

Dieter Fensel · Holger Lausen
Axel Polleres · Jos de Bruijn
Michael Stollberg · Dumitru Roman
John Domingue

Enabling Semantic Web Services

The Web Service Modeling Ontology

 Springer

Enabling Semantic Web Services

Dieter Fensel · Holger Lausen · Axel Polleres
Jos de Bruijn · Michael Stollberg · Dumitru Roman
John Domingue

Enabling Semantic Web Services

The Web Service Modeling Ontology

With 41 Figures and 2 Tables

 Springer

Dieter Fensel
Holger Lausen
Jos de Bruijn
Michael Stollberg
Dumitru Roman
Technikerstr. 21a
6020 Innsbruck, Austria
dieter.fensel@deri.org
holger.lausen@deri.org
jos.debruijn@deri.org
michael.stollberg@deri.org
dumitru.roman@deri.org

Axel Polleres
Área de Ciencia de la Computación e Inteligencia Artificial
Universidad Rey Juan Carlos
28933 Móstoles (Madrid), España
axel@polleres.net

John Domingue
Knowledge Media Institute
The Open University
Walton Hall
Milton Keynes, MK7 6AA, United Kingdom
j.b.domingue@open.ac.uk

Library of Congress Control Number: 2006932416

ACM Computing Classification (1998): H.4, H.3, D.2, I.2

ISBN-10 3-540-34519-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-34519-0 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable for prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media
springer.com

© Springer-Verlag Berlin Heidelberg 2007

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: by the Authors
Production: LE-TeX Jelonek, Schmidt & Vöckler GbR, Leipzig
Cover: KünkelLopka, Heidelberg

Printed on acid-free paper 45/3100YL - 5 4 3 2 1 0

Preface

Motivation

The constant driving factor behind the development of the Internet from its very beginning has been the combination of distributed data and software applications. The distribution of data has reached unforeseen dimensions with the development of the World Wide Web. On the basis of agreed standards, people are able to share and distribute information in a globally accessible, scalable fashion.

The distribution of applications however, has more complex needs. You need agreed protocols and interfaces between distributed software components and, last but not least the data exchanged by these components must be machine-readable and understandable.

To this end, service-oriented computing has become one of the predominant factors in current IT research and development efforts over the last few years. Standardization in this area has already made its way out of the research labs into industrial strength technologies and tools. Again, Web technologies prove to be a good starting point: *Web services* seem to be the middleware solution of the future for enabling the development of highly interoperable, distributed software solutions: the new technologies subsumed under this common term promise easy application integration by means of languages such as XML, and a common communication platform by relying on widely used Web protocols.

While developments around Web services and service-oriented architectures provide the underlying infrastructure, another field which promises nothing less than the next generation of the Web is gaining momentum: researchers worldwide are currently working on the *Semantic Web*, a Web for machines, where not only is data distributed for human consumption, but also the data on the Web will be machine-processable.

Naturally, these two lines of research fit together; still it seems unclear how to combine Web services and the Semantic Web in the most fruitful way.

However, several promising results from numerous recent EU projects and efforts within the World Wide Web Consortium show the direction.

Goals

The goal of this book is to provide an insight into and an understanding of the problems faced by Web services and service-oriented architectures, as well as the promises and solutions of the Semantic Web. We focus particularly on the Web Service Modeling Ontology (WSMO), which provides a comprehensive conceptual framework for the fruitful combination of Semantic Web technologies and Web services.

With the present book we want to give an overall understanding of the WSMO framework and show how it can be applied to the problems of service-oriented architectures. It is not a ready-to-install “user manual” for Semantic Web services that is provided with this book, but rather an in-depth introduction. While many of the related technologies and standards are still under development we nevertheless think it is not too early for such a book: it is important to create an awareness of this technology and think about it today rather than tomorrow. The technology might not be at an industrial strength maturity yet, but the problems are already.

Intended Audience

This book is aimed at providing beneficial insights to persons with various levels of knowledge. On the one hand, by giving an exhaustive overview of the history and development of the underlying technologies, we aim to guide nonexperts in realizing the potential benefits of Semantic Web services, and to give them a good overview of the field. On the other hand, we provide plenty of detail about the Web Service Modeling Ontology, the state of its realization, its underlying language, and ongoing tool support and implementation efforts. By a thorough analysis of and comparison with all major related approaches, we aim also to give the reader a glance at different ideas, and at the possibility of future convergence of these technologies.

Organisation of this Book

We have divided the book in three main parts: Part I provides an introduction to the field and its history. We cover basic Web technologies, Web services and their predecessors, and the state of research and standardization in the Semantic Web field. Readers familiar with these basics or parts of them, can choose to skip all or parts of this Part.

Part II is dedicated to the realization of Semantic Web services. At the core of this part of the book is a description of the Web Service Modeling Ontology and its language. We shall discuss in detail how WSMO and related technologies aim to address conceptually the problems of service-oriented architectures by exploiting semantic annotations.

Part III is devoted to tools and applications and illustrates the practical developments around WSMO and Semantic Web service technologies in general. As opposed to the more abstract views in Part II, we aim to provide pointers to ready-to-use tools and to interesting prototypes in this part, and hope to encourage many interested readers to exploit and possibly deploy the technologies presented in practice.

Acknowledgments

The work presented in this book has been funded by the European Commission under the SWWS Project (IST-2001-37134), in addition to contributions from several other EU projects, namely, Knowledge Web (FP6-507482), DIP (FP6-507483), and SEKT (IP-2003-506826). The majority of the research that is described in this publication must be accredited to the tireless efforts of the WSMO, WSML, WSMX, and WSMT working groups, to whom we remain gratefully indebted for their valuable discussion and helpful advice. We must also express the same gratitude to several members of the OASIS consortium, particularly the SEE technical committee. Though we are unable to mention so very many whose contributions deserve acknowledgment, this section would be incomplete – as would the respective sections of this book – without recognizing the contributions of our colleagues, Uwe Keller, Mick Kerrigan, Jacek Kopecký, and Barry Norton. Special thanks also goes to Graham Hench for his enduring proofreading and patient editorial efforts. Finally, to all those not mentioned – and to any we may have forgotten we offer our sincerest thank you.

The authors, April 2006

Contents

Part I Foundations

1	Introduction	3
2	The World Wide Web	7
2.1	History	7
2.2	The Building Blocks: URIs, HTTP, and HTML	10
2.3	From HTML to XML	14
2.4	Summary	24
3	The Semantic Web	25
3.1	Ontologies and the Semantic Web	27
3.2	The Resource Description Framework	31
3.3	The Web Ontology Language OWL	33
3.4	Rules for the Semantic Web	34
3.5	Summary	36
4	Web Services	37
4.1	Terminology and Principles	39
4.2	The Origins of Web Services	40
4.3	The Web Service Technology Stack	42
4.4	Web Services in Reality	49
4.5	What's Missing in Web Services?	50
4.6	Summary	53

Part II The Web Service Modeling Ontology

5	Introduction to WSMO	57
5.1	WSMO Design Principles	57
5.2	Top-Level Elements of WSMO	59

5.3	The Language for Defining WSMO	60
6	The Concepts of WSMO	63
6.1	Ontologies	63
6.2	Web Services	67
6.3	Goals	74
6.4	Mediators	75
6.5	Nonfunctional Properties	78
6.6	Summary	81
7	WSML – a Language for WSMO	83
7.1	The WSML Layering	84
7.2	General WSML Syntax	85
7.3	WSML Semantics	93
7.4	WSML Exchange Syntaxes	94
7.5	Key Features of WSML	97
7.6	Relation to RDF(S) and OWL	98
7.7	Summary	99
8	Related Work in the Area of Semantic Web Service Frameworks	101
8.1	OWL-S	101
8.2	SWSF	104
8.3	WSDL-S	107
8.4	Summary	109

Part III Tools and Applications

9	Semantic Web Service Usage Tasks in WSMO	113
9.1	The Virtual Travel Agency Scenario	113
9.2	Discovery	115
9.3	Mediation	124
9.4	Composition	132
9.5	Grounding and Execution	135
10	Tools	141
10.1	Infrastructure	141
10.2	Design Tools	145
10.3	Execution Environments	151
10.4	Summary	156

11 Applications of WSMO 157

 11.1 E-Commerce 157

 11.2 E-Government 165

 11.3 E-Banking 166

 11.4 Summary 168

12 Conclusion and Outlook 169

 12.1 Semantic Web Services Using WSMO 169

 12.2 Standardization Efforts 170

 12.3 Industrial Collaboration 173

 12.4 Alternatives to Classical Web Services 174

References 177

Index 187

List of Figures

2.1	Gopher – the first “net browser”	8
2.2	Mosaic – the first graphical Web browser	10
2.3	HTML document and corresponding layout in a Web browser	13
2.4	An HTML table: tags do not reflect the meaning of the content	15
2.5	An XML tree	17
2.6	XSLT: converting between different XML formats	23
3.1	The Semantic Web language layer cake	26
3.2	Example is-a hierarchy (or taxonomy)	28
3.3	Example RDF graph	31
3.4	Example RDF/XML serialization	32
3.5	RDFS ontology of persons and working group members	32
4.1	Web service architecture	38
4.2	Structures of WSDL 1.1 and WSDL 2.0	45
4.3	The GlobalWeather service viewed in an UDDI browser	48
4.4	The evolution of the Web	50
5.1	WSMO core elements	59
5.2	The relation between WSMO and MOF	60
6.1	WSMO Web service – general description	67
7.1	WSML variants and layering	84
8.1	OWL-S conceptual model	102
8.2	The layered structure of SWSL-Rules [9]	106
8.3	Schematic illustration of how WSDL-S provides links to domain models	107
9.1	Overview of the use of the VTA	114
9.2	The three major processes of heuristic classification	118

9.3	The three major processes in service discovery	119
9.4	Matchmaking notions for semantically enabled discovery	124
9.5	Dimensions of mediation	125
9.6	Process mediation patterns	128
9.7	Example of process-level mediation	129
9.8	WSMO mediator topology	131
9.9	Approaches to data grounding	137
10.1	Architecture of the WSML2Reasoner framework.....	144
10.2	The WSML online reasoning service	145
10.3	The Eclipse platform and the WSMO tools developed on it	146
10.4	The Web Service Modeling Toolkit	148
10.5	Creating mappings in the data mediation plug-in	148
10.6	Concept editor in WSMO Studio	149
10.7	DERI Ontology Management Environment (DOME)	150
10.8	WSMX components	152
11.1	Overview of a VISP	163
11.2	Semantic Web services in e-government	166

Part I

Foundations

Introduction

More than three decades ago, the information revolution started with ARPANet, the so-called grandfather of the Internet, with originally only four connected servers. The idea of developing this network came from the need for scientists to communicate with one another independently of geographical barriers. Just as the telephone allowed them to communicate their words, ARPANet enabled them to freely transfer data or use remote computers. While the Internet grew continuously, several useful developments such as email, FTP, and TELNET took place within ARPANet. However, in addition to such developments, it took another revolutionary idea to cause the real take-off. Only in the early 1990s, when the Internet as such and communication via email, file transfer via FTP, and remote server access via TELNET were already fairly well established in the scientific community, did a young scientist working at CERN spark the next revolution. Tim Berners-Lee combined several innovative ideas into a distributed hypertext system, and provided building blocks such as a simple underlying protocol (HTTP), unique identifiers for linkable information (URIs), and an easy-to-use language (HTML) to create human-readable, interlinked documents accessible all across the Internet. And with these monumental technological foundations the *World Wide Web* was born.

The idea of allowing persistent publication of information on your server, which would be publicly available via an open protocol, combined with the possibility of linking this information arbitrarily, encouraged people to publish enormous amounts of data making the Web the biggest data collection ever: Google alone has more than 9 700 000 000 Web pages indexed at the time of publication, and is, of course, rapidly growing.

Facing this unmanageable amount of data, but also recognizing its big potential, it was again Tim Berners-Lee who coined the name for his vision of the next generation of the Web, the *Semantic Web*. His vision proposed to enrich the human-readable data on the Web with machine-readable annotations thereby allowing the Web to evolve into the world's biggest database: "If HTML and the Web made all the online documents look like one huge

book, [the Semantic Web] will make all the data in the world look like one huge database” [12].

However, making the data on the Web machine-readable is still only half of the story of revealing the full potential of the Web. Within conventional databases, for example, machine-readable annotations usually adhere to completely different schemes and structures which remains the biggest obstacle to automation of data integration tasks. The Semantic Web faces the same problem.

Ontologies, which formally define the structure and meaning of machine-readable metadata while simultaneously providing common, consensual metadata vocabularies to be used on the Semantic Web, will close this gap. To this end, the W3C has developed metadata standards such as the Resource Description Framework (RDF) and the Web Ontology Language (OWL). These languages not only allow us to annotate the data on the Web in a machine-understandable way but also enable additional inferences and allow for the integration of this data via automated reasoning over its ontological structure.

However, we are not at the end of the road, but it is only here where this book hooks in: the original idea of the Internet was about sharing not only static data but also dynamic resources and services such as using procedures and whole applications on remote machines. This idea is also starting to reveal its full potential on the current Web. Humans interact with programs that interact via Web interfaces in order to purchase books, to book flights, to check their current stock quotes, etc. in a much more dynamic fashion than only sharing and requesting static data. Moreover, besides these business-to-customer (B2C) services exposed as Web interfaces on the current Web, there is an even stronger need for easy-to-use, easy-to-integrate services on the business-to-business (B2B) level. In order to achieve full automation of these services we must go beyond the universal database that the current Semantic Web proposes. What we require is more like what could be called a “service-oriented Semantic Web”.

The emerging Web service technologies are the first step in the direction of such service-oriented architectures (SOAs). A unified protocol (SOAP), a common interface description language (WSDL), the de facto standard of a service registry (UDDI), and consistent use of a common XML-based syntax provide the basic building blocks for such an infrastructure. Unfortunately, even with such technologies, the integration of services still depends largely on human experts. Web services as such are limited, in so far as they operate on a merely syntactic level.

The proposed solution towards full automation of service usage, namely facilitating the seamless integration of services that are published and accessible on the Web, is called *Semantic Web services* [90]. Semantic Web services are simply a semantic annotation of the functionalities and interfaces of Web services. In the very same way that ontologies and metadata languages will facilitate the integration of static data on the Web, the annotation of services

will help to facilitate the automation of service discovery, service composition, service contracting, and execution.

Part I of this book is dedicated to introducing the building blocks which constitute Semantic Web services. After giving an introduction to the basic Web technologies in Chapter 2, we present a primer on current Semantic Web standards in Chapter 3. This is followed by an introduction to Web service technologies and their origins in Chapter 4, where we also explain the need for convergence of these technologies towards Semantic Web services.

The chapters in Part I do not aim to give exhaustive details about the technologies discussed, but rather attempt to provide the reader with adequate foundational understanding. The goal is to present the relevant technologies in enough depth to prepare the reader for the later chapters. However, should the explanations seem too brief, or readers' minds be too inquiring, and whenever else deemed necessary, we refer the interested reader to the relevant literature in the respective areas.

The World Wide Web

This chapter will provide a short history of the World Wide Web (Section 2.1) and an introduction to the basic concepts behind the scenes (Section 2.2). Thereafter we shall discuss in Section 2.3 the first step towards a machine-readable language, the eXtensible Markup Language (XML), and its possible impact on the current Web landscape on the one hand and its impact on information integration in general on the other.

2.1 History

Since its early days, the main goal of the Internet was the deployment of a platform for humans to interact and exchange information. Nevertheless, it took sometime before it took off in the way we know and use the Internet today. This volume is not about the basic technologies and protocols which were necessary to build up the necessary infrastructure of the Internet as such,¹ but we start at the stage when technologies that we now consider as the basis of the modern Internet had already begun to spread around the globe, i.e. when the Internet as a medium for communication via email, file transfer via FTP, and remote server access via TELNET were already fairly well established in specific communities. Decades ago, the telephone enabled analog spoken communication across geographic boundaries. The Internet now enabled collaboration and free exchange of digital information with several advantages compared with its analog predecessor. By allowing human users to exchange and communicate via email, it allowed people to communicate information asynchronously through free exchange of messages. That is, unlike telephone communication, information exchange through messages was no longer constrained to take place at the same time. However, the user still had to know very well in advance where to find certain information, and persistent publication or free availability of information on remote servers was not common.

¹ We refer the interested reader to the literature on computer networks, for instance [59, 127], for these fundamental aspects.

Public (anonymous) FTP servers were a first step in this direction. Yet finding and, particularly, referencing information still required human interaction to a large extent. However, starting in the late 1980s and early 1990s, several parallel developments changed this situation. The need for directory services and user-friendly search facilities of the information available on the Internet was recognized and taken into account by what we can – in a sloppy way – call the first “net browser”: Gopher² was the first program that allowed the user to browse and search for data and computational resources and tried to abstract away from cryptic commands, hiding network details completely from the user.

Along with the first menu-driven graphical user interfaces and operating systems, Gopher, introduced in 1991 at the University of Minnesota, provided browsing facilities through hierarchies of application links, files, directories, a phone book server (X.500), graphics, etc. on a remote machine, plus facilities to search indexing servers. Fig. 2.1 shows an “ancient” Gopher browser.

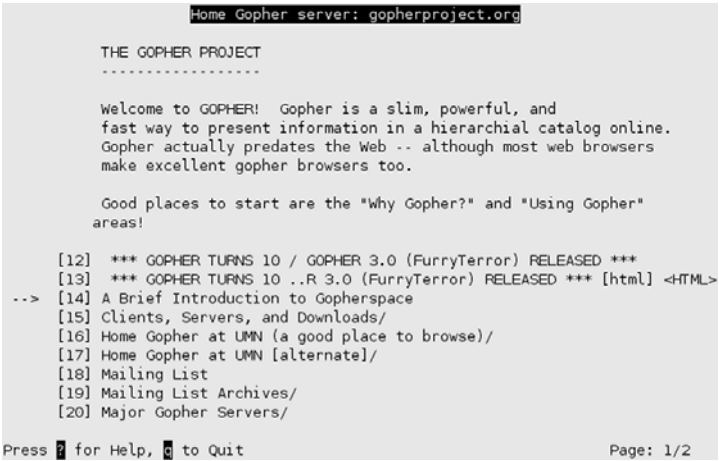


Fig. 2.1. Gopher – the first “net browser”

However, in order for the idea to really take off, another revolutionary idea – or, more precisely, the combination of existing revolutionary ideas – was necessary, and it took some two more years until what we refer to as the “Web” and a “Web browser” were fully developed. The whole idea of the Web dates back to a small application called Enquire. Tim Berners-Lee, then a young researcher at CERN, first developed Enquire, an efficient, easy-to-use information system, mainly for his own use. Enquire³ already used terms such as “universal document identifier” and “hypertext”.

² <http://gopherproject.org/Software/Gopher>.

³ Inspired by the title of a book with the flowery title *Enquire about everything*.

The idea of hypertext, i.e. the idea of documents that contain references that allow one to jump to another text or document, or to a different part of the same document, was not completely new. The concepts can be viewed as dating back to Vannevar Bush's Memex, presented in an article in the *Atlantic Monthly* back in 1945, where he described a vision – though never actually constructed – of a mechanical device that allowed texts in books to be linked with each other and allowed one to navigate via these connections.⁴

Tim Berners-Lee recognized the potential of these appealing ideas used in hypertext for building an easy-to-use, easy-to-extend, easy-to-access global information system: the World Wide Web. The real innovation of Tim Berners-Lee was the combination of three main ingredients to create the world's most successful distributed information system:

- A *standard protocol* for retrieving hypertexts and other documents, accessible over any server on the Internet which supports this protocol (HTTP).
- Each information item has a *globally unique identifier* by which it can be retrieved/dereferenced (the URI).
- A simple format for creating and laying out human-readable, interlinked hypertext documents, refined to include the possibility to include graphics, and extensible to other multimedia formats. (HTML)

These ingredients allowed a global network of interlinked information items published over the Internet, to grow in a constantly expanding way, with over 9 700 000 000 Web pages indexed by Google alone at the time of writing of this book.

The first Web page was created in 1990 by Tim Berners-Lee himself on a CERN server.⁵ The first graphical Web browser, Mosaic (Fig. 2.2), was developed by the NCSA (National Center for Supercomputing Applications) and entered the market in 1993.

In May 1994, the first International Conference on the World Wide Web led to the foundation and first meeting of the World Wide Web Consortium (W3C) later that year, in December. Since then, the W3C has set up many important standard recommendations (e.g. HTML, XML, RDF, and OWL) – some of which we shall revisit in the course of this chapter – in the hope of steering the development of the Web towards its next generation.

In the following section we shall take a closer look at the fundamental building blocks of the Web.

⁴ The term “hypertext” itself, though, was coined later in the 1960s by another IT pioneer, Ted Nelson, in the Xanadu project.

⁵ A copy of this page is still available at <http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html>.

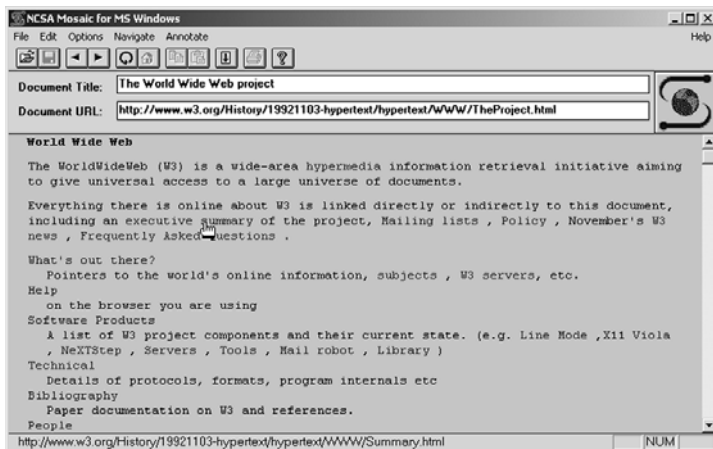


Fig. 2.2. Mosaic – the first graphical Web browser

2.2 The Building Blocks: URIs, HTTP, and HTML

The success of the early Web architecture was based on three main components: Uniform Resource Identifiers (URIs), the Hypertext Transfer Protocol (HTTP), and the Hypertext Markup Language (HTML). We shall now briefly describe these ingredients in Sections 2.2.1–2.2.3, after which we shall discuss some limitations on the automation and flexibility of the basic Web infrastructure, particularly with respect to HTML. These limitations have led to the development of the more general eXtensible Markup Language (XML), presented in more detail in Section 2.3.

2.2.1 Unique Identifiers for the Web – URIs and IRIs

Uniform Resource Identifiers (URIs) mark the first building block of the Web, following a very simple but effective principle: global naming leads to global network effects. First defined in RFC⁶ 1630 in June 1994 as “Universal Resource Identifiers”, URIs were originally distinguished between Uniform Resource Locators (URLs) and Uniform Resource Names (URNs). URLs were conceived as addresses of a dereferenceable resource, whereas URNs are not necessarily dereferenceable.

A simplified version of the URI syntax looks as follows:

$$scheme : [//authority][/path][?query][#fragid]$$

⁶ The RFCs (Requests for Comments) published by the Internet Engineering Task Force (IETF) play an important role in the Internet standardization process. They include mainly standards on a more fundamental technical level than what the W3C deals with. For more details see <http://www.ietf.org>.

Here, *scheme* denotes, for instance (but not necessarily), a protocol over which the respective resource is accessible. Nowadays, the `http://` scheme is most commonly used. The owning entity of the resource is denoted by *authority*. In most cases this is a server or domain name which has the practical side effect that domains indeed belong to legal entities. Explaining the details of the *query* and *fragid* (fragment identifiers) parts would go beyond the scope of this book, but most readers will have experienced them already in the common use of HTTP. For example, when a search engine is accessed, the query for a keyword is usually coded in the URI, such as `http://www.google.com/search?q=wsmo`; this URI encodes a query for the search string `wsmo` at Google. Fragment identifiers denote a secondary resource with no direct path where the interpretation of fragment IDs depends on the type of medium. For instance, the usual use in HTML denotes an anchor within a hypertext document. URIs without scheme parts are called relative URIs, where “relative” means that these URIs are understood relative to a base URI. What this base URI is or defaults to depends on the particular application of the URI; in HTML, for instance, it is the document URI, where the document is accessible on the Web.

Note that we learn from the now-deprecated distinction between URNs and URLs that although URIs/IRIs identify resources, it does not follow that any URI is a deferencable, *web-accessible* resource. As we shall see in later chapters, the principle of unique identification via globally unique identifiers can and is used in ways other than dereferencing Web resources accessible over a particular protocol on a particular server. Still, it is often useful to take advantage of some of the useful characteristics of URIs (such as ownership by an authority by the use of domains) for unique identification of resources other than documents on the Web. Usual uses of URIs include:

- addresses on the Web (which include documents, service endpoints, etc.);
- namespaces in XML QNames or other languages (see Section 2.3);
- identifiers of things and concepts (e.g. RDF, see Section 3.2);
- unique keys (e.g. MIME message ID).

The specification of URIs reached its current state in RFC 3986 in January 2005. IRIs (International Resource Identifiers) are a recent extension of URIs which allow a richer character set, i.e. full Unicode, as defined in RFC 3987.

2.2.2 A Protocol for Hypertext Transfer – HTTP

The Hypertext Transfer Protocol (HTTP) is a simple protocol that provides all necessary means for simple client/server interaction in a stateless request–response manner. Such a manner allows a client to open the connection with a request, whereafter the server responds and closes the connection again. HTTP is the protocol of choice for most Web-based communication for delivering files and other data (collectively called resources) on the World Wide

Web. A resource is generally understood as a piece of information that can be identified by a dereferenceable URI.

Request messages in HTTP consist of a method invocation, some header lines and an optional message body. The basic method of requesting a resource from an HTTP server is HTTP GET. For instance, when you type the URI `http://www.wsmo.org` in your browser, it will initiate a GET request message of the form

```

      request method  path to resource  HTTP version
      {      }      {      }      {      }
      GET           /           HTTP/1.1
Host: www.wsmo.org

```

The second line (`Host: ...`) is the header information for the GET request. GET requests always have an empty message body. Apart from GET, some other important request methods are:

- HEAD: returns only the headers of what GET would return (useful for testing);
- PUT: to replace or create a resource;
- DELETE: to remove a resource;
- POST: to submit data to a resource for processing, or to manipulate another resource.

An HTTP response in turn contains again the HTTP version, a status code and, analogously to the request, headers and a message body which, in the case of a website request, would be the respective HTML document, e.g.

```

HTTP/1.1 200 OK
Date: Fri, 10 Feb 2006 16:05:38 GMT
Content-Length: 1000
Content-Type: text/html

Connection: close
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head>
:

```

For further details on the two most commonly used versions of HTTP, we refer to the respective latest RFCs 1945 (HTTP 1.0) and 2616 (HTTP 1.1). The differences between these versions include, among other things, cache control or keeping the socket connection alive for reuse after a response.

2.2.3 Hypertext for the Masses – HTML

In his quest for an easy-to-use layout language for the Web, Tim Berners-Lee decided to base it on an application of Standard Generalized Markup Language (SGML). SGML allows the definition of markup languages using a format of text structured using tags delimited by `<>`, of the following form:

`<tagname>` plain text or nested tags `</tagname>`

Tags distinguish between an opening tag `<tagname>` and a closing tag `</tagname>`.⁷ Attributes can be attached to opening tags in the form `<tagname attributename1="value" . . .attributenamen="value">`. The allowed tags and attributes for a particular SGML application such as HTML are defined by document type definitions (DTDs).

The DTD for HTML defines tags and attributes for layout and for interlinking text and multimedia documents. So, HTML is a description language for the layout of documents, tailored for display in Web browsers, which provides a portable format independent of the actual rendering in a specific browser or other application. The general structure of an HTML document is shown by a small example in Fig. 2.3.

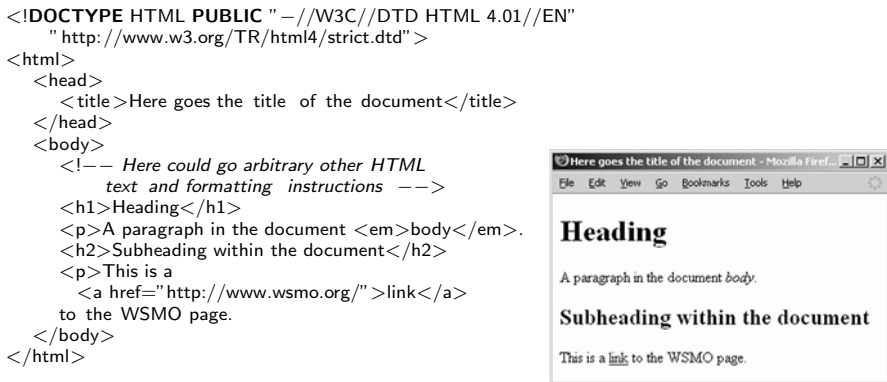


Fig. 2.3. HTML document and corresponding layout in a Web browser

HTML and most browsers are tolerant of minor, sloppy mistakes such as omitting closing tags. For example, the closing paragraph tag `<P>` in Fig. 2.3 is optional in HTML. Browsers often accept even invalid HTML and will try to render it, although the W3C highly recommends that one should stick with the standards when authoring an HTML page.

Basic features of HTML comprise:

- linking to other documents or parts of documents;
- structuring text (lists, tables, various levels of headings, etc.);
- formatting text (italic/boldface/underlined, etc.);
- adding graphics to the document;
- creating simple user interfaces using forms (including radio buttons, text fields, check boxes, etc.).

⁷ Note that in HTML it is not necessary that every tag must be closed: for instance, `
`, denoting a new line.

HTML can define links using absolute and relative URIs (see Section 2.2.1 above) using the document location as the base URI. Using the above-mentioned fragment identifier, links to different parts of a document can also be defined. Links are created using an anchor tag `<a>` which specifies the location of the link in the `href` attribute, for example, `link` in Fig. 2.3.

This simple mechanism of linking from one HTML page to another has been one of the success factors of the current Web. Such links enable crawlers to find and index pages, an essential prerequisite for efficient searching. Moreover a link also contains semantic information: if page *A* links to page *B* this means that page *B* supposedly has some valuable information. Search engines such as Google use this information to rank their search results.

The ease of use and the availability of many WYSIWYG editors for HTML and cascading style sheets (CSS) have made it possible for almost anyone to publish information on the Web. This active, widespread participation has contributed significantly to today's success of the Web. Apart from standardization of basic features by the W3C, HTML has been extended to include support for scripting languages, such as JavaScript, developed by Sun Microsystems, multimedia formats, such as Macromedia Flash, developed and distributed by Adobe Systems (formerly by Macromedia), and other vendor-specific extensions which are widely supported by modern browsers, either natively or via plug-in mechanisms. Nevertheless, despite many vendor-specific extensions and widely adopted features, the W3C is trying to keep standard HTML within controlled bounds in order to enable widest possible accessibility and interoperability. To this end, the W3C has published Web Content Accessibility Guidelines,⁸ which provide several levels of conformance in order to guarantee accessibility of Web content.

From its first standardized version 2.0 described in RFC 1866 of November 1995, HTML evolved to the last version of "classic" HTML, version 4.1, published as a W3C Recommendation on December 24, 1999. From that time on, the further evolution of classic HTML was frozen in favor of XHTML, the XML version of HTML to be discussed in the next section.

The need for separation of content and layout, just as in templates in usual word processors, led to extension of the basic HTML standard by cascading style sheets, where the word "cascading" indicates that one style sheet can inherit from another, permitting the combination of several stylistic preferences in one document.

2.3 From HTML to XML

The freedom of HTML has its drawbacks. First, although CSS take the first step towards separation of content and layout, the tags and attributes in

⁸ <http://www.w3.org/TR/WAI-WEBCONTENT>.

HTML are merely designed to convey layout information for a browser. To be precise, there are a few exceptions such as the <META> tag, which allows some limited form of meta-information to be added within HTML documents. Nevertheless, the language is mainly tailored for human consumption and does not impose any strict rules for facilitating machine readability. However, the increase in the number of dynamically generated Web pages and the sheer amount of information available on the Web have created a strong need for automatically processing Web content.

Via the detour of HTML, much of the structural information conveyed in a Web page gets lost. Take, for instance, the table in the HTML document in Fig. 2.4 which shows some members of the WSMO Working Group and their affiliations. The human reader can easily see that the rows of the table

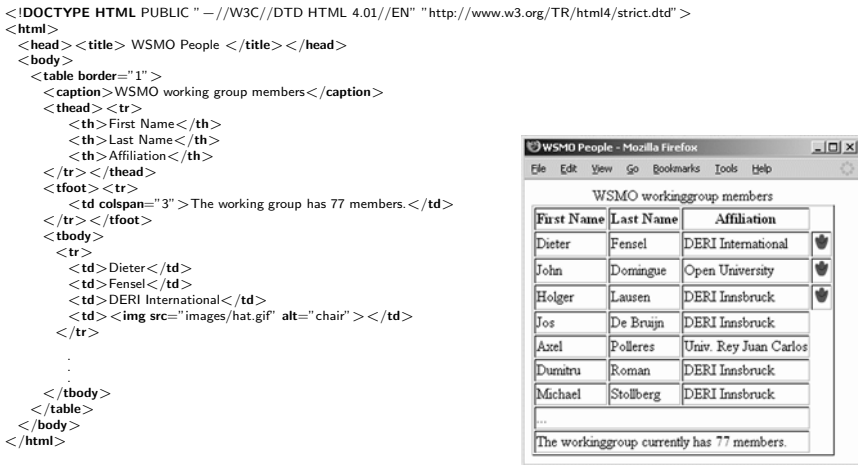


Fig. 2.4. An HTML table: tags do not reflect the meaning of the content

body contain the first names, last names, and affiliations of WSMO Working Group members. The table footer gives information about the total number of Working Group members. However, the HTML tags themselves do not give any hint towards this conclusion.

If, however, we could use arbitrary SGML tags, we could indicate the meaning of particular data items directly within the document source, thereby allowing tasks such as ordering by last name or computing the total number automatically, using simple script languages. So, what was needed was a general markup language tailored for the Web; to this end, the eXtensible Markup Language (XML) was developed.

XML, in short, is a restricted version of SGML which imposes certain restrictions (e.g. tag names disallow certain symbols, all tags have to be closed, and names are case-sensitive). These restrictions were chosen, in order to ease

```

<?xml version="1.0"?> <!DOCTYPE people SYSTEM
" http://www.wsmo.org/workinggroup.dtd" > <!-- This XML document gives
information about working group members of the WSMO working group
--> <people xmlns="http://www.wsmo.org/namespace" >
  <title>WSMO working group members</title>
  <member chair="yes" >
    <firstname>Dieter</firstname><lastname>Fensel</lastname>
    <affiliation >DERI International</affiliation >
  </member>
  <member chair="yes" >
    <firstname>John</firstname><lastname>Domingue</lastname>
    <affiliation >Open University</affiliation >
  </member>
  <member >
    <firstname>Axel</firstname><lastname>Polleres</lastname>
    <affiliation >Univ. Rey Juan Carlos</affiliation >
  </member>
  :
</people>

```

Listing 2.1. An XML document

parsing and have made XML a widely accepted interchange format for various (semi-)structured data over the last few years. A possible XML version of the example in Fig. 2.4 is shown in Listing 2.1

Obviously, designated tag names and attributes in XML facilitate easier processing of the content by a machine. XML allows the application designer to design his/her own language with specific tags and attributes, leaving almost all the freedom of SGML. The specific elements and attributes used are, just as in SGML, declared in a DTD.

However, how can a Web browser, for instance, know how to display the content in an XML document? More generally, how can machines using different tags and attributes still exchange data? To solve this, XML comes with several accompanying standards for querying XML documents (XPath and XQuery) and translating between different XML formats (XSLT). Using these mechanisms, the document in Listing 2.1 can, for instance, be transformed to XHTML, the XML version of HTML, which can be displayed by Web browsers.

In the following, we shall briefly outline some of the essentials of XML, such as namespaces and DTDs, and come to the conclusion that DTDs are sometimes insufficient for describing the structure of the data in an XML document. To overcome this, another standard for defining the structure of XML documents, XML Schema, has been defined. After taking a look at XML Schema, we shall also briefly introduce XPath and XSLT to familiarize the reader with the fundamental ideas. We shall close this section and chapter with a discussion of the outlook for XML applications and some bridging remarks concerning the Semantic Web technologies described in more detail in the next chapter.

2.3.1 XML Basics

Let us return to the simple XML file in Listing 2.1 to briefly explain the basic structure of an XML document.

