the JBI specification is an incomplete attempt to standardize the interfaces between multi-vendor infrastructure and contributes little to the usefulness of the Java platform for business application integration, one of the real pain point for our customers. It's unfortunate that [its] name alone will result in significant confusion in the marketplace" [[JCP, 2005a](#)].

WCF is a connected-systems framework that is built solidly on SOAP 1.2 and aims to integrate different types of technology for distributed computing with a federation model. Chappell explains:

.. with the universal agreement among vendors on Web services, the long-standing problem of application interoperability can .. be solved. Because Indigo's fundamental communication mechanism is SOAP, Indigo applications can communicate with other software running in a variety of contexts" [[2005](#)].

One of the most encouraging features is the clear adherence by WCF to the standards that differentiate web services from distributed object technologies. Box points out that WCF is built on the following four cornerstones [[2004](#)]: that

- *boundaries are explicit* (a movement away from technologies such as RPC, which aimed to make the boundaries transparent)
- *services are autonomous* (the service and the client are independent, using the WSDL as their interface)
- *services share schema and contract, not class* (the object-oriented behaviour of passing classes and methods across boundaries was a feature of distributed object technologies)
- *service compatibility is determined based on policy* (choices are determined according to agreed web-service standards).

WCF offers two different interfaces, one internal and one for communication with non .NET-based services. The internal interface is "optimized" to use a binary format of XML for message exchanges, while the external interface will use a normal text-based XML SOAP format.

Officially Microsoft's BizTalk server implementation (which uses BPEL but transforms it internally into Microsoft's own XLANG) will still function as the main Microsoft service integration model, but there is talk of integrating this too into WCF further down the line.

Microsoft emphasizes that its new approach towards web services, as embodied in WCF, will involve the developer in writing much less code because the heavy work is carried out by means of the annotations and XML configuration files. As with other, less sophisticated, toolkits, this removes the developer even further from what is actually happening at a lower level and

contradicts the advice from Slama *et al.* [2004], cited in Chapter 2, that problems arise when the developer is too far removed from the distribution. What is refreshing, however, is that there is a stated aim and effort to interoperate with other platforms.

6.5 Summary

This very brief look at web-service integration ventures and specifications has demonstrated that many of the attempts at a meta-level unsurprisingly echo the problems that are often found at the root-level: tight-coupling and vendor-competitiveness. It is difficult at this stage to measure the likely success of integration technologies and specifications because of competing approaches and versioning problems, and also because the web-service technologies upon which they are built are still in flux.

CHAPTER 7: VALIDATION OF QUALITATIVE FINDINGS

7.1 Introduction

The services described in this chapter are a validation of the qualitative findings expressed in Chapters 4 and 5. The services described below were constructed for a variety of platforms, and clients for each of them were also run from a variety of platforms. Mindreef's *SOAPscope* and Parasoft's latest *SOA Test* tools, were also used. As well as testing the services for interoperability against the Basic Profile 1.1, *SOA Test* also created unit tests for each method of a service. The services were developed to supply proofs-of-concept and were not intended to be of production quality. While the Hello World service and the calculator service are versions of examples that will be found in many places, the others were developed by the author.

A choice was taken not to implement these services with the *Java Web Services Developer Pack* (JWSDP) because of its lack of transparency: in the past its WSDLs and web-service methods have been automatically generated (with no WSDL-First policy), and the numerous generated files are not only very lengthy but also not user-friendly. A general observation, which applies to all the tested frameworks and kits is that the further the developer is removed from what is actually happening on the wire, the greater the potential for problems introduced by complexities in the environment.

After giving an overview of the platforms used for development, this chapter will examine in turn each of the web services and the clients used to make calls on them. The focus of the chapter will be on any interoperability problems that arose from the implementations, the efforts that were made to resolve them and the conclusions that it was possible to draw concerning interoperability. The chapter will conclude with a brief survey of the findings presented as a table to draw them together.

7.1.1 The Platforms

This section provides some explanatory details about the web servers used in the development and testing of web services and the reasons for choosing them.

7.1.1.1 Apache Axis 1.2 (Java)

A desire to avoid environment-introduced complexity was one of the reasons for the selection of Axis (*Apache Extensible Interaction System*) as the initial Java testbed. Axis is an open-source SOAP engine or "framework for constructing SOAP processors such as clients, servers, gateways" [[Axis user-guide, 2005](#)], originally donated to the Apache foundation by IBM as *SOAP4J*.³⁶ Axis is a specialized servlet that can run inside a servlet container such as Apache Tomcat. An earlier version of Axis was part of the reference implementation of JAX-RPC. Axis is moving towards a fuller implementation of SOAP 1.2, for which it currently has some support. While release 1.2 Final of Axis was used for most of the Axis development, features of version 1.3 (released in October, 2005) were examined for changes.

The main advantages of Axis, according to its 1.2 user-guide, are that it is easily-configurable, flexible, stable, component-oriented, standards-compliant and fast, providing extensive WSDL support and a transport framework, while being essentially transport-neutral – not a paltry list by any standard. Axis may run as a server in standalone mode or within a web server. The latter was the form chosen, with Jakarta Tomcat as the chosen web container because it is open-source, easily configurable and standards-compliant. Many vendors of SOAP web services claim that Axis is supported by them. It might be said to represent the best that is available in SOAP toolkits in that it aims to incorporate current standards and specifications without commercial interests.

The only significant drawback of using Axis for the creation of interoperable web services is that the service proxies it generates are compatible only with Axis, a drawback that would probably be replicated in other systems and toolkits, although Sun's announcement in section 7.1.1.3 below sets its artefacts aside as portable. Some versions of Axis before version 1.2 Final did not

³⁶ IBM was co-author of SOAP 1.1 [[Box et al., 2000](#)].

implement the Basic Profile requirement of a `SOAPAction` WSDL attribute and therefore omitted the `SOAPAction` header in the SOAP message structure, but this is not the case with the current version. The default WSDL style even in Axis 1.3 is still RPC/encoded.

7.1.1.2 GlassFish (Java)

GlassFish is the name given to the source-code release of Sun's Application Server 9, not yet commercially released, but made available to the Java Community for their contribution to its development. What puts GlassFish apart from earlier releases of Sun's AppServer is its inclusion of state-of-the-art Java technology including an early-access release of the JAX-WS (formerly called JAX-RPC). The default WSDL style in GlassFish is document/literal.

As with many toolkits, GlassFish offers two approaches to creating web services. The first, closer to earlier versions of J2EE, is code-based and uses annotations to create the mappings between Java and XML. While this approach is clearly opposite to that encouraged by those who support the theory of WSDL First for interoperability, it does have the advantage of saving the developer from having to write, and rewrite, boilerplate code and configuration files. The second approach, following the WSDL-First policy, starts with the WSDL file and from that generates an interface. The documentation for the JAX-WS describes the trade-off that each approach involves:

If you start from a Java class, you can make sure that the endpoint implementation class has the desirable Java data types, but the developer has less control of the generated XML schema. When starting from a WSDL file and schema, the developer has total control over what XML schema is used, but has less control over what the generated service endpoint and the classes it uses will contain [[Sun, 2005b](#)].

7.1.1.3 Sun Application Server 8.1 (Java)

Although version 8.1 of the Sun Application Server was designed to use JAX-RPC 1.1, it is possible to build into it the functionality of the early-access release of JAX-WS. Because JAX-WS introduces support for SOAP 1.2, as well as for the use of metadata annotations, it was

possible to examine the extent to which the new developments aid web-service interoperability either with or without the SOAP stack. Sun announced that the artefacts generated by its new tools were portable and might therefore be run on any J2EE-compliant server. Sun's (former) technical lead for JAX-WS recently announced the achievement in JAX-WS of a 52% reduction in the number of files generated and a 77% reduction in the size of these files [[Kohlert, 2005](#)].

With JAX-WS, the creation of a web service becomes more transparent – not just a point and click exercise. Ant tasks may be used as aids in the development process and the possibility exists of starting a web service either from a Java class or from a WSDL. A change is that no service endpoint interface needs to be created. Notably, the structure of the samples bundled with JAX-WS reveals that, even when a service is created from a Java class, the implementation should include a schema as a joint starting point. The default WSDL style is document/literal.

A close examination of the code generated for clients by the Application Server engine reveals the creation of a client proxy or stub for the client, along with the generation of interface artefacts and extensive use of `java.rmi.Remote`. The `AddNumbersImpl` sample interface extends `Remote` and the client class throws a `RemoteException`. This is not the only type of client that may be created, however. JAX-WS offers a `Dispatch` API which allows the developer to work at the level of the XML messages, but at the expense of more complexity in terms of the APIs involved.

7.1.1.4 BEA WebLogic 9.0 (Java)

This latest version of the WebLogic server (released in November, 2005) comes with an extensive implementation of the web-services metadata specification, not surprisingly because the head for the specification was from BEA [see [Zotter, 2005](#)]. The server documentation explains clearly how it is possible easily to customize the generation of the WSDL file without the user having to know exact WSDL details, although he or she would need to know something of WSDL files in order to take advantage of the metadata facilities. It rapidly became evident that, for a developer working from code, the metadata features are the next best option to WSDL. First in that they invite him or her to step outside the code and think in terms of the XML

elements that will be created. Code and WSDL may be developed in parallel. While most of the new metadata annotations are present, the `WebFault` metadata annotation has not yet been implemented and so exception-handling falls back to the earlier JAX-RPC 1.1 model. Unlike GlassFish and the Sun AppServer, the default WSDL style for WebLogic is `document/wrapped` literal.

7.1.1.5 Web Matrix (C#)

Web Matrix is an older free development tool, based on the Microsoft .NET 1.1 framework and Software Development Kit (SDK). It is also a lightweight alternative server to IIS. It was chosen for C# development, not only because it was free, but also initially because the learning curve for using it to create C# web services was not as steep as that for learning *Visual Studio 2005* which at the time of initial development was still in Beta 2. A further advantage of Web Matrix is that it is not project-based and has a small footprint. It does not implement the "code behind" features of *Visual Studio* and its web-service creation is more transparent. It uses the default Microsoft WSDL style of `document/wrapped` literal.

7.1.1.6 Visual Studio 2005 (C#)

Although *Visual Studio 2005* (released in November, 2005) cannot be described as open-source, it offers Microsoft's latest implementation of web-service standards and was therefore useful for both comparison and development. It is based on version 2.0 of the .NET Framework and SDK. Its default service style is `document/wrapped` literal.

7.1.1.7 Apache 2 (PHP)

Apache was used as the web server for the PHP implementations. The latest version of PHP (5, released in 2005) was used because of its new support for web services. Some development was done with the recently released (November, 2005) Zend Studio 5, which has introduced a WSDL generator. The default WSDL style for both PHP 5 and Zend Studio appears to be `RPC/encoded`, although this is not explicitly stated.

7.1.1.8 Mozilla *Firefox* (JavaScript)

Firefox was chosen as the browser because its open-source code makes it possible to create extensions to use with it. There are a number of different technologies involved in the creation of *Firefox* extensions, the most prominent of which are XUL (*XML User Interface Language*), RDF (*Resource Description Framework*) and JavaScript. The difficulty in creating this extension lay in the fact that, while some sources did describe the creation of simple extensions, these extensions did not involve web services. Further exploration of the Mozilla SOAP API produced a sample application [[Rosenberg, 2003](#)] (though not an extension) that used the API to call a web service. Even better, there was a reference to a sample web service call, which made it possible to determine that *Firefox* did support the Mozilla SOAP API. Further details on the development of the extension may be found on page 179 in Appendix B.

Although no service development was carried out with JavaScript, it was used to create a client in the form of an extension to the Mozilla browser, *Firefox*. The Mozilla JavaScript API offers support for SOAP 1.1, with a default style of RPC/encoded. JavaScript SOAP browser implementations are more limited in that they require special permissions in order to be able to run. Extended details of the *Firefox* extension described below are given in APPENDIX B: *Creating A Firefox Extension*.

A recent (2004) excellent addition to the Mozilla APIs is ECMAScript for XML (E4X), originally created by BEA as a means of incorporating literal XML into JavaScript, and the basis of a technology with a new (2005) name but ancient roots, Ajax (*Asynchronous JavaScript and XML*). It is possible to create web services with Ajax that may be deployed within Axis on the basis of a deployment descriptor file containing Ajax script, without any further code. The WSDL generated by Axis from this descriptor does not need a `types` section because the XML is incorporated into the language directly and therefore needs no conversion. Some experimentation was done, but Ajax web-service implementations are not at a mature stage and were not developed for this study.

7.2 Web Services

7.2.1 The Simple Calculator Service

The calculator service was one of the samples built into Axis and was used for basic testing. Because of their simplicity, SOAP messages generated by this service were used to illustrate the different messaging styles in section 4.2.4.1 of Chapter 4. This service was implemented in each of the three styles, RPC, document and wrapped. While the RPC and wrapped style implementations behaved as expected, returning appropriate responses to a programmed client, the document style implementation encountered problems:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <in0 xmlns="http://calc">7</in0>
    <in1 xmlns="http://calc">8</in1>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope
```

Method name omitted

The service had difficulty interpreting the SOAP message without the method name which it was expecting and kept returning an `IllegalArgumentException`, whichever method was invoked. Other SOAP implementations (for example WebLogic) have also followed Microsoft's example in making all document style services *wrapped*, which makes sense in terms of including the method name for identification.

That this was the case was confirmed when another approach was tried, in the form of generating the WSDL file with the Axis `Java2WSDL` tool. Using the tool with the document option caused a warning to be generated that problems might be encountered and, when the WSDL so created was used as the basis for SOAP messages generated within *XMLSpy* and *Oxygen*, the form of the message was identical to the previous document-style SOAP message that had been intercepted on the wire:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <in0 xmlns="urn:calc">INT</in0>
    <in1 xmlns="urn:calc">INT</in1>
  </SOAP-ENV:Body>
```

Method name omitted

```
</SOAP-ENV:Envelope>
```

and the same error message was received in the SOAP response.

Although SOAP 1.1 does not support the HTTP GET method, there is elementary support for it within Axis. After invoking, for example, a document/literal version of the calculator service in a browser with the URL,

```
http://localhost:8080/axis/services/Calculator?method=add&in0=6&in1=2
```

the following successful SOAP message appears as a response in the browser window:

```
<soapenv:Envelope>
  <soapenv:Body>
    <addResponse>
      <ns1:addReturn>8</ns1:addReturn>
    </addResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

An interesting feature of the GET method here is that, even with a document/literal style, it enables the use of the method name, bypassing the confusion about which method was being called that was experienced with the SOAP 1.1-compliant and WSDL-generated POST method.

7.2.2 An Even Simpler Date Service

Because of the confusion caused by using an inappropriate `Date` object in Java (mentioned on page 48), the simplest possible Java RPC/literal service was created using this code based on the `java.util.Date` class:

```
public class Timings {
  private aDate = new Date();

  public Date getTime(Date d) throws JAXRPCException{
    aDate = d;
    return aDate;
  }
}
```

Axis generated a WSDL which did, as expected, make the conversion to a `dateTime` schema date type:

```
<wsdl:part name="getTimeReturn" type="xsd:dateTime"/>
```

and calling the service with a canonical schema `dateTime` rendering of a date:

```
1994-11-05T13:15:30Z
```

produced an accurate response:


```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getTimeResponse xmlns="http://myTime">
      <getTimeReturn>1994-11-05T13:15:30.000Z</getTimeReturn>
    </getTimeResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Inserting into the SOAP message anything other than the canonical rendering of the schema `dateTime` data type caused the service, as expected, to throw an exception. It was still surprising that a normally formatted `dateTime` worked, when it had been anticipated that it would fail because of an expected conversion, on the server side, of the XML `dateTime` type into a `java.util.Calendar` object, which should have been incompatible with the `java.util.Date` object required by the service. It is possible that behind the scenes Axis deserializers make a correct conversion, but this assumption has not been substantiated.

The service was also deployed into WebLogic with the same correct result. When the WebLogic version was called with a .NET 1.1 client having a `C# DateTime` object set to the current date and time, an accurate response was also received, again showing that correct conversions to and from XML had been made. Using a .NET 1.1 client to call the Axis service caused a different problem when a message appeared warning that .NET 1.1 did not support the RPC/literal style which had been used. This was in spite of the fact that the WSDL for this service had been run through *SOA Test* and had received a clean bill of health in terms of the Basic Profile, as had the WSDL for the WebLogic version.

A `C#` version was also created in *Visual Studio 2005*, this time with the service class making a comparison between the date that was received and a date that had been hard-coded. This time a `java.util.Calendar` object was used to create a date and the WSDL, which used the default Microsoft style of document/wrapped literal, defined the date object that was sent inside a schema `complexType` representing the method. Java clients were tested against this service. As before, when an actual date (or in this case `Calendar`) object was sent, there was no problem.

But, as anticipated³⁷, when the `Calendar` object was set to null and sent across, exceptions were thrown on the server side.

7.2.3 The Indexing Application: Java

This service was based on SOAP 1.1 and WSDL 1.1, developed for Axis, and built to display more complex interactions inside the service. The purpose of the indexing web service was to make it possible for a user both to maintain a record of significant web pages, independent of the machine on which he or she was browsing and also to query that record with keywords. The index was maintained on a web server to which the user was identified before either indexing a file or querying an index.

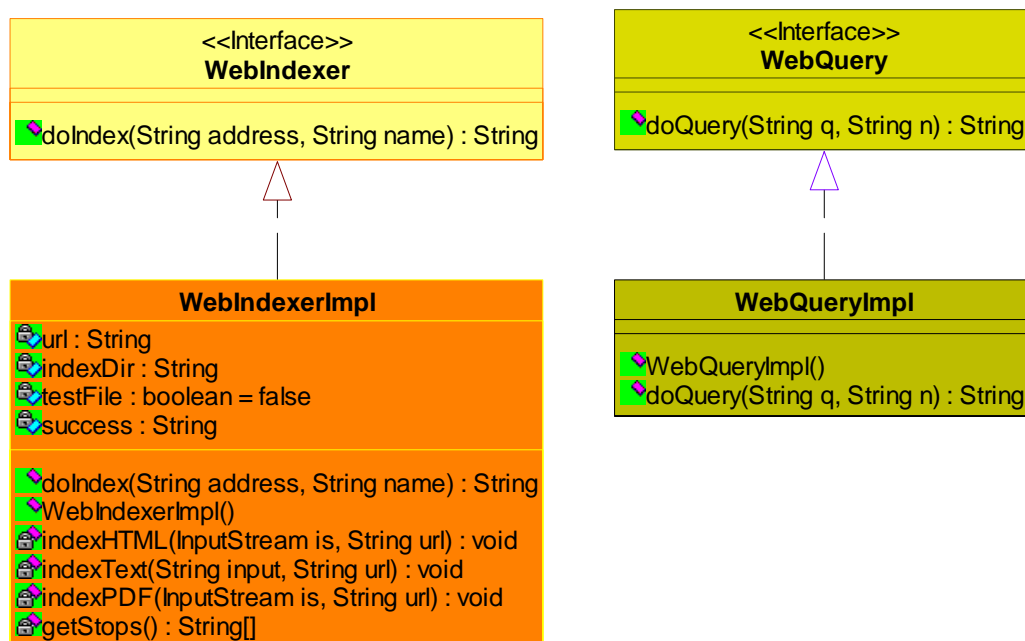


Figure 7-1: UML Diagram of the Index and Query Application

The UML diagram in Figure 7-1 outlines the main service classes for the indexing and query service. The query class was very simple, accepting a query as a string and then returning a response, which either stated that the query term could not be found in the user's index or

³⁷ This is a well-known problem, arising from the fact that in Java a `Calendar` object is a reference variable which may be set to null, while in C# `DateTime` is considered a value type which may not be null.

contained hyperlinks to the web pages on which the query term could be located. Essentially the messages for the indexing service were also simple, accepting a hyperlink and an indexing directory name as input strings and returning confirmation of the index creation or extension, but the inner workings of the service were more complex in that the page to be indexed had first to be fetched in memory, its file type deciphered, a conversion to text format achieved, and then it had to be indexed. Conversions to text from HTML and PDF formats were written, but the application could easily be extended to include the conversion of, for example, MS Word and PowerPoint formats.

Although the indexing service was a fairly straightforward SOAP service, with SOAP messages passed over HTTP, there was a secondary need for web access so that the web pages with input URLs might be obtained from host servers and cached in memory, in order for them to be processed. For simplicity and efficiency, the Jakarta Commons `HttpClient` API was chosen to implement the HTTP calls to the external web servers. These calls were idempotent, as were the service calls. The `HttpClient` API also provided an easy means of accessing remote files through the university proxy server. The mime content-type HTTP header delivered by the `HttpClient` instance determined the kind of processing for each file type.

Apache Lucene is another open-source API, also developed under the Jakarta project, for text indexing and querying. A Lucene index is a data structure, stored in a series of files, which permits fast random access and may also be mapped to any type of database. Because the web service must run independently of the machine used for browsing, the index files were stored on the web server which responded to user requests: they could, of course, be stored on a server independent of the web server processing the service calls. The more intricate details of how this service was constructed may be found on page 182 in Appendix D.