The first part of the document, called the *prolog*, consists of an XML declaration which denotes the XML version, and optionally the character encoding, etc., plus an optional document type definition. This DTD can be defined externally (as in the example), or within the document itself. Immediately after the prolog, the XML document starts with the designated document element (the root). Each XML element is delimited by start and end tags and can contain one or more subelements, text, or a mixture of both text and elements. Each element must end with a closing tag (an element without content can be abbreviated to `<tagname/>` instead of writing `<tagname></tagname>`); attribute values must be quoted and must not appear in end tags, and elements may not overlap. That is more or less about it. Any document following this syntactical structure is considered a *well-formed* XML document. The information contained in a well-formed XML document can be viewed as a *tree*, having the document element as its root, as shown in Fig. 2.5.

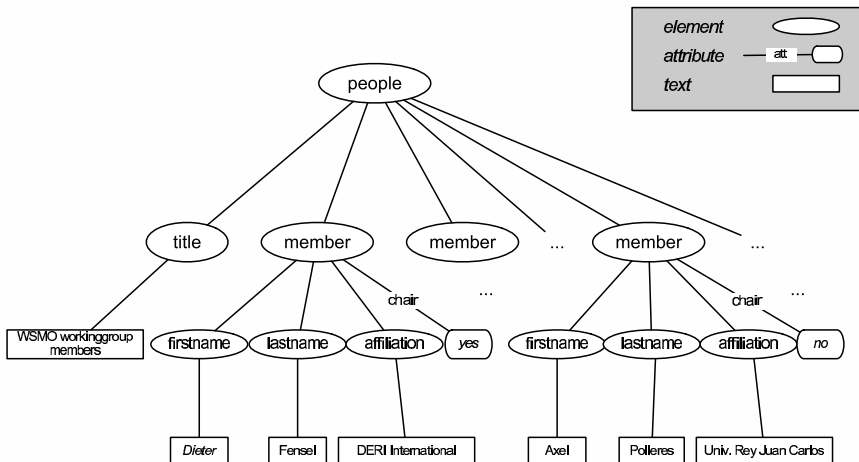


Fig. 2.5. An XML tree

Furthermore, XML documents which obey the rules for usable tag names and attributes defined in an associated DTD (or other schema definition such as XML Schema, see below) are called *valid*.

Document Type Definitions in XML

As we have mentioned DTDs several times already in the context of SGML and XML, it is worthwhile to illustrate DTDs with a small example. As already

mentioned, DTDs provide a basic mechanism for defining an XML language, i.e. the usable tags and attributes in a specific document obeying a particular DTD, and which elements may appear as subelements of another, etc. Listing 2.2 shows the DTD `workinggroup.dtd`, which defines the elements and attributes used in our running example.

```

<!DOCTYPE people [
  <!ELEMENT people (title,member+)>
  <!ELEMENT member (firstname,lastname,affiliation+)>
  <!ATTLIST member chair (yes|no) "no">
  <!ELEMENT title (PCDATA)>
  <!ELEMENT firstname (\#PCDATA)>
  <!ELEMENT lastname (\#PCDATA)>
  <!ELEMENT affiliation (\#PCDATA)>
]>

```

Listing 2.2. DTDs define the allowed tags and attributes in an XML file

DTDs allow one to define the structure of elements by regular expressions using the usual symbols ‘?’ ‘*’, ‘+’, and ‘—’ to denote optional, arbitrary, at least one, or alternative occurrences, respectively, of elements. Attributes can be assigned a default value and can be set as required or optional. For text elements, DTDs do not provide specific datatypes but only the generic PC-DATA (parseable character data). For attributes, there are several possible types apart from the generic character data (CDATA), such as IDs (which need to be unique within a document), and NMTOKENS (essentially, allowed XML attribute and element names). In addition, DTDs allow one to define commonly used macros as ENTITIES. The well-known `>` which is replaced by the greater-than sign ‘>’ within the text in an HTML document is an example for such a macro, and is defined in the HTML DTD as follows:

```

<!ENTITY gt CDATA "&\#62;">

```

Obeying to a common schema such as that provided by a DTD (e.g. HTML documents obey the HTML DTD) allows software to exchange and process the corresponding documents. The same holds for exchange formats in XML defined by common usage of a shared DTD. However, DTDs, which originate from SGML, have several drawbacks with respect to expressivity. In the context of XML, these have been overcome by another schema definition language, namely XML Schema, which we shall cover in Section 2.3.2.

Namespaces

The `xmlns` attribute in the document element of the XML document in Listing 2.1 plays a special role. This attribute assigns a *namespace* URI to the element. Unlike other attributes which can be freely defined within DTDs (or in XML Schema as below), the namespace attribute has a special meaning in XML. When given, it defines the namespace for all the attributes and tags within its scope, which means the element, where it is defined, all its attributes and subelements, and the attributes of subelements. Namespaces

serve the purpose of disambiguating tags by reusing the fundamental idea of URIs once again: Global naming allows universal IDs for tag and attribute names.

If you consider our example above, a mathematician who did not know about our WSMO Working Group DTD might easily come up with another XML document if she wanted to represent the set $\{a, b, c\}$, using the tag name `member` ambiguously, with a completely different meaning:

```
<?xml version="1.0"?>
<set xmlns="http://www.example.org/mathstuff/sets/" >
  <member>a</member>
  <member>b</member>
  <member>c</member>
</set>
```

Fortunately, by declaring different namespaces, an XML parser will recognize that the `member` in Listing 2.1 is something different from that in this short example. Different namespaces can also be mixed within one document, since each tag or attribute is uniquely identified by a *qualified name*, which is obtained from the combination of the namespace plus the tag/attribute name. However, if different namespaces are used within one XML document, they must be disambiguated by using different prefixes. Assuming that we wanted to reuse the set notation of our unknown mathematician for people with several affiliations among the WSMO Working Group members, we could come up with a modified XML document as in Listing 2.1.

```
<?xml version="1.0"?>
<people xmlns="http://www.wsmo.org/namespace"
  xmlns:math="http://www.example.org/mathstuff/sets/" >
  <title>WSMO working group members</title>
  <member chair="yes" >
    <firstname>Dieter</firstname><lastname>Fensel</lastname>
    <affiliation ><math:set>
      <math:member>DERI International</math:member>
      <math:member>DERI Innsbruck</math:member>
      <math:member>DERI Galway</math:member>
    </math:set></affiliation>
  </member>
  <member chair="yes" >
    <firstname>John</firstname><lastname>Domingue</lastname>
    <affiliation >Open University</affiliation >
  </member>
  :
  :
  <member chair="yes" >
    <firstname>Axel</firstname><lastname>Polleres</lastname>
    <affiliation ><math:set>
      <math:member>Univ. Rey Juan Carlos</math:member>
      <math:member>DERI Innsbruck</math:member>
    </math:set></affiliation>
  </member>
  :
  :
</people>
```

Listing 2.3. Alternative XML format for Listing 2.1

Here, the XML processor has no difficulty in disambiguating different `member` tags, since they are all uniquely qualified by namespaces; for unprefix tags and attributes, the default namespace `http://www.wsmo.org/namespace` is used, whereas the prefix `math` is used to point to the namespace `http://www.example.org/mathstuff/sets/`.

We can see from this simple example that namespaces and unique identification are crucial underpinnings for combining and exchanging XML data in a Web context or in other open environments where the partner is not known upfront.

As a side remark, note that namespace declarations are an example where URIs are used not only to identify Web-accessible documents, but also as abstract resources.

2.3.2 XML Schema

DTDs impose several restrictions with respect to expressivity. This means that only very simple languages can be defined by means of DTDs alone. For instance, one cannot define restrictions on how often a certain element may appear by only regular expressions; moreover, datatypes are hardly supported; etc.

These limitations are overcome by another mechanism for defining XML grammars, XML Schema. Beyond DTDs, XML schema allows advanced features such as the following:

- support for a basic set of datatypes (numbers, strings, and dates, etc.), which can be restricted further;
- definition of one's own element or attribute types, available for reuse via an inheritance mechanism which allows extension/restriction;
- namespace support;
- XML Schema is an XML language itself, allowing developers to profit from tool support, the inherent extensibility of XML, the combination of several XML Schema files, etc.

Listing 2.4 shows a schema for the XML file shown in Listing 2.1, which exposes some of these features. For instance, it specifies a common type `namestring`, which is based on the basic XML Schema string datatype restricted to strings starting with an upper-case letter. This type is assigned to both the `firstname` and the `lastname` elements. In order to dereference the schema in Listing 2.4, the DTD reference in Listing 2.1 would need to be replaced with a reference to the XML Schema document. For more details of XML Schema we refer to the XML Schema Standard, see <http://www.w3.org/XML/Schema>.

Note, however, that XML Schema still operates on a purely syntactic level in defining the structure of XML data and, more importantly, there are certain things that are beyond the expressivity of XML Schema. These shortcomings will be partly resolved by technologies to be discussed in Chapter 3:

```

version="1.0" encoding="UTF-8"?> <xs:schema
xmlns="http://www.wsmo.org/namespace"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="qualified" targetNamespace="http://
  www.wsmo.org/namespace" >
  <xs:element name="people">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string" maxOccurs="1"/>
        <xs:element name="member" type="person" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="person">
    <xs:sequence>
      <xs:element name="firstname" type="namestring" minOccurs="1" maxOccurs="2"/>
      <xs:element name="lastname" type="namestring" minOccurs="1" maxOccurs="2"/>
      <xs:element name="affiliation" type="namestring" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="chair" default="no">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="yes"/>
          <xs:enumeration value="no"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
  <xs:simpleType name="namestring">
    <xs:restriction base="xs:string">
      <!-- This pattern says that names are strings
      starting with an uppercase letter -->
      <xs:pattern value="\{p}\{Lu\}.*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>

```

Listing 2.4. XML Schema offers features beyond DTD: datatypes, reuse of element structures, etc.

- Although XML Schema allows one to define reusable types for attributes and tags, this does not mean that one can indeed define refined tags, i.e. not true inheritance on a semantic level is supported [38].
- No complex checks relating different elements or attributes are supported (for instance, in comparison arithmetic, one element may need to be greater than another, the sum of the contents of elements of a particular type should not exceed a certain amount, etc.).

2.3.3 XPath and XSLT

So far, we have only scratched the surface, limiting our discussion to defining the structure of XML documents. We have not yet touched on how to query, transform, and integrate various XML files/formats: the forte of the family of standards that surround XML.

Switching between different XML formats, easily generating mediators or wrappers between different formats, merging data, and combining and filtering information are the hallmark of the Web as it evolves from a network of

human-consumable information towards machine-readability. These issues are partly being addressed by XPath and XSLT.

XPath is a lightweight query language for extracting parts of XML trees. It allows to specify complex conditions on child or parent nodes, content, or attribute values, and provides simple arithmetic and aggregation (such as summing or averaging element values). The details of XPath are beyond the scope of this book, but, most importantly, XPath is one of the main components of Extensible Stylesheet Language Transformations (XSLT) which allows one to rearrange and synthesize the results of XPath queries into new XML documents; thus, the transformation of an arbitrary XML format into another one is facilitated.

Here also, we do not want to go into details, but it is worth noting the evident concepts behind XSLT (Fig. 2.6). An XSLT stylesheet defines templates which match certain parts of the original XML file(s) and convert them into the output format. These templates can be recursively applied or explicitly called.

The example in Fig. 2.6 shows an XML stylesheet which converts our running example XML file back to XHTML, which can be displayed in your Web browser.

Again, we emphasize that XSLT is not restricted to XHTML transformations but can also be used for arbitrary conversions between different XML formats.

2.3.4 Applications and Tools for XML

In this subsection, we discuss, without claiming to be exhaustive, some available tools and applications for XML, illustrating the great success of XML in the past few years. Application designers and developers are profiting from the large number of available APIs and tools for XML.

Tools

As shown in Fig. 2.5 XML documents are often conceived as trees. Application programming interfaces (APIs) for XML such as DOM⁹-based APIs indeed store XML documents in a tree-like data structure while parsing and allow programmers to conveniently deal with XML data in the DOM tree in memory, navigating between child and parent elements, along attributes, etc. Such an in-memory representation of XML data might become too inflexible or space-consuming for large XML documents. Therefore other more lightweight APIs, such as SAX,¹⁰ have been developed in order to parse and process XML files sequentially using an event-based model.

⁹ <http://www.w3.org/DOM>.

¹⁰ <http://www.saxproject.org>.


```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  xmlns:wsmo="http://www.wsmo.org/namespace"
  xmlns="http://www.w3.org/1999/xhtml" >

  <xsl:template match="/" >
    <html xmlns="http://www.w3.org/1999/xhtml" >
      <title>The chairs of the WSMO Working Group</title>
      <body>
        <h1>WSMO WG Chairs</h1>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="wsmo:member[@chair='yes']" >
    <em><xsl:value-of select="."/wsmo:lastname"/></em>,
    <xsl:value-of select="."/wsmo:firstname"/><br/>
  </xsl:template>

  <xsl:template match="text()|@"*/>
  </xsl:stylesheet >

```

(a) An XSLT Stylesheet ...



(b) ... which converts the XML file from Listing. 2.1 into XHTML

Fig. 2.6. XSLT: converting between different XML formats

XPath and XSLT processors are also available off the shelf, e.g. XERCES¹¹ from Apache. The simplicity of the underlying data model combined with the increasing tool support has made XML the most successful data exchange format for information integration applications (not restricted to Web applications alone).

Applications of XML

Nowadays, XML has reached the status of a universal data exchange format with high industry support. In applications such as data integration or Web

¹¹ <http://xerces.apache.org>.

services, XML is a fundamental basis. The simplicity of the format and the standardization by a neutral body with wide vendor support (the W3C) are the cornerstones of the success of XML. Nevertheless, as already pointed out in the discussion of XML Schema, the format only enables syntactic interoperability, to a large extent. The meaning of tags cannot be described with XML as such, and transformation between different XML formats, albeit facilitated through XSLT and related languages, remains mainly a manual task. However, the manual deployment of XSLT transformations does not scale to mediation between possibly hundreds of thousands of different XML formats. Standardized vocabularies and intelligent technologies, such as the Semantic Web, promise to tackle these challenges by building on top of XML standards. In the following chapters, we shall describe these in more detail.

2.4 Summary

In this chapter, we have briefly recapitulated the history of the current Web and outlined its most substantial developments and success factors. The traditional Web is based on three main building blocks: global identification of resources via URIs, a simple client-server-based protocol for persistent publication of globally accessible data (HTTP), and an easy-to-use language for creating interlinked hypertext documents (HTML).

Standardization is the key for technologies such as the Web in reaching their current level of success and interoperability. The World Wide Web consortium (W3C) and other standardization bodies are working towards the continuous progress of Web-related standards, by controlling accessibility and compliance with basic Web principles.

As we have seen, a first step towards the next generation of the Web and a more strict separation between content and layout has already been taken with the introduction of XML and the related family of standards, of which we have sketched a few. As we shall see in the following chapters, however, this takes us only halfway towards real machine-processable Web content and interoperable services. We need to bring back the computer as a device for computation and assistance to help us deal better with the rapidly growing amount of information available on the Web.

Note that unlike other documents in the literature, we do not include XML per se in the “Semantic Web”, but instead consider the family of standards around XML as an intermediate step towards a real machine-processable Web infrastructure.

The Semantic Web

A major drawback of XML is that XML documents do not convey the meaning of the data contained in the document. Exchange of XML documents over the Web is only possible if the parties participating in the exchange agree beforehand on the exact syntactical format (expressed in XML Schema) of the data. The Semantic Web [13] allows the representation and exchange of information in a meaningful way, facilitating automated processing of descriptions on the Web.

Annotations on the Semantic Web express links between information resources on the Web and connect information resources to formal terminologies – these connective structures are called ontologies. Ontologies [38] form the backbone of the Semantic Web; they allow machine understanding of information through the links between the information resources and the terms in the ontologies. Furthermore, ontologies facilitate interoperation between information resources through links to the same ontology or links between ontologies.

The term “ontology” originates from philosophy and has been adopted in the field of Computer Science with a slightly different meaning [53]:

An ontology is a formal explicit specification of a shared conceptualization.

In the late 1990s the idea of a Semantic Web [13] boosted interest in the development of ontologies even further. The general conviction held by the W3C is that the Semantic Web needs an ontology language that is compatible with current Web standards and is in fact layered on top of them. The language needs to be expressed in XML and, preferably, should be layered on top of RDF(S) (an overview of these languages is provided later in this chapter).

An often used depiction of the vision of Semantic Web languages is the “Semantic Web layer cake.” The original layer cake¹, which featured a rules language layered on top of the ontology language, was presented at XML2000 by Tim Berners-Lee, director of the World Wide Web Consortium (W3C). It

¹ <http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html>.

turns out that rules languages cannot be layered on top of the Web Ontology Language OWL in a straightforward manner [70]; this triggered a refinement of the layer cake, depicted in Fig. 3.1, where rules feature next to OWL, on top of a common layer. The revised layer cake² was presented by Tim Berners-Lee at his keynote address at the WWW2005 conference.

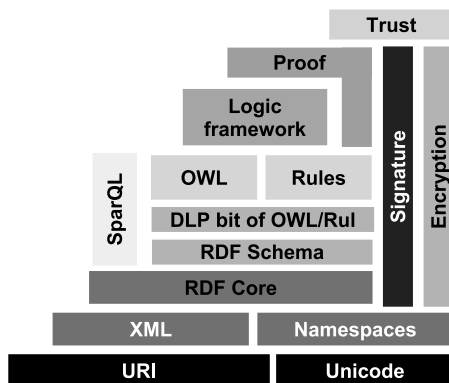


Fig. 3.1. The Semantic Web language layer cake

The bottom layers in the layer cake, i.e. Unicode and URI and XML (Schema), consist of existing Web standards and provide a syntactical basis for Semantic Web languages. Unicode provides an elementary character-encoding scheme, which is used by XML. The URI (uniform resource identifier) standard provides a means to uniquely identify and address documents and, more generally, resources on the Web. All concepts used in the languages located higher in the layer cake can be specified using Unicode and are uniquely identified by URIs.

We shall describe the RDF(S), OWL, and rules layers below. We shall not cover the logic, proof, and trust layers here. Placing the logic layer on top of the OWL and rules layer is somewhat controversial, since OWL and rules languages are grounded in logic. Some argue that a more expressive logic language should be layered on top of the ontology language [108]. It could also be argued that this is not an appropriate layering; that is, that OWL and rules should be the top languages and that applications should use that layer directly. The proof and trust layers are not well-understood, but most likely refer to the application and not to any specific language. For instance, the application could prove some statement by using deductive reasoning, and a statement could be trusted if it had been proven and digitally signed by some trusted third party. The user would very likely play an important role in the trust layer because it is the user that should decide whether or not an information source should be trusted.

² <http://www.w3.org/2005/Talks/0511-keynote-tbl/>.

In the remainder of this chapter we first describe the role of ontologies in the Semantic Web and then proceed by discussing the languages of the layer cake.

3.1 Ontologies and the Semantic Web

A key feature of ontologies is that, through formal, real-world semantics and consensual terminologies, they interweave human and machine understanding [38]. This important property of ontologies facilitates the sharing and reuse of ontologies among humans, as well as among machines.

A major reason for the recent increasing interest in ontologies is the development of the Semantic Web [13], which can be seen as knowledge management on a global scale. Tim Berners-Lee, inventor of the current World Wide Web and director of the World Wide Web Consortium (W3C), envisions the Semantic Web as the next generation of the current Web. This “next generation” will expand upon the prowess of the current Web by adding machine-readable information and automated services. According to [38], “The explicit representation of the semantics underlying data, programs, pages, and other Web resources will enable a knowledge-based Web that provides a qualitatively new level of service.” Ontologies provide such an explicit representation of semantics. The combination of ontologies with the Web has the potential to overcome many of the problems in knowledge sharing and reuse and in information integration.

Ontologies interweave human and computer understanding of symbols. These symbols, also called terms and relations, can be interpreted by both humans and machines. The meaning for a human is represented by the term itself, which is usually a word in natural language, and by the semantic relationships between terms. An example of such a human-understandable relationship is a superconcept – subconcept relationship (often referred to by the term “is-a”). Such a relationship denotes the fact that one concept (the superconcept) is more general than another (the subconcept). For instance, the concept **Person** is more general than **Student**. Figure 3.2 shows an example “is-a” hierarchy (or taxonomy), where the more general concepts are located above the more specialized concepts.

Concepts describe a set of objects in the real world. For example, the concept **PhD-Student** aims to capture all existing PhD students. One such PhD student is **Mary**, who is modeled in Fig. 3.2 as a box, and has an **instance-of** relation to the concept **PhD-Student**. This **instance-of** relationship means that the actual object is captured by the concept **PhD-Student**. And because of the formal **is-a** relationships between the concepts **PhD-Student**, **Researcher**, **Student**, and **Person**, **John** must also be an instance of the concepts **Researcher**, **Student**, and **Person**.

These relationships are fairly easy to understand for the human reader and, because the meanings of the relationships are formally defined, a machine

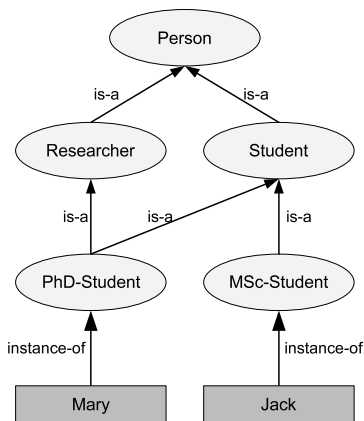


Fig. 3.2. Example is-a hierarchy (or taxonomy)

can reason with them and draw the same conclusions as a human can. These relationships, which are implicitly known to humans (e.g. a human knows that every student is a person) are encoded in a formally explicitly way so that they can be understood by a machine. In a sense, the machine does not gain real “understanding”, but the understanding of humans is encoded in such a way that a machine can process it and draw conclusions through logical reasoning.

3.1.1 Types of Ontologies

There exist many different ontologies, built for many different types of applications; they vary in both the amount of detail they express and the generality of their use.

WordNet [37], for example, aims to cover all of the English language by providing a (natural-language) description for every English term and by specifying simple relationships, such as synonym (equivalent terms) and hyper-/hyponym (more/less general terms) relationships. The scope of WordNet is very broad (all of the English language) and the level of detail is relatively low. There are only natural-language descriptions for the terms, which are not machine-understandable, and there exist only very simple relationships between the terms.

CYC [82] is another example of an ontology with a very broad scope, which attempts to capture all commonsense knowledge (e.g. space and time), but with a high level of detail. There are many very strict formal relationships between different terms. These formal relationships are machine-understandable.

We shall refer to the scope of an ontology as the “generality” and the level of detail as the “expressiveness”. We provide a more detailed description of the generality and expressiveness of ontologies below and use these as dimensions to classify existing ontologies.

3.1.2 Generality of Ontologies

An ontology is a specification of a shared conceptualization. Therefore, domain experts, users, and designers need to agree on the knowledge specified in an ontology so that the ontology may be shared and reused. It is hard to get such agreement. It is therefore advantageous to layer the knowledge in different ontologies on the basis of generality, so that not everybody needs to agree to all ontologies. Agreement is required only between specific domain and application ontologies and between the higher-level ontologies that are being used [91].

In the literature [38, 56, 62, 125], we generally find three common layers of knowledge. On the basis of their levels of generality, these three layers correspond to three different types of ontologies, namely:

- *Generic* (or top-level) ontologies, which capture general, domain independent knowledge (e.g. space and time). Examples are WordNet [37] and CYC [82]. Generic ontologies are shared by large numbers of people across different domains.
- *Domain* ontologies, which capture the knowledge in a specific domain. An example is UNSPSC,³ which is a product classification scheme for vendors. Domain ontologies are shared by the stakeholder in a domain.
- *Application* ontologies, which capture the knowledge necessary for a specific application. An example could be an ontology representing the structure of a particular Web site. Arguably, application ontologies are not really ontologies, because they are not really shared.

The separation between these three levels of generality is not always strict. WordNet, for example, contains some domain-specific relations and CYC contains domain-specific microtheories (modules of the ontology).

Although sometimes other types of ontologies, such as representational ontologies or task ontologies are distinguished, the above three types of ontologies are common in the literature and are, in our opinion, a useful separation of types of ontologies along the dimension of generality.

3.1.3 Expressiveness of Ontologies

Orthogonal to the generality of ontologies is their expressiveness. We distinguish the following levels of expressiveness (partly on the ontology spectrum introduced in [89]):

- *Thesaurus*. Relations between terms, such as synonyms, are additionally provided. Again, WordNet [37] is an example.

³ <http://www.unspsc.org>.

- *Informal taxonomy.* There is an explicit hierarchy (generalization and specialization are supported), but there is no strict inheritance; an instance of a subclass is not necessarily also an instance of the superclass. An example is the Yahoo! Dictionary.⁴
- *Formal taxonomy.* There is strict inheritance; each instance of a subclass is also an instance of a superclass. An example is UNSPSC.
- *Frames.* Frame (or class) contains a number of properties and these properties are inherited by subclasses and instances. Ontologies expressed in RDFS [20], described below, fall into this category.
- *Value restrictions.* The values of properties are restricted. Ontologies expressed in OWL Lite (see Section 3.3) fall in this category.
- *General logic constraints.* Values may be constrained by logical or mathematical formulas using values from other properties. Ontologies expressed in OWL DL (see Section 3.3) fall into this category.
- *Expressive logic constraints.* Very expressive ontology languages such as those seen in Ontolingua [36] and CycL [82] allow first-order logic constraints between terms and more detailed relationships such as disjoint classes, disjoint coverings, inverse relationships, and part-whole relationships. Note that some of these detailed relationships, such as disjointness of classes, are also supported by OWL DL (and even OWL Lite), which indicates that the borders between the levels of expressiveness remain fuzzy.

3.1.4 History of Ontology Languages

In the areas of knowledge engineering and knowledge representation, interest in ontologies really started taking off in the 1980s with knowledge representation systems such as KL-ONE [19] and CLASSIC [18].

An important system for the development, management, and exchange of ontologies in the beginning of the 1990s was Ontolingua [36], which uses an internal KIF (knowledge interchange format)⁵ representation, but is able to interoperate with many other knowledge representation (ontology) languages, such as KL-ONE, LOOM, and CLASSIC.

The languages used for ontologies were determined by the tool used to create the ontologies. Systems such as KL-ONE, CLASSIC and LOOM each used their own ontology language, although the Ontolingua system was capable of translating ontologies between different languages, using the KIF language as an interchange language. We can see the languages and tools as being interdependent, but also as being somewhat orthogonal, where we have the language on one axis and the tool on the other. For example, KL-ONE, CLASSIC, and LOOM all have their basis in description logics [5], while KIF has its basis in first-order logic.

⁴ <http://www.yahoo.com>.

⁵ <http://logic.stanford.edu/kif/kif.html>.

In the early 1990s, KIF could be seen as a standard for ontology modeling. This language was used in prominent tools such as Ontolingua and in important ontology engineering projects, such as TOVE (Toronto Virtual Enterprise) [55] and the Enterprise Ontology [129].

Later on in the 1990s, ontologies began to be applied to the World Wide Web. SHOE [62], for example, used ontologies to annotate Web pages using formal ontologies embedded in HTML documents. Ontobroker and its successor On2broker [40] use ontologies not only to annotate Web pages, but also to formulate queries and derive answers. Ontobroker and On2broker provide an annotation language which is used to annotate HTML documents with references to ontologies. The ontology, the terminology used by the annotation language, is specified using a representation language, based on F-Logic [72]. Finally, Ontobroker uses a query language for the retrieval of documents based on their annotations. This query language is a subset of the representation language. All these languages have an impact on the current languages of the Semantic Web.

3.2 The Resource Description Framework

The Resource Description Framework (RDF) [75] is the first language developed especially for the Semantic Web. RDF was developed as a language for adding machine-readable metadata to existing data on the Web. RDF uses XML for its serialization in order to realize the layering depicted in the Semantic Web language layer cake (Fig. 3.1). RDF Schema [20] extends RDF with some basic (frame-based) ontological modeling primitives. There are primitives such as classes, properties, and instances. Also, the instance-of, subclass-of, and subproperty-of relationships have been introduced, allowing structured class and property hierarchies.

RDF has the subject–predicate–object triple, commonly written as P(S,O), as its basic data model. An object of a triple can, in turn, function as the subject of another triple, yielding a directed labeled graph, where resources (subjects and objects) correspond to nodes, and predicates correspond to edges. Furthermore, RDF allows a form of reification (a statement about a statement), which means that any RDF statement can be used as a subject in a triple. An example RDF graph is shown in Fig. 3.3. The corresponding RDF/XML serialization is shown in Fig. 3.4.

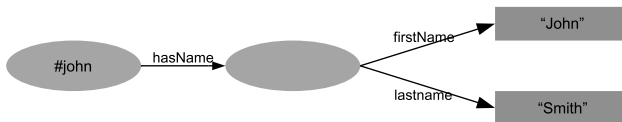


Fig. 3.3. Example RDF graph

```

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ex="http://example.org/#"
  xml:base="http://example.org/">
  <rdf:Description about="#john">
    <ex:hasName rdf:parseType="Resource">
      <ex:firstName>John</ex:firstName>
      <ex:lastName>Smith</ex:lastName>
    </ex:hasName>
  </rdf:Description>
</rdf:RDF>

```

Fig. 3.4. Example RDF/XML serialization

3.2.1 RDF Schema

RDF Schema (RDFS) is a lightweight ontology language for defining vocabularies for RDF. Unlike XML Schema, which prescribes the order and combinations of tags (the structure) in an XML document, RDF Schema only provides information about the interpretation of the statements given in an RDF data model. RDF Schema does not say anything about the syntactical appearance of the RDF description. RDFS can in fact be seen as an extension of RDF with a vocabulary for defining classes, class hierarchies, properties (binary relations), property hierarchies, and property restrictions. RDFS classes and properties can be instantiated in RDF. For a more detailed comparison of XML Schema and RDF Schema we refer the reader to [74]. Figure 3.5 shows the graph which corresponds to the is-a hierarchy of Fig. 3.2.

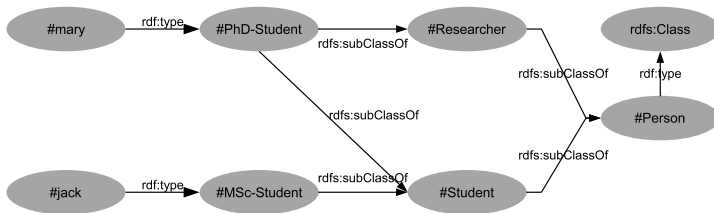


Fig. 3.5. RDFS ontology of persons and working group members

RDF(S) (referring to the combination of RDF and RDF Schema) is not very expressive compared with many other ontology languages, as it allows only the representation of concepts, concept taxonomies, and binary relations. The expressive limitations of RDF(S) were a major motivation for developing more expressive languages for the Semantic Web. In the remainder of this chapter, we describe the ontology and rules components which are layered on top of RDF(S).

3.3 The Web Ontology Language OWL

The Web Ontology Language OWL [32] is an expressive ontology language which extends RDFS. OWL itself consists of three *species* of increasing expressiveness:

- *OWL Lite*. The least expressive of the OWL species. Compared with RDFS it adds local range restrictions, existential restrictions, simple cardinality restrictions, equality, and various types of properties (inverse, transitive, and symmetric).
- *OWL DL*. Compared with OWL Lite, OWL DL adds full support for (classical) negation, disjunction, cardinality restrictions, enumerations, and value restrictions. The element “DL” comes from the resemblance to an expressive description logic language [5], namely *SHOIN(D)*.
- *OWL Full*. Whereas OWL Lite and OWL DL impose restrictions on the use of vocabulary and the use of RDF statements, OWL Full does not have such restrictions. Therefore, OWL Full allows both the specification of classes-as-instances and the use of language constructs in the language itself, which thereby modifies the language.

For OWL Lite, it turns out that although there are many syntactic restrictions in the language, the actual expressiveness of the language is very close to the expressiveness of OWL DL [65]. OWL Full is a very expressive language and, because of the syntactic freedom which is allowed in the language, key inference problems are undecidable. In the remainder of this section, we shall focus on OWL DL.

Whereas statements in RDF(S) are triples, statements in OWL DL are either axioms or assertions. An axiom is either a class definition, a class axiom, or a property axiom. Class definitions can be used to define subclass relationships, as well as certain property restrictions which hold for a particular class. With class and property axioms, one can express more complex relationships between classes and between properties such as boolean combinations of class descriptions and transitive, inverse, and symmetric properties. Individual assertions can be used to express class membership, property values, and (in)equality of individuals.

OWL DL is defined in terms of an abstract syntax [109]. However, the normative syntax for OWL is RDF/XML. The RDF representation of an OWL DL ontology can be obtained through a mapping from the abstract syntax. Listing 3.1 shows an OWL DL ontology, in abstract syntax, where we define a class `Person` with a property `hasChild`, of type `Person`, and a class `Parent`, which is defined as a person who has a child. Finally, we define an individual `John`, who has a child `Mary`. OWL DL allows us to infer that `John` is a `Parent`.

Note that from the type restriction on `hasChild` in the definition of `Person`, and the fact that `Mary` is a value for this property, `Mary` is in fact a member of

```

Class(Person partial
  restriction (hasChild allValuesFrom(Person)))

Class(Person complete
  Person
  restriction (hasChild someValuesFrom(Person)))

ObjectProperty(hasChild)

Individual (John type(Person)
  value(hasChild Mary))

```

Listing 3.1. OWL example using abstract syntax

the class `Person`. Although this inference seems to be intended in this scenario, it cannot be assumed that this kind of inference is intended in general [31].

3.4 Rules for the Semantic Web

From the early days of the Semantic Web, rules have been seen as an important paradigm for representing and reasoning with knowledge on the Semantic Web.⁶ However, at the time of writing, activities have just gotten underway towards standardization of a rules language for the Semantic Web.⁷

With ontologies, one can express knowledge about classes, class hierarchies, properties, etc. Rules have a complementary expressiveness: with rules, one can express knowledge in the form “if A then B ”. An example which is often used to motivate the use of rules is the “uncle” example, which says that “the brother of a person’s parent is that person’s uncle”:

$$person(?x) \wedge parent(?x, ?y) \wedge brother(?y, ?z) \Rightarrow uncle(?x, ?z).$$

3.4.1 SWRL

SWRL [64] is an extension of OWL DL which adds the expressive power of rules (without negation) to OWL; the above “uncle” example can be expressed in SWRL.

The basic SWRL constructs are Horn-like rules. However, whereas Horn rules have a conjunction of atomic formulas in the antecedent (body) of the rule and a single atomic formula in the consequent (head) of the rule, SWRL allows any OWL class description, property, or individual assertion in both the body and the head of the rule. In this way, SWRL diverges from traditional rules systems, which are based on Logic Programming or Deductive Databases.

⁶ <http://www.ruleml.org>.

⁷ <http://www.w3.org/2005/rules>.

```

person[ child =>> person].
parent :: person.

X:parent :- X[child ->> Y], X:person.

```

Listing 3.2. Example F-Logic ontology

Because SWRL combines the full expressive power of function-free Horn logic with an expressive description logic language, the key inferences tasks (e.g. satisfiability and entailment) are in general undecidable for SWRL.

3.4.2 F-Logic

F-Logic [72], and, more specifically, the Horn subset of F-Logic extended with negation, has been proposed as an ontology and rule language for the Semantic Web [69]. Rules in F-Logic are similar to Horn rules, with the distinction that besides atomic formulas, F-Logic rules *also* allow molecules in place of atomic formulas. Note that although the syntax of F-Logic seems higher-order, the language remains semantically in the first-order framework.

There are various kinds of molecules. An is-a assertion $C:D$ states that C is an instance of the class D ; a subclass assertion $C::D$ states that C is a subclass of D . Data molecules of the form $C[D \rightarrow E]$ have the meaning that the attribute D of the individual C has the value E . Signature molecules of the form $C[D \Rightarrow E]$ indicate that the class C has an attribute D and that all values associated with this attribute are of type E .

To simplify matters, we shall focus only on a subset of F-Logic. We do not consider parameterized methods or functional (single-valued) methods, but instead consider only noninheritable methods. We also do not consider compound molecules.

An important concept in F-Logic is *object identity* [68]. Each object (e.g. a class, instance, or method) has a unique object identifier, where an object identifier is in fact a term. In F-Logic, classes and methods are interpreted *intentionally*, which means that class identifiers and method identifiers are interpreted by themselves and not directly as sets or as binary relations, as is the case with concepts and roles in description logics. Classes and methods are first interpreted as objects in the domain, and these objects are then related to sets of objects and sets of binary tuples, respectively.