Different ways of implementing the application as a service, and of clients to consume it, were tested. For this service, which was a learning experience because it was the first to be developed, the application classes were written first, with Apache Axis then being used to generate as many different WSDL files as was possible for all the different service types, with the WSDL files

being refined and changed as the process continued. Figure 7-2 gives the *XMLSpy* graphical rendering of the document/wrapped version of the WSDL for the indexing service

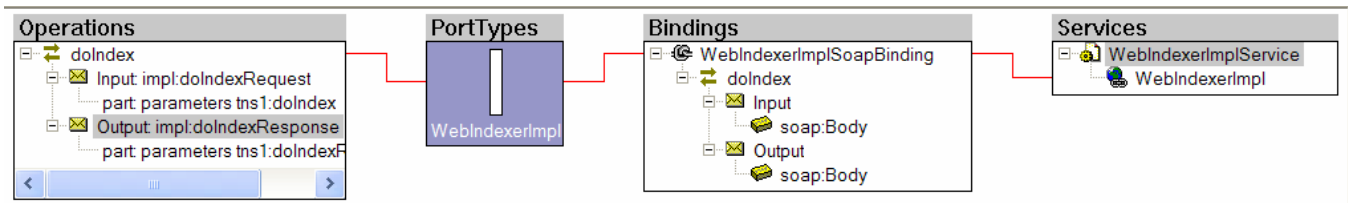


Figure 7-2: A Graphical Rendering of the Web-Service WSDL

Because of the simplicity of the input and return parameters, the indexing service was also able to be called successfully from a PHP client running on Apache, and from a C# client running on the internal web server provided with *Visual Studio 2005*. Although it was possible to implement the service with either RPC or document programming styles, the nature of the application meant that it was closer to RPC in that it required a method call that passed and returned parameters. The WSDL file was very easy to use in the machine generation of clients because of its use of strings as parameters.

7.2.3.1 A Firefox Front End

One way of testing the interoperability of the service was to create a client as an extension for the *Firefox* browser, in which calls to web services are made in JavaScript with the Mozilla SOAP APIs that support it written in C++.

The extension worked as successfully as had the web service run using Axis, going some way to prove that calling the service in another language was of no consequence, and that it was interoperable. Although the sequence of interactions for this service was not simple, the parameters to the service method and the return values were simple, no interoperability difficulties with the use of text strings was either anticipated or found.

7.2.4 The Dictionary Service: Java

The dictionary service was much simpler in its interactions than the indexing service. The focus of exploration with this service was data-type serialization. For the initial Java version of this service, which involved a user-defined data type, the starting point was a W3C Schema and JavaBeans, as a means of comparison against the classes generated by data-binding tools such as the AXIS WSDL2Java tool, Castor and XMLBeans.

There have been complaints about the .NET `wsdl.exe` code generated for multidimensional arrays or arrays of complex types, particularly when the complex types are themselves nested inside a schema [Ingalls, 2004]. The complaints were refuted in the same listing but it was pointed out that constructs such as jagged arrays³⁸ require "a particular style of XSD type definition in the WSDL" for them to be compatible with .NET, with "different programming models" on the client and the server – not exactly interoperability!

With this service, the first step was to create a schema for the data type required:

```
<xs:schema targetNamespace="http://dct"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="DictType">
    <xs:sequence>
      <xs:element name="headword" type="xs:string" nillable="true"/>
      <xs:element name="plural" type="xs:string" nillable="true"/>
      <xs:element name="pos" nillable="true">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="noun"/>
            <xs:enumeration value="verb"/>
            <xs:enumeration value="adjective"/>
            <xs:enumeration value="adverb"/>
            <xs:enumeration value="article"/>
            <xs:enumeration value="preposition"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Here a `DictType` was defined as being composed of a sequence of three other elements, a `headword`, a `plural` and a `pos` or part of speech. Because the part of speech may have a limited

³⁸ .NET arrays which have arrays for elements and where the elements may be of varying dimensions and sizes

number of values, it was made into a simple type that was a restriction on a basic string in its enumeration of six possible values. The element structure is illustrated in Figure 7-3. It was believed originally that the availability in Java 5 of the `enum` data type would make the programming language-to-XML translation easier for both Java and C# clients – but there are problems with enumerated types in that they are treated differently in programming languages and XML. (Page 146 takes this discussion further.)

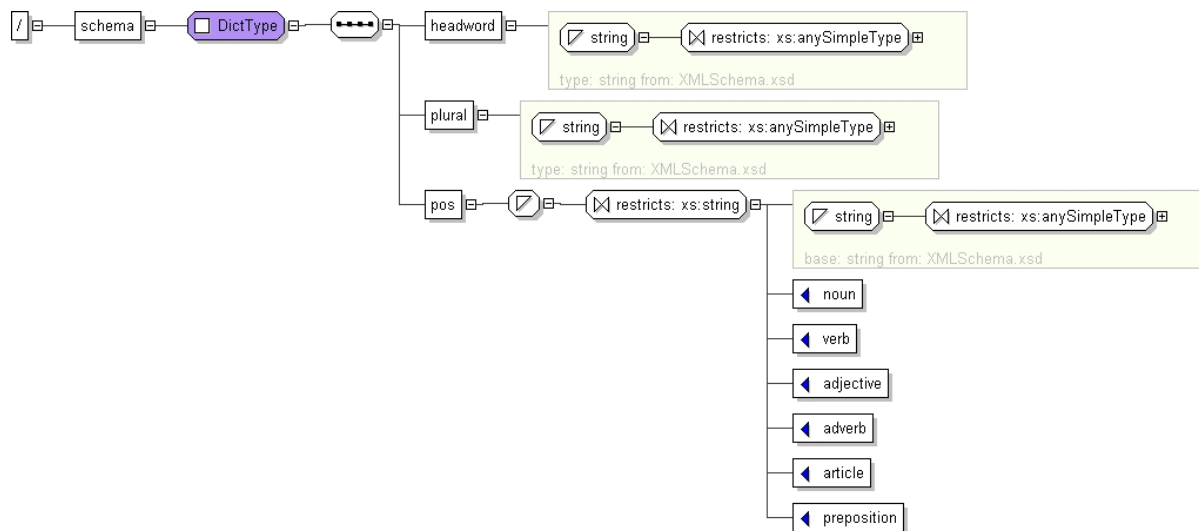


Figure 7-3: Graphical Depiction of the Schema for the Dictionary Service

The bean class to represent this type was designed as follows:

```
public class DictEntry {
    private String headword;
    private String plural;
    private enum PartOfSpeech {noun, verb, adjective, adverb, article,
                               preposition};
    private PartOfSpeech pos;

    public DictEntry () { }

    public String getHeadword(){return headword;}
    public void setHeadword(String head) {headword = head;}
    public String getPlural() {return plural;}
    public void setPlural(String pr) {plural = pr;}
    public String getPos(){return pos.toString();}

    public void setPos(String s) throws Exception{
        for (PartOfSpeech sp : PartOfSpeech.values()){
            if (s.equals(sp.toString())){
                pos = PartOfSpeech.valueOf(s);
            }
        }
    }
}
```

```

    }
    if (pos == null)
        throw new Exception("Constant does not exist");
    }
}

```

With this service a mixture of approaches was employed, approximating to the WSDL-First approach recommended by many developers. The Axis `wsdd` architecture³⁹ was first used to generate a WSDL, with the output being tweaked initially with as many `wsdd` input parameters as were available. A graphical representation of that WSDL appears as Figure 7-4.

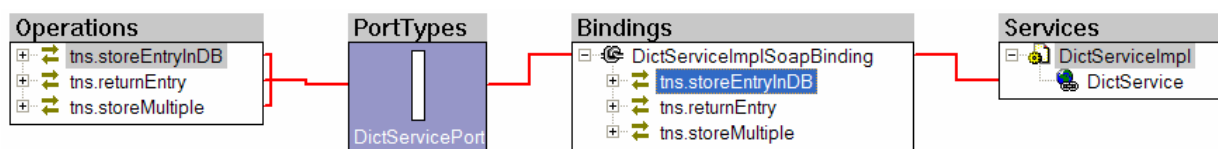


Figure 7-4: XMLSpy Graphical Representation of the WSDL for the Dictionary Service

Because of inexperience, a schema that would represent the whole service, including the messages, was not at first implemented. This would have been much more useful and was implemented later. The omission illustrates the problems caused by thinking in a programming language (Code First) and thinking of Java-data-type-to-XML conversion, rather than thinking more abstractly in terms of a contract (WSDL or Schema First) in which the messages exchanged may also be represented in XML as types. The omission was later rectified by the creation of a schema that represented the service more fully.

7.2.4.1 Using Exceptions in the Dictionary Service

Experiments were made with exception throwing, included in section 3.4.3 of Chapter 3 as one of the hindrances to interoperability. First a new `TransportFault` class was created that extended `RemoteException` and implemented `java.io.Serializable`. The fault was thrown within the implementation class. The fault details were included in the Axis `wsdd` file and from this a SOAP fault was generated in the WSDL file, which should have provided (and to a limited extent did indeed provide) a portable means of returning an exception. A coherent SOAP

³⁹ The Axis `wsdd` (web service deployment descriptor) file is an XML configuration file which allows the user to give specific instructions to the Axis WSDL generation engine.

`faultstring` was, however, difficult to generate by this means. All that was returned initially was the name of the fault, which would be of little assistance to a user on a service failure.

Following the suggestions of Wang and Butek [2004], the next fault generation was through the use of a simple `javax.xml.rpc.JAXRPCException` which was thrown from the main service method. Because `JAXRPCException` inherits from `Exception` (which inherits from `Throwable`), a Java (or C#) client class may simply enclose the attempted service call in a `try..catch` block for the all-embracing `Exception` and use the `Throwable` method `getMessage()` to discover a wide range of runtime problems. `JAXRPCException` has a constructor which takes a simple `String` message outlining the reason for the problem. A further advantage of `JAXRPCException` is that it is automatically translated by the Axis runtime into a SOAP fault which is returned in the body of the SOAP response as follows:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <soapenv:Fault>
      <faultcode>
        soapenv:Server.userException
      </faultcode>
      <faultstring>
        javax.xml.rpc.JAXRPCException: Unable to make connection
      </faultstring>
      <detail>
        <ns1:hostname xmlns:ns1="http://xml.apache.org/axis/">
          Mad
        </ns1:hostname>
      </detail>
    </soapenv:Fault>
  </soapenv:Body>
</soapenv:Envelope>
```

A third experiment with exceptions was the use of a simple user-created exception, `DictException`, for this service. Later experimentation in WebLogic was carried out using a `SOAPFaultException`. Although the `DictException` was automatically included in a generated WSDL file, changing this to a `SOAPFaultException` meant that it was excluded from the WSDL, even though the functionality within the method was essentially the same. Exceptions are the focus of section 7.2.10 below.

7.2.4.2 Testing on Other Servers

The dictionary service was tested on all the servers, including WebLogic. A feature in WebLogic that differs from the other Java servers is its handling of arrays, which is close to the Microsoft approach in that it models an array of user-defined data types as a separate, "wrapped" data type in itself, which makes for good interoperability with Microsoft. When WSDLs used with the Web Logic service were tested against the Basic Profile, using Parasoft's latest version of its *SOA Test* tool, a fault found was the use of "the convention `ArrayOfXXX`", which caused a warning, but not a failure, to be issued. (Parasoft uses the latest (June, 2005) version of the WS-I Test Assertion Document.) The "fault" was actually a mistake: assertion BP2110 [[Lauzon and Wagh, 2005](#)] is meant to apply to *RPC-style* WSDL documents and the style of the WebLogic WSDL was document. BEA have also created their own testing tool which may be used for any web service, including those run on other servers. When run against the Axis services, it found fault with the omission of the `SOAPAction` header from the WSDL files, but otherwise had no problem.

An inspection of the message request and response flow to and from WebLogic revealed that the HTTP `Content-Type` header was `text/xml` but, because it carried a `charset=utf-8` addition, the XML content, had it required non-ASCII characters, would have been correctly interpreted. What the `Content-Type` header also revealed was an adherence to SOAP 1.1, which was to be expected in an implementation adhering to the current Basic Profile.

All the unit tests on this service, generated by *SOA Test*, succeeded, with the exception of the attempt to store a `DictEntry` that was actually empty of data, which was expected to fail and did so. An attempt to store multiple entries succeeded where it should have failed (there was, again, no content) – perhaps because the array wrapper was there even though it contained nothing.

7.2.4.3 C# Clients for the Dictionary Service

In order to create a C# client for a service, the .NET SDK *wsdl* executable has to be given the location of the service WSDL file as follows:

```
wsdl /l:CS /protocol:SOAP
http://localhost:8080/axis/services/DictServiceImpl?WSDL
```

where the "l" (language) "CS" refers to C# and the "protocol" is the default, "SOAP".

Theoretically it is also possible to use "HttpGet" or "HttpPost" with the `wsdl` tool as alternatives for this protocol, but neither with .NET 1.1 nor with .NET 2.0 is any code generated if either of these options is selected. Choosing the "SOAP12" option with .NET 1.1 generates a message that it is not yet supported and with .NET 2.0 the message just that no classes were generated and that warnings were encountered – though it does not say what these were. Choosing "SOAP" causes the `wsdl` tool in both versions to generate a proxy class, which contains details of the service methods and any user-defined service types, in the case of this Axis service the `DictEntry` type. Although the methods for the dictionary service were designed to be synchronous, the proxy class, even with the early .NET 1.1, makes it possible for calls to be made asynchronously, providing "begin" and "end" methods which correspond to the outbound service call, and the collection of the response, for each synchronous method in the original. This is also the case with WebLogic code generation.

The proxy class, which contains no `Main()` method, may then be turned into a .dll file with the following command:

```
csc /t:library /r:System.Web.Services.dll /r:System.Xml.dll
DictServiceImplService.cs
```

A simple client which uses the proxy class by creating an instance of it must then be written:

```
using System;

namespace TestDictServiceImpl {
    public class Client {

        public Client() {
        }

        public static void Main () {
            // Create a proxy
            DictServiceImplService dsis = new DictServiceImplService();
            DictEntry de = new DictEntry();
            de.headword="class";
            de.plural = "classes";
            de.pos = "noun";

            // Invoke storeEntryInDB(de) over SOAP and get the result
            string result = dsis.storeEntryInDB(de);
        }
    }
}
```

```

        // Print out the value
        Console.WriteLine ("The result is :"+ result);
    }
}

```

Compiling and running this class with a reference to the linked library produced a successful result and the efficiency of the process and the detail in the generated files was impressive. In this simple instance, everything was left to the system defaults, but it is possible to customize every facet of the client, including the details of the SOAP messages.

Running the C# client with a more complex data type inside the dictionary service, however, produced some discrepancies and outlined the truth of the claims that developing a web service from code, rather than from the WSDL, introduced the potential for interoperability errors. The `returnMultiple()` method of the dictionary service requires access to the class variables inside the `DictEntry` class, in order for the entry to be decomposed into its parts. When the methods were called that gave access to these (programmatically private) variables inside the C# client, they were found to be absent and the contents of the array returned from the service could not be accessed. In the end it transpired that this was not the real problem. The C# generated proxy class should have given straight access to them in its representation of the `DictEntry` type, as it had converted them to public variables, in accord with the service WSDL, as they were revealed inside its embedded schema, and types so revealed are assumed to be public.

Examination of the SOAP messages on the wire revealed that all the data was being transmitted:

```

<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <returnMultipleResponse xmlns="http://dct">
        <returnMultipleReturn>
          <ns1:headword xmlns:ns1="urn:dct">man</ns1:headword>
          <ns2:plural xmlns:ns2="urn:dct">men</ns2:plural>
          <ns3:pos xmlns:ns3="urn:dct">noun</ns3:pos>
        </returnMultipleReturn>
        <returnMultipleReturn>
          <ns4:headword xmlns:ns4="urn:dct">woman</ns4:headword>
          <ns5:plural xmlns:ns5="urn:dct">women</ns5:plural>
          <ns6:pos xmlns:ns6="urn:dct">noun</ns6:pos>
        </returnMultipleReturn>
      </returnMultipleResponse>
    </soapenv:Body>
  </soapenv:Envelope>

```

```

    <ns7:headword xmlns:ns7="urn:dct">child</ns7:headword>
    <ns8:plural xmlns:ns8="urn:dct">children</ns8:plural>
    <ns9:pos xmlns:ns9="urn:dct">noun</ns9:pos>
  </returnMultipleReturn>
</returnMultipleResponse>
</soapenv:Body>
</soapenv:Envelope>

```

The problem lay in the conversion of the WSDL representation of the method, which was as follows:

```

<element name="returnMultipleResponse">
  <complexType>
    <sequence>
      <element maxOccurs="unbounded" name="returnMultipleReturn"
type="impl:DictEntry"/>
    </sequence>
  </complexType>
</element>

```

A sequence of an element marked as `maxOccurs="unbounded"` is the XML signature for an array. It can be seen from this therefore that what is contained inside a `returnMultipleResponse` is an array of `DictEntry` elements. `DictEntry` is itself represented as a complex type, without an array signature, although it does contain an element sequence, which is *part* – but not usually the whole – of the XML representation of an array:

```

<complexType name="DictEntry">
  <sequence>
    <element name="headword" nillable="true" type="xsd:string"/>
    <element name="plural" nillable="true" type="xsd:string"/>
    <element name="pos" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>

```

In the SOAP message it may be seen that what was actually returned did not carry the name `DictEntry` but was represented as a sequence of `DictEntry contents`. C# interpreted this to mean that what was returned was *one array element* (`DictEntry[0]`), which itself contained *three array elements* (`DictEntry`), each of which was *itself* represented as an array containing elements represented by `headword`, `plural` and `pos`. This was a practical example of the array difficulties mentioned on pages 48 and 94, which explain the reasons for this problem. With the use of the sophisticated debugging features available in *Visual Studio 2005*, it was possible to

track down what was happening, but machine generation of a proxy and the construction of a client did not bring the expected result.

It was only through the creation of a C# service corresponding to the Java service that the problems encountered in the previous client were illuminated. There was only one difference between the .NET generation of types and that in Axis, but it was a very significant difference and it concerned the data-typing of arrays. Whereas Axis had described an array of `DictEntry` as just that – an array of a previously defined type, in a typical schema declaration:

```
<element maxOccurs="unbounded" name="returnMultipleReturn"
  type="impl:DictEntry"/>
```

.NET separated out the definition of an array into the now-deprecated SOAP 1.1 encoding:

```
<s:element minOccurs="0" maxOccurs="1" name="returnMultipleResult"
  type="tns:ArrayOfDictEntry" />
```

with the `ArrayOfDictEntry` type being given its own `complexType` definition:

```
<s:complexType name="ArrayOfDictEntry">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="DictEntry"
      nillable="true" type="tns:DictEntry" />
  </s:sequence>
</s:complexType>
```

This was one example of the confusion that may arise from the autogeneration of artifacts mentioned by Shah and Apte on page 80. As has been mentioned many times, arrays are problematic in web services and it was not surprising that the problem arose in this context.

7.2.4.4 Metadata and WSDL Generation

New metadata features in WebLogic were particularly helpful in the ability they gave to control the generation of the WSDL file. The `class` declaration was preceded by:

```
@WebService (name="DictPortType", serviceName="DictService",
  targetNamespace="http://logdct")
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,
  use=SOAPBinding.Use.LITERAL,
  parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)
@WLHttpTransport(contextPath="dict", serviceUri="DictService",
  portName="DictServicePort")
```

Here the parameters supplied to the annotations enabled the developer to define values for WSDL elements without having to trip up over namespace issues. There was also the ability to define input to the `SOAPAction` header and to give user-defined names to return values through the use of annotations on *methods*:

```
@WebMethod(operationName="StoreEntryOp", action="")
@WebResult(name="StoreEntryOutputString",
           targetNamespace="http://logdct")
```

Even though only one namespace was specified in the selections given above, the generator was able to make distinctions where they were needed, using different namespace prefixes in some cases and creating a new, related namespace `java:logdct` to refer to types that had been defined in their own classes, such as the `DictException` and the `DictEntryBean`. These two types were allocated their own schema in the WSDL `types` section, which was distinct from the schema used for the methods and parameters.

As mentioned in the previous section, Shah and Apte are cited on page 80 as arguing that using toolkit technologies to generate a WSDL on the server, from which a proxy could be automatically generated for the client, would produce different versions of the API, resulting in a loss of control over the business interface. Experimentation was made with the `WSDL2Java` and `Java2WSDL` tools in the Axis release and specifically mentioned by Shah and Apte [2004a]. It was not possible to use this approach to generate a WSDL for the indexing service, but the procedure worked for all the other clients and services, with the exception of the PHP implementation of SOAP, which contains no tools for generating clients.

The usually unsatisfactory results of using a WSDL that was completely auto-generated have already been mentioned in the earlier sections of this chapter. They give the developer no control over the service contract and, as was seen on page 48, may even produce a WSDL that may not be accepted as valid. The current trend in favour of WSDL or Contract First is at the other end of the spectrum, requiring detailed knowledge of the specification, not impossible to achieve but time-consuming and error-prone for less experienced developers.

Gauging the accuracy of the conversion from WSDL to implementation code was a more complicated task. Code produced by various WSDL generators was often lengthy and complex, reflecting a necessity to produce serializers and deserializers for complex data types, particularly for the array of `DictEntry` types. There was a surprising level of accuracy in the generated code which remained true to the original WSDL file.

When the WSDL was being generated from the annotations that may be used in programs written for WebLogic, the *array* of `DictEntry` types appeared as a separate entity, as it had in the C# implementation,

```
<xs:complexType name="ArrayOfDictEntryBean_literal">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0"
name="DictEntryBean" nillable="true" type="java:DictEntryBean"
xmlns:java="java:logdct"/>
  </xs:sequence>
</xs:complexType>
```

The singly defined `DictEntry` was implemented afresh and without enumerations in WebLogic:

```
<xs:complexType name="DictEntryBean">
  <xs:sequence>
    <xs:element minOccurs="1" name="Headword" nillable="true"
type="xs:string"/>
    <xs:element minOccurs="1" name="Plural" nillable="true"
type="xs:string"/>
    <xs:element minOccurs="1" name="Pos" nillable="true"
type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Only one of the toolkits, *MapForce*, generated all of the code that was required for a client or a service. In varying degrees, the others required user input in terms of filling in the business logic, with the most being required by PHP. The `Axis Java2WSDL` was just a WSDL generator, while `WSDL2Java` created the classes necessary to construct a client, but not the actual client, and created more of a service template. The user-guide comments on this: "It is intended that the service writer fill out the implementation from this template" [[Axis user-guide, 2005](#)]. There is, therefore, no loss of control over the business interface in round-tripping with Axis.

The same is true for WebLogic, where the `wsdl2service` Ant task generates only partial code, leaving the business logic to be written by the programmer. When the `.NET wsdl` tool is run,

code is generated that acts as a library to enable client code to be written. The programmer must write the actual code him- or herself. It is possible to run an auto-generated client from a web page but this is intended for testing purposes and would not be used to perform any serious business function.

7.2.5 The Enumeration Service

Various tools for code generation from WSDL were tested, such as the Axis `WSDL2Java` tool and the ThinkTecture `jwscf` tool, both implemented as plugins inside *Eclipse*. For one test, a schema was developed as the basis for an enumeration service, which was a shortened variant of the dictionary service:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:nextEnum"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="urn:nextEnum">
  <xs:element name="storeEnum">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="de" type="tns:postType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="postType">
    <xs:restriction base="xs:integer">
      <xs:enumeration value="2"/>
      <xs:enumeration value="4"/>
      <xs:enumeration value="6"/>
      <xs:enumeration value="8"/>
      <xs:enumeration value="10"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="storeEnumResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="storeEnumReturn" type="tns:postType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The reason for the change in the schema enumeration value to integers was partly to test the accuracy of the code representation of schema enumerations, which may be of any basic schema data type except boolean and, whatever their typing inside a schema, *are handled in Java as strings*. It may be seen from Table 3-4 in Chapter 3 that schema enumerations are not included in JAXB mappings and that serialization and deserialization for them must be custom-built. The

reason for starting with the schema was not only that this is good practice. It also meant that the WSDL could be generated from the schema, which was achieved with the `jwscf` Eclipse plugin, and that then service code could be generated in a completely separate process, so that the two-stage disjunction of business logic and code might be some test of the Shah and Apte criticism already cited twice in this chapter.

The Java code generation from the WSDL file was achieved with the aid of *Altova MapForce 2006*, which takes a WSDL file (plus a schema and an instance document of the schema), and requires some mappings to be made before the code is generated, then compiled and deployed to Axis running on Tomcat (the default for *MapForce*) with the aid of a generated Ant build file. An examination of the generated code revealed that integer enumerations had indeed been converted to string data types represented as an array. Regardless of this, when the service was deployed via Axis on Tomcat, and SOAP request messages constructed for it, the string conversions were transparent and both the request and response messages contained integer representations of the enumeration types:

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:storeEnumResponse xmlns:ns1="urn:nextEnum">
      <storeEnumReturn>8</storeEnumReturn>
    </ns1:storeEnumResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Although the generated WSDL file contained no exceptions, these were correctly generated for the service and were returned appropriately if numbers outside those assigned to the enumerations were posted to the service. All this confirms the findings that, if the business logic is carefully constructed in the form of accurate WSDL and schema files, and these (not code) are used as the starting point for building a service, reputable code-generation mechanisms will succeed. If there are inaccuracies in the `types` section of the WSDL or the schema instance document (usually checked by the program before code generation), the code will be inaccurate and will not compile.

A concern with this kind of code generation, which does not involve interoperability, is the length of the generated code. The directory size for the small, generated service just described was 448KB against handwritten service directory sizes ranging from 1KB (the calculator service), to 7.94KB (the BEA service), to 63KB (the indexing service) and to 158KB (the dictionary service). Code generated to enable the construction of clients on Axis was only 33KB. Code generation does have the advantage of satisfying the requirement mentioned on page 28 that a web service should be capable of being machine processed.

Starting from WSDL or schema is one way of avoiding the potential for conversion difficulties such as those listed on page 81. The developer then has to think not in code but in terms of the service contract. REST-style web services, along with document-style programming models, offer another kind of solution by avoiding or at least reducing the hazards of data typing. Many of the concerns regarding RPC, raised in Chapter 4, were the result of research carried out *before* the release of the Basic Profile which "outlawed" RPC/encoded. The current rejection of RPC may be seen as a confirmation of qualitative findings about RPC made in Chapter 4 and typified in statements such as this by Hasan:

Web services are not optimized for RPCs. This is not what they are best at. Web services work best when they respond to messages, not to instructions [2004].

7.2.6 A Simple Hello Service

This service was built within GlassFish on lines suggested by Dochez in his presentation at JavaOne, 2005 [[Dochez, 2005](#)]. The code could hardly have been simpler, consisting merely of:

```
package endpoint;

import javax.jws.WebService;

@WebService
public class Hello {

    public String sayHello(String param) {
        return "Hello " + param;
    }
}
```

Apart from the use of metadata, this simple class differs greatly from the kind of web-service classes that would have been deployed into earlier Sun Application Servers. The class no longer implements `java.rmi.Remote`, nor does it throw a `RemoteException`. The parameters and return types must now comply with JAXB 2.0 mapping schema elements, given in Table 3-4 of Chapter 3. The service was deployed immediately into GlassFish's autodeploy directory and the annotation, `@WebService`, made it possible for the WSDL file to be automatically generated from the URL: <http://localhost:8080/Hello/HelloService?WSDL>.

At this point, however, the situation became more complicated. Most of the web services were initially tested by opening the WSDL in the XML editors *Oxygen* and *XMLSpy*. Both these editors will create and send to the service SOAP messages based entirely on the WSDL file and then display the SOAP response. When, however, the generated WSDL for the Hello service was pasted into these editors, they were unable to generate the SOAP messages from it, because the schema referenced as an import in the WSDL file was unreachable and could not at first be discovered programmatically. After some correspondence with the developer, this problem was located and fixed and found to be an issue with proxy settings in the application. Although the separation of the schema file from the WSDL (making it an import) follows the practice of loose coupling, schema and WSDL are so tightly bound that making changes to the one will necessarily impact on the other and it might make more sense to clients for the whole service definition to be in one location.

Although, with the schema properly discovered, a SOAP request message could now be sent, an error was generated on the server, as it was also subsequently when an attempt was made to run the equally simple client. It was discovered from a developer that this might be related to an updated version of JAX-WS, and to some changes in the APIs, so an even more recent nightly build of GlassFish was downloaded and installed, only for similar errors to be discovered when there was another attempt either to send a SOAP message directly in the XML editors, or to run the client. This situation has been mentioned in detail because, although whatever occasioned these problems will be fixed as the development continues, it does illustrate the issue of versioning changes (referred to by many as "versioning hell") and the difficulties these can create.

Because the rapid development of GlassFish makes it difficult to use for stable development⁴⁰, testing of JAX-WS was transferred to an earlier and stable build of the Sun Application Server. While WebLogic 9.0 implements the metadata annotations of Java 1.5 which are a significant part of JAX-WS, it is still an implementation of JAX-RPC and makes no use, for example, of the `Provider` object mentioned in the next section.

7.2.7 A Different Programming Style with JAX-WS `Provider` and `Dispatch`

This section's testing with the JAX-WS libraries and samples was necessarily tentative because of problems caused by versioning issues. The samples that were most stable were those of the early access release but these did not contain the functionality of later (nightly) builds which did not claim to be bug-free. The reason for studying these builds was that they contained Sun's latest web-service development, including a potentially REST-style interface, discussed below, and they also provided a means for studying cutting-edge interoperability.

The JAX-WS programming style with the `Provider` and `Dispatch` interfaces differs from the styles so far defined for JAX-RPC. Implementing the `Provider` interface means creating a class that does not act as a service endpoint. Whereas before in Java we have seen a web-service class that implements the service methods and is defined by a `@WebService` annotation, the different annotations used by `Provider` for *services* (`Dispatch` is an API mainly intended for *clients*) offer the opportunity to work with XML messages, depending on the `Provider` style selected (the choices are between `Source` for XML messages and `SOAPMessage`):

```
@ServiceMode(value=Service.Mode.PAYLOAD)
@WebServiceProvider(wsdlLocation="WEB-INF/wsdl/AddNumbers.wsdl")
```

The programming constructs for this style are not always straightforward. If, in an attempt to test round-tripping, an attempt to generate a WSDL file from the sample class implementing `Provider` is made, the JAX-WS `wsgen` tool complains rightly that it has not been given a service endpoint interface. The provided WSDL is valid, passes all the Basic Profile tests, and captures

⁴⁰ A late November release was also tried – this time the Ant build file setting up the application contained numerous inaccuracies making the correct installation impossible. The filename mistakes were easy to fix, but indicated that more serious problems might be encountered – and they were.

the service operations which do not appear in the `Provider` implementation class. In line with the principles of WSDL First, it is intended that the WSDL should be the starting-point not only for the generation of clients but also for the actual service classes, including a service endpoint interface named as a `..PortType` class. All of these classes are generated by the `wsimport` tool.

Unlike the other WSDLs described in this chapter, this one may not be used by a SOAP message generator to cause an accurate response to be received. Both *Oxygen* and *SOAPScope* used the WSDL to generate what looked like appropriate SOAP request messages – their wording was identical to SOAP messages revealed by a TCP Monitor in a successful C# *programmed* request – but both received the same error: *Error in provider endpoint*. Even when the exact message sent from the programmed request was copied into the generated SOAP output message dialog within *Oxygen*, it still received: *Unable to create envelope from given source*, whereas the original programmed request had received an accurate response.

The `Provider` sample from the JAX-WS release comes with its own client class, which uses the generated classes to call the service and receive the expected response. More insight into its workings was provided by the `Dispatch` interface, which can be seen as the distinct client side of `Provider`. The given `Dispatch` client class was altered to make its request, not of a `Dispatch` service at which it was originally aimed, but of the `Provider` service and this worked.

The construction of a C# SOAP client from the WSDL was achieved through the use of the `.NET wsdl` tool, which constructs a client-side proxy from the WSDL it is given. As was mentioned in section 7.2.4.3, it is then up to the programmer to use this proxy file, later compiled into a `.dll` file, to create the actual client. This proxy file contained three classes, two of them used for the XML serialization of the `addNumbers` and `addNumbersResponse` data types given in the WSDL:

```
<complexType name="addNumbersResponse">
  <sequence>
    <element name="return" type="xsd:int" />
  </sequence>
</complexType>
<element name="addNumbersResponse" type="tns:addNumbersResponse"/>

<complexType name="addNumbers">
  <sequence>
    <element name="arg0" type="xsd:int" />
    <element name="arg1" type="xsd:int" />
  </sequence>
</complexType>
```

```

    </sequence>
  </complexType>
  <element name="addNumbers" type="tns:addNumbers"/>

```

and the third for the service class, `AddNumbersService`, so named in the WSDL. All the client class was required to contain was the initialization of three object instances from each of these classes and the final casting of the response to an integer type:

```

using System;

namespace TestNumbers {
    public class Client {

        public Client() {
        }

        public static void Main () {
            AddNumbersService ans = new AddNumbersService();
            addNumbers anum = new addNumbers();
            anum.arg0 = 6;
            anum.arg1 = 3;

            addNumbersResponse anr = new addNumbersResponse();
            @anr = ans.addNumbers(anum);
            int result = (int) anr.@return;
            Console.WriteLine ("The result is :"+ result);
        }
    }
}

```

This may be seen as proof of the interoperability of this new style of programming, even though the C# implementation knew only of the WSDL and nothing of the `Provider` interface implemented on the server. JAX-WS suggests an optional customized Java `Provider` binding in the WSDL file (or in another file), which would be picked up only by other Java implementations able to handle it, but this would not be interoperable across languages [Sun, 2005c].

7.2.8 REST-Style Messaging

A REST-style service was constructed using a schema similar to that already created for the dictionary service, and XML documents were created that conformed to that schema. The schema was extended slightly to enable it to function as the basis for instance documents, not just for data types.

The schema document functions as an alternative to WSDL. As was mentioned on page 31, WSDL was conceived before the W3C Schema model had been properly standardized and it may be argued that WSDL might not have come into being had the W3C Schema pre-dated it. The first WSDL specification was released by IBM and Microsoft in September 2000, but the W3C Schema proposal did not achieve Recommendation status until 2001, although the idea of an XML schema had been in discussion from at least 1997, even before XML itself had been formally recognized as a standard.

W3C Schema was chosen because it is the most widely adopted schema language, despite its complexity and a growing use, even by W3C specifications, of schema alternatives particularly RELAX NG. In REST-style applications, the XML content which forms the body of the HTTP message is an alternative to SOAP and provides the arguments to the service. If an XML document conforms to a schema, it is as easily understood by its recipient as any SOAP message, and applications may be built to consume it. A schema can be exposed over the internet in the same way as a WSDL. It is no more difficult to send a document that conforms to a known schema than to send a SOAP message that conforms to a WSDL. The schema is no more subject to extension, change and revision than is the WSDL. WSDLs that are the contracts for document/literal service types must in any case incorporate schemas in their `types` sections.

Shah and Apte argue for using "the XSD Schemas for the business module as the API signature in the web services" [[Shah and Apte, 2004a](#)]. They insist that the use of schemas and their validation in the business module (as opposed to the web-services infrastructure) puts the ownership and validation of the exchange solidly in the hands of those responsible for the business logic, where they believe it belongs, and not in the hands of IT specialists.

The basis for the implemented REST web service was a servlet, `HandleInputServlet`, which handled both HTTP POST and GET requests. There was also a schema for a dictionary model. There were no SOAP messages and no WSDL. A difference from the other services discussed here was the need to code the marshalling and unmarshalling of the XML content passed between the client and the service. There are no toolkits for REST-style web services, although

Axis2 includes a REST model⁴¹. The StAX (*Streaming API for XML*) pull parser was chosen because it does not have the memory overheads of a DOM parser and, unlike SAX parsers, which require callback handlers, it leaves the programmer in complete control. StAX was originally a Java Community Process parser that, according to its specification lead, "grew out of the need to read and write XML in an efficient manner in the context of XML Binding and Web Services" [Fry, 2004]. The BEA reference implementation (BEA provided the specification lead for StAX) was chosen as being currently the most stable and user-friendly.

In the web service, the servlet is capable of responding to both POST and GET requests. If a GET request is received, the servlet simply returns an XML document in the body of the response message, much as something as mundane as a purchase order or catalogue might be returned by a manufacturing company. If a POST request is received, containing XML content, the content is parsed for some of its content and an XML message confirming the content is sent in response. In a more realistic context, the message might be parsed for details needed by, for example, a manufacturing company, and this would result in further external processing.

The reference implementation of StAX does not handle XML document *validation*. In the simple documents exchanged in this application, no errors would usually be caused by faulty documents, because the service looks for only those XML structures it needs to use. The approach of the StAX parser is closer to that of the XML Infoset, upon which the latest versions of SOAP and WSDL are built, in its concern with the presence of particular XML events, or elements, rather than with document structure. It is particularly suited to the parsing of documents where the document structure is known in advance, as is usually the case with web-service document exchanges. It is also particularly well suited to situations in which only sections of the document need to be processed, and also, as with the present web service, to situations in which XML needs to be both read and written.

Clients for this service may be written either as simple applications or as web pages with form input. Both the service and the client applications are extremely simple. Because the data being

⁴¹ Axis2 (version 0.92) was installed within Tomcat (version 5.5.9), but attempts to use it failed because of configuration problems within Tomcat that could not be resolved.

exchanged is basic XML and not programmed objects, there is no question of misrepresenting data types or of incompatible serialization. Any platform, language or program that can send and receive text over HTTP may use this application and it is therefore considered to be the most interoperable service developed in this study.

7.2.9 A Simple HelloWorld Service: PHP

Although technically a scripting language, PHP offers much wider control and functionality than its competitors largely because of the development of community libraries and its popularity with developers of the most widely used, open-source web server, Apache. The involvement of PHP in web services is a recent development in that only in the latest version (version 5, released in 2005) is SOAP incorporated as a core PHP module, although at earlier stages PEAR (the *PHP Extension and Application Repository*), and other library modules offered (and still offer) web-service functionality.

With first attempts at web services in PHP, the findings of Maynard, Charters and Peters [2005] were confirmed, that the information returned from the `getTypes()` method of the PHP `SoapClient` object was insufficient to construct an acceptable method call. Although a PHP call to the simpler Java indexing service was correctly interpreted and both sent and received accurately constructed SOAP messages, the more complex `DictEntry` type used in the Java dictionary service created initial SOAP encoding problems. The WSDL for the service had to be examined in more detail and a PHP class representation of `DictEntry` created before the client could access the service successfully.

Although the core PHP distribution contains no toolkits that will automatically enable the creation of a SOAP client from a WSDL file, SourceForge hosts a project named NuSOAP which aims to fill the gap by providing a means of building SOAP clients and servers in PHP, but NuSOAP predates PHP 5 and is based on a different set of modules, so was not included in this study. In November, 2005, *Zend Studio 5* introduced some basic support for SOAP 1.1 web services in the form of a WSDL generator. Despite the prohibitions of the Basic Profile, *Zend*

Studio 5 still offers the old SOAP encoding format (its default) as an alternative to "literal". It offers no possibility of choice between document and RPC, making all services RPC by default.

The document/literal, wrapped, and RPC styles of WSDL were each tested for a very simple PHP `HelloWorld` server. The service took the input of a name and returned that name concatenated with ", Hello". The following document/wrapped-style WSDL was created by hand:

```

1.<?xml version="1.0" encoding="UTF-8"?>
2.<wSDL:definitions xmlns:soap="http://schemas.xmlsoap.org/wSDL/soap/"
  xmlns:tns="urn:SayHelloPHP"
  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
  xmlns:wSDLsoap="http://schemas.xmlsoap.org/wSDL/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="urn:SayHelloPHP"
  xmlns:tns1="http://data"
  name="SayHelloPHP">
3.  <wSDL:types>
4.    <schema targetNamespace="http://data" xmlns="http://www.w3.org/2001/XMLSchema">
5.      <element name="sayHello">
6.        <complexType>
7.          <sequence>
8.            <element name="user" type="xsd:string"/>
9.          </sequence>
10.         </complexType>
11.       </element>
12.       <element name="sayHelloResponse">
13.         <complexType>
14.           <sequence>
15.             <element name="return" type="xsd:string"/>
16.           </sequence>
17.         </complexType>
18.       </element>
19.     </schema>
20.   </wSDL:types>
21.   <wSDL:message name="sayHelloRequest">
22.     <wSDL:part name="parameters" element="tns1:sayHello"/>
23.   </wSDL:message>
24.   <wSDL:message name="sayHelloResponse">
25.     <wSDL:part name="parameters" element="tns1:sayHelloResponse"/>
26.   </wSDL:message>
27.   <wSDL:portType name="SayHelloPHPPortType">
28.     <wSDL:operation name="sayHello">
29.       <wSDL:input name="sayHelloRequest" message="tns:sayHelloRequest"/>
30.       <wSDL:output name="sayHelloResponse" message="tns:sayHelloResponse"/>
31.     </wSDL:operation>
32.   </wSDL:portType>
33.   <wSDL:binding name="SayHelloPHPSoapBinding" type="tns:SayHelloPHPPortType">
34.     <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
35.     <wSDL:operation name="sayHello">
36.       <soap:operation/>
37.       <wSDL:input>
38.         <wSDLsoap:body use="literal"/>
39.       </wSDL:input>
40.       <wSDL:output>
41.         <wSDLsoap:body use="literal"/>
42.       </wSDL:output>
43.     </wSDL:operation>
44.   </wSDL:binding>
45.   <wSDL:service name="SayHelloPHP">
46.     <wSDL:port name="SayHelloPHPPort" binding="tns:SayHelloPHPSoapBinding">

```

```

47.         <soap:address location="http://localhost/SayHelloExampleServer.php"/>
48.     </wsdl:port>
49. </wsdl:service>
50.</wsdl:definitions>

```

As was outlined in Chapter 4, document-style WSDLs usually contain or reference a schema within a `types` section (ll.3-20 above). A *wrapped* document style schema represents a method call and its response as parent elements (ll.5-11,12-18), with input and return parameters as child elements (ll.8, 15). Any schema element that contains child element(s) is represented as a `complexType` (ll.14-18, 21-25), containing, in this case, a `sequence` (ll. 15-17, 22-24) of parameter or return elements, regardless of the fact that only one may be required. The client code has a level of complexity, as some lines from the actual method call illustrate:

```

$sayHelloParams = array('userName' => $userName);
$sayHelloResponse = $soapClient->sayHello($sayHelloParams);
$sayHelloReturn = $sayHelloResponse->sayHelloReturn;
print $sayHelloReturn;

```

while the server code looks like this:

```

function sayHello($sayHello) {
    return array('sayHelloReturn' => $sayHello->userName . ", Hello");
}

```

With the earliest attempts at creating code for this service, although an accurate SOAP call could be made to the service, and a well-formed SOAP message was received in response, it was not possible to incorporate into the reply the user name that had been sent. The writer is grateful to Dmitri Stogov, one of the writers of the PHP 5 SOAP extension, for examining the code she sent to him and for making suggestions, but it was still not possible to make a working client and server for the *wrapped* service unless they were both in the same file, which did not qualify as a web service. The wrapped WSDL did pass the *SOA Test* interoperability tests (as did both of the other WSDL files for this service) and the conclusion was drawn that the PHP 5 implementation does not yet handle wrapped-style services.

When an attempt with the much simpler code, given below, was used with this WSDL, SOAP messages were sent and returned, but the response contained an *object* rather than a string and a way was not found of "*unwrapping*" the object:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://data">
  <SOAP-ENV:Body>
    <ns1:sayHelloResponse/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The WSDL was then converted to document style, by simplifying the types section to include only elements for the input and return parameters. Even though this style does not include the method name in the SOAP message,

```

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <user xmlns="http://data">Horace</user>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

because there was only one possible method, it received an appropriate response:

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://data">
  <SOAP-ENV:Body>
    <ns1:return>Hello, Horace!</ns1:return>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The document and RPC services both used a much simpler method call:

```
$s->sayHello('Gandalf');
```

based on a similarly simplified service method:

```

function sayHello($user) {
  if (strlen($user)) {
    return "Hello, {$user}!";
  } else {
    throw new SoapFault("Server", "No name sent as input");
  }
}

```

Later, simplifying the WSDL even further, by converting it to RPC-style, meant removing the types section altogether plus other minor changes appropriate to this style, such as using the type attribute rather than the element attribute for the part sub-element for each message element.