The simple F-Logic ontology shown in Listing 3.2 models the concept `person`, which has an attribute `child` with type `person`. The concept `parent` is a subconcept of `person`, and the rule states that every person with a child is a `parent`.

Note that in F-Logic there is no distinction between classes and instances. An *object identifier* can denote a class, an instance, or an attribute, but there is no separation in the signature Σ of the identifiers denoting any of these. The advantage of such an overloaded concept of an object is that objects denote

classes, instances, and attributes depending on the syntactic context, thereby allowing certain kinds of metastatements. For example, we might define an attribute `parent`, which is not related to the class `parent`:

```
person[parent =>> person].
```

Both SWRL and F-Logic have been proposed as rules languages for the Semantic Web. The main difference between the two proposals is that in SWRL, the rules language is seen as an extension of the ontology language OWL DL, whereas in the F-Logic (programming) proposal, ontologies are modeled using rules.

3.5 Summary

In this chapter we have seen that Semantic Web technologies have the potential to overcome many of the limitations of the current Web by introducing machine-processable annotations and formal languages for expressing knowledge on the Web. Ontologies play a key role in enabling communication and interoperation on the Web because they represent consensual terminologies which can be used for automated processing.

RDF [75] is the language recommendation from the W3C for writing machine-processable annotations on the Web. RDFS [20] and OWL [32] are the language recommendations for writing ontologies on the Web. RDFS is an inexpressive ontology language which offers the possibility to define classes, class hierarchies, properties, property hierarchies, and domain and range restrictions. OWL extends RDFS towards a more expressive ontology language. Some examples of the features of OWL are transitive, symmetric, and inverse properties; local restrictions on properties; cardinality and existential property restrictions; and boolean combination of classes.

Rules can be seen as a knowledge representation paradigm which is orthogonal to that of ontologies, yet can also be combined with ontologies (e.g. SWRL [64]). However, it has also been argued that rules should be used as the basis for an ontology language [69].

Web Services

Computer science is now in an important period of abstraction. A generation ago, we learned to abstract from hardware – now we are progressing by learning to abstract from software. This abstraction is achieved through service-oriented architectures (SOAs), which allow us to reuse and combine software components via standardized interfaces, arbitrarily combining them to serve our needs.

Web services and service-oriented architectures are rather new technologies with the common goal of making software components and business applications available via standardized interfaces, within or across enterprise boundaries. These technologies will facilitate simpler development of distributed applications and the integration of existing software solutions. The two major application areas of Web service technologies are as follows:

- *Enterprise application integration (EAI)*. The integration of legacy applications and systems, which have often been used reliably over years, is becoming a significant, if not the most important, factor in IT. There is a strong need for the development of flexible EAI solutions instead of expensive reimplementations.
- *E-commerce*. The existing Web interfaces which various providers offer to their customers mark only the beginning of an increasing economic sector. Online stores and electronic marketplaces – empowered by Web service technologies – could profit greatly by integrating and combining their current services into more powerful, value-added services via common software interfaces.

Web service technology promises to offer the technical infrastructure for both these key areas.

Basically, Web services allow access to a functionality via the Web using a set of open standards that make the interaction independent of implementation aspects such as the operating system platform and the programming language used. The core set of Web service standards enable a set of basic interactions required in a service-oriented architecture, as depicted in Fig. 4.1.

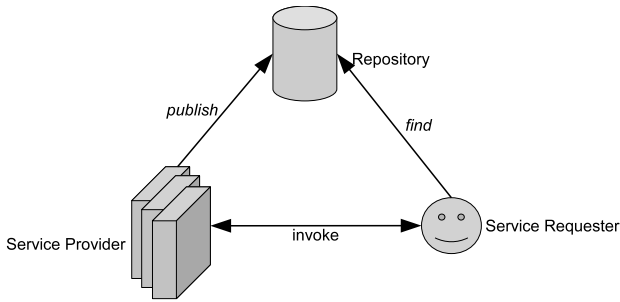


Fig. 4.1. Web service architecture

This figure illustrates the basic building blocks of a service-oriented architecture. In order for a service to be discovered, a service provider must first publish the service. In most of the current proposals, this is done in some central repository. A service requester is then able to issue queries to obtain a reference to the desired service. This interaction is regulated by, for example, the UDDI (Universal Description, Discovery and Integration) standard.

To invoke a service the requester needs to know how to do this (i.e. it needs the interface definition). This information is provided by WSDL (Web Service Description Language)[27]. Together with the SOAP standard, used for transmitting data, this gives us three core standards (UDDI, WSDL, and SOAP) that enable the basic interactions required by a service-oriented architecture.

It has to be emphasized that so far mainly syntactical aspects have been addressed. Therefore, these technologies provide only a set of rigid services that cannot adapt to changing environments without human intervention. The human programmer has to be kept in the loop, and the scalability and economy of Web services are limited [41].

Nevertheless, Web service technologies represent a milestone in making the development of distributed applications in an SOA possible on large scale. They provide, for the first time, widely accepted and widely used standards to govern the communication between applications, facilitating their combination into more complex entities. This is a major step beyond previous and existing middleware standards, which have not reached the same level of consensus and acceptance. This failure was partly due to a lack of agreement between global players in the industry, and partly due to the lack of simple widespread, application-independent protocols and standard formats for data exchange – protocols and standards such as HTTP and XML upon which Web services are built.

In this chapter, we first give an overview of the terminology used and the underlying principles in Section 4.1, followed by a brief history and overview of Web service technologies in Section 4.2. We proceed to a more detailed analysis of UDDI, WSDL, SOAP, and other relevant Web service standards

in Section 4.3. We summarize the current status of the most widely used Web service technologies in Section 4.4.

4.1 Terminology and Principles

Before we continue, let us clarify the terms “service” and “Web service” themselves. The English word “service” is overloaded in its meaning [111]. In the business world, a “service” normally denotes the provision of a general business activity which provides a certain value to the customer [6]. In computer science, in contrast, the term “service” is often used synonymously with “Web service”, i.e. a software component accessible over the Internet via a standardized interface. Preist [111] defines these terms as follows:

- *Service*. A service is defined as the provision of a concrete product or abstract value in some domain. As an example, let us consider a user who wants to book a flight from Madrid to Innsbruck. Following our definition of a service, we mean the actual transport, fulfilling user constraints such as a certain date. The provision of the service as such, and the contractual issues around this service provision, are independent of how the supplier and the provider interact. It is irrelevant whether the requester goes to an airline ticket office or uses the airline’s Web site to book the flight. We understand the term “service” in this sense, that is, as a provision of value.
- *Web service*. Following Preist’s definition, Web services are defined as computational entities accessible over the Internet (using Web service standards and protocols) via platform- and programming-language-independent interfaces. Returning to our previous example, an airline might provide a software component accessible via Web service standards, i.e. a Web service to request the booking of a flight; the Web service is an electronic means to request a service, but not the service itself. We understand the term “Web service” as a means to request a service over standard protocols, described using widely accepted standards. Thus, a Web service is a means to consume an actual service, or to place a contract for an actual service.

This book is about Web services, and we shall use the terms “service” and “Web service” synonymously to mean the latter in the remainder of this book unless explicitly indicated otherwise.

A further term often used in connection with Web services is the “service-oriented architecture”, which we introduced at the beginning of this chapter. Since the term and its definition are still heavily discussed, we shall merely reiterate and outline the basic principles that are common to all definitions:¹

¹ This list has been extracted from a message by J.P. Morgenthal on service-orientated-architecture@yahoogroups.com and subsequent discussions; see <http://groups.yahoo.com/group/service-orientated-architecture/message/2709>.

1. *Loose coupling*. Every service is atomic, self-describing, accessible, declarative, stateless and composite.
2. *Contracted*. Services in an SOA are represented by a contract that describes its inputs, outputs, access policies, quality-of-service requirements, and error-handling procedures.
3. *Discoverable*. Services should be able to be discoverable at the time of execution.
4. *Addressable*. Services should be uniquely identifiable in a network.
5. *Distributed*. Services in an SOA are separated by geographic and machine boundaries and, therefore, must be good netizen applications, i.e. they must be developed with the ability to recover from loss of communication.
6. *Point-to-Point*. At any point in time one consumer uses one and only one producer.

In particular, the stateless nature of services (1), the pure point-to-point behavior (6), and discovery at execution time (3) deserve a closer look. We shall not go into the details in this book, but refer to [42, 79] for a more in-depth discussion of the related issues.

The term “service”, in connection with SOAs, is again often used synonymously with “Web service”. However, in general, services in an SOA are not necessarily Web services in the strict sense. Rather, they simply denote any software component that is accessible over an arbitrary protocol. Platform independence, programming-language independence, and the use of WSDL are not necessary criteria.

4.2 The Origins of Web Services

Efforts towards the development of common standards for the development of distributed applications have a long history in computer science. Following the usual terminology in software engineering, technologies which offer an abstraction layer over the actual communication media are often subsumed under the general term “middleware”.

Numerous earlier attempts to establish a single standardized middleware solution in the past either failed completely or were restricted to use within enterprise boundaries. In particular there has not been vendor-spanning agreement on how such a standardization should look.

In the early 1980s, Birell and Nelson [15] introduced the Remote Procedure Calls (RPC), the first widespread technology for invoking procedures in distributed systems. The RPC definition was released in an abstract fashion, hiding implementation details from the developer. In combination with a standardized interface description language, RPC offers a platform- and programming-language-independent mechanism for embedding distributed procedure calls in applications.

The additional requirements of real-world applications and the wide acceptance of the object-oriented paradigm have led to various extensions of

RPC, such as Transactional RPC, Object Brokers, Remote Method Invocation (RMI), Transaction Processing (TP) monitors, and message-oriented middleware (MOM).

Comprehensive middleware architectures such as CORBA – introduced by the Object Management Group (OMG) in the early 1990s – already allowed for advanced features such as dynamic selection and binding of services. Unfortunately, these architectures did not reach the expected degree of acceptance that would be required to reach the actual goal of a standardized middleware platform: effective, simple means of communication for distributed software components catering for painless solution of integration problems within an organization on the one hand and across enterprise boundaries on the other hand.

The technologies which are nowadays subsumed under the term “Web services” have several advantages compared with their predecessors. These advantages are the reason for the wider acceptance of Web services and guarantee a higher probability of long-term success:

- Communication between Web services relies on widely accepted protocols such as HTTP.
- SOAP [92] offers a protocol for message exchange between Web services, where XML is used as a unified exchange format. The growing acceptance of XML as a data interchange format and of related technologies considerably eases the pain of establishing syntactic interoperability between Web services.
- Similarly to RPC and CORBA, there exists a standardized interface description language, WSDL [27], which is currently gaining increasing acceptance.
- As opposed to a pervasive, complex infrastructure, which would solve all the problems of the interplay of distributed applications at once, the “Web services Technology Stack” consists of a modular family of standards, each of which tackles certain subproblems.
- Finally, the “global players” seem to have come to an agreement that the solution to interoperability problems cannot be found in dozens of proprietary approaches; instead, their research and development departments are jointly engaging in the development of standard recommendations in international standardization bodies.

Summarizing, Web service technologies in the narrow sense build upon four main components:

- An agreed *transport protocol*.
- A platform-independent *message description format*.
- A language for *Web service interface description* that describes which operations, and with which messages a service can offer.
- A *registry* for publication and discovery of available services.

The first component can, in principle, be realized by any of the common transport protocols such as SMTP or FTP. The most popular protocol in the context of Web services is HTTP. HTTP has the advantage of allowing the reuse of existing infrastructure while catering for the preexisting know-how of potential users.

The second component is realized by SOAP [92]. SOAP is a specification for standardization of the exchange of XML-coded messages. Moreover, SOAP specifies the binding to HTTP of an underlying communication protocol between two addressable service partners (“endpoints”). Communication via SOAP over HTTP is, in a sense, a solution to existing problems with approaches such as RMI and CORBA. The shortcomings of the latter two include imposing tight coupling between service partners and the inability to reuse the open “Web infrastructure”: firewalls usually block special proprietary communication protocols in common IT across enterprise boundaries.

The Web Service Description Language (WSDL) [27] serves as the third component for XML-based description of interfaces. WSDL allows a distinction between the description of the abstract functionality (operations) that a Web service provides and the details of how to access the service. On the one hand, operations, along with their input and output parameters, are described; on the other hand, the addresses at which the service is invocable (“endpoints”) and the supported transport protocols can be described independently.

Finally, Universal Description, Discovery and Integration (UDDI) [10] provides interfaces to publish (advertise), browse, and query existing Web services. UDDI thus realizes the fourth main component in Web-services-based middleware solutions.

Excellent overviews of the predecessors of current Web service technologies and the current state of the art of “syntactical”² Web service standards are provided by Alonso et al. [2] and Weerawarana et al. [133].

4.3 The Web Service Technology Stack

In this section we give further details on the current Web service technology stack, covering the three main standards: SOAP, WSDL and UDDI.

4.3.1 SOAP

The overall goal of the Simple Object Access Protocol (SOAP), as the first version of this protocol, published as a W3C Note in May 2000, was called, is the provision of a messaging framework for peers communicating messages

² We talk about “syntactical” standards here, to emphasize the fact that these technologies do not yet tackle the semantic annotation of (Web) services.

in XML format. In June 2003, SOAP 1.2³ became a W3C Recommendation and has been gaining momentum ever since, being adopted in most Web-service-based applications. While in theory (and also in practice), instance messages of services communicating over HTTP could be encoded directly in HTTP/get or post requests, SOAP adds a more rigid framework especially tailored for XML message exchange on top, in fact, nowadays the term “Web services” is often used synonymously for “Web services communicating over SOAP” which gives us justification enough to further explain the basics of this fundamental standard protocol here.

SOAP aims to achieve maximum acceptance and flexibility by the provision of a framework in which application-specific information may be conveyed in an extensible manner, without making any up-front commitment to the semantics of application-specific data. Also, SOAP provides a full description of the required actions that must taken by a SOAP node upon receiving a SOAP message. This can, for example, be achieved by including a `mustUnderstand` header, i.e. an XML element that describes processing instructions such as requiring support for some transaction protocol. Without specifying a concrete set of instruction, the SOAP standard specifies that if such an element cannot be understood by a receiving node, it must not process the request but instead issue a fault message.

A SOAP message itself is just an XML document. Listing 4.1 shows a simple example of a typical SOAP document that is used to invoke a weather service. Within a SOAP envelope, optional header and body elements are present. To keep the example short, we have omitted the header element, which normally contains processing instructions as explained in the previous paragraph. The body contains the actual payload of the message. In this case it is an XML node – conforming to the schema definition of the `GetWeather` message of some service within the example – containing two strings indicating a city, described by its name and country. Additionally, SOAP defines a standardized way of handling faults in a message exchange by defining specific XML elements for it (such as `faultcode` and `faultstring`, which eases automatic processing).

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
  <soap:Body>
    <GetWeather xmlns="http://www.webservicex.net" >
      <CityName>Innsbruck</CityName>
      <CountryName>Austria</CountryName>
    </GetWeather>
  </soap:Body>
</soap:Envelope>
```

Listing 4.1. An example SOAP message

Particularly interesting in the Web context is the protocol-binding framework of SOAP. Basically, SOAP messages can be sent using a variety of pro-

³ For various reasons, the original acronym SOAP no longer represents the name in the most recent version.

ocols, although in the Web context SOAP is usually used over HTTP. There are of course other frequently used and well-defined protocols, for example email⁴. When a SOAP message is sent over HTTP, basically just an HTTP header is added to the SOAP envelope. In this way, SOAP can leverage the infrastructure that is already in place for the underlying protocol. Thus, the problems with firewalls and Internet routers that occur when using traditional Remote Procedure Calls are eliminated.

```
POST: /stockquote.aspx
SOAPAction: "http://www.webservice.net/GetQuote"
Content-Type: text/xml; charset=UTF-8
Content-Length: 3000
[...]
```

Listing 4.2. An example SOAP HTTP header

Listing 4.2 shows a typical header that is used when a SOAP message is sent over HTTP. Only the header is dependent on the actual transport protocol used; the SOAP messages transferred do not need to change for different protocols. The header contains standard elements of an HTTP header, such as POST, Content-Type, and Content-Length. Only the SOAPAction field is an addition (it is mandatory in SOAP 1.1).

In summary, SOAP is an XML format for message exchange that can be bound to different underlying transport protocols. It allows one to wrap XML messages in an envelope, possibly containing additional routing or fault-handling information. SOAP has become a popular way to transport XML messages in the current Web service stack.

4.3.2 WSDL

The Web Service Description Language is an XML format to describe interfaces of software components. WSDL essentially allows one to describe a set of operations and their input and output messages. The initial WSDL 1.1 standard was published by the W3C in March 2001 as a Note. The successor, version 2.0, is (at the time of writing) a candidate recommendation. The scope, of describing interfaces remains in version 2.0, although some essential details have changed.⁵ The changes include, for example, the removal of the message construct; the support for operator overloading has been discontinued; and some constructs have been renamed and their usage further clarified. Figure 4.2 illustrates the structure of both versions of the standard and how they relate.

Currently there is not yet mature tool support for the WSDL 2.0 specification, and thus we limit ourselves for the more detailed descriptions to WSDL 1.1. It is an extensible standard, allowing different typing mechanisms

⁴ <http://www.w3.org/TR/2002/NOTE-soap12-email-20020626>.

⁵ Initially it was planned to use the version number 1.2, but after the number of changes was recognized it was decided to use version number 2.0.

and several binding technologies, but in this book we shall restrict our explanations to typical usage in the Web context. In the following we will use exclusively XML Schema as the language for typing and SOAP as as binding technology.

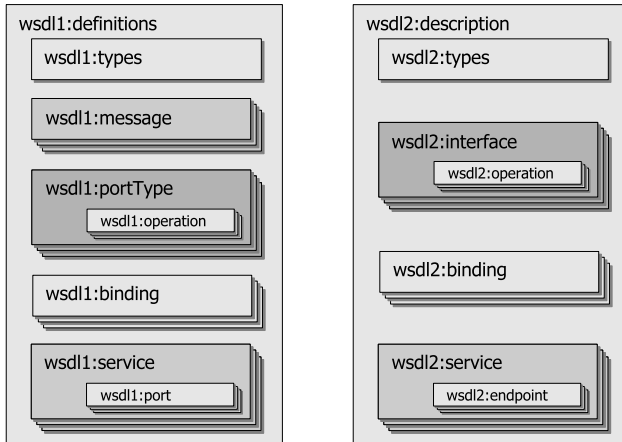


Fig. 4.2. Structures of WSDL 1.1 and WSDL 2.0

The left box in Fig. 4.2 shows the various elements of WSDL 1.0. The top three boxes in the figure show generic elements of WSDL. PortTypes can be seen as interfaces that may define several operations. The definition of the inputs and outputs is encapsulated in a message construct that refers to types defined in the generic type section of a WSDL file. The type section can, basically, include one or more XML schemas that contain typing information.

The bottom two boxes in Fig. 4.2 represents the binding of the interface definition (PortType) to a specific endpoint address and message format. The endpoint indicates the physical address where messages can be sent to. In this way, a WSDL specification separates the signature description of an interface, i.e. operation names with inputs and outputs in a PortType, from the bindings to a concrete transport protocol and an actual endpoint.

Listing 4.3 shows how inputs and outputs are defined in a WSDL file⁶ and how they can be linked to an operation. The type section can contain multiple XML Schema definitions. The relevant definition in our example is the element definition for `getWeather`. It indicates that there is a child element `CityName` and `CountryName` of the type “string”. At the bottom of the listing one can see a `portType` definition, alongside one operation `GetWeather`. The input and outputs are not directly linked to XML elements, but to messages, which in turn reference the previously defined types.

⁶ Taken from <http://www.webservices.net/globalweather.asmx?WSDL>

```

[...]
<wsdl:types>
  [...]
  <s:element name="GetWeather" >
    <s:complexType>
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="CityName" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:element>
  [...]
</wsdl:types>
<wsdl:message name="GetWeatherIn" >
  <wsdl:part name="parameters" element="tns:GetWeather" />
</wsdl:message>
[...]
<wsdl:portType name="GlobalWeather" >
  <wsdl:operation name="GetWeather" >
    <wsdl:input message="tns:GetWeatherSoapIn" />
    <wsdl:output message="tns:GetWeatherSoapOut" />
  </wsdl:operation>
  [...]

```

Listing 4.3. A signature definition in a WSDL file

```

[...]
<wsdl:binding name="GlobalWeatherSoap" type="tns:GlobalWeather" >
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <wsdl:operation name="GetWeather" >
    <soap:operation soapAction="http://www.webservice.net/GetWeather" style="document" />
    <wsdl:input><soap:body use="literal" /></wsdl:input>
    <wsdl:output><soap:body use="literal" /></wsdl:output>
  </wsdl:operation>
  [...]
</wsdl:binding>
<wsdl:service name="GlobalWeather" >
  <wsdl:port name="GlobalWeatherSoap" binding="tns:GlobalWeatherSoap" >
    <soap:address location="http://www.webservice.net/globalweather.asmx" />
  </wsdl:port>
  [...]
</wsdl:service>

```

Listing 4.4. Sample WSDL binding

Listing 4.4 maps the abstract definition to a concrete protocol. The `GlobalWeatherSoap` binding defines a binding using SOAP over HTTP. Finally, in the service element, the actual endpoint (`soap:address`) is declared.

With this short example, we have illustrated a number of features of WSDL: (1) XML Schema can be reused for type declaration; (2) interfaces (operation + PortType) are defined independently of the transport protocol; (3) an interface can be easily reused for several different bindings.

4.3.3 UDDI

UDDI offers an interface which allows business partners and (external) service providers to dynamically find one another. An UDDI server is accessible via a Web service interface and offers operations such as publication and querying of services.

UDDI provides a data model for services and business entities. More concretely, it provides three categories of information: white, yellow, and green pages. This model provides related information to a service such as the name, address, telephone number, and other contact information of a given application; basically, any information that categorizes applications, and technical information about the Web services provided by a given application.

Initially, big vendors such as Microsoft and SAP operated public UDDI services and connected them to an UDDI Business Registry (UBR) as a single entry point for service searches. However, UDDI has not been successful in the area of public Web services. UBR has been discontinued, and only smaller providers such as XMethods still provide an UDDI interface as an *additional* means for accessing their service repository.

The continued development of the UDDI product, however, indicates that the technology is still relevant for intranet and extranet usage. We shall briefly take a more detailed look at what information can be advertised:

- *Application information.* This is contained in a business entity object, which in turn contains information about services, categories, contacts, URLs, and other things necessary to interact with a given application.
- *Service information.* This describes a group of Web services. These are contained in a business service object.
- *Binding information.* The technical details necessary to invoke a Web service. This includes URLs, information about method names, argument types, and so on. The binding template object represents this data.
- *Service specification detail.* This is metadata about the various specifications implemented by a given Web service. These are called tModels in the UDDI specification.

To illustrate the categorization information in UDDI we have browsed the XMethods UDDI repository using the Web Service Explorer⁷ and depicted the information for the GlobalWeather service used in the previous subsection (Listing 4.3). Using key value pairs, one can assign a service to specific categories, similar to using a geographical code or a standard product classification such as UNSPC.⁸

An interesting remark on the annotation can be made here: the key value pairs in the annotation provide good hints about what the service does; however, just reusing standard taxonomies and not making the context explicit does leave some questions open. For example, the `geo` taxonomy unambiguously describes a location; however, by just placing it in the context of a service interface the intended meaning remains unspecified. In this case the

⁷ The Web Service Explorer is a client that can browse and edit UDDI entries as part of the Eclipse Web Tools Platform (WTP) project; see <http://www.eclipse.org/webtools>.

⁸ To illustrate the annotation, we have manually added this information to the service, since the original public service has only minimal information.

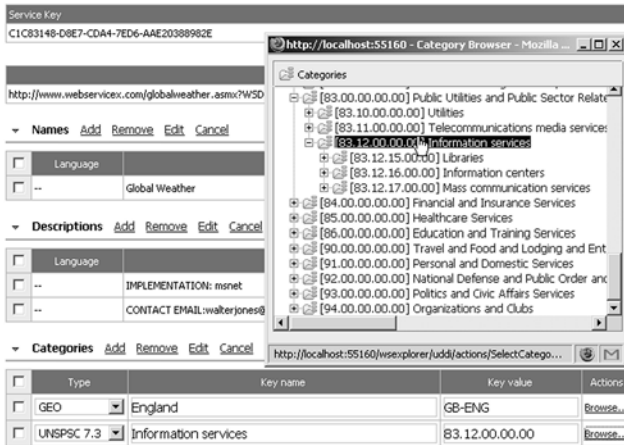


Fig. 4.3. The GlobalWeather service viewed in an UDDI browser

location specifies the physical location of the service endpoint, however one might also interpret it as the “scope” of the service. A human can easily deduce from the interface name `GlobalWeather` that this is not the case, but for a software program interpreting the description, this is less obvious.

The description of published services also contains a link to a WSDL file and, specifically, also to concrete interfaces (port types). Using this link, UDDI provides a machine-processable access to the interface description of an service and therefore, theoretically, also provides the possibility of automatic invocation of services. However we face the problem that both WSDL and UDDI offer a mainly syntactic description.

4.3.4 Related WS-* Standards

A series of related standards and recommendations complete the technologies that are nowadays commonly referred to as “Web services”. The components described so far allow only the description of interactions that follow a simple in/out process. However, typical business processes are more complex and require descriptions of more complex interaction patterns.

The Business Process Execution Language for Web Services (BPEL4WS) [4] allows one to describe such processes on the basis of services described using WSDL. BPEL is an XML-based language for describing an executable process over a series of Web service operations. This allows one to aggregate atomic service operations into a more meaningful service component.

Note that WSDL, in principle, already allows one to group a set of related operations together in a description; however, it provides no means to describe any restriction on the possible sequences that those operations can be invoked in. BPEL4WS can describe such concrete sequences, for example to first invoke first a login operation, followed by subsequent search and buy operations.

The further standards that complement the core specification are often aggregated under the acronym “WS-* standards”. They have been developed by standardization bodies such as the W3C⁹ and OASIS.¹⁰ Without claiming completeness, or discussing them extensively, we shall mention some of them here:

- WS-Security [97] standardizes a SOAP extension to encode security-related information.
- WS-Policy [8] is a standard that enables the description of supported properties of a service, such as security level, and language.
- WS-PolicyAttachment [7] allows one to link policy descriptions with WSDL descriptions.
- WS-Trust [3] can describe trust relationships between business partners.

4.4 Web Services in Reality

It is hard to assess the total spread of this technology. First, it is at present mainly used within or between companies, and second, there is no reliable way to count the publicly available Web services. However, if one considers the behavior of software vendors as an indicator for the adoption of the technology, it is a tremendous success. There are numerous different platforms that allow the easy publication of software components as Web services. The technology is an integral part of the Microsoft .NET environment and of various platforms for Java-based solutions. Even script languages such as PHP provide libraries for easy publication and consumption of Web services. The support is not limited to the technology providers; application vendors such as SAP and Oracle have made Web services an integral part of their products.

This indicates that the technology has indeed been widely adopted; however, of these many services, relatively few are made publicly available. The big public UDDI-based repositories from IBM, Microsoft, and SAP have recently been shut down. The majority of registered services have been toy examples or simple tests by technology enthusiasts. Currently, several portal sites have begun to fill this void by providing a central point of registration. One such portal is StrikeIron,¹¹ which not only allows the registration of Web services, but also serves as a reseller of Web service usage contracts. Having contracts with multiple smaller providers allows StrikeIron to offer companies the ease of aggregated invoicing and contract management. The services available range from currency conversion services, and various forms of SMS or fax services (many available without charge), to commercial offers, such as address validation and services that perform printing and mailing of electronic letters via surface mail delivery.

⁹ <http://www.w3.org>.

¹⁰ <http://www.oasis.org>.

¹¹ <http://strikeiron.com>.

Although UDDI does not at present play a huge role in the market for public repositories, it seems (given the product range of vendors) that it plays an important role in intra- and inter-enterprise usage. Nearly all of the publicly available services use SOAP as the transport protocol and XML Schema for type information.

4.5 What’s Missing in Web Services?

So far, we have presented the pillars of the next generation of Web technologies: we have introduced the basic Web technologies, followed by the vision of the future Web, the Semantic Web. We then discussed Web services as a means to provide an interoperable platform for application integration using Web technologies. However, as we can see in Fig. 4.4, the next obvious step is still missing, and this is what this book is all about. The term “Semantic Web

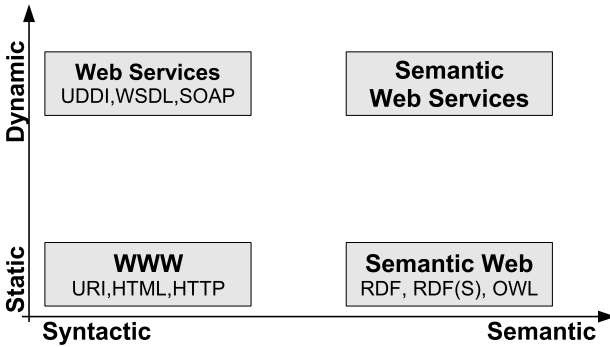


Fig. 4.4. The evolution of the Web

services”, probably first coined in [90], stands for the automation of service usage tasks such as discovery, selection, composition, and enactment of suitable services. This task is accomplished by making services themselves machine-processable. Just like the way in which the Semantic Web promises to make the static content of the Web machine-processable via semantic annotations, the idea of applying similar techniques to Web services is very appealing.

The potential benefits of Semantic Web services have led to the establishment of an important research area, both in academia and in industry. Several initiatives have appeared for semantically annotating Web services, providing various descriptions of Web services and their related aspects, which, in turn, provides various kinds of support for discovery and composition.

Before going into the details of our approach, namely the Web service Modeling Ontology, and related efforts in the remainder of this book, we shall spend the rest of this first introductory part reviewing some more general considerations.

The term “Semantic Web services” comprises two building blocks that could be fruitfully combined towards the automation of service usage:

- The Semantic Web is about adding machine-processable semantics to data. The computer can “understand” the information and therefore process it on behalf of the human user.
- Web services aim to employ the Web as a global infrastructure for distributed computation, for integrating various applications, and for the automation of business processes. The Web will be not only the place where human-readable information is published, but also the place where global computing is realized.

Both trends are visualized in Fig. 4.4. This illustrates the two natural aspects of the evolution of the Web. Web services, with their ability to provide an abstraction to computational resources, turn the existing Web into a more dynamic environment, yet the Web maintains its purely syntactical nature.

The vision of Semantic Web services is to describe and annotate the various aspects of a Web service using explicit, machine-understandable semantics, enabling the automatic location, combination, and use of Web services. The work in the area of the Semantic Web is being applied to Web services in order to minimize the intervention of the human user. Semantic markup will be exploited to automate the tasks of discovering services, executing them, composing them, and enabling seamless interoperation between them [87], thus, enabling intelligent Web services.

The description of Web services in a machine-understandable fashion is expected to have a great impact in the areas of e-commerce and enterprise application integration, as it can enable dynamic and scalable cooperation between different systems and organizations.

Current description standards for Web services such as WSDL have one significant drawback. To a large extent, they are restricted to the syntactic aspects of service interaction. They allow the description of *how* the service can be invoked, which operations may be called, which policies may be supported, etc. However, *what* the service does and in what order its operations have to be called in order to achieve a certain functionality is only, if ever, described in natural language in the entries or comments of a WSDL description or UDDI entry.

We can imagine that the trend of Web services populating the Web will explode in a similar way to the way in which the number of static pages on the Web did before. Thus, when looking for suitable services to achieve a certain task, we face obstacles similar to those we face as in filtering information from the overwhelming amount of data on the current human-readable “static” Web. Annotations by means of Semantic Web technologies are again the big hope. The final convergence of these technologies is essential for the success of Web service technologies on the scale of the Web.

The requirements for semantic descriptions of Web services correspond in principle to the requirements for common Web service technologies. However,

there are small but important differences, such as the use of ontologies as shared vocabularies, which promises a higher degree of automation. In particular, Semantic Web technologies are expected to have the most potential in the following service-related tasks:

- *Discovery.* Before a service published on the Web or within an enterprise can be used in a distributed application, it must first be located. Current technologies such as UDDI support this step only with keyword search, although limited forms of standardized vocabularies such as UNSPC¹² are starting to be more widely used. As opposed to such standards, *semantic annotations of service capabilities* via decentralized ontologies, interconnected via logical axioms, by which inference can determine which services match a goal, can be expressed equally well in ontological terms. These descriptions could also involve more fine-grained notions such *formal descriptions of preconditions and postconditions*, and of the *inputs and outputs* of the service using terms specified in an ontology.
- *Negotiation.* Whenever a suitable provider has been determined to serve a certain goal, it is necessary to negotiate a service instance from the possibly many services that a particular provider can offer. This includes establishment of trust policies, determination of payment modalities, selection of offers, etc., where corresponding semantic annotations are required. For the purpose of automating this task, it is important that a semantic service description not only specifies the functionality of a service, but also includes *nonfunctional aspects*, such as supported policies, and security protocols.
- *Composition.* In cases where a particular goal cannot be achieved by means of a single Web service, semantic descriptions can at least help to determine a promising combination of multiple Web services to achieve the requested functionality. Having learned the lessons of 20 years of AI research, we would be naive to expect promises to be fulfilled that efforts towards automated programming have not yet fulfilled. Nevertheless, clever selection of services and lightweight planning or scheduling can, hopefully, ease the pain of manual integration. Composition requires not only the semantic annotation of the overall capabilities of a service, but also a *behavioral description* of how to interact with the Web service, in order to achieve a certain functionality.
- *Invocation.* After a (combination of) services has been selected, the final step is execution. To this end, *possible input and output values* need to be extracted from the semantic goal description and *adapted* to the negotiated message formats and communication protocols. Ideally, *monitoring and transaction control* should be supported during invocation.

¹² United Nations Standard Products and Services Code; see <http://www.unspsc.org>.

Most approaches towards achieving Semantic Web services, i.e. automation of the above-mentioned tasks, aim at the annotation of Web service descriptions per service, by extending or complementing current technologies around WSDL, SOAP, UDDI, etc. with semantic annotations.

4.6 Summary

In this chapter, we have learned about Web service technologies. We discussed some fundamental principles of what service-oriented architectures consist of, such as loose coupling, contractibility, discoverability, and addressability in a distributed, open environment. We found that Web service technologies could to some extent to be viewed as “yet another middleware solution”, though with some differences from their predecessors:

- Web services, for the first time have broad support in joint standardization efforts by the global players instead of previous vendor-specific solutions (DCOM, etc.).
- Web services are not an all-in-one framework to solve it all (like CORBA), but a family of (i) simple, (ii) modular, (iii) interrelated, and (iv) extensible standards.
- Web services rely on and build upon Web protocols such as HTTP, thus allowing the use of widely supported technologies with a large community of developers and users behind them.
- All Web service technologies are based fundamentally on XML as the exchange and message format of choice.

We have discussed the three pillar technologies for making the Web services vision of interoperable services on the Web scale work: SOAP provides an XML-based protocol and messaging mechanism; WSDL allows one to describe and advertise Web service interfaces in a way that exposes enough detail about the supported operations and protocol bindings to enable a developer to integrate the service into his/her own application, abstracting away from implementation-specific details. UDDI has been proposed as a registry infrastructure which allows one to register and publish service advertisements and to support discovery. More standards for policy negotiation, and for describing security and trust-related information, etc. are on the way to complete the family of technologies subsumed under the term “Web services”.

There is still a big obstacle to automation of the use of these technologies and service-oriented architectures in general though: Web services described through WSDL provide service endpoints in such an architecture, but there could soon be several thousands of services capable of composing any required specific application.

Web services allow easy adaption and integration by deploying machinery of XML; however, they do not let you express what a service actually

does. For instance, if services use different XML tags to annotate their operations/messages, then it is not possible to find out that two operations offered by different services in different WSDLs actually have the same function.

Similarly, keyword-based search in natural-language descriptions in a repository, as provided by UDDI, might well be insufficient in such realistic scenarios. The XML message formats used by different services might still be syntactically incompatible, leaving the integration effort to a human programmer.

Finally, we have outlined the potential of combining the ideas of the Semantic Web and Web services. The two are complementary trends that are about to transform the Web from being for humans only into a network that involves automated communication between computers. These advances will provide support for human interaction at a much higher level than what is possible with current Web technologies:

- The Semantic Web adds machine-processable semantics to data.
- Web services promise to provide smooth and painless integration of arbitrary applications.

Eventually, the combination of these two trends in what is called “Semantic Web services” promises a fully mechanized Web for computer interaction, where humans organize their collaboration and their business relationships.

So, what we really need are provisional metadata annotations about the semantics of services, using formal languages with an agreed meaning. Reflecting what we have already learned in Chapter 3, this is just what Semantic Web technologies are about. The tempting challenge that this book is aiming at is to combine these two cornerstones, the Semantic Web and Web services.

The Web Service Modeling Ontology

Introduction to WSMO

The Web Service Modeling Ontology (WSMO) [116, 117] is aimed at describing all relevant aspects related to general services that are accessible through a Web service interface. Its ultimate goal is to enable the (total or partial) automation of the tasks (e.g. discovery, selection, composition, mediation, execution, and monitoring) involved in both intra- and inter-enterprise integration of Web services. WSMO has its conceptual basis in the Web Service Modeling Framework (WSMF) [41], and aims at refining and extending this framework and developing a formal ontology and set of languages. Before we describe all of the concepts of WSMO in detail, we give an overview of the design principles of WSMO, its core elements, and the language used to define the elements that are part of the conceptual model of WSMO.

5.1 WSMO Design Principles

WSMO provides ontological specifications for the core elements of Semantic Web services. In fact, Semantic Web services are aimed at producing an integrated technology for the next generation of the Web by combining Semantic Web technologies and Web services, thereby turning the Internet from an information repository for human consumption into a worldwide system for distributed Web computing. Therefore, appropriate frameworks for Semantic Web services need to integrate the basic Web design principles, the design principles defined for the Semantic Web, and design principles for distributed, service-oriented computing on the Web. WSMO is therefore based on the following design principles:

- *Web compliance.* WSMO inherits the concept of the URI (Universal Resource Identifier) for unique identification of resources as the essential design principle of the World Wide Web. Moreover, WSMO adopts the concept of namespaces for denoting consistent information spaces, and supports XML and other W3C Web technology recommendations, as well as the decentralization of resources.

- *Ontology-based.* Ontologies are used as data models throughout WSMO, meaning that all resource descriptions and all data interchanged during service usage are based on ontologies. Ontologies are a widely accepted state-of-the-art knowledge representation, and have thus been identified as the central enabling technology for the Semantic Web. The extensive usage of ontologies allows semantically enhanced information processing and support for interoperability; WSMO also supports the ontology languages defined for the Semantic Web.
- *Strict decoupling.* Decoupling denotes that WSMO resources are defined in isolation, meaning that each resource is specified independently without regard to possible usage or interactions with other resources. This complies with the open and distributed nature of the Web.
- *Centrality of mediation.* As a complementary design principle to strict decoupling, mediation addresses the handling of the heterogeneities that naturally arise in open environments. Heterogeneity can occur in terms of data, the underlying ontology, the protocol, or the process. WSMO recognizes the importance of mediation for the successful deployment of Web services by making mediation a first-class component of the framework.
- *Ontological role separation.* Users or more generally, clients exist in specific contexts which will not be the same as those of the available Web services. For example, a user may wish to book a holiday according to preferences about weather, culture, and childcare, whereas Web services will typically cover airline travel and hotel availability. The underlying epistemology of WSMO differentiates between the desires of users or clients and the available services.
- *Description versus implementation.* WSMO differentiates between the descriptions of elements of Semantic Web services (description) and executable technologies (implementation). While the former requires a concise, sound description framework based on appropriate formalisms in order to provide concise semantic descriptions, the latter is concerned with the support of existing and emerging execution technologies for the Semantic Web and Web services. WSMO aims to provide an appropriate ontological description model, and to be compliant with existing and emerging technologies.
- *Execution semantics.* In order to verify the WSMO specification, the formal execution semantics of reference implementations such as WSMX, as well as other WSMO-enabled systems, provide the technical realization of WSMO.
- *Services versus Web services.* A Web service is a computational entity which is able to achieve a user's goal by invocation. A service, in contrast, is the actual value provided by this invocation [6, 111].¹ WSMO provides

¹ Note that [111] also distinguishes between a computational entity in general and a Web service, where the former does not necessarily have a Web-accessible interface. WSMO does not make this distinction.

a means to describe Web services that provide access (searching, buying, etc.) to services. WSMO is designed as a means to describe the former and not to replace the functionality of the latter.

5.2 Top-Level Elements of WSMO

Following the key aspects identified in the Web Service Modeling Framework, WSMO identifies four top-level elements as the main concepts: ontologies, services, goals, and mediators (Fig. 5.1). These have to be described in order to describe Semantic Web services:

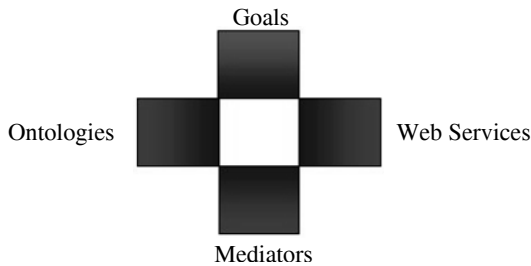


Fig. 5.1. WSMO core elements

As introduced in Chapter 3, *ontologies* provide the terminology used and are the key element for the success of Semantic Web services. WSMO describes an epistemology of ontologies, i.e. the conceptual building blocks such as concepts and relations.

The actual *Web services* are computational entities that provide some value in a certain domain. Web service descriptions consist of their capability, their interfaces, and their orchestration—the internal workings of the service.

Goals describe aspects related to user desires with respect to the requested functionality; again, ontologies can be used in order to define the domain terminology used to describe the relevant aspects of goals. Goals model the user's view of the Web service usage process and therefore are a separate top-level entity in WSMO.

Finally, *Mediators* describe elements that handle interoperability problems between different elements, for example two different ontologies or services. We envision mediators as a core concept for resolving incompatibilities on the data, process, and protocol levels, i.e. they will be used in order to resolve mismatches between different terminologies (data level), to communicate between services (protocol level), and to combine Web services and goals (process level).

These top-level elements of WSMO are discussed and defined in Chapter 6.

5.3 The Language for Defining WSMO

WSMO is meant to be a metamodel for aspects of Semantic Web services. To introduce this model, we make use of the Meta-Object Facility (MOF) [103] specification, which defines an abstract language and framework for specifying, constructing, and managing technology-neutral metamodels. MOF defines a metadata architecture consisting of four layers, namely:

- The *information layer* comprises the data we want to describe.
- The *model layer* comprises the metadata that describe the data in the information layer.
- The *metamodel layer* comprises the descriptions that define the structure and semantics of the metadata.
- The *metametamodel layer* comprises the description of the structure and semantics of the metamodel.

In terms of the four MOF layers, the language in which WSMO is defined corresponds to the metamodel layer; WSMO itself constitutes the meta-model layer; the specifications of the ontologies, services, goals, and mediators constitute the model layer; and the actual data described by the ontologies and exchanged between Web services constitute the information layer. Figure 5.2 shows the relation between WSMO and the MOF layered architecture.

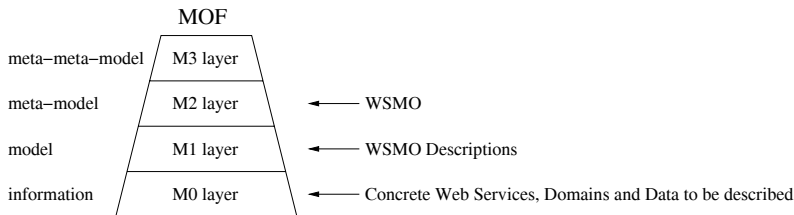


Fig. 5.2. The relation between WSMO and MOF

The MOF metamodeling construct used most in the definition of WSMO is the *Class* construct (and, implicitly, its class generalization (*sub-Class*) construct), together with its *Attributes*, the *type* of the *Attributes*, and their *multiplicity* specifications. In defining WSMO, the following assumptions are made:

- Every *Attribute* has its *multiplicity* set to “multi-valued” by default; when an *Attribute* requires its *multiplicity* to be set to “single-valued”, this is explicitly stated in the listings where WSMO elements are defined.
- Some WSMO elements define *Attributes* that take values from a union of several types, a feature that is not directly supported by the MOF metamodeling constructs; this can be simulated in MOF by defining a new *Class* as a *super-Class* of all the types required in the definition of

the *Attribute*, representing the union of the individual types, with the *Constraint* that each instance of this new *Class* is an instance of at least one of the types which are used in the union; to define this new *Class* in WSMO, we use curly brackets, and enumerate the *Classes* that describe the types required for the definition of the attribute.

In the remainder of this Part we use listings constructed with the MOF metamodel to illustrate the structure of WSMO where this will support the understanding of the overall structure.

The Concepts of WSMO

In the case of complex systems, one cannot make sense of the constituent parts without first understanding the whole. Parts can only make sense in the context of a greater whole.

Our hope is that the previous chapter has provided the reader with some sort of conceptual understanding of WSMO (although only a general overview at best), at least enough to be able to understand the contention that WSMO does not fall prey to the problems of the family of complex systems. Nevertheless, the present chapter will follow the “grandfatherly” advice of complexity theory just the same, attempting to “make sense of the constituent parts”.

In this chapter, we describe further the core elements of WSMO: we present ontologies in Section 6.1, Web services in Section 6.2, goals in Section 6.3, and mediators in Section 6.4. Section 6.5 presents an informal discussion of the nonfunctional properties that are part of the WSMO conceptual model.

6.1 Ontologies

Ontologies were introduced in Chapter 3 and although there are currently several standardizations efforts for ontology languages [60, 32, 64], none of them have the desired expressivity and computational properties that are required to describe Web services at a sufficient level of granularity. In the following, we shall define an epistemological model which is general enough to intuitively capture existing languages. In the next chapter of this book we shall present a concrete language specifically designed to express this metamodel.

Now we present the conceptual model and introduce the elements that constitute an ontology using the MOF notation, defining the class “ontology” with its attributes. The following listing specifies the building blocks for an ontology. Note that all attributes are optional. Thus an ontology can have multiple instances of all elements; however, this is not a requirement.

```

Class ontology
  hasNonFunctionalProperty type nonFunctionalProperty
  importsOntology type ontology
  usesMediator type ooMediator
  hasConcept type concept
  hasRelation type relation
  hasFunction type function
  hasInstance type instance
  hasAxiom type axiom

```

6.1.1 Nonfunctional Properties

Nonfunctional properties are applicable to all the definitions of WSMO elements. Note that in some cases WSMO specifically recommends that certain properties should be applicable to an element, but this does not impact the general considerations presented here. Nonfunctional properties are mainly used to describe nonfunctional aspects such as the creator and the creation date, and to provide natural-language descriptions, etc. The elements defined by the Dublin Core Metadata Initiative [134] are taken as a starting point. Dublin Core is a set of attributes that define a standard for cross-domain information resource description. In WSMO, we use URIs for identification of elements, which are reused. WSMO proposes several extensions to this set by introducing new attributes, such as the version element which can contain information about the particular version of an element. The set of nonfunctional properties that WSMO recommends are informally presented in Section 6.5.

6.1.2 Imported ontologies

Building an ontology for a particular problem domain can be a rather cumbersome and complex task. One standard way to deal with the complexity is modularization. Imported ontologies allow a modular approach to ontology design and can be used as long as no conflicts need to be resolved between the ontologies. When ontologies are imported, all statements of the imported ontology will be virtually included in the importing ontology. Every WSMO top-level entity may use this import facility to include the logical definition of the vocabulary used.

6.1.3 Mediators

When ontologies are imported in realistic scenarios, some steps are needed for aligning, merging and transforming the imported ontologies in order to resolve ontology mismatches. For this reason and in line with the basic design principles underlying the WSMF, ontology mediators (OO mediators), which are described in detail in Section 6.4, are used when an alignment of an imported ontology is necessary. Such an alignment can be done by simply renaming concepts, attributes or the like. Just like the *importsOntology* statement, the *usesMediator* statement is applicable to all top-level elements, however depending on the element, different mediators may be used.

6.1.4 Concepts

Concepts constitute the basic elements of the agreed terminology for a problem domain. From a high-level perspective, a concept - described by a concept definition - provides attributes with names and types. Specified more formally in MOF, a concept is made up of the following elements:

```

Class concept
  hasNonFunctionalProperties type nonFunctionalProperties
  hasSuperConcept type concept
  hasAttribute type attribute
  hasDefinition type logicalExpression multiplicity = single-valued

```

Furthermore, a concept can be a subconcept of several (or possibly none) direct superconcepts, as specified by an “is-a” relation. In the WSMO model, each *concept* can have a finite number of *concepts* that serve as a superconcept for some other concept. Being a subconcept of some other concept means, in particular, that a concept inherits the signature of this superconcept and the corresponding constraints.

A concept provides a (possibly empty) set of attributes that represent named slots for the data values of instances. An attribute specifies a slot of a concept by fixing the name of the slot and a logical constraint on the possible values filling that slot, which, in a simple case, can be another concept. Hence, this logical expression can be interpreted as a typing constraint.

Additionally, a concept is defined by a logical expression. This means that axioms can be asserted about a concept that refine its meaning, for example with nuances that are not expressible by attributes or an “is-a” hierarchy. A logical expression can be used to refine the semantics of the concept. More precisely, the logical expression defines (or restricts) the extension (i.e. the set of instances) of the concept. More details of the concrete language in which this is done are given in Chapter 7.

6.1.5 Relations

Relations are used in order to model interdependencies between several concepts (or instances of these concepts). The arity of relations is not limited. Specified more formally in MOF, a relation is made up of the following elements:

```

Class relation
  hasNonFunctionalProperties type nonFunctionalProperties
  hasSuperRelation type relation
  hasParameter type parameter
  hasDefinition type logicalExpression multiplicity = single-valued

```

Every *relation* can have a finite set of *relations*, of which a defined relation is declared as being a subrelation. Being a subrelation of some other relation means, in particular, that a relation inherits the signature of this superrelation and the corresponding constraints. Furthermore, the set of tuples belonging to the relation (and the extension of the relation) is a subset of each of the

extensions of the superrelations. In the example given earlier, we can define airline distance as a subrelation of the general distance relation.

Similarly to attributes for concepts, each relation has a possibly empty set of named parameters. If no named parameters are given, a unnamed, ordered list is assumed. Each parameter is single-valued and can have a range restriction in the form of a concept. As for concepts, a logical expression defining the set of instances (n -ary tuples, if n is the arity of the relation) can be specified.

6.1.6 Functions

A function is a special relation, with a unary range and an n -ary domain (the parameters inherited from relations), where the range specifies the return value. Functions can be used to represent and exploit built-in predicates of common datatypes. Their semantics can be captured externally by means of an oracle or the semantics can be formalized by assigning a logical expression to a particular relation. The logical representation of functions is almost the same as for relations. A typical example of the use of oracles is provided by “built-ins”. These are functions such as “round” that are evaluated outside the logical theory.

6.1.7 Instances

Instances are defined either explicitly or by a link to an instance store, i.e. an external storage of instances and their values. Specified more formally in MOF, an instance is made up of the following elements:

```
Class instance
  hasNonFunctionalProperties type nonFunctionalProperties
  hasType type concept
  hasAttributeValues type attributeValue
```

Instances of relations (with arity n) can be seen as n -tuples of instances of the concepts that are specified as the parameters of the relation.

In general, instances do not need to be specified using an explicit notation of the kind used above. In particular, when a huge number of instances exist, a link to a data store can be used [73]. Basically, the approach is to integrate large sets of instances which already exist on storage devices by means of sending queries to external storage devices or oracles.

6.1.8 Axioms

An axiom is a logical expression together with its nonfunctional properties. A detailed discussion of axioms can be found in Chapter 7.

6.2 Web Services

The *Web service* element of WSMO provides a conceptual model (a meta-model in MOF terms) for describing, in an explicit and unified manner, all aspects of a service, including its nonfunctional properties, its functionality, and the interfaces needed to obtain it. An unambiguous model of services with a well-defined semantics can be processed and interpreted by computers without human intervention, enabling the automation of the tasks involved in the usage of Web services, for example discovery, selection, composition, mediation, execution, and monitoring.

As observed in [111], the word “service” can be understood in several different ways, with slightly different meanings: as the provision of value in some domain, as a software entity able to provide something of value, and as a means of interacting online with a service provider.

WSMO provides a unified view of a service; the value that the service can provide is captured by its *capability*, and the means to interact with the service provider to request the actual performance of the service, or to negotiate aspects of its provision, is captured by its *interfaces*. The software entity that provides the service is transparent to us. We are concerned only with its style of interaction and with what other services are used in order to provide the value described in the capability.

Notice that in WSMO, the interaction with a service can be realized by using Web services in the WSDL sense. However, we are not restricted to WSDL for the grounding of services. The elements identified in WSMO are equally applicable to other underlying technologies, such as Triple Space computing (Section 12.4). The directions that we have taken for the grounding to WSDL are described in Section 9.5.

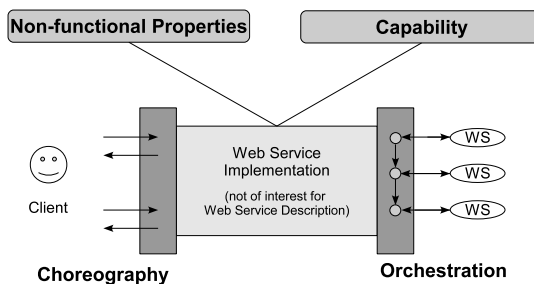


Fig. 6.1. WSMO Web service – general description

The main elements of a service description, depicted in Fig. 6.1, are a capability, describing the value that the service can provide, and one or more interfaces, in which the choreography and the orchestration of the service are described. The choreography specifies how the service achieves its capability

by means of interaction with the user – i.e. the communication with the user of the service. The orchestration specifies how the service achieves its capability by making use of other services – i.e. the coordination of other services.

More precisely, the WSMO Web service element is defined as follows:

```

Class service
  hasNonFunctionalProperty type nonFunctionalProperty
  importsOntology type ontology
  usesMediator type {ooMediator, wwMediator}
  hasCapability type capability multiplicity = single-valued
  hasInterface type interface

```

- *Nonfunctional properties.* The nonfunctional properties of a Web service are the aspects of the Web service that are not directly related to its functionality. Besides the nonfunctional properties already presented, they consist of Web-service-specific elements such as *accuracy* (the error rate generated by the service), *financial* (the cost-related and charging-related properties of a service [105]), *network-related QoS* (QoS mechanisms operating in the transport network that are independent of the service), *owner* (the person or organization to which the service belongs), *performance* (how fast a service request can be completed), *reliability* (the ability of a service to perform its functions, i.e. to maintain its service quality), *robustness* (the ability of the service to function correctly in the presence of incomplete or invalid inputs), *scalability* (the ability of the service to process more requests in a certain time interval), *security* (the ability of a service to provide authentication, authorization, confidentiality, traceability/auditability, data encryption, and nonrepudiation), *transactional* (the transactional properties of the service), *trust* (the trustworthiness of the service), and *version*. The nonfunctional properties are to be mainly used for the discovery and selection of services; however, they contain information that is also suitable for negotiation.
- *Imported ontologies.* Imported ontologies are used to import the explicit, formal vocabulary used in the specification of a Web service.
- *Use of mediators.* A Web service uses mediators in the following situations. (1) When heterogeneous terminologies are used and conflicts between them arise, a service can import ontologies using ontology mediators, as explained in Section 6.1.3. (2) When the service needs to cope with process and protocol heterogeneity when interacting with other services. In this case, a *wwMediator* is used. For a more detailed description of mediators, see Section 6.4.
- *Capability.* The capability describes the real service provided for example booking of train tickets. A more detailed description of capabilities is given in Section 6.2.1.
- *Interface.* An interface describes the interface of the Web service to be used to achieve the service described. Further details are given in Section 6.2.2.

6.2.1 Capability

The functionality offered by a given service is described by its capability. This is expressed by the state of the world before the service is executed and the state of the world after successful service provision. The capability of a service can be used for discovery and selection purposes, i.e. the capability is used by the requester to determine whether the service meets its needs.

Specified more formally in MOF, a capability is made up of the following elements:

```

Class capability
  hasNonFunctionalProperty type nonFunctionalProperty
  importsOntology type ontology
  usesMediator type ooMediator
  hasPrecondition type axiom
  hasAssumption type axiom
  hasPostcondition type axiom
  hasEffect type axiom

```

The set of nonfunctional properties that can be attached to a capability is presented in Section 6.5. The attributes of *imported ontologies* and *mediators* play the same role for ontologies.

- *Preconditions.* Preconditions in the description of a capability specify the required state of the information space before the execution of the service, i.e. they specify what information a Web service expects in order to provide the service. Preconditions constrain the set of states of the information space such that any state satisfying these constraints can serve as a valid starting state (in the information space) for executing the service in a defined manner.
- *Assumptions.* Assumptions in the description of a capability describe the state of the world which is assumed before the execution of the service. Otherwise, the successful provision of the service is not guaranteed. As opposed to preconditions, assumptions are not necessarily checked by the service. We make this distinction in order to allow the explicit idea of conditions on the state of the world that are outside the information space.
- *Postconditions.* Postconditions in the description of a capability describe the state of the information space that is guaranteed to be reached after the successful execution of the service; they also describe the relation between the information that is provided to the service and its results.
- *Effects.* Effects in the description of a capability describe the state of the world that is guaranteed to be reached after the the successful execution of the service, i.e. if the preconditions and the assumptions of the service are satisfied.

6.2.2 Interfaces

An interface describes how the functionality of a service can be achieved (i.e. how the capability of a service can be fulfilled) by providing a twofold view

of the operational competence of the service: (1) *choreography* decomposes a capability in terms of interaction with the service, and (2) *orchestration* decomposes a capability in terms of the functionality required from other services.

This distinction reflects the difference between communication and cooperation. The choreography defines how to communicate with the service in order to consume its functionality. The orchestration defines how the overall functionality is achieved by the cooperation of more elementary service providers.

The Web service interface is meant primarily for the description of the behavior of Web Services and is presented in a way that is suitable for software agents to determine the behavior of the service and reason about it. It might be also useful for discovery and selection purposes, and in this description, a connection to existing Web service specifications, for example WSDL, could also be specified.

The definition of an interface is given below:

```
Class interface
  hasNonFunctionalProperty type nonFunctionalProperty
  importsOntology type ontology
  usesMediator type ooMediator
  hasChoreography type choreography
  hasOrchestration type orchestration
```

Choreography

The choreography interface describes the behavior of the service from the client's point of view; this definition is in accordance to the one given in the W3C Glossary:¹ Web service choreography concerns the interactions of services with their users. Any user of a Web service, automated or otherwise, is a client of that service. These users may, in turn, be other Web services, applications, or human beings.

The state-based mechanism for describing WSMO choreography and orchestration interfaces is based on the abstract state machine (ASM) [58] methodology. An ASM is used to abstractly describe the behavior of a service with respect to an invocation instance of that service. We have chosen an ASM model for the description of such interfaces since such a service invocation (e.g. the purchase of a book from Amazon.com) may consist of a number of interaction steps. These interactions can be described by a stateful abstract machine.

ASMs have been chosen as the underlying model for the following three reasons:

- *Minimality.* ASMs provide a minimal set of modeling primitives, i.e. they enforce minimal Ontological commitments. Therefore, they do not intro-

¹ <http://www.w3.org/2003/glossary>.

duce any ad hoc elements that would be questionable if they were to be included into a standard proposal.

- *Maximality.* ASMs are expressive enough to model any aspect around computation.
- *Formality.* ASMs provide a rigid framework to express dynamics.

In the following, we present a short overview of ASMs and then address the core conceptual model for WSMO choreographies.

Basic Abstract State Machines

Abstract state machines, formerly known as evolving algebras [58], provide a means to describe systems in a precise manner using a semantically well-founded mathematical notation. The core principles are the definition of ground models and the design of systems by refinements. Ground models define the requirements and operations of the system, expressed in mathematical form. Refinements allow the expression of the classical divide-and-conquer methodology for system design in a precise notation, which can be used for abstraction, validation, and verification of the system at any given stage in the development process.

Abstract state machines can be divided into two main categories, namely basic ASMs and multiagent ASMs. The former express the behavior of a system within a given environment. Multiagent ASMs express the behavior of the system in terms of multiple entities that are collaborating to achieve a functionality. For the description of the behavior of a single party, we are interested only in basic ASMs.

A basic ASM is defined in terms of a state signature plus a finite set of transition rules, which are executed in parallel. It may involve nondeterministic behavior. Finite state machines can be viewed as a special case of such basic ASMs. The state signature for a classical ASM usually consists simply of a set of static and dynamic functions, which can be locally updated by update rules. Dynamic functions can be classified into four other categories, namely, *controlled*, *monitored* (or *in*), *interaction* (or *shared*), and *out*. Controlled functions are directly updatable by the rules of a machine M only. Thus, they can be neither read nor updated by the environment. Monitored functions can only be updated by the environment and read by the machine M , and hence constitute the externally controlled part of the state. Shared functions can be read and updated by both the environment and the rules of the machine M . Out functions can be updated and read by the environment, but remain unreadable by M . Furthermore, ASMs define *derived functions*. These are functions that are updatable by neither the machine nor the environment but are instead defined in terms of other static and dynamic (and derived) functions.

WSMO Choreography Model

WSMO choreography deals with the interactions of the Web service from the client's perspective. We base the description of the behavior of a single service exposed to its client on the basic ASM model. WSMO choreography interface descriptions inherit the core principles of such ASMs, which can be summarized as follows: (1) they are state-based, (2) they represent a state by a signature, and (3) they model state changes by transition rules that change the values of functions and relations defined by the signature of the algebra (which, in our context, is a nonempty set of ontologies).

In order to define the signature we use a WSMO ontology, i.e. definitions of concepts, their attributes, and relations and axioms over these. Instead of dynamic changes of function values as represented by dynamic functions in ASMs, we allow the dynamic modification of instances and attribute values in the state ontology. Note that the choreography interface describes the interaction with respect to a single instance of the choreography. The key extension compared with the basic ASMs described above is that the machine signature is defined in terms of a WSMO ontology (or possibly more than one), and the logical language used for expressing conditions is WSML. This leads us to the notion of evolving ontologies (derived from the notion of evolving algebras, the name initially used for ASMs), since the choreography ASM in fact changes the values of concepts and relations within ontologies.

Taking the ASM methodology as a starting point, a WSMO choreography consists of three elements, which are defined as follows:

```

Class choreography
  hasNonFunctionalProperties type nonFunctionalProperties
  hasStateSignature type stateSignature
  hasTransitionRules type transitionRules

```

The set of *nonfunctional properties* that can be attached to a capability is presented in Section 6.5. The *state signature* defines the state ontology used by the service, together with the definition of the types of modes that the concepts and relations may have.

State Signature

The signature of the machine is defined by (1) importing an ontology which defines the state signature over which the transition rules are executed, (2) an optional set of OO mediators if the imported state ontologies are heterogenous, (3) a set of statements defining the modes of the concepts and relations, and (4) a set of update functions. The default mode for concepts of the imported ontologies which are not listed explicitly in the mode statements is static. Note it is not allowed to assign either “in” or “out” to concepts which have explicitly defined any instance data in the imported ontologies by the state signature. The definition of the state signature is expressed as follows:


```

Class stateSignature
  hasNonFunctionalProperties type nonFunctionalProperties
  importsOntology type ontology
  usesMediator type ooMediator
  hasStatic type mode
  hasIn type mode
  hasOut type mode
  hasShared type mode
  hasControlled type mode

Class mode sub—Class {concept, relation}
  hasGrounding type grounding

```

The state for a given signature of a WSMO choreography is defined by all legal WSMO identifiers, concepts, relations, and axioms. The elements that can change and are used to express different states of a choreography, are instances of concepts and relations, which are used similarly to locations in ASMs. These changes are expressed in terms of the creation of new instances or changes of attribute values.

In a similar way to the classification of locations and functions in ASMs, the concepts and relations of an ontology are marked to support a particular role (or mode). These roles are of five different types: static, in, out, shared, and controlled.

Since Web services deal with actual instance data, the classification is inherited by instances of the correspondingly classified concepts and relations. That is, instances of controlled concepts and relations can only be created and modified by the choreography interface; instances of “in” concepts can only be read by the choreography interface; instances of “out” concepts can only be created by the choreography interface, and cannot be read or modified after their creation. Instances of shared concepts and relations can be read and written by both the choreography interface and, possibly, the environment; they can also be modified after creation. In the future, we expect shared concepts to be particularly important for grounding alternatives to WSDL which do not rely on strict message passing, such as semantically enabled Tuple Spaces [39].

Transition Rules

Unlike basic ASMs, the most basic forms of rules are not assignments, but instead deal with basic operations on instance data, for example adding instances to the signature ontology, removing instances, and updating instances. To this end, we define atomic update functions to add, delete, and update instances. This in turn allows us not only to add and remove instances to/from concepts and relations, but also to add and remove attribute values for particular instances. In WSMO choreography, these basic updates are defined as a set of fact modifiers, which are of four different types:

- add(fact);
- delete(fact);

- `update(factnew);`
- `update(factold → factnew).`

More complex transition rules are defined recursively, analogously to classical ASMs, by “if-then”, “forall” and “choose” rules:

```

if Condition then Rules endif
forall Variables with Condition do Rules endforall
choose Variables with Condition do Rules endchoose

```

Orchestration

Orchestration describes how a service makes use of other services in order to achieve its capability. In many real scenarios, a service is provided by using and interacting with services provided by other applications or businesses. For example, the booking of a trip might involve the use of another service for validating a credit card and charging it with the correspondent amount, and the user of the booking service may want to know which other business organizations they are implicitly going to deal with.

WSMO introduces the orchestration element into the description of a service to reflect such dependencies. WSMO orchestration allows the use of statically or dynamically selected services. In the former case, a concrete service will be selected at design time. In the latter case, the service will only describe the goal that has to be fulfilled in order to provide the service. This goal will be used to select at runtime an available service that fulfills it, and the service user could influence this choice.

It is envisioned that orchestration should make use of the multiagent asynchronous ASM model to describe the interactions between Web services and goals. These aspects are still to be investigated further, and will be defined in future publications.

6.3 Goals

Goals are used in WSMO to describe a user’s desires. They provide the means to specify the requester-side objectives when a Web service is consulted, describing at a high level a concrete task to be achieved.

Goals are representations of objectives for which fulfillment is sought through the execution of Web services. They can be descriptions of services that would potentially satisfy the user’s desires.

Note that WSMO completely decouples the objectives that a requester has, i.e. his/her goal, from the services that can actually fulfill such a goal. Specified more formally in MOF, a goal is made up of the following elements:

```

Class goal
hasNonFunctionalProperty type nonFunctionalProperty
importsOntology type ontology
usesMediator type {ooMediator, ggMediator}
requestsCapability type capability multiplicity = single-valued
requestsInterface type interface

```

- *Nonfunctional properties.* Given the fact that a goal can represent a service that would potentially satisfy the user's desires, the set of nonfunctional properties that can be attached to a goal is similar to that attached to a Web services (see Section 6.2). An extra nonfunctional property can be attached to a goal, namely *type of match*, which represents the type of match desired for a particular goal (under the assumption of set-based modeling, this can be an exact match, a match where the goal description is a subset of the Web service description or a match where the Web Service description is a subset of the goal description). For a detailed discussion, see Section 9.2.
- *Imported ontologies.* A goal uses imported ontologies as a terminology to define the other elements that are part of the goal, as long as no conflicts need to be resolved.
- *Nonfunctional properties.* A goal uses mediators in the following situations. (1) When heterogeneous terminologies are used, conflicts between them may arise; in these cases, a service can import ontologies using ontology mediators (*OO mediators*). (2) When a goal reuses already existing goals, for example by refining them, *GG mediators* are used (these are explained in more detail in Section 6.4).
- *Requested Capability.* The requested capability in the definition of a goal describes the capability of the services that the user would like to have.
- *Interface.* The interface in the definition of a goal describes the interface of the service that the user would like to have and interact with.

6.4 Mediators

Mediation is concerned with handling heterogeneity, i.e. resolving possible mismatches between resources that ought to be interoperable. Heterogeneity naturally arises in open and distributed environments, and thus in the application areas of Semantic Web services, WSMO defines the concept of mediators as a top-level concept.

Mediator-orientated architectures, as introduced in [135], specify a mediator as an entity for establishing interoperability of resources that are not compatible prior to resolving mismatches between them at run time. The approach to mediation that is aspired to relies on declarative description of resources, whereupon mechanisms for resolving mismatches work on a structural and semantic level. This allows the defining of generic, domain-independent mediation facilities and the reuse of mediators. Concerning the need for mediation within Semantic Web services, WSMO identifies three levels of mediation:

1. **Data-level mediation:** mediation between heterogeneous data sources and transfer protocols; within ontology-based frameworks such as WSMO, this is mainly concerned with ontology integration.
2. **Functional-level mediation:** mediation between heterogeneous functionalities requested and provided; in WSMO, this relates to capability descriptions of goals and Web services that are similar but do not match precisely.
3. **Process-level mediation:** mediation between heterogeneous communication protocols and business processes; in WSMO, this relates mainly to choreographies of Web services that ought to interact with the mismatch handling on the business logic level of Web services (which is related to the orchestration of Web Services).

WSMO mediators realize a mediation-orientated architecture for Semantic Web services, thereby providing an infrastructure for handling heterogeneities that may possibly arise between WSMO components and realizing the design concepts of strong decoupling and strong mediation. We explain the WSMO mediation architecture in further detail in Section 9.3.

6.4.1 Description of a Mediator

A WSMO mediator connects WSMO components and resolves mismatches between them. The following gives the general definition:

```

Class mediator
  hasNonFunctionalProperty type nonFunctionalProperty
  importsOntology type ontology
  hasSource type {ontology, goal, webService, mediator}
  hasTarget type {ontology, goal, webService, mediator}
  hasMediationService type {webService, goal, wwMediator}

```

- *Nonfunctional properties.* As a mediator can be provided as a service, the same nonfunctional properties as for Web services are used (see Section 6.2 for what these nonfunctional properties consist of).
- *Imported ontologies.* Imported ontologies are used to import the explicit, formal vocabulary used in the specification of a mediator; they play the same role as in the case of ontologies or any other element defined by WSMO.
- *Source.* The source component of a mediator defines the resources for which heterogeneities are resolved; a mediator can have several source components.
- *Target.* The target component of a mediator is the component that receives the mediated source components.
- *Mediation service.* The mediation service defines the mediation facility applied for resolving mismatches. This can be defined in various ways: directly (i.e. explicitly linking to a mediation service); via a goal that

specifies the desired mediation facility which is then detected by a discovery mechanism; or via another mediator when a mediation service is to be used that is not interoperable with the first mediator.

6.4.2 WSMO Mediator Types

In order to allow the resolving of heterogeneities between different WSMO components, WSMO defines several different types of mediators for connecting different WSMO components and overcoming the heterogeneities that can arise between those components: OO mediators, GG mediators, WG mediators, and WW mediators. All mediators are subclasses of the general WSMO mediator class defined above, where the prefix indicates the components connected by the particular type of mediator. The following explains the various WSMO mediator types.

OO Mediators

OO mediators resolve mismatches between ontologies and provide mediated domain knowledge specifications to the target component. The source components are ontologies or other OO mediators that are heterogeneous and able to be integrated, and the target component is any WSMO top-level concept that applies to the integrated ontologies. The following shows the description specialization of an OO mediator:

```
Class ooMediator sub–Class mediator
  hasSource type {ontology, ooMediator}
```

OO mediators are used to import the terminology required for a resource description whenever there is a mismatch between the ontologies to be used. The mediation technique used by OO mediators is mainly ontology integration, i.e. merging, aligning, and mapping ontology definitions in order to retrieve integrated, homogeneous terminology definitions.

GG Mediators

A GG mediator connects goals, allowing one to create a new goal from existing goals and thus defining goal ontologies. GG mediators are defined as follows:

```
Class ggMediator sub–Class mediator
  usesMediator type ooMediator
  hasSource type {goal, ggMediator}
  hasTarget type {goal, ggMediator}
```

A GG mediator might use an OO mediator to resolve terminology mismatches between the source goals. Mediation services for GG mediators reduce or combine the descriptions of the source goals into the newly created target goal.

WG Mediators

A WG mediator links a Web service to a goal, resolves terminological mismatches, and states the functional difference (if any) between the two. WG mediators are defined as follows:

```

Class wgMediator sub—Class mediator
  usesMediator type ooMediator
  hasSource type { service , wgMediator }
  hasTarget type { goal, ggMediator }

```

WG mediators are used to prelink services to existing goals, and for the handling of partial matches within Web service discovery. As with GG mediators, WG mediators can be applied for resolving terminological mismatches.

WW Mediators

A WW mediator is used to establish interoperability between Web services that are otherwise not interoperable. The definition in MOF is as follows:

```

Class wwMediator sub—Class mediator
  usesMediator type ooMediator
  hasSource type { service , wwMediator }
  hasTarget type { service , wwMediator }

```

A WW mediator mediates between the choreographies of Web services that ought to interact with one another, where mediation might be required on the data, protocol, and process levels. As with the other WSMO mediator types, OO mediators can be applied for resolving terminological mismatches.

6.5 Nonfunctional Properties

Nonfunctional properties are used in the definition of WSMO elements. Which nonfunctional properties apply to which WSMO element is specified in the description of each WSMO element. In the following, we informally describe the nonfunctional properties recommended to be attached to WSMO elements.

- *Accuracy*. This represents the error rate generated by a Web service. It can be measured by the numbers of errors generated in a certain time interval.
- *Contributor*. An entity responsible for making contributions to the content of the element. Examples of the property `dc:contributor` include a person, an organization, and a Web service. The Dublin Core specification recommends that typically, the name of a `dc:contributor` should be used to indicate the entity.²

² In order to point unambiguously to a specific resource, we recommend the use of an instance of `foaf:Agent` as the value type [21].

- *Coverage.* The extent or scope of the content of the element. Typically, dc:coverage will include spatial location (a place name or geographic coordinates), a temporal period (a period label, date, or date range), or a jurisdiction (such as a named administrative entity).³
- *Creator.* The entity primarily responsible for creating the content of the element. Examples of dc:creator include a person, an organization, and a Web service. The Dublin Core specification recommends that, typically, the name of a dc:creator should be used to indicate the entity.
- *Date.* The date of an event in the life cycle of the element. Typically, dc:date will be associated with the creation or availability of the element.⁴
- *Description.* An account of the content of the element. Examples of dc:description include, but are not limited to, an abstract, a table of contents, a reference to a graphical representation of the content and a free-text account of the content.
- *Financial.* This represents the cost-related and charging-related properties of a Web service [105]. This property is a complex property, which includes charging styles (e.g. per request or delivery, per unit of measure, granularity, etc.), aspects of settlement such as the settlement model (transactional vs. rental) and a settlement contract, payment obligations, and payment instruments.
- *Format.* The physical or digital manifestation of the element. Typically, dc:format may include the media type or dimensions of the element. The format may be used to identify the software, hardware, or other equipment needed to display or operate the element. Examples of dimensions include size and duration.
- *Identifier.* An unambiguous reference to the element within a given context. Recommended best practice is to identify the element by means of a string or number conforming to a formal identification system. In Dublin Core, formal identification systems include, but are not limited to, the Uniform Resource Identifier (URI) (including the Uniform Resource Locator (URL)), the Digital Object Identifier (DOI), and the International Standard Book Number (ISBN).
- *Language.* The language of the intellectual content of the element. The language tags are defined in ISO Standard 639 [43], for example “en-GB”. In addition, the logical language used to express the content should be mentioned; for example this could be OWL.

³ For more complex applications, consideration should be given to using an encoding scheme that supports an appropriate specification of information, such as DCMI Period, DCMI Box, or DCMI Point.

⁴ We recommend using an encoding defined in ISO Standard 8601:2000 [44] for date and time notation. A short introduction to the standard can be found at <http://www.cl.cam.ac.uk/~mgk25/iso-time.html>. This standard is also used by the XML Schema definition (YYYY-MM-DD) [16], and thus one is automatically compliant with XML Schema too.

- *Network-related QoS*. This represents the QoS mechanisms operating in the transport network that are independent of the Web service. They can be measured by network delay, delay variation, and/or message loss.
- *Owner*. A person or organization to which a particular WSMO element belongs.
- *Performance*. This represents how fast a Web service request can be completed. According to [112], performance can be measured in terms of throughput, latency, execution time, and transaction time. The response time of a Web service can also be a measure of the performance. High-quality Web services should provide higher throughput, lower latency, lower execution time, faster transaction time and faster response time.
- *Publisher*. The entity responsible for making the element available. Examples of dc:publisher include a person, an organization, and a Web service. The Dublin Core specification recommends that, typically, the name of a dc:publisher should be used to indicate the entity.
- *Relation*. A reference to a related element. Recommended best practice is to identify the referenced element by means of a string or number conforming to a formal identification system.
- *Reliability*. This represents the ability of a Web service to perform its functions (that is, to maintain its Web service quality). It can be measured by the number of failures of the Web Service in a certain time interval.
- *Rights*. Information about rights held in and over the element. Typically, dc:rights will contain a rights management statement for the element, or reference a Web service providing such information. Rights information often encompasses intellectual property rights (IPR), copyright, and various property rights. If the rights element is absent, no assumptions may be made about any rights held in or over the element.
- *Robustness*. This represents the ability of the Web service to function correctly in the presence of incomplete or invalid inputs. It can be measured by the number of incomplete or invalid inputs for which the Web service still functions correctly.
- *Scalability*. This represents the ability of the Web service to process multiple requests in a certain time interval. It can be measured by the number of requests resolved in a certain time interval.
- *Security*. This represents the ability of a Web service to provide authentication (entities (users or other Web services) that can access a Web service, and their data, should be authenticated), authorization (entities should be authorized so that only they can access the protected Web services), confidentiality (data should be treated properly so that only authorized entities can access or modify the data), traceability/auditability (it should be possible to trace the history of a Web service after a request has been serviced), data encryption (data should be encrypted), and nonrepudiation (an entity cannot deny requesting a Web service or data after the fact).
- *Source*. A reference to an element from which the present element is derived. The present element may be derived from an element referenced in

dc:source in whole or in part. Recommended best practice is to identify the referenced element by means of a string or number conforming to a formal identification system.

- *Subject.* A topic of the content of the element. Typically, dc:subject will be expressed as keywords, key phrases, or classification codes that describe a topic of the element. Recommended best practice is to select a value from a controlled vocabulary or a formal classification scheme.
- *Title.* The name given to an element. Typically, dc:title will be a name by which the element is formally known.
- *Transactional.* This represents the transactional properties of the Web service.
- *Trust.* This represents the trustworthiness of a Web service or an ontology.
- *Type.* The nature or genre of the content of the element. The dc:type includes terms describing general categories, functions, genres, or aggregation levels of content.
- *Version.* As many properties of an element might change over time, an identifier of the element at a certain moment in time is needed.

6.6 Summary

In this chapter we have exhaustively introduced all elements relevant to the WSMO framework. We have done this in a semiformal fashion using the UML-based Meta-Object Facility. By separating the model from the language used to implement it, this provides significantly more freedom than other competing approaches (see Chapter 8).

Among the guiding principles we have presented Web compliance, which will be very prominent in the next chapter, which introduces the concrete language for WSML. Aspects of the Semantic Web are represented mainly by introducing an epistemological model for ontologies and by using elements of this model for refining other aspects such as capabilities through logical expressions.

We have not yet made the relationship between the WSMO model and Web service technology in the sense of WSDL explicit. The reason is that (1) WSMO, used as a conceptual model to describe services, is independent of the concrete technology used to implement them, and (2) this relationship (grounding) is, in our context, only relevant to actual Web service execution and not essential for an overall understanding. Information about grounding can be found in Section 9.5.

WSML – a Language for WSMO

The Web Service Modeling Language (WSML) is a formal language for the description of ontologies and Semantic Web services which takes into account all aspects of Web service descriptions identified by WSMO. The language comprises various formalisms in order to allow one to investigate their applicability to the description of Semantic Web services. Since our goal is to investigate the applicability of various formalisms to the description of Semantic Web services, it would be too restrictive to base our effort on existing language recommendations such as OWL [32]. A concrete goal in our development of WSML is to investigate the usage of various formalisms, most notably description logics and logic programming, in the context of ontologies and Web services.

We see three main areas which will benefit from the use of formal methods in service descriptions: *ontology description*, *declarative functional description of goals and Web services*, and *description of dynamics*. In its current version, WSML defines a syntax and semantics for ontology descriptions. The underlying formalisms mentioned above are used to give a formal meaning to ontology descriptions in WSML. For the functional description of goals and Web services, WSML offers a syntactical framework, with Hoare-style semantics in mind. However, WSML does not yet formally specify the exact semantics of the functional descriptions of services. The description of the dynamic behavior of Web services (choreography and orchestration) in the context of WSML is currently under investigation, but has not yet been integrated into WSML. Thus, in this chapter we focus primarily on ontology description in WSML, where we find that WSML already includes many potentially useful features that were lacking in previous approaches.

We give an overview of WSML and its layering of languages in Section 7.1. The normative human-readable syntax of WSML is described in Section 7.2, followed by a rough sketch of the WSML semantics in Section 7.3 and the WSML exchange syntaxes in Section 7.4. The key features of WSML are described in Section 7.5. Section 7.6 describes related approaches to the

description of Semantic Web services and ontologies. Section 7.7 contains a summary of the Chapter and outlines future work which is planned for WSML.

The WSML specification and related documents can be found on line at <http://www.wsmo.org/wsml/wsml-syntax>.

7.1 The WSML Layering

Figure 7.1a shows the different variants of WSML and the relationships between them. These variants differ in logical expressiveness and in the underlying language paradigms and allow users to make a trade-off between provided expressiveness and the implied complexity of ontology modeling on a per-application basis. The variants are as follows:

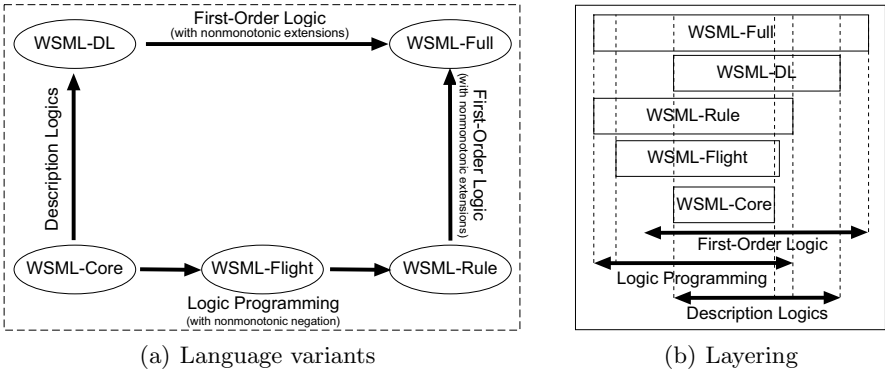


Fig. 7.1. WSML variants and layering

- *WSML-Core* is based on the intersection of the description logic *SHIQ* and horn Logic, called DLP (description logic programs) [50]. It has the least expressive power of all the WSML variants. The main features of the language are concepts, attributes, binary relations, and instances, as well as concept and relation hierarchies and support for datatypes.
- *WSML-DL* captures the description logic *SHIQ(D)*, which is a major part of the DL species of OWL [32].
- *WSML-Flight* is an extension of *WSML-Core* which provides a powerful rule language. It adds features such as metamodeling, constraints, and nonmonotonic negation. *WSML-Flight* is based on a logic programming variant of F-Logic [72] and is semantically equivalent to datalog with inequality and (locally) stratified negation. *WSML-Flight* is a direct syntactic extension of *WSML-Core*, and it is a semantic extension in the sense that the *WSML-Core* subset of *WSML-Flight* agrees with *WSML-Core* on ground entailments (see [70]).

- *WSML-Rule* extends WSML-Flight with further features from logic programming, namely the use of function symbols, unsafe rules and unstratified negation under the well-founded semantics [45].
- *WSML-Full* unifies WSML-DL and WSML-Rule under a First-Order umbrella with extensions to support the nonmonotonic negation of WSML-Rule. The semantics of WSML-Full is currently an open research issue.

As shown in Fig. 7.1b, WSML has two alternative layerings, namely, $\text{WSML-Core} \Rightarrow \text{WSML-DL} \Rightarrow \text{WSML-Full}$ and $\text{WSML-Core} \Rightarrow \text{WSML-Flight} \Rightarrow \text{WSML-Rule} \Rightarrow \text{WSML-Full}$. For both layerings, WSML-Core and WSML-Full mark the least and most expressive layers, respectively. The two layerings are to a certain extent disjoint, in the sense that interoperation in WSML between the description logic variant (WSML-DL) and the logic programming variants (WSML-Flight and WSML-Rule) is only possible through a common core (WSML-Core) or through a very expressive superset (WSML-Full).

7.2 General WSML Syntax

In this section, we introduce the general WSML syntax, which encompasses all features supported by the different language variants. We describe the restrictions imposed on this general syntax by the variants. These restrictions follow from the logical language underlying the specific language variant, as described in the previous section.

WSML makes a clear distinction between (a) the modeling of the various conceptual elements and (b) the specification of complex logical definitions. To this end, the WSML syntax is split into two parts: the conceptual syntax and the logical-expression syntax. The conceptual syntax was developed from the perspective of the user, and is independent of the particular underlying logic; it shields the user from the peculiarities of the underlying logic. Having such a conceptual syntax allows easy adoption of the language, since it allows an intuitive understanding of the language for those not familiar with logical languages. When the full power of the underlying logic is required, the logical-expression syntax can be used. There are several entry points for logical expressions in the conceptual syntax, namely axioms in ontologies and capability descriptions in goals and Web services.

We shall first describe the use of Web identifiers and concrete data values in Section 7.2.1. The various kinds of WSML definitions and a general explanation of the conceptual syntax are given in Section 7.2.2. The logical-expression syntax is described in Section 7.2.3.

7.2.1 Identifiers in WSML

WSML has three kinds of identifiers, namely IRIs, sQNames (abbreviated IRIs), and data values.

An *IRI* (Internationalized Resource Identifier)¹ uniquely identifies a resource in a Web-compliant way. The proposed IRI standard is the successor to the popular URI standard and has already been adopted in various W3C recommendations. IRIs are delimited using an underscore and a double quote “_” and a double quote “””, for example `_”http://www.wsmo.org/wsmi/wsmi-syntax#”`.

In order to enhance legibility, an IRI can be abbreviated to an *sQName*, which is short for “serialized QName”, and is of the following form: *prefix#localname*. The prefix and the local part may be omitted, in which case the name falls into the default namespace. Our concept of an “sQName” corresponds to the use of QNames in RDF and is slightly different from QNames in XML, where a QName is not merely an abbreviation for an IRI, but a tuple `<namespaceURI, localname>`.

Data values in WSML are either strings, integers, decimals, or structured data values, reflecting the XML Schema datatypes. WSML defines constructs which reflect the structure of data values. For example, the date “March 15th, 2005” is represented as `_date(2005,3,15)`. In logical expressions, constructed data values can be used in the same way as constructed terms. However, logical expressions do not allow constructed terms to be nested inside constructed data values.

7.2.2 Conceptual Syntax

The WSML conceptual syntax allows the modeling of ontologies, Web services, goals, and mediators. It is shared between all variants, with the exception of some restrictions which apply to the modeling of ontologies in WSML-Core and WSML-DL.

Ontologies

An ontology in WSML consists of the elements **concept**, **relation**, **instance**, **relationInstance**, and **axiom**. Additionally, an ontology may have nonfunctional properties and may import other ontologies. We start the description of WSML ontologies with an example which demonstrates the elements of an ontology in Listing 7.1, and then provide a detailed explanation.

Concepts

The notion of concepts (sometimes also called “classes”) plays a central role in ontologies. Concepts form the basic terminology of the domain of discourse. A concept may have instances and may have a number of attributes associated with it. The nonfunctional properties and the attribute definitions, are grouped together in one frame, as can be seen from the example concept `book` in Listing 7.1.

¹ See IETF RFC 3987: <http://www.ietf.org/rfc/rfc3987.txt>

```

wsmml:variant _" http://www.wsmo.org/wsmml/wsmml-syntax/wsmml-flight"
namespace { _" http://example.org/bookOntology#" ,
  dc _" http://purl.org/dc/elements/1.1/" }
ontology _" http://example.org/bookOntology"
nonFunctionalProperties
  dc:title hasValue "Example Book Ontology"
  dc:description hasValue "Example ontology about books and shopping carts"
endNonFunctionalProperties
concept book
  title ofType _string
  hasAuthor ofType author
concept author subConceptOf person
  authorOf inverseOf(hasAuthor) ofType book
concept cart
  nonFunctionalProperties
    dc:description hasValue "A shopping cart has exactly one id
      and zero or more items, which are books."
  endNonFunctionalProperties
  id ofType (1) _string
  items ofType book
instance crimeAndPunishment memberOf book
  title hasValue "Crime and Punishment"
  hasAuthor hasValue dostoyevsky

relation authorship(impliesType author, impliesType document)
  nonFunctionalProperties
    dc:relation hasValue authorshipFromAuthor
  endNonFunctionalProperties

axiom authorshipFromAuthor
  definedBy
    authorship(?x,?y) :- ?x[authorOf hasValue ?y] memberOf author.

```

Listing 7.1. An example WSMML ontology

Attribute definitions can take two forms, namely *constraining* (using **ofType**) and *inferring* (using **impliesType**) attribute definitions.² Constraining attribute definitions define a typing constraint on the values of an attribute, similar to integrity constraints in databases; inferring attribute definitions imply that the type of the values of the attribute is inferred from the attribute definition, similar to the range restrictions on properties in RDFS [20] and OWL [32]. Each attribute definition may have a number of features associated with it, namely, transitivity, symmetry, reflexivity, and the inverse of an attribute, as well as minimal and maximal cardinality constraints.

Constraining attribute definitions, and also cardinality constraints, require closed-world reasoning and are thus not allowed in WSMML-Core and WSMML-DL. As opposed to the features of roles in description logics, features of attributes such as transitivity, symmetry, reflexivity, and inverse attributes are local to a concept in WSMML. Thus, none of these features may be used in WSMML-Core and WSMML-DL. For a description of the motivation for the use of constraining attributes, see [31].

² The distinction between inferring and constraining attribute definitions is explained in more detail in [31, Section 2].

Relations

Relations in WSML can have an arbitrary arity, they may be organized in a hierarchy using **subRelationOf**, and the parameters may be typed using parameter type definitions of the form (**ofType** *type*) and (**impliesType** *type*), where *type* is a concept identifier. The use of **ofType** and **impliesType** corresponds to their use in attribute definitions. Specifically, parameter definitions with the keyword **ofType** are used to check the type of parameter values, whereas parameter definitions with the keyword **impliesType** are used to infer concept membership of parameter values.

The allowed arity of a relation may be constrained by the underlying logic of the WSML language variant. WSML-Core and WSML-DL allow only binary relations and, similar to attribute definitions, they allow only parameter typing using the keyword **impliesType**.

Instances

A concept may have a number of instances associated with it. The instances explicitly specified in an ontology are those which are shared as part of the ontology. However, most instance data exist outside the ontology in private databases. WSML does not prescribe how to connect such a database to an ontology, since different organizations will use the same ontology to query different databases and such corporate databases are typically not shared.

An instance may be a member of zero or more concepts and may have a number of attribute values associated with it; see, for example, the instance `crimeAndPunishment` in Listing 7.1. Note that the specification of concept membership is optional and that the attributes used in the instance specification do not necessarily have to occur in the associated concept definition. Consequently, WSML instances can be used to represent semistructured data, since without concept membership and constraints on the use of attributes, instances form a directed labeled graph. Because of this possibility to capture semistructured data, most RDF graphs can be represented as WSML instance data, and vice versa.

Axioms

Axioms provide a means to add arbitrary logical expressions to an ontology. Such logical expressions can be used to refine concept or relation definitions in the ontology, and also to add arbitrary axiomatic domain knowledge or to express constraints. The axiom `authorshipFromAuthor` in Listing 7.1 states that the relation `authorship` exists between any author and any book of which he/she is an author; consequently, `(dostoyesky, crimeAndPunishment)` is in the relation `authorship`. Logical expressions are explained in more detail in Section 7.2.3.

Web Services

A Web service has a capability and a number of interfaces. The capability describes the functionality of the Web service by expressing conditions on

```

webService _" http://example.org/bookService"
nonFunctionalProperties
  dc#title hasValue "Example Book Buying Service"
  dc#description hasValue "A simple example Web service for adding items to a shopping cart"
endNonFunctionalProperties

importsOntology _" http://example.org/bookOntology"
capability
  sharedVariables {?cartId , ?item}
  precondition
    definedBy
      ?cartId memberOf _string and ?item memberOf book.
  postcondition
    definedBy
      forall ?cart (?cart[id hasValue ?cartId] memberOf cart implies
        ?cart[items hasValue ?item]).

```

Listing 7.2. A WSML Web service description

its prestates and poststates³ using logical expressions; the interfaces describe how to interact with the service. Additionally, WSML allows one to specify nonfunctional properties of a Web service. Listing 7.2 describes a simple Web service for adding items to a shopping cart.

Capabilities

Preconditions and assumptions describe the state before the execution of a Web service. While preconditions describe conditions on the information space, i.e. conditions on the input, assumptions describe conditions on the current state of the world which, cannot necessarily be directly checked. Postconditions describe the relation between the input and the output, for example a credit card limit with respect to its value before the execution of the service. In this sense, postconditions describe the information state after execution of the service. Effects describe changes in the real world caused by the service, for example the physical shipment of goods. The **sharedVariables** construct is used to identify variables which are shared between the preconditions and postconditions and between the assumptions and effects. Shared variables can be used to refer to the same input and output values in the conditions of the capability. In the service described in Listing 7.2, given a shopping cart identifier and a number of items, the items are added to the shopping cart with this identifier.

Interfaces

Interfaces describe how to interact with a service from the requester’s point of view (**choreography**) and how the service interacts with the other services and goals that it needs to fulfill in order to fulfill its function from the provider’s point of view (**orchestration**). Choreography and orchestration descriptions are

³ “Prestate” and “poststate” refer to the state before and after, respectively, the execution of the Web service.

external to WSML; WSML allows reference to any choreography or orchestration identified by an IRI.

Goals

Goals are symmetric to Web services in the sense that goals describe the desired functionality and Web services describe the offered functionality. Therefore, a goal description consists of the same modeling elements as does a Web service description, namely nonfunctional properties, a capability, and a number of interfaces.

Mediators

Mediators connect different goals, Web services, and ontologies, enabling interoperation by reconciling differences in representation formats, encoding styles, business protocols, etc. Connections between mediators and other WSML elements can be established in two different ways:

1. Each WSML element allows the specification of a number of mediators to be used through the keyword **usesMediator**.
2. Each mediator has, depending on the type of mediator, zero, one, or more sources and zero or one target. The source and target are optional, in order to allow generic mediators.

A mediator achieves its mediation functionality either through a Web service, which provides the mediation service, or a goal, which can be used to dynamically discover the appropriate (mediation) Web service.

7.2.3 Logical-Expression Syntax

We shall first explain the general logical-expression syntax, which encompasses all WSML variants, and then describe the restrictions on this general syntax for each of the variants. The general-logical expression syntax for WSML has a first-order logic style, in the sense that it has constants, function symbols, variables, predicates and the usual logical connectives. Furthermore, WSML has extensions based on F-Logic [72] in order to model concepts, attributes, attribute definitions, and subconcept and concept membership relationships. Finally, WSML has a number of connectives to facilitate the logic programming based variants, namely default negation (negation-as-failure), LP-implication (which differs from classical implication), and database-style integrity constraints.

Variables in WSML start with a question mark, followed by an arbitrary number of alphanumeric characters, for example `?x`, `?name`, or `?123`. Free variables in WSML (i.e. variables which are not explicitly quantified), are implicitly universally quantified outside of the formula (i.e. the logical expression

in which the variable occurs is the scope of quantification), unless indicated otherwise, through the **sharedVariables** construct (see Section 7.2.2).

Terms are either identifiers, variables, or constructed terms. An atom is, as usual, a predicate symbol with a number of terms as arguments. Besides the usual atoms, WSML has special kinds of atoms, called *molecules*, which are used to capture information about concepts, instances, attributes and attribute values. There are two types of molecules, analogous to the situation in F-Logic:

- An *isa* molecule is a concept membership molecule of the form A **memberOf** B or a subconcept molecule of the form A **subConceptOf** B , where A and B are arbitrary terms.
- An *object* molecule is an attribute value expression of the form $A[B$ **hasValue** $C]$, a constraining-attribute signature expression of the form $A[B$ **ofType** $C]$, or an inferring-attribute signature expression of the form $A[B$ **ofType** $C]$, where A, B , and C are arbitrary terms.

WSML has the usual first-order connectives: the unary negation operator **neg**, and the binary operators for conjunction **and**, disjunction **or**, right implication **implies**, left implication **impliedBy**, and dual implication **equivalent**. Variables may be universally quantified using **forall** or existentially quantified using **exists**. First-order formulas are obtained by combining atoms using the above connectives in the usual way. The following are examples of first-order formulas in WSML:

```
//Every person has a father
forall ?x (?x memberOf Person implies exists ?y (?x[father hasValue ?y])).
//John is member of a class which has some attribute called 'name'
exists ?x,?y (john memberOf ?x and ?x[name ofType ?y]).
```

In addition to first-order formulas, WSML allows the use of the negation-as-failure symbol **naf** on atoms, the special logic programming implication symbol **:-**, and the integrity constraint symbol **!-**. A Logic Programming rule consists of a *head* and a *body*, separated by the symbol **:-**. An integrity constraint consists of the symbol **!-** followed by a rule body. Negation-as-failure **naf** is allowed to occur only in the body of a logic programming rule or an integrity constraint. The further use of logical connectives in Logic Programming rules is restricted. The following logical connectives are allowed in the head of a rule: **and**, **implies**, **impliedBy**, and **equivalent**. The following connectives are allowed in the body of a rule (or constraint): **and**, **or**, and **naf**. The following are examples of logic programming rules and database constraints:

```
//Every person has a father
?x[father hasValue f(?y)] :- ?x memberOf Person.
//Man and Woman are disjoint
!- ?x memberOf Man and ?x memberOf Woman.
//In case a person is not involved in a marriage, the person is a bachelor
?x memberOf Bachelor :- ?x memberOf Person and naf Marriage(?x,?y,?z).
```

Particularities of the WSML Variants

Each of the WSML variants defines a number of restrictions on the logical-expression syntax. For example, logic programming rules and constraints are not allowed in WSML-Core and WSML-DL. Table 7.1 presents a number of language features and indicates in which variant each feature can occur:

Table 7.1. WSML variants and feature matrix

Feature	Core	DL	Flight	Rule	Full
Classical negation (neg)	–	X	–	–	X
Existential quantification	–	X	–	–	X
(Head) disjunction	–	X	–	–	X
n -ary relations	–	–	X	X	X
Metamodeling	–	–	X	X	X
Default negation (naf)	–	–	X	X	X
LP implication	–	–	X	X	X
Integrity constraints	–	–	X	X	X
Function symbols	–	–	–	X	X
Unsafe rules	–	–	–	X	X

- *WSML-Core* allows only first-order formulas which can be translated to the description logic programming (DLP) subset of $\mathcal{SHIQ}(\mathbf{D})$ [50]. This subset is very close to the 2-variable fragment of first-order logic, restricted to Horn logic. Although WSML-Core might appear in Table 7.1 as featureless, it captures most of the conceptual model of WSML, but has only limited expressiveness within the logical expressions.
- *WSML-DL* allows all first-order formulas which can be translated to $\mathcal{SHIQ}(\mathbf{D})$. This subset is very close to the two-variable fragment of first-order logic. Thus, WSML DL allows classical negation, and disjunction and existential quantification in the heads of implications.
- *WSML-Flight* extends the set of formulas allowed in WSML-Core by allowing variables in place of instance, concept, and attribute identifiers and by allowing relations of arbitrary arity. In fact, any such formula is allowed in the head of a WSML-Flight rule. The body of a WSML-Flight rule allows conjunction, disjunction and default negation. The head and body are separated by the LP implication symbol. WSML-Flight additionally allows metamodeling (e.g. using a class identifier simultaneously as an instance) and reasoning over a signature, because variables are allowed to occur in place of concept and attribute names.
- *WSML-Rule* extends WSML-Flight by allowing function symbols and unsafe rules, i.e. variables which occur in the head or in a negative body literal do not need to occur in a positive body literal.

- *WSML-Full*. The logical syntax of WSML-Full is equivalent to the general logical expression syntax of WSML and allows the full expressiveness of all other WSML variants.

The separation between conceptual and logical modeling allows a gentle learning curve for nonexperts, since the conceptual syntax does not require expert knowledge in logical modeling, whereas complex logical expressions require more familiarity and training with the language. Thus, WSML allows the modeling of various aspects of Web services on a conceptual level, while still offering the full expressive power of the logic underlying the chosen WSML variant. Part of the conceptual syntax for ontologies has an equivalence in the logical syntax. This correspondence is used to define the semantics of the conceptual syntax. Note that since only parts of the conceptual syntax are mapped to the logical syntax, only part of the conceptual syntax has a semantics in the logical language for ontologies. For example, nonfunctional properties are not translated (hence the name “nonfunctional”). The translation between the conceptual and the logical syntax is sketched in Table 7.2.

Table 7.2. Translating the conceptual to the logical syntax

Conceptual	Logical
concept A subConceptOf B	A subConceptOf B.
concept A B ofType (0 1) C	A[B ofType C]. !- ?x memberOf A and ?x[B hasValue ?y, B hasValue ?z] and ?y != ?z.
concept A B ofType C	A[B ofType C].
relation A/n subRelationOf B	A(x ₁ , ..., x _n) implies B(x ₁ , ..., x _n)
instance A memberOf B C hasValue D	A memberOf B. A[C hasValue D].

7.3 WSML Semantics

The semantics of WSML ontologies is defined through a mapping between the WSML logical syntax and the formalism which underlies the variant. WSML-Core and WSML-DL logical expressions are mapped to first-order predicate logic (FOPL). The restrictions on the WSML syntax for these variants ensure that the expressions stay inside the expressiveness of the *SHIQ* description logic. The semantics of WSML-Flight and WSML-Rule is defined through a mapping to F-Logic programming [72, Appendix A].

In order to facilitate the mapping between a WSML variant and the underlying logic, the WSML ontology is first transformed to a set of normalized logical expressions, according to Table 7.2. Then, this normalized set of logical expressions is mapped to the logic using a mapping function π . Finally, satisfiability and entailment are defined with respect to this transformed WSML ontology.

Rather than providing a direct semantics, the semantics of WSML is defined through several mappings to well-known formalisms in order not only to facilitate understanding of the language, but also so that complexity results follow immediately from known definitions and the accepted literature. To give an idea of the semantics,⁴ we show the example ontology `booksOntology` (Listing 7.1), transformed into F-Logic programming:

```

← x[y⇒z] ∧ w[y→v] ∧ not v : z
book[title⇒_string]
book[hasAuthor⇒author]
author :: person
author[authorOf⇒book]
y[hasAuthor→x] ← x : author ∧ x[authorOf→y]
y[authorOf→x] ← y : author ∧ x[hasAuthor→y]
cart[id⇒_string]
book[hasAuthor⇒author]
← x : cart ∧ not x[id→y]
← x : cart ∧ x[id→y] ∧ x[id→z] ∧ y ≠ z
cart[items⇒amazonBook]
crimeAndPunishment : book
crimeAndPunishment[title→"CrimeandPunishment"]
crimeAndPunishment[hasAuthor→dostoyevsky]
x : author ← authorShip(x, y)
y : document ← authorShip(x, y)
authorship(x, y) ← x[authorOf→y] ∧ x : author

```

Note that in this example, a rule without a head is an integrity constraint. A model of the rule base violates an integrity constraint if the body of the constraint is true in the model. We call a transformed WSML-Flight ontology O *satisfiable* iff the transformed rule base $\pi(O)$ (without the constraints) has a perfect model M_O [72, Appendix A] and this model does not violate any of the integrity constraints. Furthermore, a satisfiable WSML-Flight ontology O *entails* a ground formula F iff $\pi(F)$ is true in M_O .

7.4 WSML Exchange Syntaxes

For exchange over the Web and interoperation with RDF-based applications, WSML defines an XML syntax and an RDF syntax.

7.4.1 The WSML XML Syntax

The Extensible Markup Language (XML)⁵ is a markup language for interchange of structured and semistructured data over the Web. WSML specifies a serialization in XML for exchange of WSML specifications over the Web.

The WSML XML syntax is similar to the human-readable syntax, both in keywords and in structure. We have defined the XML syntax through

⁴ Space limitations prevent us from giving the complete semantics of WSML. Instead, we have given a rough sketch of the WSML semantics and refer the interested reader to [30].

⁵ <http://www.w3.org/XML>.

a translation from the human-readable syntax. Because of space limitations, we give only a partial example (Listing 7.3) of the XML version of the WSML example in Listing 7.1. The complete translation can be found in [30]. Typically, WSML keywords, which are in the WSML namespace (<http://www.wsmo.org/wsml/wsml-syntax#>), map to XML elements. Note also that sQNames are written as full IRIs in XML. We have defined the WSML XML syntax in the form of an XML Schema.⁶ The XML syntax for logical expressions is close to RuleML, and we expect further alignment between the two initiatives in the next version.

```

<!ENTITY book "http://example.org/bookOntology#" >
<wsml xmlns="http://www.wsmo.org/wsml/wsml-syntax#
  variant="http://www.wsmo.org/wsml/wsml-syntax/wsml-flight" >
  <ontology name="http://example.org/bookOntology" >
    <nonFunctionalProperties>
      <attributeValue name="http://purl.org/dc/elements/1.1/title" >
        <value type="http://www.wsmo.org/wsml/wsml-syntax#string" >
          Example Book Ontology
        </value>
        [...]
      </attributeValue>
    </nonFunctionalProperties>
    <concept name="&book;book" >
      <nonFunctionalProperties>[.]</nonFunctionalProperties>
      <attribute name="&book;title" type="constraining" >
        <range>http://www.wsmo.org/wsml/wsml-syntax#string</range>
      </attribute>
      [...]
    </concept>
    [...]
  </ontology>
</wsml>

```

Listing 7.3. WSML XML example

7.4.2 The WSML RDF Syntax

The Resource Description Framework (RDF)⁷ defines a data structure based on triples for metadata description on the Web. RDF Schema [20] is a simple ontology language intended to provide a vocabulary for RDF descriptions.

WSML provides a serialization in RDF of all its conceptual modeling elements. The WSML RDF syntax reuses and extends the RDF Schema vocabulary to allow existing RDF(S)-based tools to understand parts of a WSML specification. For logical expressions, we reuse the XML serialization of WSML, since an encoding of complex constructs does not fit well into the RDF triple-based data model.

Owing to limited space, we show here only a small example of the WSML RDF syntax, in Listing 7.4. The complete RDF serialization of WSML and a larger example can be found in [30]. In Listing 7.4, “rdf” stands for the

⁶ <http://www.wsmo.org/TR/d16/d16.1/v0.21/xml-syntax/wsml-xml-syntax.xsd>.

⁷ <http://www.w3.org/RDF>.

RDF namespace `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, “dc” stands for the Dublin Core namespace `http://purl.org/dc/elements/1.1/`, “wsml” stands for the WSML namespace `http://www.wsmo.org/wsml/wsml-syntax#`, and “book” stands for the namespace `http://example.org/bookOntology#`. We use the sQ-Name abbreviation mechanism, where *prefix#localname* is an abbreviation for the concatenation of the namespace IRI and the local name. The RDF triple is written as *subject predicate object*, and blank nodes are indicated by “_.” in front of the name.