It would be naïve to expect that the popularity of the document and wrapped styles will completely supersede RPC style, especially on the corporate internet, where legacy systems demand method calls. The PHP RPC example displays one advantage of this style, which is the greater simplicity it enables in both client and server code. Quite apart from the coding complexities involved with the wrapped service, each of the other styles was seen to be adequate. The possibility of confusion between methods in document style could not be tested with only one method. A further method, `sayGoodbye`, was therefore added to the WSDL and to the code, with the same input parameters, and messages were sent to the service using both coded clients and generated SOAP messages, this time from *XMLSpy*. As expected, the document-style service was unable to distinguish between the two. When the same approach was tried with the RPC service, the method was detected accurately both by the generated SOAP message inside *XMLSpy* and by the client code. Without the possibility of the wrapped style, therefore, the RPC (literal) style has definite advantages for method calls.

Currently the only toolkit for working with the PHP 5 SOAP extension is that provided by *Zend Studio* in terms of its WSDL generator, which currently supports only RPC encoded or literal. There is, however, built-in use and interpretation of WSDL files in PHP 5 which greatly simplifies the creation of PHP web services and clients and appears to be an acknowledgment of the significance of the WSDL-First position, although alternative constructors do make it possible to create a SOAP client without a WSDL file.

7.2.10 A CheckNumbers Service

Section 7.2.4.1 examined exception-throwing in Java. It is easier to think of throwing and catching exceptions with services and clients in the same language because they resolve to the same object. The CheckNumbers service was therefore written to examine what happens when exceptions are thrown in one language and caught in another. The service was extremely simple by design because its focus was on the exception and not on the programming logic. The first version was written in C# and deployed on Web Matrix. The service took an `int` input parameter. If the number was less than 10, the service returned a string message. If, however, the number was greater than 10, a `SoapException` was thrown. `SoapException` is the main

inbuilt exception class for web services in .NET and it is used automatically by the framework to wrap any other exception class that may be chosen or created [Microsoft, 2003]. Despite this, `SoapExceptions` do not appear as faults in the WSDL file generated for a service that uses them, on either .NET 1.1 or .NET 1.2. They are, however, correctly interpreted by both C# and Java clients. In *Oxygen* the SOAP response to a number larger than 10 was:

```
org.apache.commons.httpclient.HttpException :
<?xml version="1.0" encoding="utf-8"?>
  <soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd=http://www.w3.org/2001/XMLSchema">
    <soap:Body>
      <soap:Fault>
        <faultcode>soap:Client</faultcode>
        <faultstring>System.Web.Services.Protocols.SoapException: Your
          number was greater than 10 at
          CheckMyNumberService.Check(Int32 a)</faultstring>
        <detail />
      </soap:Fault>
    </soap:Body>
  </soap:Envelope>
```

When the Axis `WSDL2Java` tool was used to help generate a static stub client in Java, the implementation stub class was shown to throw a `RemoteException`, as a result of the JAX-RPC catch-all requirement that `RemoteException` must be thrown by service methods. Because JAX-WS does not include this requirement, JAX-WS client classes were generated with the `wsimport` tool and a reference to the online WSDL. An examination of these pre-compiled classes in a decompiler revealed that, as in Axis, the service method was made to throw a `RemoteException`.

The service was also implemented in Java on WebLogic, where a `javax.xml.rpc.soap.SOAPFaultException` was used instead of the C# `SoapException`. As with the C# service, no mention of a fault appeared in the WSDL file. .NET does not automatically generate an exception when a web service proxy class is being generated from the service WSDL, unless the WSDL actually mentions the exception as a fault. When the WebLogic service was used by a C# client, no problems occurred when correct parameters were sent to the service, but, when a parameter greater than 10 was sent, the service returned a message that there was an unhandled `SOAPHeader` exception. The exception did contain the fault string but was easier to understand when the message was trapped on the wire than when it was received programmatically.

Another C# client (.NET 2.0) version was then written which used a `try..catch` block round the service call with a `SOAPException`. This returned the exception to the client programmatically and in the SOAP message on the wire. It is perhaps a little obvious to say that a client program must be prepared to catch an exception, but the last example was an illustration of the fact that a generated client may not always do this if the fault is not included in the WSDL file which is used. This was only a simple example and the unhandled exception caused no problem, but it could have been more serious in a production service. This is one example in which the default behaviour of Java stub-creation tools, by including exceptions automatically, pre-empt difficulties better than those for C# and are therefore more interoperable.

7.3 Implementations of SOAP 1.2

Because SOAP 1.2 is not yet supported by the Basic Profile, its implementations are few and often still buggy. *Visual Studio 2005* offers default support for SOAP 1.2 in tandem with support for SOAP 1.1 by including two different bindings and ports inside automatically generated WSDL files, so that a basic example Hello World service has two bindings:

```
<wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">
<wsdl:binding name="ServiceSoap12" type="tns:ServiceSoap">
```

and two related port definitions:

```
<wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">
<wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">
```

Either binding may be switched off inside a web configuration file. Because both bindings are present, either of them may be used by SOAP messages.

The *Visual Studio* environment makes a comparison possible between the SOAP messages sent as a result of each of the example bindings. For SOAP version 1.2, the HTTP request content was:

```
POST /WebSite3/Service.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: ...
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <HelloWorld xmlns="http://tempuri.org/" />
  </soap12:Body>
</soap12:Envelope>
```

while for version 1.1 the equivalent message was:

```
POST /WebSite3/Service.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: ...
SOAPAction: "http://tempuri.org/HelloWorld"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <HelloWorld xmlns="http://tempuri.org/" />
  </soap:Body>
</soap:Envelope>
```

Here it can be seen that in SOAP 1.2 the `SOAPAction` header has been replaced with a more useful definition of the `Content-Type` header, solving the HTTP encoding problems mentioned on page 45. Apart from the namespace differences, the messages are otherwise identical.

Running the service through *SOA Test* produced no interoperability problems (it discounted the extra binding), but the program complained that it could not read the schema from the namespace given for the 1.2 binding, although the namespace, <http://schemas.xmlsoap.org/wsdl/soap12/> is correct according to the WSDL 1.1 binding for SOAP 1.2 [[XMLSOAP, 2002](#)].

Using *Oxygen* to generate SOAP clients produced only a SOAP 1.1 client, with the 1.2 binding being completely ignored by the program, although the development team have said that they will consider including handling mechanisms in a later release. *XMLSpy* did recognize the bindings as distinct and asked the user to choose between them, but then ignored the 1.2 namespace, possibly for the same reason as *SOA Test* above.

7.4 Different Types of Java Client

7.4.1 Static Stub Client

As was mentioned in section 4.2.4.5.1 of Chapter 4, static stub clients rely on the prior generation of classes from a WSDL or a service endpoint interface, usually with a tool such as the AXIS `WSDL2Java`. One of the generated classes will usually include the word "stub" in its name, and will act as a go-between for the client and the client's representation of the service class, so that the client looks as though it is communicating with the service directly when this is not the case. This RPC programming style suffers from the RPC problems of tight-coupling and the deceptive blurring of the distinction between local and remote methods outlined in section 4.2.4.3.

A static stub client was written for a C# version of the dictionary service running on Web Matrix. The stub classes were first generated with the AXIS `WSDL2Java` tool and then used to create the client logic. As the following few lines of client code show, this style of client has the advantage of being brief because most of the working code lies in the generated classes:

```
public class CSharpStubClient {
    public static void main(String args[]) throws Exception {
        DictEntry de = new DictEntry("song", "songs", "noun");
        ExceptService dict = new ExceptServiceLocator();
        ExceptServiceSoap port = dict.getExceptServiceSoap();
        String ret = port.storeEntry(de);
        System.out.println(ret);
    }
}
```

This code made a successful call on the C# service and demonstrates that this style of client is likely to be interoperable with services written in other languages. It is also very easy to write because it is so brief.

7.4.2 Dynamic Proxy Client

As was mentioned in section 4.2.4.5.2 of Chapter 4, dynamic proxy clients differ from static stub clients in that the proxy class is created *runtime* and requires no pre-generated code.

Otherwise the procedure is much the same as the static stub in that the client appears to be communicating with the service but is actually exchanging messages with a proxy through which

contact with the real service is made. This type of client is even more tightly coupled in that its creation depends not only on an awareness of details in the WSDL file but also on a knowledge of the service endpoint interface and of any data types represented as JavaBeans. It cannot function without an interface written and compiled in Java and so can no more interoperate with non-Java services than can RMI.

A not very serious attempt was made to interact with a C# service, using its service class as an interface, but, as expected, the client code would not even compile because of the language difference in the supporting classes. A dynamic proxy client was able to be created for the Axis version of the dictionary service and the significant parts of its code, illustrating its use of the service interface appear in bold below:

```
String urlString =
    "http://localhost:8080/axis/services/DictServiceImpl?wsdl";
String namespaceUri =
    "http://localhost:8080/axis/services/DictServiceImpl";
String serviceName = "DictServiceImplService";
String portName = "DictServiceImpl";
URL dictUrl = new URL(urlString);
ServiceFactory serviceFactory = ServiceFactory.newInstance();
Service dictSve = serviceFactory.createService(dictUrl, new
    QName(namespaceUri, serviceName));
DictService myProxy = (DictService) dictSve.getPort(new
    QName(namespaceUri, portName), dp.DictService.class);
System.out.println(myProxy.storeEntry(de));
```

7.4.3 Dynamic Invocation Client

As mentioned in section 4.2.4.5.3, dynamic invocation clients are meant to be able to be generated at *runtime*, although a method of achieving this has not been discovered. What makes these clients different from static stub clients is that there is no generation of stub classes, and they differ also from dynamic proxy clients in not calling upon a service interface. They are similar to both previous client types in that they do need a knowledge of the service WSDL.

Complexity is introduced, however, with the use of user-defined data types, as with the dictionary service `DictEntry` type, for which the only available definition was the WSDL file. An attempt to fudge the difficulty by creating a `DictEntry` class as an inner class within the client succeeded in compiling, but received a message from the service that the *Server did not*

recognize the value of HTTP Header SOAPAction. Checking the SOAP messages that went back and forth on the wire revealed the usual Axis style of an empty `SOAPAction` header, but this was considered unlikely to be the real problem when the static stub client messages, also generated by Axis, had worked. Further examination revealed that the `DictEntry` parameter had not been sent, which was not surprising when its source had been an inner class lacking the real namespace required for the construction of the SOAP message.

The conclusion drawn was that, while this style of client would interoperate with a service using simple built-in schema types as parameters, it would not work for a service in another language that required the use of a user-defined data type for a parameter. The line of code that makes it work for Java classes,

```
call.registerTypeMapping(DictEntry.class, qn, new
    BeanSerializerFactory(DictEntry.class, qn), new
    BeanDeserializerFactory(DictEntry.class, qn));
```

requires a Java class to work, as may be seen from the *three* references to a class file.

7.5 Summary: a Synthesis of Findings

This chapter has presented findings from practical implementations of different web services and clients in a variety of styles. Each of the implementations has made a contribution to the survey of interoperability for web services, either by providing an opportunity to examine a feature marked as problematic for interoperability, such as the use of array types, or, as in the case of the REST-style service, by demonstrating that a simpler, alternative approach using pre-existing technologies might pre-empt such problems altogether. The following table presents a brief summary of findings with regard to problematic features made in this chapter:

PROBLEM	OUTCOME
Data Types	
<i>Array types</i>	There were problems with the representation of array types. Microsoft systems created a "packing" system for an array of user-defined types which made it difficult to access the actual data if the service it was reaching had not also used this technique. PHP had no mechanisms for handling this.

	WebLogic also uses a packing strategy for arrays, increasing its interoperability with other systems. This representation of arrays is frowned on by the Basic Profile, but not totally disallowed. Testing of wrapped arrays made it appear that the sending of an empty array to a method that was expecting content succeeded where it should have failed because of the wrapper.
<i>Date Java/C# data types</i>	Surprisingly date objects (even the <code>java.util.Date</code> object) turned out to be more portable than had been expected, and no interoperability problems were encountered.
<i>Custom data types</i>	While each of the systems was able to handle complex data types internally, there were cross-system problems and these were probably the most significant failures for all the systems.
<i>Enumeration types</i>	No problem with the internal representation of XML enumerations as strings.
<i>Exceptions</i>	<ol style="list-style-type: none"> 1. Difficult to generate a fault string that could be captured using <code>RemoteException</code>. 2. Using <code>JAXRPCException</code> was more successful in capturing the fault string for a Java Client. 3. <code>SOAPFaultException</code> was efficient.
SOAP features	
<i>SOAPAction header</i>	<ol style="list-style-type: none"> 1. Default rendering of WSDL with beta versions of Axis did not include it: Stylus Studio will not validate the WSDL. Later versions include it as an empty value. 2. The WebLogic <code>testclient</code> program refused to validate a service that did not contain it.
<i>SOAP versioning</i>	Currently not an issue because so few of the toolkits implement SOAP 1.2. This is an issue to watch.
<i>HTTP XML headers</i>	<ol style="list-style-type: none"> 1. Used <code>text/xml</code> but carried a charset indicator. 2. Good support for SOAP version 2 content-type headers.
Toolkit features	
<i>Proprietary features</i>	Planning to introduce proprietary WSDL headers that will not be used by other platforms.
<i>Versioning problems</i>	Not the only culprit but by far the most noticeable, as might

	be expected when looking at nightly builds.
<i>Customization of SOAP messages</i>	None found.
<i>Different language interpretations</i>	The fact that the service was in a different language <i>per se</i> made no difference to simple service calls from clients written in different languages. Language difference in itself (apart from language-specific data types, dealt with above) is no barrier to web services.
<i>WS-I interoperability</i>	None of the generated or hand-created WSDL files failed the tests, despite the fact that very different approaches were used.
RPC-Style	
<i>Run-time binding</i>	Difficult to reproduce the automatic generation of a client at runtime.
<i>Different method signature on the client from the one on the server</i>	Either not found or not a problem. Axis client code generation sometimes produced method names run together with a namespace but even this did not produce an error when the service was called.
Code First	
<i>Flaws in the WSDL file</i>	Running WSDL files through a validation engine should be a requirement before they are used to create a web service. No flaws were found in the WSDL files used. Problems experienced with document-style method invocations where there was more than one method with a similar number and type of parameters could be avoided by using the wrapped style.
<i>Limited perception of types</i>	The metadata features can give the developer the opportunity to step outside his code and enable code and WSDL construction in parallel, which is much less likely to result in data-type mismatches. Simpler data types are the better choice and REST, using plain text, is perhaps best of all.

Table 7-1: Findings from the Experimental Services

The final chapter draws conclusions from the findings represented in this table and suggests that different styles of web service may coexist and complement each other.

CHAPTER 8: CONCLUSION

8.1 Introduction: the Changing Scene

It is not possible to examine the table given at the close of the previous chapter and still consider that web-service interoperability has been achieved. There is no doubt that it is a work in progress. The evidence drawn together in the previous chapter, validating the concerns raised in Chapter 4, points to the fact that, while interoperability is a widely-held ideal, its realization rests on fragile foundations and also indicates that it may be demolished by such incidental factors as versioning problems or by the inclusion of proprietary extensions. It is also true that small details can create havoc, such as naming capitalization in WebLogic, and how generating JavaBeans from a schema which contains a boolean type can cause an *isXXXX* method to be generated in place of a *getXXXX* method⁴²[[Guest, 2004](#)].

Despite the fragility (and often complexity) of the foundations currently underpinning interoperability, what cannot be ignored is the will to interoperate, evidenced in all the latest toolkits – a phenomenon not seen before in the history of middleware. Although this study was begun with the expectation that SOAP and WSDL might fail in the way that earlier middleware technologies such as CORBA can be said to have done, through being too complex and over-specified, it has ended on a note of surprised optimism and cautious anticipation that cooperation might yet turn web services into something workable.

Implementations now available are making it easier to write web services. The shift in attitude described in this thesis away from the view that web services are a distributed object technology and towards the perception that document exchange is a better model has taken place gradually over the last two years. It should be noted that the author's discussion of the disadvantages of RPC pre-dated this shift. There is no longer such a chasm between the proponents of REST and web-service "evangelists" and it will be interesting to see what further developments will arise from greater emerging support for SOAP 1.2. The writer's conclusions about the benefits of the

⁴² A default code style setting in the 3.1 Eclipse IDE is: "use 'is' prefix for getters that return boolean".

REST approach also pre-dated both the support it now has from the major companies that use it (see page 107) and the more frequent discussion of its advantages in general online forums, as opposed to those in which the original REST/SOAP debate took place⁴³. Late 2005 has also seen developers of, for example, Python and of the Rails framework showing an interest in REST⁴⁴.

The introduction to this study asserted that web services are a rapidly moving target, with implementations, modifications and specification releases following fast on one another. This study has been completed at a significant point for web services. Not only are the vendors of web-service implementations trying to ensure greater interoperability⁴⁵, but also the end of 2005 has seen major new releases of many web-service platforms which offer testing against the Basic Profile and, looking to the future, implementations of both versions of SOAP and, in Axis2, a tentative implementation of REST.

8.2 No Single Solution

Web services promise an ability to deliver the same functionality regardless of platform, language or device, and present a combination of new and legacy code in unexpected ways. Seen this way, it is no wonder that the hype over web services has been so great over the last two or three years. Of course terming it "hype" implies a form of deception or misinformation. It suggests deception surrounding a publicity stunt that does not really warrant the amount of attention generated. The concrete fact about web services is that they are just that: *services*, not meant themselves to be the focus of attention. Shah and Apte were very insistent that web services are a *packaging* technology or a messaging technology (see page 30). The significance

⁴³ A Google search, for *REST 'web services' 2005*, on December 13th, 2005, produced 2,650,000 results. See, for example: <http://www.newsgator.com/forum/shwmessage.aspx?ForumID=8&MessageID=8170>, <http://groups.yahoo.com/group/rest-discuss/message/5065>.

⁴⁴ For Python, see: <http://www.xml.com/lpt/a/2005/08/17/restful-web.html>; and for Rails see <http://www.xml.com/lpt/a/2005/11/02/rest-on-rails.html>.

⁴⁵ Sun and Microsoft have cooperated in the arena of web services more than ever this year and have been official guests at each other's major conferences. A press release on November 4th, 2005, reveals that Sun is planning to create open-source implementations of specifications needed for interoperation with the WCF. This comes hard on the heels of the joint venture to use a single sign-on specification that will work for systems created by both vendors (see http://today.java.net/today/archive/index_11112005.html).

of the service, however, is in the way it may be used to package – or "message" – *anything*, in other words its *interoperability*. A postal system, for instance, that worked in one location but not in another, that conveyed pink parcels but not green ones, that baulked at manila packaging but loved greaseproof paper, that interfaced with courier service X but not Y, would not be very useful. Web services without major interoperability run the risk of not justifying the hype.

Unlike universal postal systems, web services are not really a "one size fits all" solution. Because web services provide an integration technology, they can in fact be mixed with middleware technologies that have preceded them. Lomow and Newcomer explain very clearly that web services should not be viewed merely as a replacement technology:

Web services are not just adding more technology to the problems of IT; they are proposing a different approach to solving some of the problems of IT, especially around integration, because of new capabilities offered by the technology. Web services are not really a replacement technology; ... Web services are not really a new middleware system in the sense that J2EE, CORBA, and the .NET Framework are middleware systems. Web services are XML-based interface technologies; they are not executable; they do not have an execution environment—they depend upon other technologies for their execution environments... Using Web services successfully requires a change in thinking about technology, not simply learning a new grammar for the same old way of building and deploying systems. Web services currently and will always require a mix of technologies. Therefore, Web services need to be understood in terms of what they add to the picture, not only in the context of what they replace [Lomow and Newcomer, 2004].

The conclusion is clear: although web services bring interoperability to distributed computing in ways that earlier technologies could not, they represent not a replacement of the earlier technologies but a paradigm shift.

However, their centrality should not be overestimated. Especially within the enterprise, they are not the only solution for distributed computing. Inside an intranet, where assumptions about language, platform and proprietary mechanisms may be made that would not hold good in a wider context, middleware solutions might perform better, as Table 8-1 suggests. There are

definitely situations in which the use of web-services technology would be inappropriate, e.g. where large numbers of small web-service messages would carry a disproportionately verbose overhead, with multiplied serialization and parsing requirements. A white paper from Intrisync (a vendor with an alternative integration product to sell), published in mid 2005, stated that: "Web Services can introduce known risks in deployment, complexity and scalability. The primary disadvantage is the performance overhead. With applications that have timing requirements and/or large amounts of data to transmit, Web Services is clearly not the best option" [Intrisync, 2005]. If, within an intranet, both systems are running, for example, Java, there would be performance advantages to using RMI with Enterprise JavaBeans, as Table 8-1 demonstrates. Interoperability is usually not a problem for an intranet.

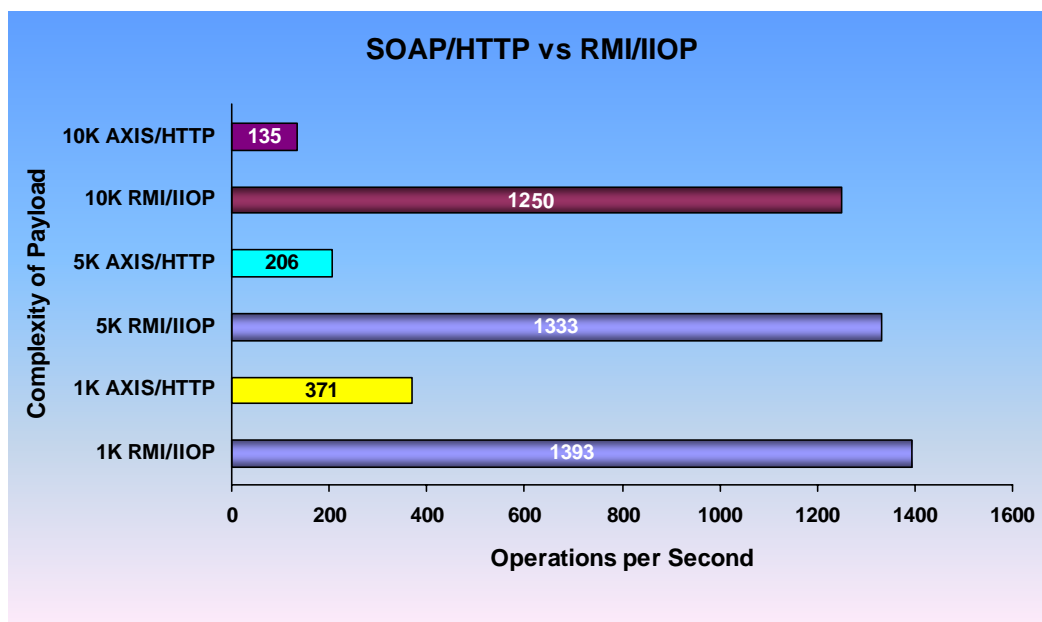


Table 8-1: Performance Differences Between SOAP and RMI [Adapted from [IBM Software Group, 2003](#)]

There are also situations in which simple XML over HTTP, as used in REST, may be all that is required to access the functionality of a fairly simple service and in these cases the extended protocol paraphernalia that has attached itself to web services (as witness the change in the W3C definitions on page 28) would be overkill. The vast majority of *service* applications probably lie somewhere in-between and for them combinations of SOAP and WSDL together with quality-of-service additions such as security and transaction management provide a useful solution.

Such exceptions aside, the paradigm shift represented by web services, as discussed in this study, represents significant potential for the entire IT enterprise; but only if interoperability is assured in future developments.

8.3 Problems Still Needing Solutions

It is just because web services do not present a "one size fits all" solution that the multiple layering of specifications upon specifications can be seen as a drawback to their acceptance and use. This multiple layering might be compared to a badly designed computer language, in which the desire to cover all eventualities swamps comprehensibility. There is a point beyond which adding to the basic functionality of web services can be seen as counter-productive, even indigestible. A greater and still unfulfilled need is for proper standardization at the implementation level. Lomow and Newcomer argue that what damaged CORBA at the outset was a failure to design a standard for interoperability. Irresponsible actions by major companies and consortiums that aim to play one group off against another for their own profit might still cause the web-services initiative to be sidelined into something that "fits" no one, although current vendor cooperation is making this look less likely. If the key to web-service functionality is interoperability, this thesis has shown that it is in everyone's long-term interest to agree upon a set of supporting standards in the same way that agreement has been reached over the core standards of SOAP and WSDL, which are, in fact, standards for interoperability.

Just as it is not possible to foresee every use (or misuse) of an application that will cause it to do unintended things, so it is not possible to say with *absolute* conviction that any particular implementation of a web service is truly interoperable. The WS-I Profile goes a long way towards guaranteeing interoperability and increasingly other testing tools are becoming popular in a way which signals serious problems for those who flout existing standards. A problem inherent in standards such as SOAP and WSDL is their extensibility, a core feature of XML languages. While this feature leaves room for future growth, it must not become the breeding ground for proprietary extensions which are by definition not interoperable.

8.4 Changing Requirements

The enterprise domains in which interoperability is currently required are different from those that were originally and perhaps naively envisaged. Instead of an open market in which services are sought and provided on the basis of a requirements "fit" rather than on source credibility, companies prefer to have dealings with other entities previously known to them. The ideal perpetuated by Sun in its slogan, "The Network is the computer" still exists as an increasingly realizable vision for the future of web services in a more general sense. In this ideal, we all have access not only to more information than we could ever hope to use but to increasing functionality exposed through services which form a web of opportunity and which, to be universally accessible, will need to conform to standards that ensure interoperability.

The corporate world is not the only domain for web services. The vision is potentially much larger. It may be that the human interaction component mentioned in the introduction to this thesis (see page 28) will grow as we live in an increasingly "intelligent" world and the objects we see around us as inanimate communicate in ways that have so far been the province of science fiction but may in the future enlarge our perception of what web services can do.

8.5 Summary of Achievements in this Study

This study has achieved what it set out to accomplish in providing

- a. a survey of factors contributing to or detracting from *interoperability* in web services and
- b. a snapshot of current progress towards achieving it, in terms of available implementations of and approaches to web services, particularly with reference to Java.

The qualitative findings of Chapters 4 and 5 have been validated by the experimental work in Chapter 7.

Failure to achieve interoperability has been shown to be a contributing factor in the relative failures of some middleware technologies. Interoperability is a feature that vendors claim increasingly for their own implementations of *services*, but less attention has been paid to the

significance of interoperability in *client* styles, as presented in Chapters 4 and 7. Chapter 5 was succinct in its presentation of REST, which in its simplicity and efficiency may yet turn out to be the winning approach, particularly if the SOAP stack becomes further enmeshed.

This study has been completed so hard on the heels of new developments in terms of JAX-WS, that it can claim to be one of the earliest studies of these new specifications and implementations. No other *critique* of the new web-services implementations in PHP 5 has been found. While no other web-service implementations in Mozilla's XUL and JavaScript were to be found at the time of the creation of the Firefox extension for the indexing service, this work can be seen to foreshadow the Ajax development that has exploded in the course of 2005.

The study also demonstrates a paradigm shift from the viewpoint encouraged by many implementations that code must be the starting point of a web service, to the understanding that, for interoperability, such a position can be held only by the service contract. The shift was an unforeseen conclusion derived from examining and experimenting with both starting points and is validated not only by an increasing consensus on this matter but also by the end results as discussed throughout Chapter 7. The study offers a new critique of the compromise achieved in this area through the use of Java web-services metadata annotations.

Finally the study clarifies the vexed question of definitions by presenting in section 3.1.3 Newcomer's working definition of web services [2005] which, unlike the latest provided by the W3C, is as applicable to the REST approach as it is to that targeting SOAP and WSDL, and is based on the notion of *interoperability*.

8.6 Future Work

An important extension of this work would focus on REST technologies and their implementations in a variety of languages across a variety of platforms: trends mentioned in section 8.1 suggest potential and exciting growth in that area. Other extensions could include an examination of value-added services on top of the basic SOAP/WSDL design, such as security

and maintaining the integrity of transaction processing. In section 7.1.1.8 it was suggested that there might be potential web-service development with the Ajax technologies, which hold out a promise of asynchrony plus a direct incorporation of XML into JavaScript. Still further extensions might focus on load-testing and on the speed and efficiency of the various approaches to web services, among them the development of asynchronous web services, possibly over messaging protocols.

One extension of particular interest to the writer is the development of web services in mobile form. With the number of mobile phones far exceeding the number of computers on the African continent, mobile computing through web services offers an interesting and relevant challenge in this particular environment.

It is accepted that the feasibility of many of the possible research paths sketched here, involving the uptake and implementation of web services, will depend in part on future implementations and what happens within the current Babel of specifications.

APPENDIX A: List Of Acronyms Used In The Text

ACID	Systems that are atomic, consistent, isolated and durable
AJAX	Asynchronous JavaScript and XML
AXIS	Apache Extensible Interaction System
BPEL	Business Process Execution Language for Web Services
BPML	Business Process Management Language
CICS	Customer Information Control System
COBOL	Common Business-Oriented Language
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DIME	Direct Internet Messaging Extensions
DOM	Document Object Model
E4X	ECMAScript for XML, an extension for JavaScript
EA	Early Access (release)
ESB	Enterprise Service Bus
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IMS	IP Multimedia Subsystems
IRI	Internationalized Resource Identifier
JAXB	Java Architecture for XML Binding
JAX-RPC	Java API for XML-Based RPC
JAX-WS	Java Api for XML-Based Web Services
JB1	Java Business Integration
JCP	Java Community Process
JMS	Java Message Service
JVM	Java Virtual Machine
JSR	Java Specification Request
JWSDP	Java Web Services Developer Pack

MEP	Message Exchange Pattern (in SOAP version 1.2)
MOM	Message-Oriented Middleware
MTOM	SOAP Message Transmission Optimization Mechanism
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open Systems Interconnect
PEAR	The PHP Extension and Application Repository
POA	Portable Object Adapter
RAND	reasonable and non-discriminatory (of patent licensing policies)
RDF	Resource Description Framework
RELAX	Regular Language Description for XML, Next Generation
REST	Representational State Transfer
RFC	Refer for Comments
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAAJ	SOAP with Attachments API for Java
SAX	Serial API for XML
SOA	Service-Oriented Architecture
SOAP	Although originally considered an acronym for Simple Object Access Protocol, its acronymic status was removed in the 1.2 Specification, at which point it became simply a name.
SPEC	Standard Performance Evaluation Corporation
STAX	Streaming API for XML (Parser)
SWA	SOAP with Attachments
TAG	The W3C Technical Architecture Working Group
UDDI	Universal Description, Discovery, and Integration
URI	Uniform Resource Indicator
W3C	World Wide Web Consortium
WCF	Microsoft's Windows Communication Foundation (formerly code-named Indigo)

WSC	Web Services Choreography
WSCF	Web Services Contract First; the acronym is also used as the name for a free tool which may be used with <i>Visual Studio</i>
WSCI	Web Services Choreography Interface
WSDD	(Axis) Web Service Deployment Descriptor – also used as a file extension and usually not capitalized
WSDL	Web Services Description Language
WS-I	Web Services-Interoperability Organization
WSIL	Web Services Inspection Language
WXS	W3C Schema (sometimes also known as "xsd")
XOP	XML-binary Optimized Packaging
XUL	XML User Interface Language, also used as a file extension

APPENDIX B: Creating A *Firefox* Extension

A sequence of files has to be created in a directory structure that looks like this [2004 Duff]:

c:\Firefox Extensions\

+ - indexFile

 + - install.rdf

 + - content

 + - contents.rdf

 + - indexFile-Overlay.xul

The .rdf (*Resource Description Framework*) file tells *Firefox* what the extension is, its identifying package (in our case *indexFile*), its name, its unique id, what versions of *Firefox* will support it and the application (e.g. *Firefox* or *Thunderbird*) inside which it is meant to run. The file also refers to user interface components (known as "chrome") which are described by an interface definition language termed XML User Interface Language (XUL)⁴⁶. A very convenient feature of *Firefox* is that it is possible to modify visible components of the browser, e.g. a toolbar or context menu, without modifying the original configuration files but by adding extra XUL files which the *Firefox* installer can read and interpret. *Firefox* uses a type of zipped file with an .xpi (cross platform installer⁴⁷) extension, which contains files in a particular order e.g. indexFile.xpi will contain at the following levels:

+ - install.rdf

+ - chrome/

 + - indexFile.jar

In its turn the jar file contains the files inside the content directory:

+ - content/

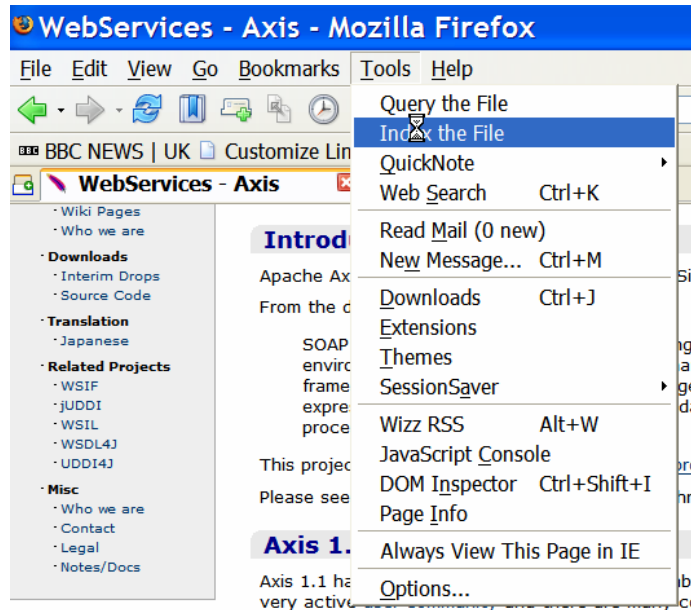
 + - contents.rdf

 + - indexFile-Overlay.xul

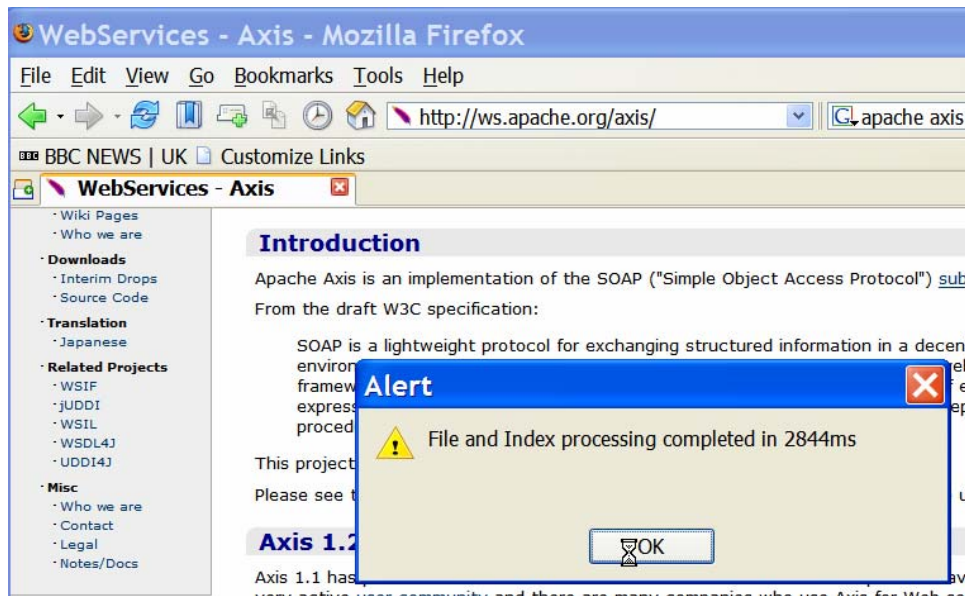
⁴⁶ pronounced *zool*.

⁴⁷ pronounced *zippy*.

The indexFile-Overlay.xul imports a JavaScript file which contains the function to call the web service. An overlay file is, as its name suggests, something that "overlies" the original window or control. It is an addition to the browser (or mail client) but its addition does not involve changing any of the original installation files.



Appendix B-1: Invoking the Indexing Service



Appendix B-2: Message Confirming that the Indexing Service has Finished

APPENDIX C: The Complexity Of Web-Service Specifications **[\[Jeon, c2005\]](#)**

The graphical insert overleaf is Appendix C.

APPENDIX D: The Inner Workings of the Indexing Service

The web service required an index writer (for storing data), an index searcher (for querying data), and an analyzer (responsible for tokenizing the data to be indexed), all of which were provided in the Lucene API. Lucene prevents race conditions by creating two possible "lock" files, one while an index is being written to or deleted from (`write.lock`) and the other when it is being either searched or updated (`commit.lock`). If this latter lock is not released by the closing of the `IndexSearcher`, any process that tries to delete files the `IndexSearcher` has searched (even if the program instance has terminated) will throw an `IOException`. While Lucene gives careful directives about closing the `IndexWriter` instance (no index will be written unless the writer is closed), no such emphasis is given to the closing of the `IndexSearcher`, as was discovered through trial and error.

Part of the analysis of a web page that can be carried out by Lucene involves the exclusion from the index of common words (known as stop words) that will not be required in a search. The `StandardAnalyzer` class contains a brief list of stop words but a simple test was first run to determine the difference of adding to this list. Running the test (which involved creating an index and then running a simple query) on two files without additional stop words took 911 ms and produced an index of 26 KB in size. With an added list of 541 words⁴⁸, the time came down to 832 ms and the index was reduced to 22 KB. Increasing the list to a total of 1762 common words⁴⁹ reduced the time to 812ms and the index size to 19KB, which was considered a worthwhile advantage: a saving of around 14% on index size is not inconsiderable when the file sizes and indices grow. The choice of words on the list was governed by the decision to limit the articles chosen for the test suite to those within the Computer Science subject area. Many common words like "aunt" or "bread", "drink" or "continent" were removed from the list as neither being likely to be found in such articles nor required in a search, and therefore not necessary for inclusion. Of course, if the choice of articles were to be broadened by more

⁴⁸ The list was originally the String Array `SMART_STOP_WORDS` contributed to Lucene by John Caron.

⁴⁹ Additional words were selected from a list of 3000 words in common use in the USA found at: <http://www.paulnoll.com/China/Teach/English-3000-common-words.html>.

general guidelines, such words might need to be included. The stop words were not hard-coded into the service but were read in from a file at runtime.

The index was optimized by supplying the *text* content of the files through a `Reader` (in this case a `FileReader`), which indexes and tokenizes the character content of a file but does not store it. Storing the content would increase the index size. String *fields* were stored instead. Fields which need to be included in a search must be indexed and transformed into tokens, as in a parser. Text analyzers are actually parsers.

The choice of an analyzer is crucial because the analyzer controls how text is tokenized; whether, for instance, words are converted all to lower case; whether terms are made more searchable by having their endings stripped through the use of a stemming algorithm, thus allowing e.g. "lose", "losing" and "lost" to be returned from a single search; and also how term boundaries (e.g. a URL or an email address) are defined.

Snowtide Informatics `PDFTextStream` API was used for converting PDF files to text. This API reads PDF Metadata attributes describing, for example, the title and the date of the file's creation, and also enables the developer to change these attribute names so that they may be given the same names as attributes in text and HTML files. Otherwise, null pointer problems would arise from searching on fields belonging to one file type but not to others included in the same search. Snowtide Informatics were generous in their gift of a free, time-limited academic licence for the project.