```

1  http://example.org/bookOntology rdf:type wsml#ontology
2  http://example.org/bookOntology wsml#variant http://www.wsmo.org/wsml/wsml-syntax/wsml-
   flight
3  http://example.org/bookOntology wsml#nfp _:nfp1
4  _:nfp1 dc:title "Example Book ontology"^^xsd:string
5  http://example.org/bookOntology wsml#hasConcept book#book
6  http://example.org/bookOntology#Person wsml#hasAttribute _:att1
7  _:att1 wsml#attribute http://example.org/bookOntology#title
8  _:att1 wsml#ofType xsd:string
9  http://example.org/bookOntology wsml#hasAxiom book#authorshipFromAuthor
10 book#authorshipFromAuthor rdfs#isDefinedBy "<impliedByLP>.."^^rdf#XMLLiteral
11 [..]

```

Listing 7.4. WSML RDF example

7.4.3 Mapping to OWL

We have defined a complete mapping from the ontology subset of WSML-Core to OWL DL and a partial mapping from OWL DL to WSML-Core. The mapping can be found in [30].

The ontology subsets of WSML-Core and WSML-DL can be completely mapped to OWL DL, but OWL can be only partially mapped to WSML-DL, since WSML-DL does not allow nominals (i.e. individuals in class descriptions). The mapping between OWL and WSML-DL has not yet been defined. However, it is a trivial extension of the mapping between WSML-Core and OWL DL.

We give below a few notes about the mappings between OWL and WSML, which should also clarify the relation between OWL and WSML:

- Since OWL is an ontology language and Web service modeling requires additional modeling elements such as goal, mediator, and Web service descriptions, the mapping between OWL and WSML can only be done for the ontology-modeling part of WSML.
- The WSML-Core variant is based on a subset of the description logic $SHIQ(\mathbf{D})$, and thus the ontology subset of WSML-Core can be completely mapped to OWL DL. WSML-DL can be completely mapped to OWL DL (with datatype extensions), whereas the reverse mapping is not complete, because WSML-Core does not allow nominals.

7.5 Key Features of WSML

There are a number of features which make WSML unique in comparison with other language proposals for the Semantic Web and Semantic Web services. These key features are mainly due to the two pillars of WSML: (1) a *language-independent conceptual model* for ontologies, Web services, goals, and mediators, based on WSMO [116] and (2) the *reuse* of several well-known logical-language paradigms in *one* syntactical framework. More specifically, we see the following as the key features of WSML:

- *One syntactic framework for a set of layered languages.* We believe different Semantic Web and Semantic Web service applications need languages of different expressiveness and that no single language paradigm will be sufficient for all cases. With WSML, we are investigating the use of description logics and logic programming for Semantic Web services.
- *Normative, human-readable syntax.* It has been argued that tools will hide the syntax of the language from the user; however, as has been seen, for example with the adoption of SQL, an expressive but understandable syntax is crucial for successful adoption of a language. Developers and early adopters of the language will have to deal with the concrete syntax. If it is easy to read and understand then it will allow easier adoption of the language.
- *Separation of conceptual and logical modeling.* On the one hand, the conceptual syntax of WSML has been designed in such a way that it is independent of the underlying logical language, such that no or only limited knowledge of logical languages is required for the basic modeling of ontologies, Web services, goals, and mediators. On the other hand, the logical-expression syntax allows expert users to refine the definitions of the conceptual syntax using the full expressive power of the underlying logic.
- *Semantics based on well-known formalisms.* WSML captures well-known logical formalisms such as datalog and description logics in a unifying syntactical framework while maintaining the established computational properties of the original formalisms through proper syntactic layering. The variants of WSML allow the reuse of tools already developed for these formalisms. Notably, WSML allows the reuse of efficient querying engines developed for datalog and efficient subsumption reasoners developed in the area of description logics. Interoperation between paradigms is achieved through a common subset, WSML-Core, based on description logic programs [50].
- *WWW language.* WSML has a number of features which integrate it seamlessly with the Web. WSML adopts the IRI standard, the successor of the URI, for the identification of resources, following the Web architecture. Furthermore, WSML adopts the namespace mechanism of XML, datatypes in WSML are compatible with datatypes in XML Schema [16],

and datatype functions and operators are based on the functions and operators of XQuery [86]. Finally, WSML defines an XML syntax and an RDF syntax for exchange over the Web. When the RDF syntax is used, WSML can be seen as an extension of RDFS.

- *Frame-based syntax.* Frame logic [72] allows the use of frames in logical expressions. This allows the user to work directly at the level of concepts, attributes, instances, and attribute values, instead of at the level of predicates. Furthermore, variables are allowed in place of concept and attribute identifiers, which enables metamodeling and reasoning over a signature in the rule-based WSML language variants.

7.6 Relation to RDF(S) and OWL

In Section 7.4 we showed that parts of WSML can be expressed in RDF and OWL, the most prominent language recommendations in the context of the Semantic Web. Now let us take a closer look at the ontological modeling capabilities that WSML offers in comparison with these two languages.

We recall that RDFS (see Section 3.2.1) allows the expression of classes, properties, class hierarchies, property hierarchies, and domain and range restrictions, all of which are provided similarly by WSML. The following two factors make the layering of more expressive ontology languages on top of RDFS difficult:

1. RDFS allows the use of the language vocabulary as subjects and objects in the language itself.
2. RDFS allows the use of the same identifier to occur at the same time in place of class, individual, and property identifiers.

We believe that the number of cases where the first feature is used, namely the use of language constructs in the language itself, is limited. However, the use of the same identifier as class, individual, and property identifiers (also called metamodeling) is deemed useful in many cases. WSML does not allow the use of the language constructs in arbitrary places in an ontology, but does allow metamodeling in its Flight, Rule and, Full variants.

WSML is an extension of a significant part of RDFS. However, it does not allow the use of language constructs in the language itself and does not allow full treatment of blank nodes because this would require reasoning with existential information (which is not allowed in the rule-based WSML variants). WSML provides a significant extension of RDFS through the possibility of specifying local attributes, range and cardinality constraints for attributes, and attribute features such as symmetry, transitivity and reflexivity. Furthermore, WSML (in its rule-based variants) provides an expressive rule language which can be used for the manipulation of RDF data.

As for OWL (see Section 3.3), we can easily verify that the logical expressions in WSML-Core form a semantic subset of OWL Lite. WSML-DL is

semantically equivalent to OWL DL. However, there is a major difference between ontology modeling in WSML and ontology modeling in OWL. WSML uses an epistemology which abstracts from the underlying logical language, whereas OWL directly uses description logic's epistemology. Also, in contrast to OWL, WSML makes a separation between conceptual modeling for the nonexpert user and logical modeling for the expert user. Arguably, these properties could make WSML easier to use as an ontology language. This is, however, merely a conjecture and would require extensive user testing to verify its correctness.

Since WSML-Flight and WSML-Rule are based on the logic programming paradigm rather than the description logic paradigm, their expressiveness is quite different from that of OWL. Advantageously, WSML-Flight and Rule allow chaining over predicates and nonmonotonic negation, but do not allow classical negation, full disjunction or existential quantification. We conjecture that both the description logics and logic programming paradigms are useful on the Semantic Web (see [70]). With WSML, we capture both paradigms in one coherent framework. And successful interaction between the paradigms is achieved through a common subset, WSML-Core.

7.7 Summary

In this chapter, we have presented the Web Service Modeling Language, WSML, a language for the specification of various aspects of Semantic Web services, based on WSMO. WSML brings together different logical-language paradigms and unifies them into one syntactical framework, enabling the reuse of proven reasoning techniques and tools. Unlike other proposals for the Semantic Web and Semantic Web service languages, WSML has a normative human-readable syntax that makes a separation between conceptual and logical syntax, thereby enabling conceptual modeling according to a language-independent metamodel (WSMO), while not restricting the expressiveness of the language for the expert user. With the use of IRIs (the successor of URIs) and the use of XML and RDF, WSML is a language based on the principles of the Semantic Web and allows seamless integration with other Semantic Web languages and applications.

The definition of an interoperability layer between the description logic and rules paradigms (i.e. WSML-Core) enables the use and extension of the same core ontology for a number of different reasoning tasks supported by a number of different reasoners, most notably subsumption reasoning using description logic reasoners and query answering using logic programming reasoners.

Related Work in the Area of Semantic Web Service Frameworks

WSMO is not the only initiative aimed towards Semantic Web services. Other major initiatives in the area are documented by recent W3C member submissions: OWL-S [87], SWSF [9], and WSDL-S [1]. In this chapter, we shall give an overview of these related efforts, highlight their differences, and explain their relation to WSMO.

8.1 OWL-S

OWL-S [87] is an OWL service ontology for describing various aspects of Web services. The initiative has its roots in the DAML Service Ontology (DAML-S), the first version of which was released in May 2001. At the time, it was the first progressive effort towards semantic annotation of Web services. OWL-S has currently reached version 1.1, which was submitted as a member submission to the W3C in November 2004 by Nokia, the University of Maryland, the National Institute of Standards and Technology (NIST), Network Inference, SRI International, France Telecom, Stanford University, Toshiba, and the University of Southampton.

By switching from DAML+OIL to OWL, OWL-S tried to adopt existing Semantic Web recommendations yet still maintain bindings to the world of Web services by linking OWL-S descriptions to existing WSDL descriptions. We shall now take a closer look at the top-level concepts of the OWL-S ontology.

The ontology itself defines the top-level concept “Service” and three OWL-S subontologies known as the “Service Profile”, “Service Model”, and “Service Grounding”, all of which are illustrated in Fig. 8.1.

8.1.1 Service Profile

Each instance of the service class *presents* zero or more service profiles. A service profile expresses “what a service does” for the purposes of advertis-

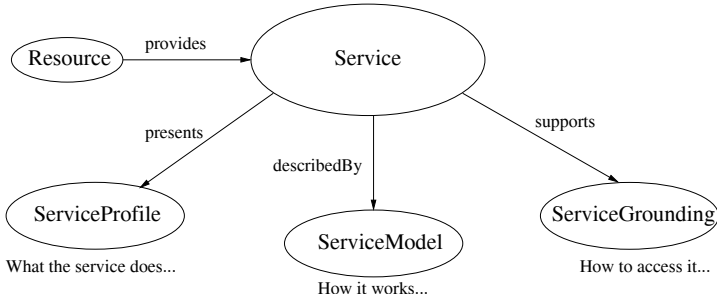


Fig. 8.1. OWL-S conceptual model

ing and serves as a template for service requests, thus enabling discovery and matchmaking. The profile comprises nonfunctional aspects such as references to existing categorization schemes (e.g. UNSPC) or ontologies, provider information, and the quality rating of the service. The most essential information presented in the profile, however, is the specification of what functionality the service provides. Information transformation is represented by *inputs* and *outputs*; the change in the state of the real world caused by the execution of the service is represented by *preconditions* and *effects*. Inputs and outputs refer to OWL classes describing the types of instances to be sent to the service and the respective responses to be expected. The format of the preconditions and effects is not fixed, but the authors of OWL-S allow various formats to be embedded into the OWL ontology. Preferably, SWRL [64]; alternatively, DRS [88] or KIF [46] expressions may be allowed.

A possible problem in this context is that the semantics of these conditions is not covered by the (description logics) expressivity of the OWL-S ontology itself, but by reference to these languages. Parties, exchanging OWL-S profile descriptions or using OWL-S for discovery, need to agree on the language for expressing conditions and the notions of a “match” which is not addressed in the standard as such.

8.1.2 Service Model

A service might be *described by* a service model which exposes “how a service works”. The main use of a service model is to enable invocation, enactment, composition, monitoring, and recovery. The service model views the interactions of the service as a process. A process is not necessarily a program to be executed, but rather a specification of ways in which a client may interact with a service. OWL-S distinguishes between atomic processes, simple processes, and composite processes. Atomic processes are simple operations, whereas simple processes are views of complex or atomic processes. Composite processes are built up from atomic or simple processes by standard workflow constructs such as sequence, split, or join to determine the control flow, plus additional dataflow information (which outputs are routed to which inputs

within the workflow). Results and outputs may depend on a condition, again expressed in a third-party logical language (SWRL, KIF, or DRS).

Again, a possible problem, besides that of conditions, is that the semantics of the workflow constructs is not expressible in the description logics underlying OWL, for which reason this semantics has been externally defined [98].

8.1.3 Service Grounding

In order to map to the Web service world, an OWL service can *support* a grounding which maps the constructs of the process model to detailed specifications of message formats, protocols, and so forth. In detail, OWL-S allows one to map atomic processes to WSDL operations and their inputs and outputs to WSDL messages. These message mappings might have XSLT transformations attached, in order to solve the problem of lifting/lowering from the ontological representation in OWL to XML Schema in WSDL.¹ Note that, although WSDL is the only completely defined grounding for OWL, OWL-S does not restrict itself to WSDL as the only underlying service technology. Rather, OWL-S should be understood as a general service description ontology, and is extensible to other grounding mechanisms.

8.1.4 Relationship Between OWL-S and WSMO

The most obvious difference between OWL-S and WSMO is probably that the former is specified using the Web Ontology Language, whereas WSMO uses an abstract MOF model (as introduced in Section 5.3). By reusing OWL as a recommended standard, OWL-S gained considerable momentum. On the other hand, OWL-S defines its meta model in the same language that it uses for concrete service descriptions, which may be viewed as a disadvantage. WSMO's basis in the abstract MOF model successfully avoids this problem.

Moreover, OWL-S has the disadvantage of the need to “retrofit” more expressive languages into the OWL framework which then opens up new research questions on how they should interact. WSMO aims to avoid this problem upfront by providing a single language framework for expressing ontologies, conditions, and also dynamic aspects of WSMO within a single language, WSML.

When the language suggestions for WSML and OWL-S are compared, it turns out that OWL-S aims at combining various notations and semantics with OWL for the description of service conditions and effects, while WSML takes a more cautious approach. WSML does not distinguish between languages used for inputs/outputs and other description elements of the Web service, but rather provides one uniform language for capability descriptions. Additionally,

¹ For a more in-depth discussion of what is meant by lifting/lowering, we refer to Section 9.5 later in this book, where this problem is discussed in the context of WSMO.

the languages suggested for OWL-S are all based on classical logic, whereas WSML also offers the possibility to use nonmonotonic features from logic programming.

We also see similarities in the conceptual models of OWL-S and WSMO. A service profile in OWL-S is close to a capability of a service or goal in WSMO. Nevertheless WSMO makes a conceptual distinction between the provider's and the requester's view, which is not made in OWL-S.

The process model of OWL-S is conceptually similar to the WSMO service and goal interfaces. However, the distinction between the description of external behavior (choreography interface) and of the internal behavior (orchestration interface) is not made explicit in OWL-S.

WSMO mediators are not present in OWL-S. The authors of OWL-S claim, for instance in [107], that mediators in OWL-S are simply to be viewed as special kind of services, whereas we make the resolution of heterogeneities on several levels (data, protocol, and process) a first-class citizen in WSMO. As for the grounding, WSMO and OWL-S adopt similar ideas with respect to binding to WSDL. However, the grounding is not a top-level concept in WSMO, but is instead integrated into the WSMO interfaces. For a detailed comparison, we refer the interested reader to [81]. Note that neither WSMO nor OWL-S is restricted to WSDL alone as the only possible underlying technology, instead they are open for bindings to other grounding mechanisms to be defined.

8.2 SWSF

The Semantic Web Services Framework (SWSF) [9] is a relatively recent attempt towards a Semantic Web service annotation framework that greatly profits from previous work with its roots in OWL-S and the Process Specification Language (PSL), standardized by ISO 18269. This framework is a joint proposal by the Semantic Web Services Language Committee and was also submitted to the W3C in September 2005. SWSF is based on two major components: an ontology (or conceptual model) and a language used to axiomatize it.

8.2.1 Conceptual Model

The Semantic Web Services Ontology (SWSO) has been influenced by OWL-S and shares its three concepts of profile, model, and grounding, as described in the previous section. Thus SWSO can be seen as an extension or refinement of OWL-S. Although there are many similarities to the OWL-S ontologies, one important difference is in the expressiveness of the underlying language, which is, instead of OWL, a richer language called the Semantic Web Service Language (SWSL). Another fundamental aspect is a richer behavioral

process model based on PSL [54]. In the SWSF approach, there are two independent formalizations of the conceptual model, both expressed in variants of the underlying SWSL.

In the following we briefly review the conceptual model as it is formalized by the FLOWS (First-order Logic Ontology for Web Services) ontology, based on the SWSL-FOL language variant. The second axiomatization, ROWS (Rule Ontology for Web Services), shares the conceptual model but provides a different set of concrete semantics, which rely on the semantics of SWSL-Rules, this semantics being closer to that of logic programming.

The process model is the part of the FLOWS ontology which offers constructs to describe the behavior of the service based on the PSL approach. In order to do so, two fundamental elements are added: (1) the structured notion of atomic processes as found in OWL-S, and (2) the infrastructure for specifying various forms of data flow. By the use of PSL, each of the constructs used inherits the semantics defined in PSL, which overcomes some initial problems with the underdefined workflow semantics of the OWL-S process constructs.

The core part of this PSL extended by FLOWS is called PSL Outer Core, and the resulting FLOWS subontology is called FLOWS-Core. The process ontology is made up of six parts, which are divided according to their expressivity such as ordering constraints and occurrence constraints.

8.2.2 Language

SWSL is a language for describing the concept of Web services and for describing individual services. As previously mentioned, SWSL comes in two variants: SWSL-FOL and SWSL-Rules. The design of both languages was driven by compliance with Web principles such as the use of URIs, integration with XML built-in types, and XML-compatible namespaces and import mechanisms. Both languages are layered languages where every layer includes a number of new concepts that enhance the modeling power of the language. This means that they are richer and more expressive than OWL-S, which is based on OWL DL. Being based on first-order logic, FLOWS makes use of logic predicates and terms to model the state of the world.

Features from situation calculus (see for instance [113]), such as the use of fluents (i.e., predicates, and terms which vary over time), have been introduced to model changes in the world, in a way compatible with PSL. However, PSL has a more elaborate ontology of time and the occurrences of activities than has pure situation calculus. For details, we refer to [9, 54].

SWSL-Rules is a logic programming language, which includes features from courteous-logic programs [51], HiLog [26] and F-Logic [72] and can be seen as both a specification and an implementation language. SWSL-Rules provides support for service-related tasks such as discovery, contacting, and policy specification. The language is layered as illustrated in Fig. 8.2.

The core of the SWSL-Rules language is represented by its pure Horn subset. This subset is extended by adding features such as disjunction in the

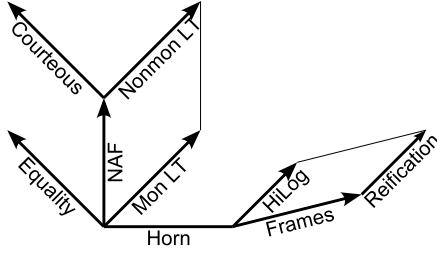


Fig. 8.2. The layered structure of SWSL-Rules [9]

body and conjunction and implication in the head (Mon LT) [85], and negation in the rule body interpreted as “negation as failure”. Other extensions are based on courteous rules, HiLog, and Frames.

The second language, SWSL-FOL, is based on first-order logic and also includes features from HiLog and F-Logic. It has a layered structure similar to that of SWSL-Rules with various extensions. Some of the extensions provided for SWSL-Rules apply to SWSL-FOL as well.

8.2.3 Relationship Between SWSF and WSMO

As for the conceptual model, SWSF shares many common features with OWL-S, and thus similar considerations to those above apply for comparison with WSMO.

A more interesting aspect concerns the language used. Many of the deficiencies of OWL-S have been overcome in SWSF by

- not being bound to the restricted expressivity of description logics;
- providing rigid definitions of semantic aspects of conditions and dynamic aspects of Semantic Web services by reuse of PSL;
- providing a single semantic language framework.

We remark that all of these points are also major design goals of WSMO and WSML.

However, as opposed to WSMO, OWL-S, and WSDL-S, which will be discussed in the next section, we are unaware of any serious tools for Semantic Web services, and ongoing implementation efforts based on SWSF so far. Since such efforts also seem unlikely to be expected in the near future, SWSF is in danger of remaining an interesting but mainly theoretical part of the development of Semantic Web services. Further adoption of the concepts of SWSF in the Web service community will probably also depend on the acceptance of the PSL standard, with which SWSF provides a solid and very valuable formal underpinning.

8.3 WSDL-S

Compared with WSMO, OWL-S, and SWSF, WSDL-S [1] is a rather minimalist approach which aims at a direct extension of the existing “traditional” Web service descriptions in WSDL with semantics (indicated by the last letter of the acronym). Although the most recent W3C member submissions have been mentioned in this chapter, the initiative has its roots in the METOR-S project [110, 132], which has been going on for several years at the LSDIS Laboratory², Athens, Georgia. WSDL-S was proposed in the form of a member submission to the W3C in November 2005 by IBM and the University of Georgia.

This approach proposes a lightweight mechanism to augment Web service descriptions in WSDL with semantics. By adding annotation tags to the XML schema of WSDL, the proposal picks those aspects similar to those in WSMO capability definitions or OWL-S profiles which appear most relevant to the authors of WSDL-S.

Using the extensibility of WSDL elements, a set of annotations can be created to semantically describe the *inputs*, *outputs* and *operations* of a Web service, as well as to specify a service category in an externally defined ontology, as is schematically shown in Fig. 8.3.

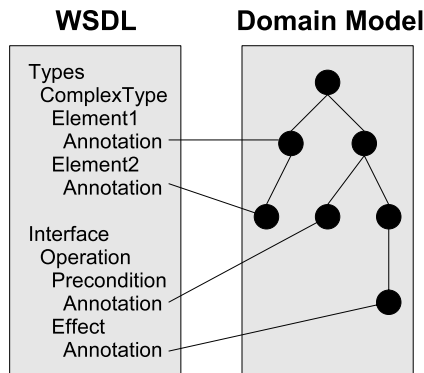


Fig. 8.3. Schematic illustration of how WSDL-S provides links to domain models

This method keeps the semantic model outside WSDL, making the approach impartial in relation to any ontology representation language. The authors of WSDL-S mention WSML, OWL, and UML as possible candidates.

The advantage of this approach is that it builds upon and stays close to existing industry standards. The underlying design principles of WSDL-S can be summarized as follows:

² See <http://lsdis.cs.uga.edu>.

1. WSDL-S aims to build upon existing Web service standards and promotes an upwardly compatible mechanism for adding semantics to Web services.
2. Annotations should be impartial with respect to the representation language for the semantic model. Consequently, WSDL-S does not prescribe which semantic representation language should be used.
3. Support for XML Schema datatype annotations needs to be added to the XML Schema, as this is the most important data definition format in the realm of Web services.

The third item, i.e. closing the gap between WSDL message descriptions (inputs/outputs) in terms of XML Schema and their semantic counterparts, is precisely the problem addressed in WSMO (and OWL-S) in the *grounding* component. However, WSDL-S differs fundamentally in being linked to WSDL and not defining semantic annotations separately. Instead, WSDL-S links from WSDL to the semantic description in terms of ontological concepts which are defined externally.

In addition, attributes for referencing mappings between XML Schema complex types and the corresponding ontological concepts are included in WSDL-S. In its specification, WSDL-S exemplifies XSLT and XQuery as possible technologies to define such lifting/lowering from XML Schema to the ontological level.

8.3.1 Language

WSDL-S does not fix a specific formalism for semantic descriptions. WSML, OWL, and UML are mentioned as examples of formalisms that could be adopted. Remarkably, unlike the other frameworks described so far, WSDL-S fixes the underlying service technology (referred to as “grounding” in the other formalisms) to WSDL by definition.

8.3.2 Relationship Between WSDL-S and WSMO

WSDL-S, with its minimalist approach can be viewed as orthogonal to WSMO. It can even use to annotations in WSML for defining conditions, etc. in its annotations or references to WSMO descriptions.

We have already mentioned that WSDL-S does not claim to be a fully-fledged description framework/ontology, but rather simply adds some useful attributes to WSDL’s XML tags in order to reference semantic annotations. All the referenced concepts such as message types, operation preconditions, effects, and service categories can likewise be annotated in WSML/WSMO.

Unlike WSMO, WSDL-S does not clearly establish how requests should be defined. As we move towards a standard Semantic Web architecture, the complete impartiality of WSDL-S with respect to the language used for defining semantic annotations might be criticized. Without a certain degree of commitment to a specific language, or, at the very least, a definition of how

different semantics of usable languages relate to one another, it is impossible to formally define requests, queries, or notions of a “match” between service requests and service descriptions.

With respect to nonfunctional properties of services, WSDL-S points to several existing proposals for standards from the Web service community. For instance, the authors of WSDL-S remark that it is necessary to “[...] investigate how to represent QoS assertions using ontologies and rules by extending the WS-Policy framework” [1]. Likewise, as for behavioral descriptions or interface descriptions, similar extensions of BPEL or WS-CDL are mentioned as being possible. The concrete integration with such existing standards has not yet been tackled, and remains open for future work. In terms of industrial adoption of these standards, the WSDL-S team definitely has a point here. In fact, the WSMO and WSMX working groups are investigating closer alignment with industry standards, and the adoption of WSDL-S/METEOR-S itself as a possible future grounding formalism for WSMO has even been considered [131].

8.4 Summary

As we have learned in this chapter, apart from WSMO, several other ongoing initiatives are progressing towards the development of frameworks for adding semantic annotations to Web services.

We also learned that despite some fundamental conceptual differences, there are several features common to these approaches. For instance, they all seem to agree on a minimal subset of necessary aspects of semantic service annotations:

1. General *service classifications* are common to all approaches.
2. A notion of *preconditions and postconditions* for successful service execution is common to all approaches, on the level of either atomic operations or complex service interactions.
3. As for the *behavioral/protocol* descriptions of a service, WSMO, OWL-S, and SWSF all allow one to encode complex behavior and interaction patterns; in WSDL-S, this can probably be simulated to a limited extent by adding preconditions and effects to individual WSDL operations, for example by embedding WSDL-S into BPEL4WS [4]
4. Nonfunctional aspects (such as quality of service, cost, and availability) can be modeled in most approaches. WSMO, OWL-S, and SWSF all provide extensible sets of nonfunctional properties in their ontologies, whereas WSDL-S sees this as out of its scope and instead points to related WS-* standards.

However, WSMO has some distinct features compared with the other approaches mentioned. First, mediators are a unique top-level concept in WSMO, whereas in OWL-S and SWSF the aspect of resolving heterogeneities

is not treated separately, rather is viewed as being dealt with by special kinds of services. Second, WSMO is the only approach that makes a clear distinction between the provider's and the requester's view by separating goal and service annotations. The motivation here is to provide a clear view on the following questions:

- How are requests/queries for particular service to be formulated?
- What are the related notions of a "match"?

Finally, we have also emphasized that for the latter question, a certain degree of language commitment seems necessary, which is provided by the persistent use of WSML within our framework.

Tools and Applications

Semantic Web Service Usage Tasks in WSMO

In this chapter, we exemplify some of the tasks that arise in the use of a generic Semantic Web service framework. Given the complexity of the matter and the state of the research, we shall not go into the deep details of realization but rather aim to give the reader a general idea of how the semantics can be usefully applied to several tasks. For the reader interested in concrete code examples, we refer to the appropriate deliverables of the respective working groups within the text.

We start this chapter by introducing a generic use case that is referred to throughout the rest of the chapter. The list of tasks covered in this chapter is not exhaustive but is intended to represent the most prominent areas in current research.

9.1 The Virtual Travel Agency Scenario

The following depicts the Virtual Travel Agency (VTA) scenario which we shall refer to to illustrate semantically enabled Web service technologies throughout this chapter. This use case scenario is frequently applied for demonstrating and illustrating the functionality of Web services (e.g. [61],[124]).

9.1.1 Overview of VTA Scenario

The Virtual Travel Agency is an e-tourism service provider which offers travel-booking services to the end user by using and interacting with other, more basic e-tourism service providers. The functionality of the VTA is that of a traditional travel agency: getting a request from a customer, dealing with various e-tourism providers to put together an appropriate offer covering the customer's request, arranging all the bookings (and payments) with the providers, and transparently offering the final trip arrangement to the customer.

In principle, the VTA is a Web service that offers information and booking facilities for several aspects related to traveling, for example booking tickets

for planes or trains, booking hotels, etc. The VTA Web service achieves its functionality by aggregating Web services from other travel service providers. Figure 9.1 provides an overview of this use case setting, and the following is a list of the actors participating in this scenario:

- *Customer*: the end user who sends a trip-booking request to the VTA.
- *Tourism service providers*: commercial companies that provide specific tourism services.
- *VTA*: the intermediary between the customer and the tourism service providers. This provides tourism packages to customers by aggregating the separate services of various tourism service providers.

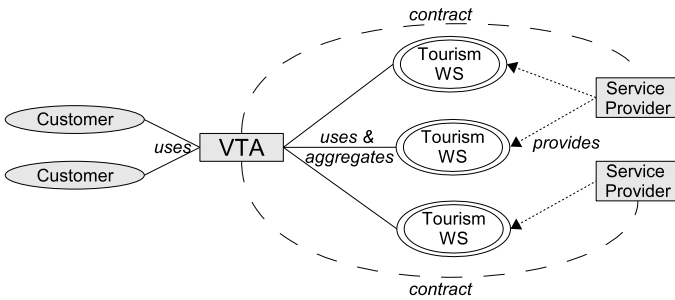


Fig. 9.1. Overview of the use of the VTA

9.1.2 Semantic Web Service Usage Tasks

Let us consider the following exemplary setting. A customer wants to take a trip to a given location (e.g., from Innsbruck to Vienna) for a given period of time (e.g., staying there from August 10 to August 15). The customer sends a request to the VTA, which then has to build a package including travel to/from Vienna and accommodation for all the nights spent in Vienna. Clearly, the hotel has to be booked according to the flight (i.e., if the flight arrives on August 9, then the hotel has to be booked from August 9).

In this context, the VTA should take care of locating the necessary tourism service providers (e.g., suitable flight providers for the trip and hotels in Vienna) and contact them. Finally, a suitable offer will be returned to the customer and, upon acknowledgment, either the accommodation, the transportation, both, or none will be booked. This booking requires some sort of weak transaction between the invoked services. We assume that the available e-tourism providers will be dynamically located by the VTA, with no need for prior agreements, and that the business process of the VTA will be dynamically composed on the basis of the request received and the available

providers. Therefore, we assume that the VTA, the customers, and all other Web service providers define WSMO descriptions of their respective resources. The main steps that need to be carried out by the VTA in order to deal with the request require the following techniques:

- *Discovery.* Out of potentially tens of thousands of Web services available on the Internet, the VTA needs to determine which of them are usable for a specific request. Here, a Web service for booking flight tickets from Innsbruck to Vienna and a Web service for booking hotels in Vienna are needed.
- *Composition.* The Web services utilized by the VTA need to be composed, meaning that a suitable execution order of the flight-booking and the hotel booking Web services needs to be determined. Here, the outputs of the flight Web service may serve as inputs to the hotel Web service, as the arrival date at the booked hotel needs to be the same as the arrival date of the flight.
- *Mediation.* The resources needed to process the request may be heterogeneous, which hampers successful interaction. For example, the providers of the hotel and flight Web services may use different ontologies in their service descriptions so that information cannot be usefully interchanged, or the Web services may require incompatible business processes for consuming their functionality. These mismatches need to be resolved in order to allow proper interaction of the Web services, the VTA, and the client.
- *Execution.* After all the necessary Web services have been discovered, composed, and mediated successfully, the VTA needs to execute them. In order to do so, information interchange as specified in the interfaces of the Web services should take place automatically.

The subsequent sections illustrate and describe semantically enabled techniques that work on WSMO descriptions of Web services, goals, and mediators. It should be noted that the VTA is a placeholder or synonym for generic Semantic Web service execution environments. As the technology matures, the VTA will no longer be necessary, because the clients will be able to skip the intermediary “middle man.” New technologies will allow clients to accomplish their goals without relying on the VTA for its mediation and aggregation service technologies.

9.2 Discovery

WSMO provides a conceptual framework for semantically describing Web services and their specific properties. In this section we discuss how WSMO can be used for service discovery. We provide a proper conceptual grounding by strictly distinguishing between service and Web service discovery and then present various techniques for realizing Web service discovery. In order to cover the complete range of scenarios that can occur in practical applications,

several approaches to achieve the automation of Web service discovery are presented and discussed. They vary in complexity and precision and require different levels of semantics in the description of Web services and requests.

9.2.1 Discovery Framework

A workable approach to automating service discovery must define precisely its conceptual model and the particular assumptions underlying the proposed solution. To describe such an approach, we start by aiming to provide a common understanding of what a service is and the levels of abstraction in its description, on the basis of [111], and we also state our assumptions about the elements involved in the process of locating suitable services for service requesters. First we recap parts of the discussion from Section 4.1 (on Web service terminology). We use the following interpretation of the term *service*, as described in the conceptual architecture for Semantic Web services presented in [111]: *the provision of value in some domain*. This definition regards a service as a provision of value in some given domain, independent of how the supplier and the provider interact. Usually, a service provider does not provide only one particular service, but a set of coherent and logically related services. For instance, a hotel usually does not provide only the possibility to book a particular room on a particular date for a given number of nights, but instead it will offer (and advertise) the general service of booking rooms at that hotel. Following the terminology of [111], we call this *collection of services* an *abstract service* offered by a provider. The smallest unit of advertisement is considered to be an abstract service.

In order to deliver a service, a service provider usually needs certain information from the requester. For instance, a hotel might require the name of the person booking the room, the requested features of the room, and a valid credit card number, as input information in order to book a room. This input data will determine what has to be provided, a *concrete service*.

Finally, *Web services* are computational entities using standardized interfaces that allow clients to interact with a provider to access, explore and consume concrete services. A service requester needs, technically, to interact with a Web service to get what he/she is actually aiming at: a concrete service, or, more generally, a set of concrete services that fulfill the requester's goal, i.e. an abstract service. Hence, we shall not distinguish between Web services and abstract services of a provider but treat the two notions as synonymous. Moreover, we can observe that a concrete service is delivered in the course of or as a result of a particular *execution* of a Web service, by invocation with specific data for the required input parameters.

Dynamics of Abstract Services Over Time

In the real world, we have a further complication in regard to discovery and Web service description. In general, an abstract service offered by a provider

does not stay the same, but changes over time. For example, a hotel will not be able to book a room with a single bed on a specific date if all such rooms in the hotel are already booked on that date.

Since clients are basically interested in finding abstract services which can actually solve their problem at hand (as specified in a WSMO goal), discovery in general needs to consider these dynamics in order to create accurate results. This basically means that purely static descriptions of Web services are not sufficient in the general case. In applications where highly accurate results are requested, Web service descriptions will have to contain a dynamic component as well.

On a description level, there are two main options to achieve a proper representation of a dynamically changing abstract service from a provider: (1) use a purely static description of the abstract service and change the description in its entirety every time the abstract service changes; (2) use a static description where some parts refer to a well-defined resource that reflects currently valid information about the dynamic part (for instance, a database table containing all available rooms at a hotel at a certain point in time). Clearly, the first approach is not really scalable, since constant changes of a stored abstract-service description are needed. The second approach is a viable one, as we can see with the dynamically generated Web pages of online stores such as Amazon.com that provide an up-to-date perspective of the prices of individual books. In the latter case, we simply externalize the details of the changing aspects of the abstract service and provide a reference in the remaining part of the Web service description. Hence, the abstract-service description, including the reference, does not need to change over time when the abstract service of the provider changes, and hence the Web service descriptions become more compact.

Nonetheless, the latter approach requires *communication* with the provider of an abstract service (e.g. via a Web service that accesses a database table) and thus can potentially be a costly step in terms of computation time. Hence, there is a clear trade-off between the accuracy of discovery results and the efficiency of the discovery process, which must be considered for any specific application that leverages Web service discovery.

A Conceptual Model for Service Discovery

Approaches to automating discovery of suitable Web services must analyze precisely *what* kinds of descriptions can be used for capturing the static and dynamic aspects of a given Web service, and *how* such descriptions can be exploited for efficiently and accurately locating a requested service. While a number of proposals are available in our area of interest, (e.g. [11, 48, 84, 106, 126]), none of them has specifically addressed these aspects. Rather, they have focused mainly on specific description languages and frameworks, partly neglecting the overall needs. Therefore, we shall first define a model that takes into account pragmatic considerations and defines borderlines between

the various steps involved in the process of locating services, namely, goal discovery, goal refinement, Web service discovery, and service discovery.

Below, we shall first briefly describe heuristic classification, summarizing its relevance to service discovery. We shall then show how this model is applied in WSMO as an underlying conceptual pattern in order to structure the service discovery process.

Heuristic Classification

In [29], Clancey provided a landmark analysis in the area of expert systems. On the basis of an analysis of numerous rule-based systems for classification and diagnosis, he extracted a pattern of three inference steps that helped to understand the various production rules implemented in the various systems.¹ The problem-solving method that he called *heuristic classification* separates abstraction, matching, and refinement as the three major activities in any classification and diagnosis task (see Fig. 9.2).

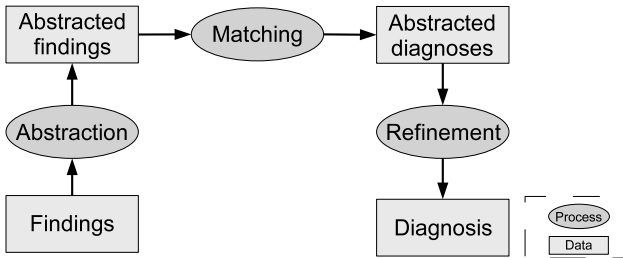


Fig. 9.2. The three major processes of heuristic classification

Abstraction is the process of translating concrete descriptions of a case into features that can be used for classifying the case. For example, the name of a patient can be ignored when making a diagnosis, his/her precise age may be translated into an age class, and his/her precise body temperature may be translated into a finding “low fever.” The process is about extracting features relevant to classification from a concrete description of a case.

Matching is the process of inferring potential explanations, diagnoses, or classifications from the extracted features. It matches the abstracted case description with abstract categories describing potential solutions.

Refinement is the process of inferring a final diagnosis, that explains the given findings. This process may include the acquisition of new features that describe the given case. However, it is now the potential solution that guides the process of acquisition of these features.

¹ In [104], heuristic classification has been used as a model for improving the translation process between different dialects of XML Schema.

As the last step indicates, the entire process can be executed in an iterative fashion. Instead of acquiring all potential findings, an initial set can be used to derive an intermediate potential explanation that can then be used to guide the next iteration of the process.

Service Discovery

The above is directly relevant because a scalable and workable approach to service discover should follow the same pattern (see Fig. 9.3).

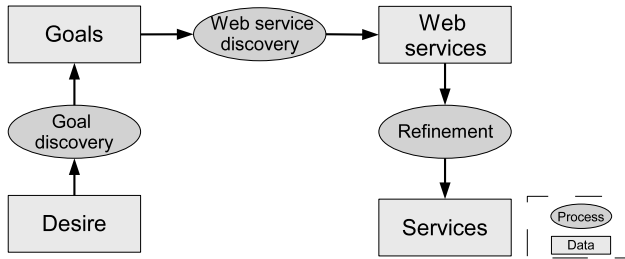


Fig. 9.3. The three major processes in service discovery

Users may describe their desires in a very personalized and specific way that makes immediate mapping to service descriptions very complicated. Therefore, each attempt at service discovery requires a process where user expectations are mapped to more generic goal descriptions. Note that this can be hidden by the fact that a discovery engine allows the user to select only from predefined goals. However, then it is simply the user who has to provide this mapping, meaning that the user must translate his/her specific requirements and expectations into more generic goal descriptions. This step can be called goal discovery, i.e. the user or the discovery engine has to find a goal that describes (with varying levels of accuracy) his/her requirements and desires. In the current literature on service and Web service discovery, this step is mostly neglected.

An example of such a user's desire (following the example of the VTA presented in Section 9.1) would be to buy a train ticket from Innsbruck to Vienna, to travel on December 12th 2004 with a departure time between 15:00 and 17:00. It can be seen that this is a very concrete and detailed desire, although goals are typically intended to be generic and reusable. Therefore, a mapping from the user's desire to a generic goal becomes necessary. In fact, Web services are means to find (and buy) services. They are to a large extent service discovery engines, and a discovery engine should not try to replace or duplicate this functionality.² Web service discovery here is about searching

² More precisely, Web services are an interface to a software artifact that may help one to find and buy services (see [111]); however, we neglect this distinction here

for Web services that can be used to buy tickets. And service discovery is about checking whether the ticket sellers offering such Web services can really provide the requested ticket. We do not think that complete, correct descriptions of all the services offered by a Web service are a realistic expectation; Web services would then be no longer necessary (at least not the discovery process). We wonder whether logical reasoning would scale under these conditions where the execution of an efficient program accessed via the Web service would be simulated by reasoning over its semantic annotation.

Alternatively, one might assume to directly query the Web service during the Web service discovery process. However, this may lead to network and server overload, and it makes a very strong assumption: in addition to data, protocol, and process mediation, the Web service must first be established before the discovery process can take place. In general, we do not think that this is a realistic assumption; however, we defer a more detailed discussion to Section 9.3. As a result, we think it is essential to distinguish between Web service discovery and service discovery in order to come up with a workable approach that scales and makes realistic assumptions in a pragmatic environment. In summary:

- Web service discovery is based on matching abstracted goal descriptions with semantic annotations of Web services. This discovery process can happen only on an ontological level, i.e. it can rely only on conceptual and reusable data. For this, to be possible, two processes are required: (1) the concrete user input has to be generalized to more abstract goal descriptions, and (2) the concrete services and their descriptions have to be abstracted to the classes of services that a Web service can provide. We believe that this twofold abstraction is essential for lifting Web service discovery to an ontological level, which is a prerequisite for a scalable and workable solution.
- Service discovery is based on the use of Web services for discovering actual services. Web service technology provides automated interfaces to the information provided by software artifacts that is needed to find, select, and eventually buy a real-world service or simply find a piece of information that somebody is looking for. Service discovery requires strong mediation and wrapping, since the specific needs of the choreography of a Web service must be met in order to interoperate with it. Note that automation of service discovery defines significantly higher requirements of mediation than does that of Web service discovery, since protocol and process mediation is also required.

and refer to the Web service interface and the accessed software artifact as one entity.

9.2.2 Web Service Discovery

We now discuss in more detail how Web service discovery is addressed within the WSMO framework. We outline several different levels that require different efforts for annotation and description of both goals and services and deliver results of different accuracy. Each approach addresses a different level of abstraction in Web service descriptions. The various techniques taken together help to create a workable solution to the problem of Web service discovery which addresses practical requirements and is based on realistic assumptions.

Web Services at Various Levels of Abstraction

The descriptions of Web services, as published by service providers, and the goals of service requesters need to be compared with one another during the discovery process in order to identify matching Web services and goals. In essence, matching in the discovery process is about finding common elements in these descriptions. Depending at the level of detail which we consider these entities, we end up with models of Web services at varying levels of abstraction:

At the most fine-grained level, we can consider services as concrete state transitions (s, s') from a state s to another state s' . The states s and s' determine precisely how the world is before and after the service has been provided. On a more abstract level, we can ignore the detailed structure of services (i.e. the respective states), understand them purely as abstract objects, and characterize their specific properties. In terms of ontologies, we would then consider services as instances and Web services as concepts. At the most abstract level, we can abstract even further in the description of abstract services by ignoring the description of the possible elements of the set. At this level, we would simply use a term or keyword for describing the abstract service and neglect any information about the fine-grained structure. Essentially, we are considering keyword-based descriptions here.

Each of these levels of abstraction implies a different description of Web services, ranging from detailed characterizations of possible state transitions, less detailed descriptions using (complex) concepts in an ontology, to simple unstructured keywords. Consequently, the achievable accuracy of a result in the discovery process varies significantly, since more or less structure is reflected in the descriptions. On the other hand, the ease of providing the descriptions varies drastically between these levels as well. Whereas simple keywords are easy to provide, the descriptions of concepts are not hard to provide but this requires more effort. The provision of detailed state-based descriptions definitely requires more skill from the people creating the formal descriptions. The more fine-grained the information is, that the descriptions reveal, the more complex the algorithms must be in order to deal with these descriptions.

Therefore, there is an interesting trade-off between the possible achievable accuracy, the ease of creating the descriptions, and the degree of computational efficiency of the discovery process.

Keyword-Based Discovery

Keyword-based discovery is a basic ingredient in a complete framework for Semantic Web service discovery. By performing a keyword-based search, the huge amount of available services can be filtered or ranked rather quickly.

In a typical keyword-based scenario, a keyword-based query engine is used to discover services. A query, which is basically a set of keywords, is provided as input to a query engine. The query engine matches the keywords in the user's input against the keywords used to describe the service. A query with the same meaning can be formulated by using a dictionary of synonyms, such as WordNet [37]. The semantics of the query remains the same but, because of the different keywords used (synonyms of previous ones), more services that possibly fulfill the user's request are found. Moreover, by using dictionaries such as WordNet, as well as natural-language processing techniques, an increase in the semantic relevance of the search results (with regard to the search request) can in principle be achieved [115]. Nonetheless, such techniques are inherently restricted by the ambiguities of natural language and the lack of semantic understanding of natural-language descriptions by algorithmic systems.

The services descriptions at least consist of a dedicated set of keywords for categorizing and indexing. Furthermore, they can include richer, semantic-based elements.

Discovery Based on Simple Semantic Descriptions

Although keyword-based search is a widely used technique for information retrieval, it does not use explicit, well-defined semantics. The keywords used to retrieve relevant information do not have an explicit formalization and, therefore, do not allow inferencing to improve the search results.

For these reasons, as a second approach, we consider the use of controlled vocabularies with explicit, formal semantics. Ontologies provide excellent and prominent conceptual means for this purpose. They provide an explicit and shared terminology, and explicate interdependencies between concepts, and thus are well suited to the description of Web services and goals. Moreover, ontologies can be formalized in logics, which enables the use of inference services for exploiting knowledge about the problem domain during discovery.

Web services at this intermediate level of abstraction, are understood as sets of (unstructured) elements, i.e. as concepts from an ontological perspective. A capability of an Web service at this level of abstraction is a description which does not depend on dynamic factors. A capability describes only *what* an advertised Web service can potentially deliver and not under which circumstances (meaning preconditions) particular services can actually be provided.

Discovery Based on Rich Semantic Descriptions

When they are a part of WSMO or another rich model description frameworks such as OWL-S [87] or SWSF [9], functional descriptions of Web services are *syntactic expressions* in some specification language using the precondition/-postcondition style of description. This description specifies the possible state transitions that are caused by service invocations. The logical formulas are constructed from terms in an ontology, in a way in which the axiomatization of these terms can be kept general (unspecific) and need not be repeated for each service description.

Each description captures specific requirements for Web services and can be used to constrain the set of all Web services to some subset that is relevant to a particular context. At this level of abstraction, the descriptions describe constraints on the valid prestates and resulting poststates. In contrast to the previous level of abstraction, we are now able to describe relations between such states; for example the balance of a bank account supplied as input may be decreased by a certain amount.

9.2.3 Matching Goals and Web Services

The general approach is to look for certain types of logical relationships between the semantic capability descriptions of Web services and requests (i.e. goals). If such a relationship holds, a Web service is considered to be usable for resolving the client's goal. On the basis of several preceding studies on semantic matchmaking [84, 106], we have identified five matchmaking notions for functional discovery as shown in Fig. 9.4. The important characteristic of these notions is that each one denotes a different logical relationship that must hold true when considering whether a Web service is suitable for achieving a given goal. For instance, an Exact Match holds if and only if each possible ontology instance that can satisfy the Web service also satisfies the goal, and there exists no possible ontology instance that satisfies only the goal or the Web service. In contrast, an Intersection Match holds if there exists one possible instance that can satisfy both the goal and the Web service. Hence, in order to precisely express client objectives with respect to discovery, WSMO goals carry an additional non functional property `typeOfMatch`, denoting the matchmaking notion to be applied.

Several prototypical realizations of semantically enabled discovery already exist: [84] and [49] apply description logic reasoners, the approach in [71] is based on frame- and transaction logic, and [123] uses a FOL theorem prover as the technical platform for semantically enabled discovery. However, all these approaches are restricted to the specific reasoning support provided by the tools used. Hence, it is expected that these approaches will converge towards an integrated discovery framework for Semantic Web services that provides appropriate reasoning facilities.

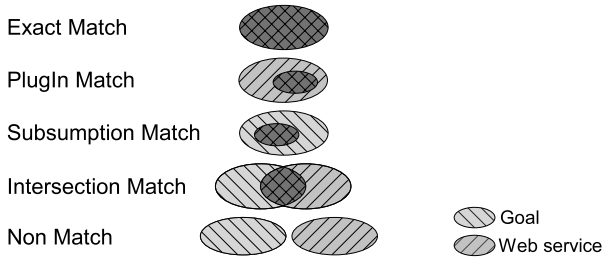


Fig. 9.4. Matchmaking notions for semantically enabled discovery

We have presented a conceptual model for service discovery which avoids unrealistic assumptions and is suitable for a wide range of applications. One of the key features of this model is the explicit distinction between the notions of Web services and services. We have outlined various approaches to the discovery of Web services, with various requirements on the description of Web services and the discovery goal itself. We have concluded by emphasizing the advantages of our favored approach: semantic-based Web service discovery.

9.3 Mediation

Owing to its design principle of decentralization, the World Wide Web is a network of decoupled, independently working computers. This makes the Web heterogeneous by nature: people create Web sites and applications independently, resulting in mismatches that hamper information interchange and interoperability. In consequence, the Semantic Web will be heterogeneous as well. The techniques for handling and resolving mismatches that hamper interoperability of Web resources require mediation, a central pillar of the next generation of Web technologies.

In the early 1990s, Wiederhold proposed so-called *mediator-oriented architectures* for handling heterogeneities in IT systems [135]. In these architectures, mediators are integrated components capable of dynamically handling heterogeneities that hamper successful interoperation of system components. For generic, application-independent mediation, the mechanisms for mismatch resolution need to work on a structural level, based on declarative resource descriptions. One of the merits of the Semantic Web is that resources carry semantic descriptions, which allows mediation techniques to be defined on a semantic level. Whereas OWL-S [87] defines an ontology for semantically describing Web services yet remains orthogonal to mediation [107], WSMO identifies mediation as a first-class citizen and, consequently, defines mediators as a core element of Semantic Web services.

In order to attain a mediator-oriented architecture in accordance with Wiederhold's conception, the WSMO mediation model presented in [122] distinguishes two dimensions: (1) *mediation techniques* for resolving various kinds

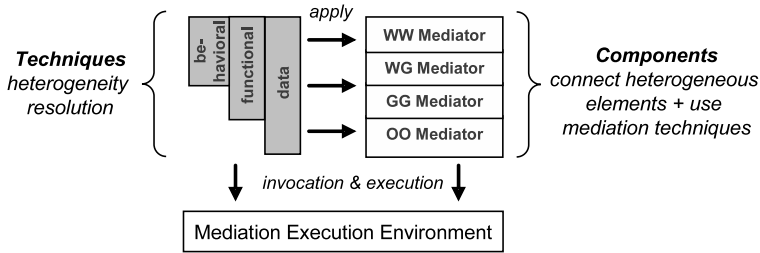


Fig. 9.5. Dimensions of mediation

of heterogeneities that can arise within the Semantic Web, and (2) *logical components* that connect resources and apply the required mediation techniques. These are embedded in a software architecture for dynamic invocation and execution. Figure 9.5 shows the structure of the mediation model, which we explicate below.

9.3.1 Mediation Levels and Techniques

The first dimension of the model is concerned with the types of heterogeneities that can occur within the Semantic Web. The various heterogeneity types require specific techniques for mismatch resolution, which we refer to as *levels of mediation*. The development of Semantic Web technology has revealed four types of heterogeneity. We shall explain this categorization and describe mediation techniques for each level developed around WSMO, referring to other material for more detailed explanation where necessary. The types of heterogeneity are as follows:

1. *Terminology.* Web services or other Web resources use different terminologies. For example one entity understands **name** to be the full name of a person, and another one defines **name** to denote only the family name. This can hamper successful interoperation on the semantic level.
2. *Representation format and transfer protocol.* Interacting resources use different formats or languages for information representation (e.g. HTML, XML, RDF, or OWL), or different protocols for information transfer (e.g. HTTP or RPC); incompatibilities on this level can obviously hamper useful information interchange.
3. *Functionality.* This is specific to Web services, and refers to functionalities of a provider and a requester that do not match exactly. This enforces complex and thus expensive reasoning procedures for detecting Web services that are usable for a given request. The need for such expensive operations can be reduced by gaining and utilizing knowledge about the functional heterogeneities, as explained in more detail below.
4. *Business process.* Also specific to Web services, this denotes mismatches in the supported interaction behavior of Web services and clients. This

can hamper successful interaction on a behavioral level for the purpose of consumption or interaction of Web services.

Data-Level Mediation

The first mediation level addresses the first two types of heterogeneity identified above. As these are strongly interconnected and can be handled by similar techniques, they are consolidated in *data-level mediation* [93]. This provides a general mediation technique for Semantic Web applications.

The most common type of mismatch in the Semantic Web occurs owing to usage of different terminologies by entities that need to interchange information. In ontology-based environments such as the Semantic Web, this results from the use of heterogeneous ontologies as the terminological basis for resource or information descriptions. A major advantage of ontologies is that such mismatches can be handled on a semantic level by *ontology integration techniques*, explained below in more detail. Regarding the second type of heterogeneity, of representation formats and transfer protocols, a suitable method of handling is to lift the data from the syntactic to a semantic level on the basis of ontologies, and then resolve the mismatches on this level [95].

Techniques for Data-Level Mediation

The central mediation techniques for the data level are semantically enabled information integration techniques. Collectively referred to as ontology integration, the main techniques are ontology mapping, alignment, and merging, which we briefly summarize in accordance with [102].

- *Ontology mapping* involves the creation of a set of rules and axioms that precisely define how terms from one ontology relate to terms from another ontology. These rules and axioms are expressed using a mapping language, as in the example given below. Ontology mapping refers to mapping definitions only; none of the ontologies involved are changed.
- *Ontology alignment* has the role of bringing the ontologies involved into mutual agreement. As in the ontology mapping technique, the ontologies are kept separate but at least one of them has to be altered so that the overlapping parts of the ontologies involved are “aligned” (i.e. they match).
- *Ontology merging* results in the creation of a new ontology that replaces the original ontologies. The merging can be done either by unification (all the terms from the ontologies involved are included, and mismatches between the overlapping terms are resolved) or by intersection (only the overlapping terms are included and their mismatches reconciled).

Example of Data-Level Mediation

In the context of the VTA, suppose that a client uses an ontology different from that of the VTA Web service description. For illustrating the handling