APPENDIX E: An Example WSDL File

```

<?xml version="1.0" encoding="UTF-8"?>
<wSDL:definitions targetNamespace="http://newdct"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://newdct"
xmlns:intf="http://newdct" xmlns:tns1="http://newdct"
xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
xmlns:wSDLsoap="http://schemas.xmlsoap.org/wSDL/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wSDL:types>
    <schema elementFormDefault="qualified" targetNamespace="http://newdct"
xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="storeEntry">
        <complexType>
          <sequence>
            <element name="de" type="impl:DictEntryBean"/>
          </sequence>
        </complexType>
      </element>
      <complexType name="DictEntryBean">
        <sequence>
          <element name="headword" nillable="true" type="xsd:string"/>
          <element name="plural" nillable="true" type="xsd:string"/>
          <element name="pos" nillable="true" type="xsd:string"/>
        </sequence>
      </complexType>
      <element name="storeEntryResponse">
        <complexType>
          <sequence>
            <element name="storeEntryReturn" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
    </schema>
    <schema elementFormDefault="qualified" targetNamespace="http://newdct"
xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://newdct"/>
      <element name="storeMultiple">
        <complexType>
          <sequence>
            <element minOccurs="0" maxOccurs="1" name="da"
type="impl:ArrayOfDictEntryBean" />
          </sequence>
        </complexType>
      </element>

```

```

    <complexType name="ArrayOfDictEntryBean">
      <sequence>
        <element minOccurs="0" maxOccurs="unbounded" name="DictEntryBean"
nillable="true" type="impl:DictEntryBean" />
      </sequence>
    </complexType>
    <element name="storeMultipleResponse">
      <complexType>
        <sequence>
          <element name="storeMultipleReturn" type="xsd:string"/>
        </sequence>
      </complexType>
    </element>
    <element name="returnEntry">
      <complexType/>
    </element>
    <element name="returnEntryResponse">
      <complexType>
        <sequence>
          <element name="returnEntryReturn" type="impl:DictEntryBean"/>
        </sequence>
      </complexType>
    </element>
    <element name="returnMultiple">
      <complexType/>
    </element>
    <element name="returnMultipleResponse">
      <complexType>
        <sequence>
          <element minOccurs="0" maxOccurs="1" name="returnMultipleResult"
type="impl:ArrayOfDictEntryBean" />
        </sequence>
      </complexType>
    </element>
  </schema>
</wsdl:types>

<wsdl:message name="returnMultipleResponse">
  <wsdl:part element="tns:returnMultipleResponse" name="parameters"/>
</wsdl:message>

<wsdl:message name="storeEntryResponse">
  <wsdl:part element="impl:storeEntryResponse" name="parameters"/>
</wsdl:message>

<wsdl:message name="storeEntryRequest">

```

```

    <wsdl:part element="impl:storeEntry" name="parameters"/>
</wsdl:message>

<wsdl:message name="storeMultipleResponse">
    <wsdl:part element="tns1:storeMultipleResponse" name="parameters"/>
</wsdl:message>

<wsdl:message name="returnMultipleRequest">
    <wsdl:part element="tns1:returnMultiple" name="parameters"/>
</wsdl:message>

<wsdl:message name="storeMultipleRequest">
    <wsdl:part element="tns1:storeMultiple" name="parameters"/>
</wsdl:message>

<wsdl:message name="returnEntryResponse">
    <wsdl:part element="tns1:returnEntryResponse" name="parameters"/>
</wsdl:message>

<wsdl:message name="returnEntryRequest">
    <wsdl:part element="tns1:returnEntry" name="parameters"/>
</wsdl:message>

<wsdl:portType name="NewDict">
    <wsdl:operation name="storeEntry">
        <wsdl:input message="impl:storeEntryRequest" name="storeEntryRequest"/>
        <wsdl:output message="impl:storeEntryResponse" name="storeEntryResponse"/>
    </wsdl:operation>
    <wsdl:operation name="storeMultiple">
        <wsdl:input message="impl:storeMultipleRequest" name="storeMultipleRequest"/>
        <wsdl:output message="impl:storeMultipleResponse" name="storeMultipleResponse"/>
    </wsdl:operation>
    <wsdl:operation name="returnEntry">
        <wsdl:input message="impl:returnEntryRequest" name="returnEntryRequest"/>
        <wsdl:output message="impl:returnEntryResponse" name="returnEntryResponse"/>
    </wsdl:operation>
    <wsdl:operation name="returnMultiple">
        <wsdl:input message="impl:returnMultipleRequest" name="returnMultipleRequest"/>
        <wsdl:output message="impl:returnMultipleResponse"
name="returnMultipleResponse"/>
    </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="NewDict" type="impl:NewDict">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="storeEntry">

```



```

    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="storeEntryRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="storeEntryResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="storeMultiple">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="storeMultipleRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="storeMultipleResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="returnEntry">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="returnEntryRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="returnEntryResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="returnMultiple">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="returnMultipleRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="returnMultipleResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<wsdl:service name="NewDictService">
  <wsdl:port binding="impl:NewDict" name="NewDict">
    <wsdlsoap:address location="http://localhost:8080/axis/services/NewDict"/>
  </wsdl:port>
</wsdl:service>

</wsdl:definitions>

```

REFERENCES

As mentioned in the *Notes* section to this study, many references were necessarily captured from the web. Web services are evolving and changing so rapidly, with implementations being reworked, that recent developments are to be found only online and items such as specifications, conference proceedings and contributions to some journals are published online and have been captured in that format. With the exception of published books, all of the references below are available on the accompanying CD to which they have been hyperlinked within the text.

- Andrews *et al.*, 2003 Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, Sanjiva Weerawarana, *Business Process Execution Language for Web Services, Version 1.1*, 5 May 2003, available for download as a PDF document from sites such as: <http://dev2dev.bea.com/webservices/BPEL4WS.html>.
- Apte and Mehta, 2002 Naresh Apte, Toral Mehta, *Web Services: A Java Developer's Guide to Using e-Speak* Prentice Hall, 2002.
- Arkin *et al.*, 2002 Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, Stefano Pogliani, Karsten Riemer, Susan Struble, Pal Takacsi-Nagy, Ivana Trickovic, Sinisa Zimek, *Web Service Choreography Interface (WSCI) 1.0*, W3C Note 8 August 2002, available online at: <http://www.w3.org/TR/wsci/>.
- Armstrong *et al.*, 2005 Eric Armstrong, Jennifer Ball, Stephanie Bodoff, Debbie Bode Carson, Ian Evans, Dale Green, Kim Haase, Eric Jendrock, *Basic JMS Concepts*, from Chapter 33 of the J2EE 1.4 Tutorial, available online at: <http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JMS3.html>.
- Australian Government, 2001 Australian Government. Paper written on the Advantages of Paperless Trade, available online at: http://www.dfat.gov.au/publications/paperless/paperless_trading.pdf.
- Axis user-guide, 2005 Axis user-guide.
- Ballinger, 2004 Keith Ballinger, Foreword to Jeffrey Hasan, *Expert Service-Oriented Architecture in C#: Using the Web Services Enhancements 2.0*, Apress, 2004, ISBN: 1-59059-390-1.
- Ballinger *et al.*, 2004 Keith Ballinger, David Ehnebuske, Christopher Ferris, Martin Gudgin, Canyang Kevin Liu, Mark Nottingham and Prasad Yendluri, *Basic*

- Profile Version 1.1*, Final Material, 2004-08-24, available online at <http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html>.
- Barr, 2003 Jeff Barr of Amazon.com in an interview with Doug Kaye, transcript available online at: <http://www.itconversations.com/transcripts/31/transcript-print31-1.html>.
- Barr, 2005 Jeff Barr in an email to the writer in response to a query concerning the relative speed of REST over SOAP.
- Barton *et al.*, 2000 John J. Barton, Satish Thatte, Henrik Frystyk Nielsen, *SOAP Messages with Attachments*, W3C Note 11 December 2000, available online at: <http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211>.
- Bertolino and Polini, 2005 Antonia Bertolino and Andrea Polini, *The Audition Framework for Testing Web Services Interoperability*, Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE 0-7695-2431-1/05.
- Birman, 2004 Kenneth Birman, Like it or not, web services are distributed objects, in *Communications of the ACM*, Volume 47, Number 12, pages 60-62. [Professor, Computer Science, Cornell University; among other positions, that of Editor-in-Chief: ACM Transactions on Computing Systems (1993-1998).]
- Biron and Malhotra, 2004 Paul V. Biron and Ashok Malhotra, *XML Schema Part 2: Datatypes Second Edition*, W3C Recommendation 28 October 2004, available online at: <http://www.w3.org/TR/xmlschema-2/#base64Binary>.
- Booth and Liu, 2005 David Booth and Canyang Kevin Liu, *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. W3C Working Draft 3 August 2005. Available online at: <http://www.w3.org/TR/wsdl20-primer/>.
- Booth *et al.*, 2002 David Booth, Michael Champion, Chris Ferris, Eric Newcomer, David Orchard, *Web Services Architecture Working Draft*, 2002, available online at: <http://dev.w3.org/cvsweb/~checkout~/2002/ws/arch/wsa/wd-wsa-arch-review2.html?rev=1.3&content-type=text/html#N1010B>.
- Booth *et al.*, 2004 David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, David Orchard, *Web Services Architecture* W3C Working Group Note 11 February 2004, available online at: <http://www.w3.org/TR/ws-arch/>.
- Bosak, 2004 Jan Bosak, cited in XML 2004 paper, *UBL: Ready for Prime Time*, available online at:

- <http://www.idealliance.org/proceedings/xml04/papers/27/UBL.html>.
- Bosworth, 2004 Adam Bosworth, blog at:
<http://www.adambosworth.net/archives/000025.html>, September 21st.
- Bosworth, 2004a Adam Bosworth, quoted in Linux World, August 30, available online at:
<http://www.linuxworld.com/story/46025.htm>.
- Box *et al.*, 2000 D. Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer, *Simple Object Access Protocol (SOAP) 1.1*, W3C Note 08 May 2000, available online at: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508>.
- Box, 2004 D. Box, *A Guide to Developing and Running Connected Systems with Indigo*, published on the MSDN website at:
<http://msdn.microsoft.com/longhorn/understanding/pillars/indigo/default.aspx?pull=/msdnmag/issues/04/01/Indigo/default.aspx>.
- Bray *et al.*, 2004 Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, *Extensible Markup Language (XML) 1.0 (Third Edition)* W3C Recommendation 04 February 2004, available online at:
<http://www.w3.org/TR/REC-xml>.
- Bray, 2000 T. Bray, 2000. Mailing on the XMLDev List, 25th July. Available online at: <http://lists.xml.org/archives/xml-dev/200007/msg00789.html>.
- Bray, 2001 T. Bray, contribution to the XML developers mailing list, xml-dev, Mon, 09 Apr 2001 15:32:55 -0700, available online at:
<http://lists.xml.org/archives/xml-dev/200104/msg00205.html>.
- Bray, 2004 T. Bray, blog, available online at:
<http://www.tbray.org/ongoing/When/200x/2004/09/18/WS-Oppo>.
- Bray, 2004a T. Bray, response to a blog entry on MTOM and XOP from Mark Nottingham. Available online at:
<http://www.mnot.net/blog/2004/05/05/boo>.
- Bray, 2004b T. Bray, *WS-Pagecount*, article available online at:
<http://www.tbray.org/ongoing/When/200x/2004/09/21/WS-Research>.
- Bray, 2005 T. Bray, interviewed by Government Computer News, November 13th, 2005, available online at: http://www.gcn.com/24_32/interview/37449-1.html.
- Britton and Bye, 2004 Chris Britton and Peter Bye, *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems*, Addison Wesley

Professional, Second Edition, ISBN: 0-3212-46942. The term *transactional component middleware* was coined in the first edition of this book.

- Burner, 2003 Mike Burner, *The Deliberate Revolution: Transforming Integration With XML Web Services*, ACM Queue vol. 1, no. 1, available online at: <http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=32>. [Burner is a .NET architect at Microsoft.]
- Butek, 2005 Russell Butek, *Which style of WSDL should I use?*, article on IBM Developer Works, 24th May, 2005 (updated), available online at: <http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/>. [Butek is an IBM WebSphere developer and was also one of the original Axis developers.]
- Butek and Scheuerle, 2004 Russell Butek and Richard Scheuerle, article, *Array gotcha—null array vs. empty array*, published on the DevX SkillBuilding, March 9, 2004, available online at: <http://www.devx.com/ibm/Article/20265>.
- Butek and Scheuerle, 2004a Russell Butek and Richard Scheuerle, article on IBM Developer Works, 2nd April, 2004, *Roundtrip issues in Java coding conventions*, available online at: <http://www.ibm.com/developerworks/library/ws-tip-roundtrip2.html>.
- Butek and Scheuerle, 2004b Russell Butek and Richard Scheuerle, article on IBM Developer Works, 18th March, 2004, *Roundtrip issues, an introduction*, available online at: <http://www-128.ibm.com/developerworks/library/ws-tip-roundtrip1.html>.
- Butterfield, 2005 Stewart Butterfield, interviewed by Richard Koman on O'Reilly Network, 2 April, 2005, *Stewart Butterfield on Flickr*, interview available online at: http://www.oreillynet.com/pub/a/network/2005/02/04/sb_flickr.html.
- Cantera, 2004 Michelle Cantera, writing for Gartner, Inc., IT Professional Services Forecast and Trends for Web Services, January 27, available online as *Web Services Testing, The RadView Difference*, at: <http://www.radview.com/products/WebServices.pdf>.
- Cape Clear, 2005 Cape Clear. The test is available online at: <http://www.esbtruthtest.com/>.
- Cape Clear, 2005a Cape Clear SOAEditor Help Files, Overview, available with the editor download.
- Cerami, 2004 Ethan Cerami, *Web Services Essentials* ISBN:0-596-00224-6, O'Reilly

- & Associates.
- Chappell, 2005 David Chappell, Introducing *Indigo: An Early Look*, article on the MSDN, available online at:
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnlong/html/introindigov1-0.asp?frame=true&hidetoc=true>.
- Chiesa, 2005 Dino Chiesa, Interoperability Notes on Apache Axis 1.1 and Microsoft .NET Framework 1.0/1.1 FAQ available online at:
<http://wiki.apache.org/ws/FrontPage/Axis/DotNetInterop>. [Chiesa is a Microsoft product manager for the .Net developer platform.]
- Chinnici *et al.*, 2003 Roberto Chinnici, Maintenance Lead. (Expert Group members: Mark Stewart, Manoj Cheenath, Krishna Sankar, Waqar Sadiq, Kazunori Iwasa, Pankaj Kumar, Russell Butek, Jim Knutson, Miroslav Simek, Dan Kulp, Miles Sabin, Shailesh Bavadekar, Glen Daniels, Steve Jones, Pierre Gauthier, Bjarne Rasmussen, Jeff Mischkinsky, Umit Yalcinalp, Amit Khanna, Dietmar Gaertner, Steve Marx, Prasad Yendluri, Matt Kuntz, James Strachan, Shawn Bayern.) *Java™ API for XML-based RPC, JAX-RPC 1.1*, available from:
<https://sdleweb3a.sun.com/ECom/EComActionServlet/LegalPage::~:com.sun.sunit.sdlc.content.LegalWebPageInfo;jsessionid=5BA0E9D9F0CC8AD2F7E01C6A52A21003;jsessionid=5BA0E9D9F0CC8AD2F7E01C6A52A21003>.
- Chinnici *et al.*, 2005 Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, Sanjiva Weerawarana, *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C Working Draft 3 August 2005, available online at: <http://www.w3.org/TR/wsdl20/>.
- Christensen *et al.*, 2001 Eric Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, *Web Services Description Language (WSDL) 1.1*, W3C Note 15 March 2001, available online at: <http://www.w3.org/TR/wsdl>.
- Columbus, 2005 Louis Columbus, *Cutting Through the Hype: Where Web Services and SOA Are Really Working*, article published on InformIT, September 16th, available online at:
<http://www.informit.com/articles/printerfriendly.asp?p=409137>.
[Columbus is an academic who has written many articles and books on computer science topics.]
- Cowan and Tobin, 2004 John Cowan and Richard Tobin, *XML Information Set (Second Edition)*, W3C Recommendation 4 February 2004, available online at:
<http://www.w3.org/TR/xml-infoset/>.
- Daniels *et al.*, 2004 Steve Graham, Doug Davis, Simeon Simeonov, Glen Daniels, Peter

- Brittenham, Yuichi Nakamura, Paul Fremantle, Dieter Koenig, Claudia Zentner, *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, 2nd Edition*, Sams Developer's Library, ISBN: 0672326418, Chapter 3. Available online to InformIT members at:
<http://www.informit.com/articles/article.asp?p=328640&fl=nl;13;2004-09-07>. (While Graham's name appears first for the printed book, Daniels' comes at the top for the online article.)
- de hÓra, 2002 Bill de hÓra. Correspondence on the xml-dev mailing list, 15th Jan., 2002. [de hÓra is Technical Architect at Propylon, which specializes in web services and XML enterprise systems.]
- Deutsch, 1991 Peter Deutsch (and James Gosling), internally published inside Sun; see JamesGosling, <http://today.java.net/jag/Fallacies/attibution.html>.
- Dochez, 2005 Jerome Dochez, joint-author of a presentation given at JavaOne, 29th June, 2005, and referenced in his blog, 30th June, 2005, *Web Services and vi*, available online at: <http://blogs.sun.com/roller/page/dochez>. A PowerPoint presentation of the talk, *Sun Java™ System Application Server 9.0 and the Java™ Enterprise Edition 5 SDK*, is referenced below. [Dochez is a Senior Staff Engineer in the Application Server group, who works mainly on the Web Services container.]
- Duff, 2004 Brian Duff, *Writing an Extension for Firefox*, 2nd October, blog available at: http://www.orablogs.com/duffblog/archives/2004_10.html.
- Ewald, 2005 Tim Ewald, *Risks to the success of Web services*, blog available online at: <http://pluralsight.com/blogs/tewald/archive/2005/03/04/6336.aspx>. [Ewald is a former .NET developer who now works for MindReef.]
- Fialli and Vajjhala, 2003 Joseph Fialli and Sekhar Vajjhala, *The Java™ Architecture for XML Binding (JAXB)*, Final, V1.0, January 8th, 2003, available online at:
- Fielding and Taylor, 2002 Roy T. Fielding, Richard N. Taylor, Principled Design of the Modern Web Architecture, article, *ACM Transactions on Internet Technology*, Vol. 2, No. 2, May 2002, Pages 115–150, available online at: <http://www.ics.uci.edu/%7Eetaylor/documents/2002-REST-TOIT.pdf>.
- Fielding, 1998 Roy Fielding, abstract for a PowerPoint presentation entitled *Representational State Transfer: An Architectural Style for Distributed Hypermedia Interaction*, (the abstract) available online at: <http://roy.gbiv.com/talks/> and cited on the RestWiki Front Page.
- Fielding, 2000 Roy Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, PhD. Thesis, UCLA, Irvine, Chapter 6.

- Fielding, 2002 Roy Fielding, contribution to the TAG mailing list, in response to Anne Thomas Manes, 23rd April, available online at: <http://lists.w3.org/Archives/Public/www-tag/2002Apr/0235.html>.
- Fry, 2004 Christopher Fry, *Pull Parsing XML*, article on the dev2dev web site, 30th January, available online at: <http://dev2dev.bea.com/pub/a/2004/01/pullparsing.html>.
- Gibbs *et al.*, 2003 K. Gibbs, Brian D Goodman, Elias Torres, *Create Web services using Apache Axis and Castor: How to integrate Axis and Castor in a Document-style Web service client and server*, article on IBM Developer Works site, available online at: <http://www-106.ibm.com/developerworks/webservices/library/ws-castor/>.
- Gray, 2004 N.A.B. Gray, *Comparison of Web Services, Java-RMI, and CORBA service implementations*, paper delivered to the Fifth Australasian Workshop on Software and System Architectures, available online at: <http://mercury.it.swin.edu.au/ctg/AWSA04/Papers/gray.pdf>. [Gray is an Associate Professor of Computer Science at the University of Wollongong, New South Wales, Australia.]
- Gudgin and Hadley, 2005 Martin Gudgin and Marc Hadley, *Web Services Addressing 1.0 - SOAP Binding*, W3C Candidate Recommendation 17 August 2005, available online at: <http://www.w3.org/TR/2005/CR-ws-addr-soap-20050817/>.
- Gudgin and Hadley, 2005a Martin Gudgin and Marc Hadley, *Web Services Addressing 1.0 – Core*, W3C Candidate Recommendation 17 August 2005, available online at: <http://www.w3.org/TR/2005/CR-ws-addr-core-20050817/>.
- Gudgin and Hadley, 2005b Martin Gudgin and Marc Hadley, *Web Services Addressing 1.0 - WSDL Binding*, W3C Working Draft 13 April 2005, available online at: <http://www.w3.org/TR/ws-addr-wsdl/>.
- Gudgin *et al.*, 2003 Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, *SOAP Version 1.2 Part 1: Messaging Framework*, available online at: <http://www.w3.org/TR/soap12-part1/>.
- Gudgin *et al.*, 2003a Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, *SOAP Version 1.2 Part 2: Adjuncts*, available online at: <http://www.w3.org/TR/2003/REC-soap12-part2-20030624/>.
- Gudgin *et al.*, 2005 Martin Gudgin, Noah Mendelsohn, Mark Nottingham, Hervé Ruellan, *SOAP Message Transmission Optimization Mechanism*, available online at: <http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/>.