```

// Heterogeneous Ontology Concepts
ontology _"http://www.example.org/ontologies/TravelRequestOntology"
  concept station
    startLocation ofType _boolean
    destinationLocation ofType _boolean
    name ofType _string

ontology _"http://www.example.org/ontologies/TravelOfferOntology"
  concept route
    from ofType (0 1) _string
    to ofType (0 1) _string

// Mapping Definitions
Mapping(http://www.example.org/ontologies/TravelRequestOntology#station
  http://www.example.org/ontologies/TravelOfferOntology#route
  classMapping(one-way station route))
Mapping(http://www.example.org/ontologies/TravelRequestOntology#destination_Location
  http://www.example.org/ontologies/TravelOfferOntology#to
  attributeMapping( one-way
    [( station ) destinationLocation => boolean] [(route) to => string]))
  valueCondition(station [( station ) destination_Location => boolean] true)
Mapping(http://www.example.org/ontologies/TravelRequestOntology#start_Location
  http://www.example.org/ontologies/TravelOfferOntology#from
  attributeMapping( one-way
    [( station ) start_Location => boolean] [(route) from => string]))
  valueCondition(station [( station ) start_Location => boolean] true)

```

Listing 9.1. Example of a definition of an ontology mapping

of terminology mismatches, consider that the ontology used by the requester contains the concept `station`, and that the one used by the provider contains the concept `route`. There are two terminological mismatches: (1) the attribute `startLocation` of the concept `station` corresponds to the attribute `from` in the concept `route`; (2) the attribute `destinationLocation` of the concept `station` corresponds to the attribute `to` of the concept `route`. In order to allow automated processing via ontology mapping, we need to create three mapping rules: one for stating the relation between the two concepts and two for imposing mappings between their attributes. Listing 9.1 shows this using the abstract mapping language presented in [119], for higher flexibility and easier maintenance of mappings. The constructs used here are mappings between concepts and mappings between attributes. Each mapping has a source and a target indicated by IRIs, a direction (here, there is one way from the source to the target), and possible conditions on the mapping (see [94] for details on syntax and semantics of the mapping language).

Process-Level Mediation

The second mediation level is concerned with mismatches on the behavioral level that can occur consumption of a Web service consumption or interaction of several Web services. For instance, at some point during the consumption of a Web service S by a requester R , R may expect an acknowledgement while S waits for the next input; therefore, the interaction process between

R and S runs into a deadlock situation. Within the WSMO framework, this mediation level refers to the interaction behavior described in the interfaces of Web services. These specify the interaction behavior supported or expected by the Web service for consuming its functionality (choreography), and for interacting with other Web services that are aggregated in order to achieve the functionality of the service (orchestration).

Techniques for Process-Level Mediation

Mismatches at the process level can occur in every interaction that a Web service is involved in. These heterogeneities can be resolved by inspecting the individual business processes of the entities that interact, while trying to establish a valid process for interaction on the basis of predefined mediation operations for the business processes. Reference [28] presents a prototype that supports the following patterns for process-level mismatch resolution, as illustrated in Fig. 9.6: (a) unexpected message stop, (b) order inversion, (c) splitting, and (d) merging.

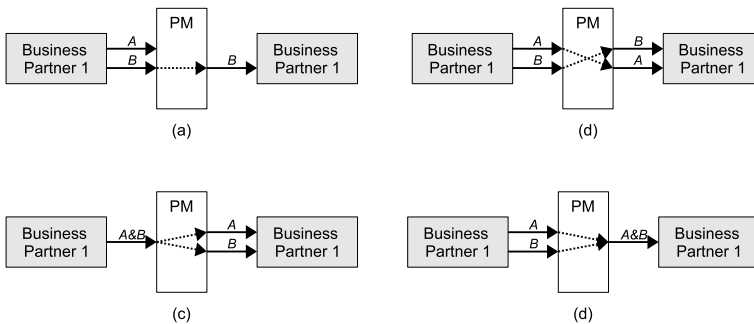


Fig. 9.6. Process mediation patterns

Example of Process-Level Mediation

The following exemplifies how these process mediation patterns can be applied to resolve a communication mismatch in the VTA scenario. A requester R wants first to send information about the travel date, followed by the start location and end location of the trip. The provider P wants to receive first the route, and then the data about the trip. R and P use ontologies containing the concepts shown in Listing 9.1.

Figure 9.7 gives an overview of this situation. The interaction between the requester and the provider is initialized by an outgoing message from R with content of type `date`. But P expects an incoming message containing a `route`. The process mediator inspects the interaction behavior of R and determines

that the second and third outgoing messages contain instances of `station` that can be mediated to `route` by the mappings defined above. Hence, the process mediator applies the order inversion pattern in order to hold the first message from R , and then – after data level mediation has taken place – uses the merging pattern. The information can now be submitted to P in the expected order and using the expected terminology.

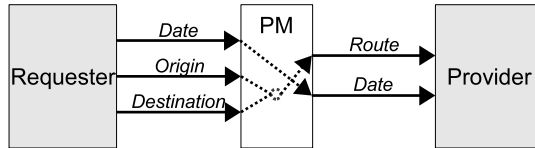


Fig. 9.7. Example of process-level mediation

Functional-Level Mediation

Heterogeneities at the functional level arise when the functionality provided by a Web service does not precisely match that one requested by a client. For instance, in our VTA scenario, a requester may define a goal of purchasing a ticket to travel from Innsbruck to Vienna without specifying the type of ticket (i.e. for a bus, train, or plane). An available Web service offers train tickets from Innsbruck to Vienna. Here, the Web service is only usable for the request under the condition that the ticket is a train ticket.

We expect situations like this to be common when Web services are used. In order to determine the usability of a Web service for a given request – commonly referred to as functional discovery, a central reasoning task for automated Web service usage – complex and thus expensive reasoning procedures are required [66]. As this hampers the efficiency of Semantic Web technologies with regard to Web scalability, we use “ Δ -relations” to denote functional heterogeneities and allow us to eliminate or reduce the need for such expensive operations.

Techniques for Functional-Level Mediation

The central technique for functional-level mediation is the use of Δ -relations that denote the explicit logical relationship between functional descriptions of Web services and goals. Functional descriptions are a central pillar of comprehensive Web service description frameworks such as OWL-S and WSMO. Defined by conditions on preconditions and effects in some ontology language, they provide a black-box description of normal runs of a Web service, omitting information about the technical details of the invocation of the service [67].

The desired relationship can be described most adequately as the semantic difference between functional descriptions. For two given functional descriptions ϕ and ψ , the Δ -relation between them is defined as $\Delta_{\phi,\psi} = (\phi \wedge \neg\psi) \vee (\neg\phi \wedge \psi)$, i.e. Δ contains those elements that are models for either ϕ or ψ and not common to them. A Δ -relation defines a symmetric relation between ϕ and ψ ; when we concatenate $\Delta_{\phi,\psi}$ with either ϕ or ψ , we obtain logical equality with the corresponding other formula.

Beneficial Usage Scenarios.

While the benefits obtainable from data- and process-level mediation are obvious, the benefits of functional-level mediation may be less obvious. Therefore, let us consider the following situation in the VTA scenario: G_1 defines a request to buy a ticket from Innsbruck to Vienna, WS_1 provides a purchase facility for train tickets in Austria, and WS_2 provides a facility for booking flight tickets in Austria and for international flights.

The Δ -relations between G_1 and each of the Web services provides additional knowledge about the circumstances under which each Web service can be used for resolving G_1 : using WS_1 will result in the purchased ticket being a train ticket; using WS_2 will result in the purchased ticket being a flight ticket. This additional information can for example be utilized for selecting the Web service to be used. Imagine that the client who defines G_1 has a preference for traveling by plane because of tight business schedules. Here, we would choose WS_2 for use as the constraint of using WS_1 contradicts the client's preference. Other beneficial mechanisms that work with Δ -relations are:

- *Goal Adjustment.* This is concerned with adjusting goal definitions that are not resolvable a priori. Within our scenario, imagine that a client defines a goal G_3 of buying a flight ticket from Innsbruck to Vienna for less than € 200. The flight booking Web service WS_2 seems to be usable, but all purchasable flight tickets cost significantly more than € 200. However, train tickets for the desired itinerary are available for less than € 200 from WS_1 . With respect to the Δ -relation between G_3 and the Web services, we can propose possibilities for adjusting the goal. Either weaken it by omitting the price constraint so that WS_2 becomes usable, or change the desired ticket type to train ticket so that WS_1 becomes usable. Note that in any case the objective of the client as specified in G_3 will be changed, so that interaction with the client is required for confirmation.
- *Goal Ontologies.* Presumably, the most effective usage scenario with respect to gainable efficiency increase would be to compute the Δ -relations between already resolved goals and Web services, and then to utilize this additional knowledge to avoid or reduce the need for complex reasoning procedures for functional discovery. For instance, we can omit runs of discovery engines for determining usable Web services for the above-defined goal G_2 if we already know WS_1 and WS_2 to be usable for G_1 , as we obtain from the Δ -relation between G_1 and G_2 the fact that they are structurally

the same – apart from different concrete literals – so that the same Web services must be usable for resolving both goals. This allows the definition of beneficial algorithmic procedures for avoiding or reducing the need for expensive reasoning operations in functional Web service discovery. We refer to [121] for details of this technique.

9.3.2 Mediation Components

The second dimension of the WSMO mediation technology relates to the logical components that utilize the mediation techniques presented above in order to resolve mismatches. As a consequence of the dynamic and evolving nature of the Semantic Web, an essential design principle for a comprehensive mediation architecture is *modularity*, i.e. delegation of mediation to distinct components in order to enable reusability of mediation facilities [41].

The WSMO metamodel defines four types of mediators that appear to be applicable within the Web service usage process. The mediator type is indicated by a prefix denoting the types of the source and target components. Each mediator type applies those mediation techniques which are required for resolving the heterogeneities that can possibly occur between the connected components. Figure 9.8 gives an overview of the WSMO mediator typology with respect to the mediation techniques used.

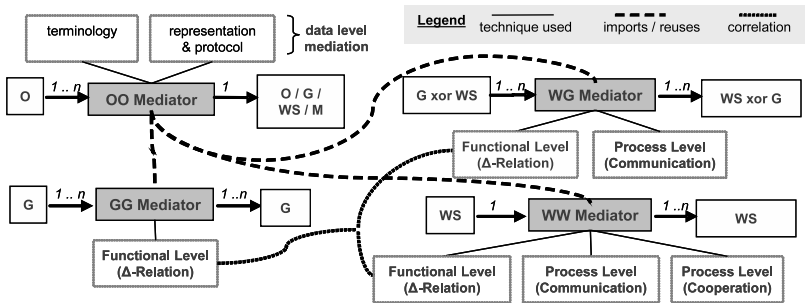


Fig. 9.8. WSMO mediator topology

OO mediators provide the data-level mediation component. The source elements are ontologies or other OO mediators, while the target can be any top-level WSMO element. The only mediation technique used is data-level mediation. OO mediators provide a general mediation component for ontology-based applications; the other mediator types are specific to Web services.

GG mediators connect WSMO goals, i.e. both the source and the target are goals. The mediation techniques used are data mediation via OO mediators and functional level mediation on the basis of Δ -relations that precisely define the logical relationship between the source and target goals.

WG mediators connect Web services and goals when a Web service is not usable for solving a goal a priori. The mediation techniques used are data-level mediation via OO mediators, functional-level mediation to establish the usability of a Web service for resolving a goal if not given a priori, and process-level mediation for resolving potential mismatches on the communication level between the source and target components.

WW mediators connect Web services that interact but are not compatible a priori. Their source and target components are Web services. The related mediation techniques are data-level via use of OO mediators, functional-level mediation for handling functional heterogeneities, and process-level mediation for resolving mismatches between the source and target service with respect to communication and to coordination of interaction. Most commonly, the source component of a WW mediator is a Web service W that aggregates other Web services $W_1, \dots, W_j, \dots, W_n$ in its orchestration, and the target component is one of the aggregated Web services W_j .

In conclusion, the WSMO mediation model realizes Wiederhold's conception and can be considered sufficient for the Semantic Web. It defines architectural components that apply mediation facilities that are appropriate for the types of heterogeneity that can appear between the core elements of Semantic Web service systems. The main merit of this model is that each mediator is *minimal* (i.e., it covers only a minimal aspect of heterogeneity handling), and *modular* (i.e., several mediators are combined for the purpose of specific applications). This enables reuse of mediation facilities and eases their maintenance within dynamic and evolving environments such as the Internet.

9.4 Composition

Composition in the context of Semantic Web services refers to the automated, online combination of existing services to meet some user goal or to the manual or (semi-) automated, offline design of some composite service from existing parts. In each case it is necessary to have some semantic description of the composite service and how it is composed; this description is referred to as an orchestration. An orchestration must necessarily define three aspects: (1) the *participants* in the composition; (2) the *control-flow* governing the order and choice between execution of these participants (3) the *data-flow* governing what is communicated between these participants.

9.4.1 Participants

It is usual within Semantic Web services that a composite service, on execution, will involve the execution of a number of participant services from different providers. Furthermore, in the context of goal-driven execution described in this section, it is possible that participants may be described as goals rather than as concrete, grounded services. Therefore, even in the context of

offline composition, it is possible that the exact combination of services that lead to the overall behavior may be left open to discovery during execution. In WSMO terms, then, a basic orchestration may describe the composition of goals, services or some combination thereof.

For example in the DIP project³ we propose a three-level WSMO compatible approach to orchestration [14]. Orchestration may be represented either in abstract state machines (ASMs) [58], in the Cashew workflow language [101], or in UML activity diagrams [52]. The activity diagram language concentrates on orchestration as composition of services, since it is intended to support an offline composition process which involves discovery, based on UML activity diagrams. The Cashew language concentrates on the hierarchical composition of goals via Workflow Patterns [130]. Abstract state machines in general and in particular the WSMO choreography language are intended to be a neutral, formal representations for orchestrations. They can compose behavior from services and goals; and both activity diagrams and Cashew workflows can be translated into the WSMO language.

In the SUPER project⁴, higher level representations for behavior are considered from the perspective of business processes. In particular, a semantic BPEL language is being constructed, preserving the features of BPEL [4] but being compatible with WSMO and taking advantage of its features, such as goal-driven execution. In BPEL, behavior from a number of partners is composed, with each partner able to play several roles within a workflow, described as WSDL port types. In order to construct a semantic equivalent, the participants in semantic BPEL processes are partners that may have several behaviors described as WSMO goals. A service discovered to fulfil the role of a partner must support all of these goals and thus the notion of participant when viewing such a process as defining an orchestration is extended.

9.4.2 Control Flow

In order to allow control-flow to be represented in a state-based manner, and to allow data-flow to be represented in a semantic manner, WSMO uses the notion of a “state signature” for orchestration. A state signature is a set of ontological concepts and relations over which state is expressed - the state at any point in the execution is understood as the set of instances of all of these concepts and relations.

Abstract state machines are defined over a state signature by allowing tests over the current instances as transition guards, and updates as transition effects. Those updates are specified as logical expressions and can be *adds*, *updates* and *deletes* of instances of concepts and relations. Control flow must therefore be represented as tests over a state. To help to make the control flow more explicit, two particular forms of ASMs are proposed: control state ASMs,

³ <http://dip.semanticweb.org>.

⁴ <http://www.ip-super.org>.

where a variable is used to represent an explicit global state; and multi-agent ASMs, where state is tracked separately for several explicit agents.

Workflow Patterns [130] survey the different types of control flow idiom present in workflow systems and provide a common language. ASMs have a clear semantics, however it has not yet been shown how to represent a suitable subset of these patterns in an explicit or compositional way, therefore higher level languages such as Cashew take the approach of explicitly representing workflows in terms of decomposition via control flow patterns and implicitly defining an ASM. The ASM defined can be thought of as a control state ASM where a process algebraic term defines the values taken by the control state, via an extension of the semantics in [100].

One drawback of the use of ASMs as only representation is the lack of a graphical representation. In the DIP project this is addressed via a language for specification of orchestrations in UML activity diagrams. Again, an ASM can be seen as implicitly defined with multiple control states, in a similar way to multi-agent ASMs, representing the “places” in a token-flow semantics for Activity Diagrams.

The semantic version of BPEL designed in the SUPER project will preserve the two forms of control flow present in BPEL. A graph-like network of control flow dependencies, in the style of WSFL [83] will be directly represented, as will an “algebraic” decompositional form similar to Cashew but with more of BPEL’s business-oriented extensions such as compensable transactions. A WSMO-oriented operational semantics for this language has not yet been defined.

9.4.3 Data Flow

Data flow in WSMO relies on the assignment of modes to the concepts and relations in the state signature, in particular the modes *in*, *out*, *shared* and *controlled*. Concepts and relations of *in*, *out* and *shared* modes represent communications outside the orchestration and are given groundings to define these physical communications.

An in-mode concept (or relation - we leave this choice implicit in the following) will be populated when a communication is received during the execution of the orchestration. Conversely, if the orchestration populates an out-mode concept, a communication will be effected. Shared-mode concepts allow such communication to be carried out via a Triple Space, where the orchestration may read or write to some concept represented externally in such a space.

An orchestration may define communications with its participants via population of, and testing on, the concepts in the state signatures their goals or services define. Within the orchestration extraction of parts from, and aggregation of parts into, messages may be defined via the use of controlled-mode concepts.

It has been proposed [24] that in the spirit of WSMO, where mediation between all dissimilar parts is explicit, dataflow between participants should be defined via mediators. While existing GG, WG and WW mediators are proposed, there remain problems of ambiguity since multiple uses may be made of e.g. a given goal in an orchestration, and entirely different dataflows may be defined between e.g. two goals in different orchestrations. Furthermore, higher-level representations of orchestrations may define dataflow between other constructs. For this reason we are currently developing a new concept to represent the prototypical performance of goals etc. [99]. To each instance a unique identifier can then be assigned, allowing a specific dataflow to be defined via the new concept of a PP mediator, i.e. a mediator between two unambiguous performances of the underlying entity. This is a subject of on-going research.

Dataflow in BPEL is defined via reading and population of variables by service invocations and other copying between variables via assign activities. It has been suggested that the semantic BPEL language being defined could replace syntactically-typed variables with semantic concepts in state signatures, and define dataflow with explicit mediation via PP mediators, but this is a subject on on-going research.

9.5 Grounding and Execution

In the previous sections we have presented the various tasks that arise in a service-oriented architecture and how semantics can increase the degree of automation. However, currently deployed Web services are generally described only on the level of syntax, specifying the structure of the messages that a service can accept or produce. In particular, Web Service Description Language [27] describes a Web service interface as a set of operations, where an operation is only a sequence of messages whose contents are constrained by XML Schema [128]. Such descriptions, introduced in Chapter 4, are referred to in the following as *syntactic descriptions* of Web services.

Certain tasks require that semantic processors have access to the information in the syntactic descriptions. For example, to invoke a discovered service, the client processor needs to know how to serialize the request message. In this section, we present this linking (i.e. grounding) between the semantic and syntactic descriptions. In general, we distinguish between two types of grounding: on the one hand, it has to be known how the content of a message relates to its ontological counterpart (data grounding); on the other hand, the order of possible invocations represented in the choreography interface needs to be mapped to the operations in the syntactic description (behavioral grounding) [76].

Web services can be understood as entities that exchange semantic (ontological) data. Data grounding is the process that must transform the semantic

data into XML messages that will be sent over the network (according to serialization details obtained from the WSDL binding), and also specify how received XML messages are interpreted as semantic data.

A behavioral grounding must handle the multiplicity of operations of a Web service interface in WSDL. WSDL itself does not specify any ordering or dependencies between operations; only the semantic description within a choreography interface tells the client what particular operations can be invoked at a specific point in the interaction.

9.5.1 Location of Grounding

We have described what kind of information must be specified in the grounding; however, we have not yet elaborated on the place of that information. There are three options for placing grounding information: (1) placing the grounding in the semantic description document, (2) embedding the grounding within WSDL documents, and (3) externalizing the grounding to a third document.

Placing the grounding within the semantic description document makes it directly accessible from the semantic data, which follows the order of the tasks in the Semantic Web service framework, i.e. discovery uses the semantic data and then relies on some sort of grounding mechanism in order to invoke the discovered service. On the other hand, placing the grounding directly in the WSDL documents (option 2) can enable the discovery of semantic descriptions in WSDL repositories such as UDDI [10]. This approach is taken by WSDL-S [1], a specification of a small set of simple WSDL extensions that can be used with any Semantic Web service modeling framework. An externalized grounding (outside both WSDL and the semantic descriptions) does not provide easy access to the grounding information from either the semantic or the syntactic side, but it may introduce more flexibility for reuse. However, at the time of publication, externalized grounding is not supported by any specification.

The options listed above are not exclusive, as grounding information can be put redundantly in both the semantic description document and the WSDL document. This could be done by using the native grounding mechanism in WSMO to point to WSDL and at the same time annotate the WSDL document with WSDL-S elements pointing back to WSMO.

9.5.2 Data Grounding

Web services generally communicate with their clients using XML messages described in XML Schema. At the semantic level, however, Web service inputs and outputs are described using ontologies. A semantic client then needs grounding information that describes how the semantic data should be written in XML and then sent to the service. Additionally, the semantic client needs grounding information that describes how XML data coming back from

the service can be interpreted semantically. In other words, the outgoing data must be transformed from an ontological format to XML and, conversely, the incoming data must be transformed from XML to an ontological format. Since the semantics of XML data is only implicit – at best it is only described in plain text – a human designer is required to specify these data transformations so that they can be executed when a semantic client needs to communicate with a syntactic Web service.

In Fig. 9.9 we present a systematic way of distinguishing between the various approaches to data grounding. The figure shows a fragment of the ontology of the semantic description of an example Web service in the upper right corner and the XML data, described in WSDL, in the lower left corner. The three different paths between the XML data quadrant and the semantic quadrant represent different options for where the transformations can be implemented: at the XML level, at the semantic level, or the direct option spanning between the two levels.

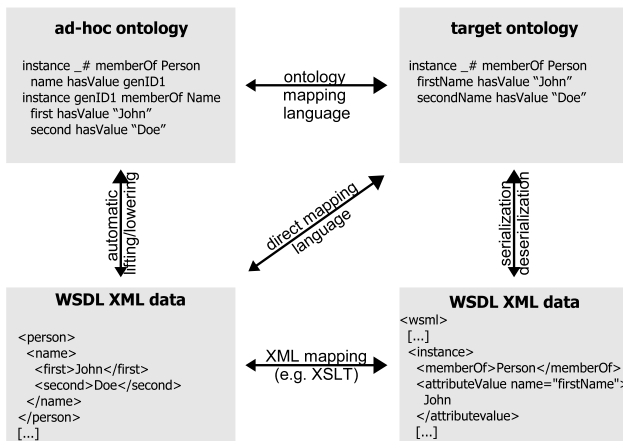


Fig. 9.9. Approaches to data grounding

First, since semantic Web ontologies can be serialized in XML, an XSLT (or similar) transformation can be created between the XML data and the XML serialization of the ontological data. Only in the case where the XML format of the ontological data is actually suitable for a particular Web service can the transformation can be avoided. Second, an ad hoc ontology can be generated from the XML Schema present in the WSDL document (upper left corner), with automatic transformations (*lifting/lowering*) between the XML data and their equivalent in the ad hoc ontology. Then a transformation using an ontology-mapping language can be designed to get to the target ontology. Finally, a direct approach to mapping between the XML data and the target semantic data is also possible.

XML-Level Transformations

In this approach, the grounding needs to link the inputs and outputs of the service to the appropriate XSLT transformations. With these links in place, the actual run-time process is very simple: assuming that a client, for example, wants to send a message to a Web service, it serializes its ontological data into the XML syntax of WSML and then runs the appropriate XSLT stylesheet, sending the result to the service. A message from the service goes through the inverse process. First it is transformed, using a reverse XSLT stylesheet, into XML. The client then parses the result directly into ontological data and processes it accordingly.

This approach is very simple and uses proven existing technologies, but it has a notable disadvantage: the XML representation of ontological data contains an unpredictable mixture of hierarchy and interlinking, as ontological data is not structured according to XML conventions (we say that ontological data is not *native* XML data). Therefore, creating robust XSLT transformations for complex data structures is a considerable task. With simple data, however, this problem is negligible, and since XSLT processors are readily available and many XML-savvy engineers have some XSLT experience, this approach is an ideal initial candidate for data grounding.

Exchanging Ontological Data

The XSLT transformation approach described above makes use of the fact that Semantic Web service ontology languages can be serialized in XML. In certain situations, this XML format could in fact be suitable for the WSDL interface of a Web service when the data will seldom be viewed as actual XML. For example, this is ideal for quick Semantic Web service prototyping of Semantic Web services, or when only semantic clients are expected to use a particular service.

In these scenarios, the WSDL description of the messages will only indicate that they contain WSML/XML, and the semantic description will say (as it does regardless of the grounding approach taken) exactly what data goes in the inputs and outputs. The grounding needs only to link the semantic inputs and outputs with particular WSDL messages, which is done implicitly by all the behavior-grounding approaches that we later describe in Section 9.5.3.

This approach does not require any human designer to create grounding transformations, which indeed saves a significant amount of effort. On the other hand, XML serializations of ontological data are not *native* XML data, and so they may be hard to comprehend or hard to process by XML tools. Consequently, services that use this grounding approach may not integrate well with non Semantic Web services.

XML Schema Ad Hoc Ontology

In this approach, an ad-hoc ontology is automatically generated from the schema of the XML messages according to a conceptual mapping between the XML metamodel and the ontological metamodel. Along with this ad-hoc ontology, a set of mapping rules is also generated for the lifting/lowering transformations between the XML messages and instances of the generated ontology.