- Guest, 2003 Simon Guest, *Microsoft® .Net and J2EE Interoperability Toolkit*, published by Microsoft Press, ISBN: 0-7356-1922-0, April 02, 2003.
- Guest, 2004 Simon Guest, article on MSDN entitled: *Web Services Interoperability Guidance (WSIG): IBM WebSphere Application Developer 5.1.2*, available online at:
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/wsinteroprecsibm-final.asp>.
- Guest, 2005 Simon Guest, Video presentation: *Top 10 Tips for Web Services Interoperability*, available online at:
<http://msdn.microsoft.com/msdntv/episode.aspx?xml=episodes/en/20050210WebServicesSG/manifest.xml> (viewable on Internet Explorer only). A (poorly rendered) transcript is available at:
<http://msdn.microsoft.com/msdntv/transcripts/20050210WebServicesSGTranscript.aspx>.
- Haas and Brown, 2004 Hugo Haas and Allen Brown, *Web Services Glossary*, available online at: <http://www.w3.org/TR/ws-gloss/>.
- Haas and Brown, 2004 Hugo Haas, Allen Brown, *Web Services Glossary*, W3C Working Group Note 11 February 2004. Available at:
<http://www.w3.org/TR/ws-gloss/>.
- Hadley and Chinnici, 2005 Marc Hadley & Roberto Chinnici, *The Java API for XML-Based RPC (JAX-RPC) 2.0*, Early Draft, March 15, available as a download from:
<http://jcp.org/aboutJava/communityprocess/edr/jsr224/>.
- Halevey, 2005 Alon Halevey, Why Your Data Won't Mix, *ACM Queue*, October, 2005, available online at:
<http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=336>.
- Hasan, 2004 Jeffrey Hasan, *Expert Service-Oriented Architecture in C#*, Apress, 2004, ISBN: 1-59059-390-1.
- Hatzidakis, 2003 Dennis Hatzidakis, *The Web Services Programming Model*, Presentation at Colorado Software Summit, Oct.26-31, available online at:
<http://www.softwaresummit.com/2003/speakers/HatzidakisJAXRPC.pdf>.
- Hinchcliffe, 2005 Dion Hinchcliffe, *The hidden battle between web services: REST and SOAP*, blog available online at:
<http://hinchcliffe.org/archive/2005/02/12/171.aspx>. [Hinchcliffe co-authored *Software Engineering: A UML Pattern Language*, Sams,

- 2000.]
- Ingalls, 2004 Bruce Ingalls, query on TheServerSide.NET, available online at: <http://www.theserverside.net/tss?service=direct/0/DiscussionThread/threadViewer.markNoisy.link&sp=130330&sp=1148280>.
- Intrinsync, 2005 white paper from Intrinsync, *Interoperability between Java and Microsoft*, available online at: http://wp.bitpipe.com/resource/org_985156716_958/Intrinsync_Whitepaper_031805_In-Network.pdf?track=DED_TSSCOM_5.18.
- Iverson, 2004 Will Iverson, *Real World Web Services*, O'Reilly, 2004, ISBN: 059600642X.
- Jacobs and Walsh, 2004 Ian Jacobs, Norman Walsh, *Architecture of the World Wide Web*, W3C Recommendation 15 December 2004, available online at: <http://www.w3.org/TR/webarch/>.
- JCP, 2005 Home Page of the Java Community Process JBI Specification, available online at: <http://www.jcp.org/en/jsr/detail?id=208>.
- JCP, 2005a JCP. *Java™ Business Integration (JBI)*, Final Approval Ballot, containing the reasons for abstention given by IBM and BEA, available online at: <http://www.jcp.org/en/jsr/results?id=3226>.
- Jeckle *et al.*, 2004 Mario Jeckle, Ingo Melzer, Michael Himsolt, presentation, *Performance of Web Services*, available online at: <http://www.mathematik.uni-ulm.de/sai/ws03/webserv/PerfWS.pdf>.
- Jeon, c2005 Jong-Hong Jeon. *Web-service specifications road-map* (image). Published online at: <http://blog.webservices.or.kr/hollobit/roadmap/ws-specs/map.htm>. [Jeon is a member of the W3C Korean Office staff and a researcher at the Korean Electronics and Telecommunications Research Institute.]
- Jiang and Systä, 2005 Juanjuan Jiang and Tarja Systä, *UML-Based Modeling and Validity Checking of Web Service Descriptions*, Proceedings of the IEEE International Conference on Web Services (ICWS'05) 0-7695-2409-5/05.
- Joseph, 2003 J. Joseph, *A developer's introduction to JAX-RPC, Part 2: Mine the JAX-RPC specification to improve Web service interoperability*, article on IBM Developer Works site, available online at: <http://www-106.ibm.com/developerworks/webservices/library/ws-jaxrpc2/>
- Keahey, 1998 Kate Keahey, *A Brief Tutorial on CORBA*, page available at:

- <http://www.cs.indiana.edu/~kksiazek/tuto.html>. [Currently employed at the Argonne National Laboratory, Keahey was the chair of the Grid Economic Services Architecture (GESA) working group at the Global Grid Forum (GGF) and also a member of the GGF Grid Resource Allocation Protocol (GRAAP) working group, where she co-authored the WS-Agreement draft specification.]
- Kleijnen and Raju, 2003 Stan Kleijnen and Srikanth Raju, An Open Web Services Architecture, *ACM Queue*, Vol.1, Issue 1, available online at: <http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=26>.
- Kohlert, 2005 Douglas Kohlert, announcements page for development of the JAX-WS specifications, available online at: <http://weblogs.java.net/blog/kohlert/>. [Kohlert cites that he is "a staff engineer in the Web Technologies and Standards division of Sun Microsystems where he is the technical lead for JAXRPC".]
- Kohlert *et al.*, 2005 D. Kohlert, K. Kohsuke, ? Mode, E. Pelegri, M. Rameshm, introductory page for the JAX-RPC Reference Implementation Project, available online at: <https://jax-rpc.dev.java.net/>.
- Kumar, Das and Padmanabhuni, 2004 K.M. Senthil Kumar, Akash Saurav Das, Dr. Srinivas Padmanabhuni, *WS-I Basic Profile: A Practitioner's View*, Proceedings of the IEEE International Conference on Web Services (ICWS'04) 0-7695-2167-3/04
- Lauzon and Wagh, 2005 David Lauzon and Shrikant Wagh, Editors, *Attachments Profile [1.0] (with Basic Profile [1.1] and Simple Soap Binding Profile [1.0]) Test Assertions Version 1.0*, Final Material, June 13, 2005, available from the WS-I site at: http://www.ws-i.org/Testing/Tools/2005/06/AP10_BP11_SSBP10_TAD.htm.
- Leavitt, 2004 Neal Leavitt, Are Web Services Finally Ready to Deliver? in *Computer*, IEEE Computer Society, November, available online at: <http://www.computer.org/computer/homepage/1104/technews/index.htm>.
- Lomow and Newcomer, 2004 Greg Lomow, Eric Newcomer, *Understanding SOA with Web Services*, Addison Wesley Professional, 2004, ISBN: 0321180860.
- Loughran and Smith, 2005 Steve Loughran and Edmund Smith, *What's wrong with the Java SOAP API?*, Paper prepared for the 2005 IEEE Conference on Web Services, sent to me by email at my request. (Published with permission as a technical paper for Hewlett Packard under the title: *Rethinking the Java SOAP Stack*, available online at:

- <http://www.hpl.hp.com/techreports/2005/HPL-2005-83.pdf>.)
- Loughran, 2003 Steve Loughran, *The Wondrous Curse of Interoperability*, presentation given to Padnug Org, January, recommended reading on the Axis website, available online at: <http://www.iseran.com/Steve/papers/interop/>.
- MacDonald, 2002 Matthew MacDonald, *Web Services: Objects or XML Endpoints?*, article on O'Reilly ONDotnet.com, available online at: <http://www.ondotnet.com/pub/a/dotnet/2002/09/03/webservices.html>. [MacDonald is the co-author of *Programming .NET Web Services*, O'Reilly, 2002.]
- Manes, 2002 Anne Thomas Manes, contribution to the TAG mailing list, 22nd April, available online at: <http://lists.w3.org/Archives/Public/www-tag/2002Apr/0229.html>.
- Manes, 2003 Anne Thomas Manes, *Web Services: A Manager's Guide*, Addison Wesley Professional, ISBN: 0321185773, Chapter 1.
- Manes, 2004c Anne Thomas Manes, replying to a question in the series *Ask The Web Services Expert* on SearchWebServices.com, available online at: http://searchwebservicestechtarget.com/ateQuestionNResponse/0,2896,25_sid26_cid596451_tax289201,00.html.
- Manes, 2005 2005, blog online at: <http://atmanes.blogspot.com/2005/03/web-services-are-not-distributed.html>.
- Maynard, Charters and Peters, 2004 Caroline Maynard, Graham Charters and Matthew Peters, *Access an enterprise application from a PHP script*, article published on IBM developerWorks, 25th February. Available online at: <http://www-128.ibm.com/developerworks/library/os-phpws/>.
- McMillan, 2003 Robert McMillan, interview with Sam Ruby (former IBM employee and contributor to several open-source projects, including Jakarta Tomcat), published as Web services visionary, on IBM developerWorks, 17th June, available at: <http://www-106.ibm.com/developerworks/webservices/library/ws-samruby.html>.
- Medicke and Pack, 2003 John Medicke and Thomas Pack, *Future-Proofing Solutions with Coarse-Grained Service-Oriented Architecture*, article in the online Web Services Journal, 19th August, available online at: <http://webservicessys-con.com/read/39848.htm>.
- Microsoft, ? Microsoft. The source for this image is Microsoft. It may be found on a number of MSDN European Language sites e.g.

- <http://msdn.microsoft.com/library/default.asp?url=/library/ITA/cpguide/html/cpconalteringsoapmessageusingsoapextensions.asp>.
- Microsoft, 2005 Microsoft. Glossary definition online, available at: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dsml/dsml/glossary.asp>.
- Microsoft, 2003 Microsoft, *How to Apply the Basic Profile*, on the MSDN website, available at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsvcenter/html/wsi-bp_chapter3.asp.
- Microsoft, 2005b Microsoft, *Why Use MTOM?* under the topic MTOM Encoding on the MSDN website available at: http://winfx.msdn.microsoft.com/library/default.asp?url=/library/en-us/indigo_con/html/1243a070-6e5d-4cbc-919c-90727f96ae3.asp.
- Microsoft, 2005c Microsoft. Explanation in the MSDN Library. Available online at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/uddi/uddi/active_directory.asp.
- Mindreef, 2004 Microsoft, *Interoperability: the Key to Quality Web Services*, MindReef whitepaper available at: <http://www.mindreef.com/products/whitepapers/whitepaper-2.html>.
- Mitra, 2003 N. Mitra, *SOAP Version 1.2 Part 0: Primer*, W3C Proposed Recommendation 07 May 2003, available online at: <http://www.w3.org/TR/soap12-part0/>. (Part 0, the *Primer*, was followed by Part 1, the *Messaging Framework* and Part 3, *Adjuncts*.)
- Mitra, 2003 Nilo Mitra, *SOAP Version 1.2 Part 0: Primer*, W3C Recommendation 24 June 2003, available online at: <http://www.w3.org/TR/soap12-part0/>.
- Morgenthal, 2003 J.P. Morgenthal, presentation, available online at: http://seminars.seyboldreports.com/2003_san_francisco/files/i/ix3_ip_morgenthal.ppt.
- Murata, 2001 M. Murata, *XML Media Types*, available online at: <http://www.ietf.org/rfc/rfc3023.txt>.
- Neward, 2005 Ted Neward, *Web + Services*, blog available online at: <http://www.neward.net/ted/weblog/index.jsp?date=20050525#1117011754831>. [Neward is a software architect who has written many books on Java.]
- Newcomer, 2005 Eric Newcomer, *Eric Newcomer's Blog*, 16th June 2005, available

- online at:
http://www.iona.com/blogs/newcomer/archives/2005_06.html.
[Newcomer served on the W3C Web Services Architecture working group and has written many books and articles on web services and SOA. He is also the CTO of Iona, the producers of the web-service toolkit, Celtix.]
- Nottingham, 2004 Mark Nottingham, *XOP and MTOM*, blog, 14th Feb, available online at: <http://www.mnot.net/blog/2004/02/14/xop>. [Nottingham works for BEA and is one of the members of the XML Protocol Working Group that released MTOM and XOP.]
- Obasanjo, 2005 Dare Obasanjo, *Contract-First XML Web Service Design is No Panacea*, blog available online at: <http://www.25hoursaday.com/weblog/CommentView.aspx?guid=e1ab8978-f0a9-4913-bee3-badc1cbefbe5>. [Obasanjo is a prominent web-services developer at Microsoft and was formerly the program manager for XML Schema technologies in the .NET Framework.]
- OED, 2004 draft addition to the OED online, available at: <http://dictionary.oed.com/cgi/entry/50282140>, Draft Editions June 2004, **web n.**
- O'Reilly, 2005 O'Reilly Networks ONJava.com Newsletter, 29th July.
- Panda, 2005 Debu Panda, article on O'Reilly On Java, *Constructing Services with J2EE*, available online at: <http://www.onjava.com/lpt/a/5868>. [Panda is a principal product manager of the Oracle Application Server development team. The Oracle Application Server supports the J2EE 1.4 standard.]
- Pandey, 2005 Dhuru Pandey, *Specification: JSR-000109 Implementing Enterprise Web Services v. 1.2 ("Specification")*, downloadable in PDF format from: <http://jcp.org/aboutJava/communityprocess/maintenance/jsr109/index.html>.
- Pasley, 2001 James Pasley. *Common WSDL Errors* at: <http://www.capescience.com/articles/commonerrors/index.shtml>, particularly the section on declaring arrays.
- Peeters, 2003 Johan Peeters, *WSDL Tales From the Trenches*, Part 3, published online by XML.com at: <http://webservices.xml.com/lpt/a/ws/2003/08/05/wsdl.html>.
- Pelz, 2003 Chris Pelz, *Web services orchestration and choreography*, IEEE Computer, October, 2003.

- Pilgrim, 2004 Mark Pilgrim, article on XML.com, *XML on the Web Has Failed*, available online at: <http://www.xml.com/pub/a/2004/07/21/dive.html>.
- Plummer *et al.*, 2004 D. Plummer, *et al.*, Strategic Planning, SPA-23-4754, *Consider WSDL a Critical Standard*, Gartner Research Note. Source unavailable.
- Ponnekanti and Fox, 2004 Shankar R. Ponnekanti and Armando Fox, Interoperability Among Independently Evolving Web Services, in H.A. Jacobsen (ed.), *Middleware*, 2004, LNCS 3231, Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware, published by Springer-Verlag, New York, Inc.
- Provost, 2003 Will Provost, *WSDL First*, article on O'Reilly XML.com, available online at: <http://webservices.xml.com/lpt/a/ws/2003/07/22/wsdfirst.html>. [Provost is a regular contributor to XML.com and this article was also published by CapeScience at: <http://www.capescience.com/articles/wsdfirst/index.shtml>.]
- Pucella, 2001 Riccardo Pucella, *The π -calculus: A Theory of Mobile Processes*, review. Available online at: <http://www.cs.cornell.edu/riccardo/pdf/sangiorgi-pi-calc.pdf>.
- Raj, 1998 Gopalan Suresh Raj, *A Detailed Comparison of CORBA, DCOM and Java/RMI*, Object Management Group (OMG) whitepaper, available at: <http://my.execpc.com/~gopalan/misc/compare.html>. [Raj is a developer and the author and co-author of several books on distributed and enterprise computing.]
- Rhody, 2004a Sean Rhody, Web Services, Part II: The Search for the Optional Standards in *Web Services Journal*, October 1, article published online at: <http://www.sys-con.com/story/?storyid=46556&DE=1>. [Rhody is the editor-in-chief of Web Services Journal and managing editor of WebLogic Developer's Journal.]
- Rhody, 2004b Sean Rhody, Going the Last Mile, *Web Services Journal*, October 28, article published online at: <http://www.sys-con.com/story/?storyid=46862&DE=1>.
- Rosen *et al.*, 2005 Laurence Rosen and 28 other signatories, *A Call to Action in OASIS*, open letter available online at: <http://perens.com/Articles/OASIS.html>. [Rosen was the first general counsel and secretary of the Open Source Initiative and was formerly its executive directory.]
- Rosenberg, 2003 Doron Rosenberg, *SOAP in Netscape Gecko-based Browsers*, Netscape Devedge, 14 March, available online at:

- <http://devedge.netscape.com/viewsource/2003/soap/01/>.
- Ruby, 2002 Sam Ruby, in:
<http://intertwingly.net/stories/2002/12/17/aBusyDevelopersGuideToWsdll11PartIii.html>.
- Ruby, 2003 Sam Ruby, an IBM developer much involved in open-source web service development and contributor to the Apache AXIS and Jakarta Tomcat projects: see the interview with him at: <http://www-106.ibm.com/developerworks/webservices/library/ws-samruby.html>.
- Ruebush, 2005 Mitch Ruebush, a Microsoft Regional Manager. A curiously disorganized MSDN webcast: *Interoperability Between .NET and Apache Axis*. Available to Microsoft-passport logged-in users on the Microsoft site.
- Seely, 2002 Scott Seely, *Using SOAP Faults*, an article in the MSDN Library, available online at:
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsservice/html/service09172002.asp>. This is only one example of the way in which steps given to handle exceptions do not translate into useful information for those using other toolkits.
- Shah and Apte, 2004a Rajal Shah, Naresh Apte, *Definition and Ownership of Web Service Interfaces: WSDL or XSD?* Prentice Hall Professional Technical Reference, June 4, available online at:
<http://www.phptr.com/articles/article.asp?p=174203>.
- Shah and Apte, 2004b Rajal Shah, Naresh Apte, *Performance and Load-Testing of Axis with Various Web Services Styles*, Prentice Hall, Professional Technical Reference, 16 July. Available at:
<http://www.phptr.com/articles/article.asp?p=177376>.
- Shah and Apte, 2004 Rajal Shah, Naresh Apte, *Web Services: A Business Module Packaging Strategy*, available online at:
<http://www.informit.com/articles/article.asp?p=170421>. [Rajal Shah is an Enterprise Architect at Cisco; Naresh Apte works with mobile solutions at Hewlett Packard and has published extensively in the area of web services.]
- Shirky, 2001 Clay Shirky, *Web Services: It's So Crazy, It Just Might Not Work*, article published on the O'Reilly XML.com, 3rd October, available online at:
<http://webservices.xml.com/lpt/a/ws/2001/10/03/webservices.html>.
- Skonnard, 2004 Aaron Skonnard, MSDN Magazine, November, article *Improving Web*

- Service Interoperability*, available online at:
<http://msdn.microsoft.com/msdnmag/issues/04/11/ServiceStation/>.
- Skonnard, 2005 Aaron Skonnard, *The virtue of contract-first*, blog available online at:
<http://pluralsight.com/blogs/aaron/archive/2005/04/25/7729.aspx>.
[Skonnard is the author of books on XML and a contributing editor to the MSDN magazine.]
- Slama *et al.*, 2004 Dirk Slama, Dirk Krafzig, Karl Banke, *Enterprise SOA: Service-Oriented Architecture Best Practices*, Prentice-Hall PRT, ISBN: 0-1314-65759.
- Srinivasan, 1995 Raj Srinivasan, *Request for Comments 1831, RPC: Remote Procedure Call Protocol Specification Version 2*, available online at:
<http://www.rfc-archive.org/getrfc.php?rfc=1831>.
- Sun Microsystems, 1988 Sun Microsystems, *RPC: Remote Procedure Call Protocol Specification*, available online at: <http://www.rfc-editor.org/rfc/rfc1057.txt>.
- Sun, 2005 *Java API for XML-Based RPC (JAX-RPC)* Sun Developer Network Site page, available online (in May) at:
<http://java.sun.com/xml/jaxrpc/index.jsp>.
- Sun, 2005a Sun, *The Java Web Services Tutorial for Java Web Services Developer Pack, v.1.6*. The tutorial comes bundled with the JWSDP. The tutorial is available online at:
<http://java.sun.com/webservices/downloads/webservicestutorial.html>.
- Sun, 2005b Sun, *JAX-WS 2.0 Early Access User's Guide*, which comes as part of the download bundle but may also be found online at: <https://jax-rpc.dev.java.net/jaxws20-ea2/docs/UsersGuide.html>. [In late November, 2005, this has moved to a public review release.]
- Sun, 2005c Sun, *JAX-WS 2.0 Beta, Provider*, which comes as part of the jaxws-ri download bundle.
- Sun-RMI, 2004 the RMI specification that is bundled with the documentation accompanying the J2SE 1.5, under the following directory structure in a Windows installation: INSTALLDIR/Java/jdk1.5.0/docs/guide/rmi/spec/rmi-intro2.html.
- Systinet, 2005 Systinet Server for Java, 6.0, *Primer*, available from the Systinet site.
- The Open Group, 1997 The Open Group, CAE Specification, DCE 1.1: *Remote Procedure Call*, referenced details available online at:

- http://www.opengroup.org/onlinepubs/9629399/chap2.htm#tagcjh_05_01.
- Thomas *et al.*, 2003 Susan Thomas, Jed Hartman, Judith Radin, Helen Vanderberg, Terry Schultz, Julie Boney, *IRIX Network Programming Guide*, Revision 110, Chapter 4: Introduction to RPC Programming, available online at: http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi/0650/bks/SGI_Developer/books/IRIX_NetPG/sgi_html/c_h04.html.
- Tyagi, 2004a Sameer Tyagi, *Patterns and Strategies for Building Document-Based Web Services*, Sun Developer Network Article, September, page 1 . Available at: <http://java.sun.com/developer/technicalArticles/xml/jaxrpcpatterns/>.
- Tyagi, 2004b Sameer Tyagi, *Lessons from the Front Line - Building Interoperable Web Services*, paper given at XML 2004 (November), and available online at: <http://www.idealliance.org/proceedings/xml04/papers/7/Interoperability.html>. [Tyagi is a senior Java Architect at Sun Microsystems.]
- Yu, Huang and Ye, 2005 Ying Yu, Ning Huang and Ming Ye, *Web Services Interoperability Testing Based on Ontology*, Proceedings of the (IEEE) 2005 Fifth International Conference on Computer and Information Technology (CIT'05) 0-7695-2432-X/05.
- Verner, 2004 Laury Verner, BPM: The Promise and the Challenge, in *ACM Queue*, March, Vol.2, No.1, p.3/9, available online at: <http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=132&page=5>.
- Vinoski, 2002 Steve Vinoski, Web Services Interaction Models, Part 2, Article in *IEEE Internet Computing*, July-August, 2002.
- Vinoski, 2004 Steve Vinoski, *WS-Nonexistent Standards*, IEEE Internet Computing, November-December, 2004.
- Vinoski, 2005 Steve Vinoski, *Middleware Matters – Focus on the Contract*, Blog 10th Feb, available online at: <http://www.iona.com/blogs/vinoski/archives/000141.html>. [Vinoski is the co-author of Addison Wesley Longman's *Advanced CORBA Programming with C++*, and has written numerous articles on Computer Science. He works with Newcomer at Iona and currently serves on the W3C Web Services Architecture Working Group.]
- Vogels, 2003 Werner Vogels, *Web Services are not Distributed Objects*, in IEEE

- Distributed Internet Computing, Nov-Dec issue, viewable online at: <http://weblogs.cs.cornell.edu/AllThingsDistributed/archives/000343.html>. [Formerly a research scientist at Cornell University, Vogels is now Chief Technology Officer at Amazon.com.]
- Wainwright, 2004 Tim Bray quoted by Phil Wainwright in a blog (recommended by Adam Bosworth - <http://www.adambosworth.net/archives/000030.html>) available online at: <http://www.looselycoupled.com/blog/lc00aa00071.html>. The original from Bray himself is no longer available at the source Wainwright cites. [Wainwright is the President of a strategy consultancy and the founder of LooselyCoupled.com, a media site with a theme of service-oriented architecture in the enterprise.]
- Waldo *et al.*, 1994 Jim Waldo, Samuel C. Kendall, Ann Wollrath, Geoff Wyant, *A Note on Distributed Computing*, research report, published by Sun Microsystems and available online at: <http://research.sun.com/techrep/1994/abstract-29.html>.
- Wang and Butek, 2004 Ping Wang, Russell Butek, *Web services programming tips and tricks: Exception Handling with JAX-RPC*, 6 Feb, published online at: <http://www-106.ibm.com/developerworks/xml/library/ws-tip-jaxrpc.html>.
- Weerawarana *et al.*, 2005 Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, Donald F. Ferguson, *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*, Prentice Hall PTR, ISBN: 0-13-148874-0.
- Wilkes, 2004 Lawrence Wilkes, *The Web Services Protocol Stack*, part of the CBDI Web Services Roadmap, available online at: <http://roadmap.cbdiforum.com/reports/protocols/>.
- XMLSOAP, 2002 This document, available online at <http://schemas.xmlsoap.org/wsd/soap12/soap12WSDL.htm>, provides WSDL mappings to SOAP 1.2 as well as the binding schema and an example WSDL. It is one of the documents provided as a link at the SOAP 1.2 namespace address: <http://schemas.xmlsoap.org/wsd/soap12/>.
- Yahoo, 2005 Frequently Asked Questions [concerning its web services] on the Yahoo Developer Network, available online at: <http://developer.yahoo.net/faq/>.
- Zotter, 2005 Zotter (then joint Specification Lead, with an Expert Members Group consisting of: Alexander Aptus, John Bossons, Charles Campbell, Alan Davies, Ted Farrell, John Harby, RajivMordani, Michael Morton,

Simon Nash, Mark Pollack, Srividya Rajagopalan, Krishna Sankar, Manfred Schneider, John Schneider, Kalyan Seshu, Rahul Sharma, Michael Shenfield, Evan Simeone), *Web Services Metadata for the Java™ Platform*, JSR-181 Java Community Process (JCP), Proposed Final Draft Specification, Version 1.0 June 1, 2005. Specification Lead: Brian Zotter of BEA Systems. Available as a downloadable PDF at: <http://jcp.org/aboutJava/communityprocess/final/jsr181/index.html>.

Zur Muehlen, Nickerson and Swenson, 2004

Michael zur Muehlen, Jeffrey V. Nickerson and Keith D. Swenson, *Developing web services choreography standards – the case of REST vs. SOAP*, Decision Support Systems, July, 2005. Available from the Elsevier Science Direct database.

INDEX

- Ajax 128, 174, 175, 176
- annotations.....49, 82, 98, 99, 121, 125, 127, 144, 145, 150, 174
- arrays v, 48, 49, 94, 135, 139, 141, 142, 143, 145, 147, 157, 165, 191, 200
- asynchronous 8, 12, 23, 24, 40, 59, 77, 83, 86, 175
- Axis v, 25, 50, 51, 55, 62, 64, 65, 66, 71, 72, 73, 77, 78, 80, 82, 85, 93, 94, 95, 98, 124, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138, 139, 140, 143, 144, 145, 146, 147, 148, 160, 163, 164, 165, 166, 167, 178, 188, 191, 192, 194, 198, 202
- Axis2..... 154, 169
- binary encoding 18, 38, 69, 70
- binary representation.....4
- binding 10, 33, 39, 47, 48, 61, 62, 64, 67, 68, 71, 73, 77, 89, 90, 92, 94, 95, 99, 106, 109, 135, 152, 156, 161, 162, 167, 205
- C# ii, 4, 7, 18, 37, 49, 50, 94, 95, 111, 127, 131, 132, 134, 136, 138, 139, 140, 141, 142, 143, 145, 151, 152, 159, 160, 161, 163, 164, 166, 188
- choreography 1, 7, 36, 114, 115
- client ...3, 4, 6, 7, 8, 9, 10, 12, 13, 14, 15, 18, 19, 22, 23, 24, 25, 46, 49, 51, 55, 59, 64, 73, 78, 79, 80, 83, 84, 85, 86, 87, 89, 92, 93, 96, 98, 107, 109, 110, 111, 115, 116, 121, 123, 124, 126, 128, 129, 131, 133, 134, 135, 136, 138, 139, 140, 141, 143, 144, 145, 146, 148, 149, 150, 151, 152, 153, 154, 155, 157, 159, 160, 161, 162, 163, 164, 165, 167, 174, 180, 194
- client/server 8, 9, 10, 13, 24
- component 2, 5, 6, 10, 13, 14, 15, 30, 34, 35, 39, 58, 80, 90, 91, 105, 111, 120, 124, 173, 179, 190
- container 15, 39, 49, 124, 193
- CORBA 8, 10, 13, 14, 18, 19, 20, 21, 22, 37, 80, 87, 89, 96, 108, 119, 168, 170, 172, 176, 194, 196, 201, 204
- data-typing.....4, 7, 18, 22, 23, 40, 43, 46, 48, 63, 81, 82, 93, 94, 96, 143, 148
 - data type ...3, 4, 7, 46, 47, 48, 59, 66, 72, 73, 75, 76, 81, 82, 89, 93, 95, 109, 125, 131, 135, 136, 139, 141, 145, 146, 147, 151, 152, 155, 164, 165, 166, 167
- deserialization 3, 7, 14, 66, 78, 79, 81, 146
- distributed computing 1, 5, 8, 9, 15, 23, 28, 43, 49, 120, 121, 170
- distributed object . 12, 13, 15, 17, 18, 23, 24, 26, 35, 52, 80, 86, 94, 108, 110, 111, 121, 168, 189
- document iii, v, 11, 18, 21, 26, 28, 33, 34, 37, 44, 45, 50, 52, 53, 54, 59, 65, 67, 68, 70, 71, 72, 73, 74, 75, 76, 77, 79, 80, 82, 83, 84, 87, 88, 90, 91, 92, 93, 94, 96, 99, 111, 118, 125, 126, 127, 129, 130, 131, 134, 139, 147, 148, 153, 154, 156, 157, 158, 159, 167, 168, 176, 188, 194, 204, 205
- document style..... 84, 194
- document-style 11, 68, 72, 74, 75, 76, 79, 80, 83, 84, 129, 148, 157, 159, 167
- wrapped-style v, 4, 49, 52, 53, 71, 73, 75, 76, 77, 79, 93, 127, 129, 131, 134, 139, 156, 157, 159, 165, 167
- Eclipse v, 54, 55, 93, 146, 147, 168
- eight fallacies of distributed computing49
- encapsulation..... 16, 18, 34, 69
- encoding... 18, 20, 38, 45, 46, 54, 65, 69, 70, 72, 73, 74, 75, 79, 94, 131, 141, 143, 146, 147, 155, 156, 157, 158, 160, 162
- endpoint 10, 15, 34, 59, 78, 90, 92, 125, 126, 148, 150, 151, 163, 164
- Enterprise Service Bus 7, 54, 119, 176
- exceptions..... 33, 43, 50, 51, 114, 127, 131, 132, 137, 138, 139, 144, 147, 159, 160, 161, 172, 188, 202
- Firefox.....ii, 7, 128, 134, 174, 179, 193
- GlassFish.....v, 125, 127, 148, 149, 150
- granularity 16, 17, 64, 83, 120
- HTTP GET..... 65, 106, 130
- HTTP POST 106, 153
- integration .. 1, 3, 7, 15, 17, 37, 38, 42, 43, 72, 93, 116, 119, 120, 121, 122, 170, 171, 194
- interface.... 10, 14, 18, 19, 23, 28, 33, 39, 54, 55, 56, 59, 67, 75, 78, 80, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 97, 101, 107, 108, 121, 125, 126, 144, 145, 150, 151, 152, 163, 164, 170, 179
- interoperability ... ii, 1, 2, 3, 4, 5, 6, 7, 13, 14, 15, 16, 17, 18, 22, 23, 26, 28, 29, 30, 31, 35, 37, 38, 39, 40, 43, 45, 46, 48, 49, 50, 51, 52, 56, 58, 59, 62, 63, 67, 69, 70, 72, 73, 75, 81, 82, 84, 88, 89, 91, 92, 95, 96, 97, 98, 101, 102, 105, 109, 111, 112, 113, 115, 120, 121, 123, 124, 125, 126, 134, 135, 137, 139, 141, 148, 150, 152, 155, 157, 161, 162, 163, 165, 166, 167, 168, 169, 170, 172, 173, 174, 196
- Java ii, xii, 2, 4, 7, 11, 12, 14, 18, 19, 20, 21, 22, 25, 37, 44, 47, 48, 49, 51, 52, 53, 54, 55, 56, 63, 71, 74, 77, 81, 82, 86, 88, 90, 91, 93, 94, 95, 98, 111, 120, 123, 124, 125, 126, 130, 131, 132, 135, 136, 137, 138, 139, 143, 146, 147, 150, 152, 154, 155, 159, 160, 161, 163, 164, 165, 166, 171, 173, 174, 176, 177, 188, 191, 192, 193, 194, 195, 196, 197, 199, 200, 201, 203, 204, 205
- JAX-WS... ii, 4, 7, 11, 18, 44, 50, 54, 55, 59, 70, 71, 80, 86, 90, 91, 125, 126, 149, 150, 151, 152, 160, 174, 176, 197, 203
- jWSOFC..... v, 147
- language-independence 30, 40, 49
- legacy system 15, 16, 17, 25, 42, 80, 83, 110, 118, 159
- loose coupling 10, 16, 20, 21, 23, 24, 26, 35, 39, 40, 64, 83, 84, 86, 94, 101, 102, 113, 117, 120, 149
- loosely coupled..... 24
- machine-readable 29
- MapForce iv, 81, 145, 147
- marshalling..... xiv, 47, 79, 83, 84, 153
- MEP44, 59, 77, 78, 177
- Message-Oriented Middleware8, 23, 24, 25, 36, 177
- messaging... 6, 12, 18, 23, 24, 25, 26, 29, 30, 34, 40, 64, 67, 68, 71, 72, 75, 77, 78, 80, 83, 93, 102, 114, 117, 129, 169, 175
- Microsoft.... 7, 14, 15, 31, 32, 34, 40, 44, 48, 52, 53, 61, 62, 63, 69, 70, 71, 73, 77, 97, 100, 101, 104, 116, 120, 121, 127, 129, 131, 139, 153, 160, 165, 169, 177, 191, 192, 195, 196, 198, 199, 200, 202

- middleware 5, 8, 9, 13, 15, 17, 24, 26, 27, 36, 113, 114, 118, 119, 170, 190
 MOM 8, 23, 24, 25, 36, 177
 MTOM 4, 30, 32, 54, 69, 70, 82, 177, 178, 190, 194, 199, 200
 namespace 43, 47, 66, 72, 73, 79, 82, 90, 91, 92, 97, 99, 103, 140, 144, 152, 162, 165, 167, 205
 Oasis 32
 object orientation . 12, 13, 14, 15, 17, 18, 23, 24, 26, 35, 37, 52, 80, 83, 86, 93, 97, 108, 110, 111, 114, 118, 121, 189
 open source 4, 41, 95, 124, 127, 128, 133, 155, 169, 198, 201, 202
 orchestration 6, 7, 113, 114, 115, 200
 Oxygen 54, 93, 99, 129, 149, 151, 160, 162
 PHP .ii, v, 4, 7, 18, 50, 73, 74, 127, 134, 144, 145, 155, 156, 157, 159, 165, 174, 177, 198
 platform independence 14
 platform-independence 11, 38
 proprietary 3, 4, 14, 16, 23, 26, 30, 35, 51, 67, 69, 70, 71, 89, 100, 101, 119, 124, 166, 168, 170, 172
 REST ii, iii, v, 6, 44, 56, 84, 86, 99, 101, 104, 105, 106, 107, 108, 109, 110, 111, 148, 150, 152, 153, 165, 167, 168, 169, 171, 174, 177, 189, 193, 195
 RMI .. xiii, 8, 10, 14, 18, 19, 20, 21, 22, 49, 50, 82, 84, 126, 149, 164, 171, 177, 194, 201, 203
 RPC .. ii, v, xiii, 8, 10, 11, 12, 15, 16, 19, 23, 24, 25, 44, 45, 50, 52, 54, 59, 63, 65, 68, 71, 72, 73, 74, 75, 76, 77, 78, 80, 81, 82, 83, 84, 86, 93, 99, 107, 111, 121, 124, 125, 127, 128, 129, 130, 131, 134, 139, 148, 150, 156, 158, 159, 160, 163, 167, 168, 176, 177, 192, 195, 196, 197, 203, 204, 205
 JAX-RPC ii, v, 11, 18, 19, 45, 50, 59, 63, 65, 71, 73, 75, 80, 82, 84, 86, 99, 124, 125, 127, 150, 160, 176, 192, 195, 196, 197, 203, 205
 proxy .53, 78, 84, 85, 126, 133, 140, 141, 143, 144, 149, 151, 160, 163, 164
 RPC-style ..11, 52, 54, 75, 78, 80, 82, 83, 111, 134, 139, 158
 stub .10, 18, 19, 78, 84, 85, 126, 160, 161, 163, 164, 165
 schema 3, 4, 16, 43, 47, 48, 52, 55, 59, 64, 66, 67, 72, 73, 74, 75, 81, 83, 84, 87, 91, 93, 94, 95, 97, 98, 109, 121, 125, 126, 130, 131, 135, 137, 141, 143, 144, 146, 147, 148, 149, 152, 153, 156, 157, 162, 165, 168, 205
 serialization 3, 7, 14, 18, 28, 46, 52, 66, 70, 78, 79, 80, 81, 83, 84, 109, 135, 145, 146, 151, 155, 171
 server ..5, 8, 9, 10, 13, 14, 15, 20, 22, 24, 42, 50, 51, 52, 55, 59, 64, 78, 79, 80, 83, 87, 94, 101, 109, 121, 124, 132, 133, 134, 135, 155, 194
 SharpDevelop 4
 SOA ...iv, 6, 7, 28, 34, 35, 36, 37, 43, 53, 54, 55, 56, 86, 93, 99, 102, 113, 119, 120, 123, 131, 139, 157, 162, 177, 188, 192, 197, 198, 199, 203, 205
 SOA Test iv, 7, 93, 99, 123, 131, 139, 157, 162
 SOAP ..v, 1, 3, 4, 6, 7, 21, 26, 28, 30, 32, 34, 35, 40, 42, 43, 44, 45, 46, 49, 50, 51, 52, 53, 54, 55, 58, 59, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 92, 93, 94, 97, 99, 100, 101, 102, 104, 106, 107, 108, 109, 110, 111, 117, 121, 124, 125, 128, 129, 130, 131, 132, 133, 134, 137, 138, 139, 140, 141, 142, 143, 144, 147, 149, 151, 153, 154, 155, 157, 158, 159, 160, 161, 162, 165, 166, 167, 168, 169, 171, 172, 174, 177, 189, 190, 192, 194, 195, 197, 199, 201, 202, 205
 SOAPScope iv, 7, 151
 specifications xii, 1, 3, 10, 14, 26, 30, 31, 37, 38, 43, 44, 47, 52, 55, 56, 59, 61, 62, 63, 67, 68, 69, 70, 71, 73, 77, 80, 87, 88, 90, 91, 94, 99, 100, 101, 102, 104, 105, 106, 109, 111, 116, 117, 118, 120, 122, 124, 126, 144, 153, 154, 169, 172, 174, 175, 188, 196, 197, 203
 standards ...ii, xiii, 1, 3, 4, 6, 7, 9, 11, 14, 26, 28, 30, 31, 33, 35, 38, 40, 41, 43, 52, 53, 54, 55, 63, 67, 98, 101, 102, 104, 113, 115, 118, 121, 124, 127, 172, 173
 state maintenance 16, 18, 20, 21, 22, 45, 102, 117
 Sun ...iv, xiii, 2, 4, 11, 14, 15, 31, 38, 40, 44, 47, 53, 63, 77, 95, 98, 116, 124, 125, 127, 149, 150, 152, 169, 173, 193, 197, 203, 204, 205
 Sun Application Server 125, 127, 149, 150
 synchronous 8, 12, 59, 77, 80, 83, 140
 tight coupling 9, 10, 15, 20, 25, 39, 80, 164
 Tomcat 53, 55, 124, 147, 154, 198, 202
 toolkit ... 1, 3, 6, 8, 13, 17, 40, 43, 44, 47, 48, 49, 50, 51, 52, 53, 54, 55, 67, 71, 72, 78, 80, 85, 94, 96, 97, 98, 104, 109, 119, 121, 124, 135, 153, 155, 202
 toolkits ii, 1, 3, 6, 8, 13, 17, 22, 40, 43, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 59, 67, 71, 72, 77, 78, 80, 85, 93, 94, 95, 96, 97, 98, 104, 109, 111, 119, 121, 124, 125, 127, 129, 135, 139, 140, 144, 145, 146, 150, 151, 153, 155, 159, 160, 163, 166, 168, 178, 199, 202
 transactional component middleware 15, 190
 UDDI 6, 28, 30, 58, 85, 99, 100, 101, 102, 177, 192
 unmarshalling xiv, 47, 153
 URI 19, 29, 44, 70, 78, 91, 106, 107, 110, 177
 Visual Studio 2005 3, 4, 34, 44, 59, 70, 77, 93, 120, 127, 131, 134, 142, 161
 W3C ... 4, 6, 7, 28, 29, 30, 31, 32, 40, 41, 63, 69, 72, 75, 77, 78, 87, 89, 95, 96, 97, 100, 102, 106, 108, 109, 115, 116, 120, 135, 153, 171, 174, 177, 178, 188, 189, 190, 192, 194, 195, 196, 199, 204
 WCF 7, 63, 120, 121, 169, 177, 190, 192
 Web Logic 44, 53, 59, 66, 77, 126, 129, 131, 138, 139, 140, 143, 145, 150, 160, 165, 166, 168, 201
 Web Matrix 4, 127, 159, 163
 web server 4, 9, 22, 55, 59, 101, 109, 124, 132, 133, 134, 155
 WSCF 95, 178
 WSDL xiii, 3, 4, 6, 17, 18, 19, 28, 30, 31, 32, 42, 43, 49, 50, 51, 52, 53, 55, 56, 58, 59, 63, 64, 66, 67, 68, 71, 72, 73, 74, 75, 76, 77, 78, 79, 82, 85, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 104, 115, 116, 117, 118, 121, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 166, 167, 168, 171, 172, 174, 178, 189, 191, 192, 194, 200, 201, 202, 205
 WSDL-First 49, 56, 59, 73, 89, 95, 96, 97, 98, 123, 125, 126, 137, 151, 159, 201
 WS-I Basic Profile ..xiii, 3, 4, 33, 45, 63, 67, 70, 72, 73, 77, 82, 89, 91, 93, 94, 97, 99, 100, 102, 109, 123, 125, 131, 139, 148, 150, 155, 161, 165, 169, 188, 197, 199
 XMLSpy iv, 54, 93, 99, 129, 134, 137, 149, 159, 162