If the designer of the semantic description finds this generated ontology suitable for describing a particular Web service, this grounding can be fully automatic. On the other hand, if the generated ad hoc ontology cannot be used, grounding involves an additional transformation between instances of the ad hoc ontology and instances of the target ontology used by the service description. This transformation step uses ontology mediation.

Similarly to the XSLT approach, the ad hoc ontology approach has the benefit of reusing existing transformation technologies (ontology mediation in this case). However, the disadvantage is that the generated ad hoc ontology is not a *native* ontology (it is structured as a restrictive schema for data validation, as opposed to a descriptive ontology for knowledge representation), and this ontology can lack or even misrepresent semantics that are only implied in the XML. This can complicate the task of mediating between the ad hoc ontology and the target ontology describing the service, similarly to the way that *non native* XML data can complicate an XSLT transformation.

Because WSMO puts a heavy focus on mediation, WSMO grounding is currently being developed in the direction of this approach, using WSMO ontology mediation for the transformations.

Direct-Mapping Language

Although we are not aware of any work in this direction in either of the Semantic Web service frameworks, we envision a third option that transforms between the XML data and the ontological data directly, using a specific transformation language.

While a new transformation language would have to be created, it could be optimized for the common transformation patterns between native ontological data and native XML. Manually created mappings would be simpler to understand, create and manage than in the previous approaches. If the disadvantages of the other approaches prove substantial, then development of such a new language should be considered.

9.5.3 Behavior Grounding

The choreography description allows the client to know what messages it can send or receive at any specific point during interaction with a service. Because

WSMO reuses WSDL, the choreography model needs to be tied to the WSDL model of operations.

In WSMO, choreography is modeled as an abstract state machine (Chapter 6), where the states are described with an ontology, and the client is (abstractly) allowed to read and write instances of certain concepts that are marked as “in”, “out”, or “shared”, the latter meaning both “in” and “out”. The state transition rules model the reactions to incoming data and the creation of outgoing data.

With this choreography model, WSMO is not limited to the message exchange view of Web services (see [39] (Triple Spaces) for an alternative). The purpose of grounding is to give the client all the necessary networking details so that it can successfully communicate with the service. When a client follows the WSMO choreography state machine, it knows from the activated transition rules that some particular data can be sent to the service. The grounding now binds this data to a WSDL operation request, and if the operation has a response message, then the client knows that it can expect specific data back.

When creating the grounding, the designer takes into account all the “in”, “out”, and “shared” concepts (since they are manipulated by the transition rules), and correlates “in” and “shared” concepts with input messages, and “out” and “shared” concepts with output messages.

WSMO is based on ontological models of Web services. These formal, logic-based models describe those aspects of Web services that are necessary for automating tasks such as service discovery, negotiation, composition, and invocation. On the other hand, the IT industry deploys Web services with only limited description capabilities, sufficient when Web services are mostly used, composed, and deployed manually. The industry standard for Web service description is WSDL, a language that views Web service interfaces as collections of operations, combined with XML Schema, a validation-oriented language for describing XML data structures.

We have indicated how WSMO grounding information should be specified. Data grounding is the main part of the grounding of Semantic Web services, as there are several different approaches to one complex problem. The current frameworks for Semantic Web services take various approaches: WSMO has initially decided to use an approach that uses an intermediate XML Schema ad hoc ontology. Together with the abstract ontology-mapping language presented above and the existing tool support, this provides the most efficient means in our settings for achieving the task.

Tools

In this chapter, we present the tools that help enable Semantic Web services using WSMO. Although Semantic Web services are fairly recent technologies, there are many software projects specifically dedicated to providing support for WSMO and WSML. This chapter is organized as a survey of existing support tools. We divide our analysis into three parts: first we look at infrastructure components such as parsers, and standard libraries, then we list the available design-time components for creating semantic descriptions, and finally we present the two major execution environments that are based upon WSMO.

10.1 Infrastructure

An essential precondition for any technology to be adopted is an infrastructure in terms of reusable software libraries. This basic software component enables both academia and business to start experimenting with a technology and to exploit it in specific application scenarios.

If one takes for example the XML technology, it is quite easy to get started. First off, there are a wide range of editors available, from commercial offerings for several hundreds of euros to numerous open source projects. Equally important as the editors are the widely accepted libraries for working programmatically with XML. One need no longer code ones own implementations, but can instead choose between existing implementations that already provide a basic functionality for parsing, validation, and manipulation of XML documents. Furthermore, there is support available in the form of higher-level processors (e.g. SAXON¹ for XSLT). The tool support available has significantly lowered the entry barrier for using XML technology.

This model can be applied similarly to the Semantic Web: a language or conceptual model gets adopted as soon as the basic infrastructure is in place.

¹ <http://saxon.sourceforge.net/>.

In the following, we shall present those tools that provide basic services for WSMO and WSML.

10.1.1 The WSMO API

The WSMO API is a Java application programming interface that provides a generic interface for programmatically accessing a WSMO description. The API is an open source project that contains functionality for WSMO ontologies, Web services, goals, and mediators. The WSMO API is bundled with a default implementation called WSMO4J.² WSMO4J implements all the interfaces of the WSMO API. The two together provide a foundation upon which other applications such as editors and reasoners are based. Its functionality it can be compared to that of the Jena library,³ which provides basic services for the ontology languages already standardized by the W3C (i.e. RDFS and OWL).

WSMO4J includes parsers and serializers for the human-readable syntax and the XML syntax of WSML. Validators for the five different variants of the WSML language are also included. These validators go beyond simply validating the syntax of the underlying WSML document and provide detailed information regarding possible misuse of features of WSML. Examples of such misuse would be specifying cardinality constraints in a WSML document when the WSML variant is WSML-Core (which does not support attribute cardinality) or using logical constructs in an axiom that is supported only by more expressive variants.

10.1.2 The WSML Reasoner Framework

Another building block of the basic infrastructure is reasoning support. In the field of artificial intelligence, various forms of reasoning have been investigated for a long time, for example for instance checking, logical entailment, consequence finding, and abductive reasoning. Corresponding algorithms have been investigated for various logics. Taking this general perspective, a traditional database task, namely finding answers to given queries with respect to a database (instance retrieval or query answering), can be seen as a reasoning task as well. It is necessary to distinguish between the various forms of reasoning because they serve significantly different purposes. These forms of reasoning are implemented by means of specific algorithms, which take a well-defined set of input information and are expected to deliver results of a certain kind. The algorithms for different reasoning tasks are based on different techniques and thus differ significantly.

The reasoning tasks that we consider as being relevant in the context of WSML are the following:

² <http://wsmo4j.sourceforge.net>.

³ <http://jena.sourceforge.net/>.

- *Logic-based reasoning tasks*: reasoning tasks that are not specific to ontologies but are usually performed with other kinds of knowledge bases. As the most relevant examples of such reasoning tasks, we consider:
 - instance retrieval;
 - logical entailment;
 - consequence finding.
- *Ontology-specific reasoning tasks*: reasoning tasks that refer specifically to ontologies and their modeling primitives. Typical examples from the description logic community are:
 - concept satisfiability: to check whether a concept is satisfiable;
 - concept subsumption: to check whether a subsumption relationship holds between two concepts;
 - relation subsumption: to check whether two relations are subsumed by each other;
 - classification tasks: for example, to compute a hierarchy of named concepts in a knowledge base (knowledge base classification), or to identify the most specific concepts.
- *Reasoning tasks specific to other top-level elements in WSMO*: this includes tasks related to Web services, goals, and mediators. Examples would be:
 - goal-capability matching;
 - choreography matching.

Many of the reasoning tasks mentioned in the third class are in fact reducible to tasks of the first and second class. Furthermore, note that there are many more reasoning tasks, such as abductive reasoning, that have not been explicitly mentioned. In fact, the current framework focuses only on the reasoning tasks associated with instant retrieval.

From an implementation perspective, various, different reasoners are already available for the tasks outline above; particularly, for the well-understood reasoning task of *instance retrieval* for use with the rule-based WSML variants (WSML-Core, WSML-Flight, and WSML-Rule). More precisely, WSML-Core and WSML-Flight can be mapped to Datalog, i.e. from the function-free Horn subset of first-order logic to Datalog[¬] (Datalog extended by a default negation). WSML-Rule can be mapped to Prolog (Datalog[¬] extended by function symbols). WSML2Reasoner simply integrates different reasoners into a common framework that uses WSML as a concrete syntax. Figure 10.1 illustrates how existing systems are integrated into the framework.

The framework defines a general API for creation and interaction with a WSML reasoner of any kind. WSML ontologies are transformed to a syntactically simpler form and finally to a pure logical representation in terms of Datalog rules. The specific semantics of the object-oriented modeling primitives in WSML is captured by auxiliary rules. An API for wrapping external Datalog engines has been defined and instantiated for a few systems. A specific auxiliary data structure that can be reused for different wrappers is a

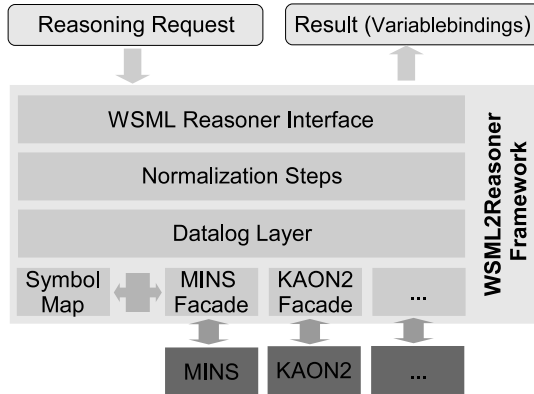


Fig. 10.1. Architecture of the WSML2Reasoner framework

symbol map, which takes care of translating the WSML-specific naming convention, in particular that of IRIs, into symbol names that are actually valid for a specific tool.

The framework is open source and can be downloaded from <http://dev1.deri.at/wsml2reasoner/>. As indicated in Fig. 10.1, MINS⁴ and KAON2⁵ are actively supported reasoners and can also be downloaded from the respective project Web sites.

10.1.3 Online Validation and Reasoning

The latter two online services provide, essentially, a simple interface on top of the components described above. Making these components available using a standard HTML interface addresses the basic needs to begin working with the Web Service Modeling Language. Instead of the need to download and install some software these services are immediately available to the interested user.

The *WSML Validator*⁶ is an online service that allows one to check a WSML specification for validity. For syntactic mistakes, it provides basic hints for fixes and moreover it also checks the specification according to the expressivity level (i.e. the WSML variant used). Additionally, it provides detailed explanations when in case a specification does not adhere to a variant. This can be particularly useful for “debugging” logical expressions.

Another online service provided is the WSML reasoning service⁷. This service supports query answering for the WSML variants Core, Flight, and Rule. Figure 10.2 shows the form fields, with the input ontology and the query in the upper frame. The lower frame shows the query result. In this example,

⁴ <http://dev1.deri.at/mins/>.

⁵ <http://kaon2.semanticweb.org/>.

⁶ <http://tools.deri.org/wsml/validator/>.

⁷ <http://tools.deri.org/wsml/rule-reasoner/>.

a simple ontology about family relationships is used. The query basically *asks* for all women who are known to have a husband. Although it is not explicitly declared that Marge has a husband named Homer, the reasoner can infer this through the *inverseOf* declaration of “hasWife/hasHusband”.

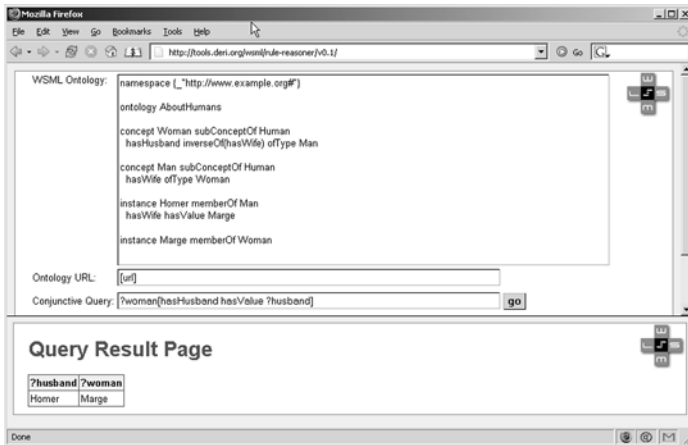


Fig. 10.2. The WSML online reasoning service

In the spirit of a service-oriented architecture both services are not only accessible via a HTML interface, but also using Web service technology⁸.

10.2 Design Tools

The creation and editing of precise formal descriptions is a cumbersome task. As the number of specifications increases, tool support is essential. The framework provided by design tools is dedicated to easing this task. All design tools at the time of writing of this book are based on the Eclipse platform. This has a number of advantages; the most striking one is that the platform allows one to combine any individual tool in a way that best suits a particular task. We first introduce the Eclipse platform and then elaborate on the various WSMO design tools.

10.2.1 The Eclipse Platform

The Eclipse⁹ framework is an open source development led by IBM. It is a framework for deploying graphical tools using the Java programming language. A common misconception is that Eclipse is a Java editing environment,

⁸ The respective WSDL definitions can be found at <http://tools.deri.org/services>.

⁹ <http://www.eclipse.org>.

when in fact it is merely the case that the Eclipse framework is used to deploy the Java Development Environment (JDE). Eclipse can be used to create complex functionalities from simple GUI plug-ins. It allows many different tool developers to create discrete editors for documents and to create views of the content contained within those documents. These editors and views can then be drawn together into a perspective, which gives the collection of plug-ins a context.

Editors, views and perspectives can be created and integrated with Eclipse quite simply, by implementing a number of extension points. In general, most Eclipse plug-ins are shipped alone for individual users to deploy in the Eclipse instance already installed on their machine. It is also possible to create a completely branded application containing only the desired plug-ins which can be shipped to users for installation. This flexibility is leading to many commercial and research tools being developed for the Eclipse framework and is resulting in a huge increase in the popularity of Eclipse.

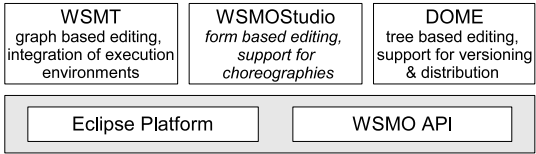


Fig. 10.3. The Eclipse platform and the WSMO tools developed on it

Figure 10.3 shows the various tool sets developed on top of the Eclipse platform. All of the tools are based on two foundational components: the plug-in framework of the Eclipse platform and the data model provided by the WSMO API. Together, they provide the possibility to combine the functionality of different tools using features of Eclipse such as a “Perspective”, which configures multiple views (provided by different sources) into one common application. The main WSMO tools are depicted by the three boxes on top of the base layer. The Web Service Modeling Toolkit (WSMT) focuses on providing a graphically based editing environment with special support for connecting to execution environments; WSMO Studio is a form-based editing environment that allows the manipulation of the various WSMO elements; and DOMÉ (Distributed Ontology Management Environment) provides a tree-based WSMO editing approach.

10.2.2 WSMT

The Web Service Modeling Toolkit is an open source project¹⁰ that is mainly driven by members of the Digital Enterprise Research Institute (DERI). The

¹⁰ <http://wsmt.sourceforge.net>.

toolkit combines a number of separate editors and views. Its main focus is on graphical visualization and support for modeling, executing and monitoring Web services.

10.2.3 Graphical Visualization and Editing

Figure 10.4 shows a screenshot of WSMT. This nicely shows how the Eclipse platform is able to combine several different views in a common perspective. The WSMML workspace, shown in the upper right corner, can be used to organize the WSMML files (in the picture titled “Navigator”). On the right-hand side, the topmost plug-in allows the user to modify WSMML descriptions using a graphical view of the WSMML elements. The ontology about family relationships used earlier to illustrate the Web-based reasoner is used here again. The graph-based view nicely illustrates various relationships such as class hierarchy (Human>Woman), instantiations (Woman>Marge), and inverse relations (hasHusband>inverseOf(hasWife)). The Visualizer is able to focus on a specific entity or to view the ontology graph as a whole. Other aspects and viewing options can be filtered depending on the user’s needs (e.g. hiding all instances). At the same time, editing the source code of the ontology is also possible (right middle window). This plug-in provides well-known features of advanced development environments, such as syntax highlighting, code folding and syntax completion. On the bottom right-hand side, an additional “Problems view” provides information about the status of the files in the project, obtained from the validation functionality. This allows an ontology engineer to quickly spot problems, such as not specifying the WSMML variant, as is shown in Fig. 10.4.

10.2.4 Integrated Reasoning Support

The functionality of the WSMML2Reasoner framework has been made available within the Eclipse environment via a specific extension point. A special view provides an interface that allows the user to execute queries over the currently open ontology. The results of the query are displayed as a list of variable bindings similar to that already presented for the online reasoner (see Fig. 10.2). However, the advantage over a simplistic HTML interface is that the results can be selected, and the view of the ontology is automatically focused on the corresponding ontology element. For example, selecting a row from the results that denotes a concept will focus WSMOViz (the WSMO Visualizer component) on the corresponding concept.

10.2.5 Data Mediation

The concepts of data mediation introduced in Section 9.3 are implemented in WSMT. The graphical editor of WSMT uses a semiautomatic approach

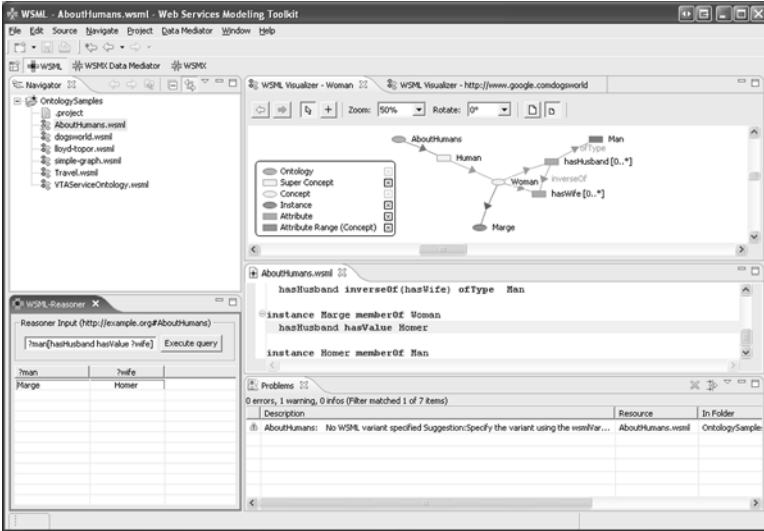


Fig. 10.4. The Web Service Modeling Toolkit

in which a component first suggests a mapping which the user must later refine. Once these mappings have been created, they can be used to transform instances of the source ontology into instances of the target ontology. The WSMT data mediation plug-in for Eclipse is the design-time component that is used to create these mappings.

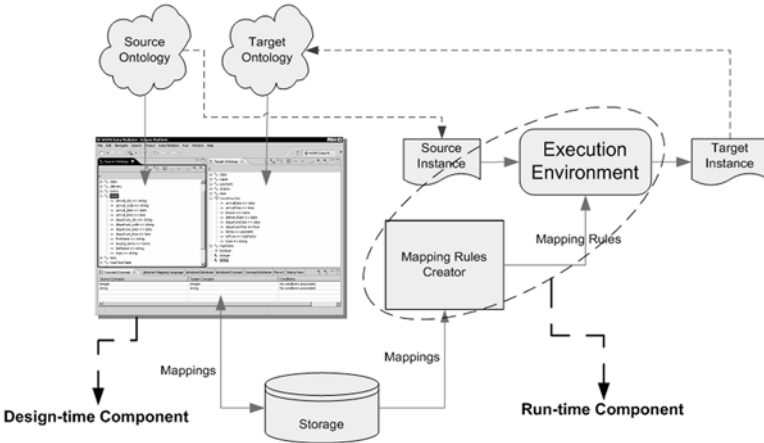


Fig. 10.5. Creating mappings in the data mediation plug-in

Figure 10.5 illustrates the two phases of data mediation: the design-time component (depicted on the left) and the run-time component (depicted on the

right). The design-time component displays two ontologies (source and target) and provides, in the bottom pane, means to specify mappings between selected entities. These mappings are stored in an execution environment, where they can be used on demand if an actual transformation between heterogeneous ontologies is needed.

10.2.6 WSMO Studio

WSMO Studio (Fig. 10.6) is a collection of tools for editing WSMO descriptions developed for the Eclipse framework by OntoText Labs¹¹, part of Sirma. WSMO Studio displays the entities contained in the WSMO document as a tree hierarchy and allows the user to edit these entities via a collection of forms. The approach taken in WSMO Studio is very similar to that taken in Protégé¹² (an ontology-editing suite developed by Stanford University); however, instead of providing a collection of tabs that can be used to edit different objects in the ontology, a unique editor is available for each object type.

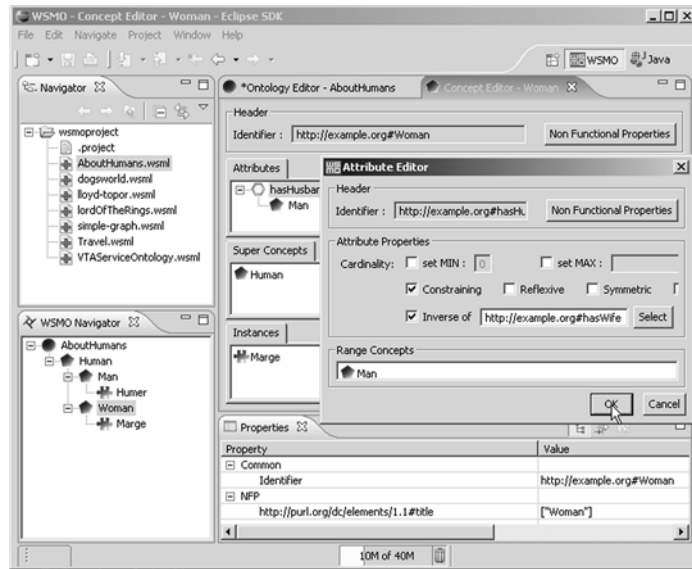


Fig. 10.6. Concept editor in WSMO Studio

Editing a given entity in a WSMO document is done by double-clicking on the entity in the tree hierarchy. When the user does this, an editor is opened specifically for this entity. The GUI widgets contained within the editor are built dynamically upon the functionality supported by the given

¹¹ <http://www.ontotext.com>.

¹² <http://protege.stanford.edu>.

WSMO entity. WSMO Studio supports ontologies, Web services, goals, and mediators and allows the user to manage all subtypes of these entities.

WSMO Studio also offers a syntax-highlighting text editor for the WSML human-readable syntax. This text editor allows more advanced users who are more comfortable editing the WSML syntax to gain some advantage from the tool set. Keywords in the WSML syntax; for example, “concept”, “instance”, “subConceptOf”, etc. are highlighted in a color of the user’s choice. Furthermore WSMO Studio provides access to a set of import and export functions from and to WSML, a subset of OWL-DL, and an XML representation of WSML. The most recent release (at the time of writing) includes a choreography designer, as well as front ends for ontology/service/goal repositories, including an integrated repository based on a triple store and an IRS-III adapter (an execution environment describe in section 10.3.2).

10.2.7 DOME

The DERI Ontology Management Environment (DOME)¹³ (Fig. 10.7) is a tool suite for managing WSML documents developed by the Ontology Management Working Group.¹⁴ DOME can be used for editing and browsing, mapping and merging, and controlling the versioning and evolution of ontologies. Ontologies are displayed using a tree hierarchy, and all editing is done via right-click menus on this tree hierarchy.

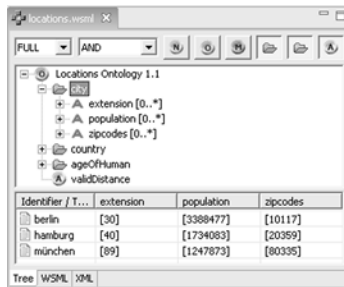


Fig. 10.7. DERI Ontology Management Environment (DOME)

The editor which contains the tree hierarchy also contains a second tab for editing the underlying human-readable WSML syntax of the visible ontology. Any changes made in the text editor are immediately visible in the tree hierarchy and vice versa. The interaction paradigm of this tool is based mainly on the “drag-and-drop” methodology.

Other views provided allow the instances of a selected concept to be seen and the nonfunctional properties of a given entity to be changed. The instance

¹³ <http://dome.sourceforge.net>.

¹⁴ <http://www.omwg.org>.

viewer allows one to see the instances of a concept without browsing through the tree hierarchy.

When two or more ontologies are opened in DOME, it is possible to create mappings between these ontologies. These mappings can later be used for query rewriting and ontology merging to resolve heterogeneity issues, i.e. in cases where two business partners use different ontologies to describe the same domain, these mappings can be used to allow the two partners to communicate.

10.3 Execution Environments

The execution environments of Semantic Web services are complete frameworks which serve as an architecture to integrate the tasks involved in the use of Semantic Web services from goal specification to execution. There exist two WSMO-compliant implementations which follow the conceptual model of WSMO, to be discussed here.

10.3.1 The Web Service Execution Environment (WSMX)

The amount of human interaction required in the process of finding and using Web services is a serious limitation on their use. Since WSMO and its WSML formalization allow Web services to be described semantically, a large portion of this process can be automated. The Web Service Execution Environment (WSMX)¹⁵ is a reference implementation of WSMO and is an execution environment within which semantic descriptions of users' goal and providers' Web service can be used to *discover*, *compose*, *mediate*, *select*, and *invoke* Web services that match the end user's requirements.

The key aim of WSMX is to define mandatory functional services, decouple their functionalities from one another, and describe components' interfaces in terms of the offered functionality. One of the design principles of service-oriented architectures (and Web services in general) is the separation of the implementation of the service from its externally visible description. Such design allows the replacement of one service with another as long as the new service supports a stable interface. Similarly, WSMX does not define services in terms of their design or implementation details, but only in terms of expected functionality and their externally visible behavior.

Components

WSMX has a component-based architecture (Fig. 10.8), with each component in the architecture performing a discrete piece of functionality. These units of functionality are drawn together by the formally described execution semantics of the system.

¹⁵ <http://www.wsmx.org>.

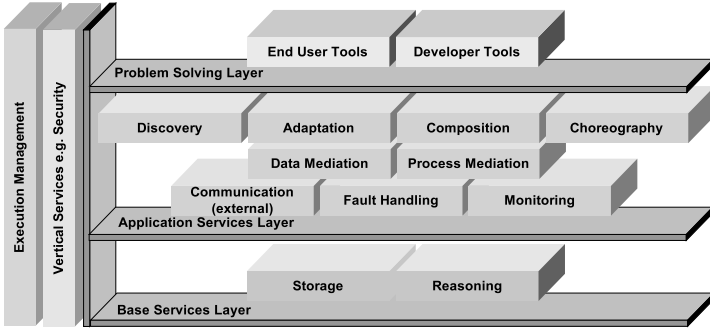


Fig. 10.8. WSMX components

Service-oriented architectures typically consist of a set of services and a coordinator that combines the services and puts them to use. Talking about SOAs in the context of WSMX can sometimes be misleading since WSMX is an SOA and at the same time it is the coordinator of another, larger SOA. The WSMX differentiates between two types of services: application services (such as discovery, choreography, and data and process mediation) and the base service layer (e.g. storage). Application services are mandatory for enabling the infrastructure to deliver its functionality as defined by its execution semantics. User services are exposed by information systems external to the WSMX infrastructure, but they are still coordinated using WSMX platform services. The WSMX recommendation defines the scope of particular platform services in terms of their functionality, whereas it remains silent about the scope and functionality of user services. Some of the components of WSMX are the following:

- *Discovery.* The discovery component of WSMX is responsible for finding Semantic Web service descriptions given a user’s semantically described goal. This component operates over all of the Web service descriptions stored in WSMX’s resource manager.
- *Adoption.* It is not feasible to presume that a Web service will be described in an exhaustive manner. Considering the Amazon book service, for instance, it would not be possible for Amazon to update the semantic description every time a new book was made available or a price changed. The adoption component of WSMX is responsible for communicating with each of the discovered services, obtaining the additional information required on the basis of the interfaces provided by the service, and selecting which of the discovered services is most suited to the user’s requirements. The user’s goal can contain preference information that can be used by the selection component to select between the various services on the basis of their nonfunctional properties. These nonfunctional properties include availability, reliability, scalability, and financial considerations.

- *Composition.* Part of a service interface describes how the service makes use of other services to achieve its functionality. Composition and choreography are related concepts but distinct from each other. The composition component is responsible for making use of other services in order to achieve its functionality. It is the WSMX implementation of the strategies discussed in Section 9.4.
- *Choreography.* The choreography part of a service interface describes the behavior of the service from a client's point of view. Any user of a Web service, automated or otherwise, is a client of that service. The choreography engine is responsible for using the choreography descriptions of both the service requester and the provider to drive the conversation between them. It is the responsibility of the choreography engine to maintain the state of a conversation and to take the correct action when that state is updated. For example, updating of the state of a choreography instance may occur as the result of a message received from a service provider. The consequent action, as described in the choreography instance, could be to forward the message unchanged to the service requester. The choreography engine in the WSMX architecture has three main responsibilities: (1) evaluating the transition rules defined in the choreography interface, (2) determining the legal instances for the last choreography step, and (3) managing invocation requests to and from the communication manager.
- *Data mediation.* The data mediation component of WSMX is responsible for performing instance transformation when the ontologies used by the goal are different from those used by the Web service. Data mediation may occur at many different points in the execution semantics. Data mediation in WSMX is a semiautomatic process, in which a domain expert must first create the mappings between the two ontologies. This manual step occurs in an offline mode. Once these mappings have been created and registered with WSMX, the data mediation component can automatically mediate between these two ontologies in real time.
- *Process mediation.* The process mediation component is used to resolve heterogeneity issues between the public processes of the requester and the provider. For example, the requester may expect the user name and password to be sent together, whereas the provider may expect them separately. The WSMX process mediation component provides an automatic mechanism for resolving these process mismatches.

This is only a subset of the components present in the WSMX architecture. Other, more low-level components include a resource manager for storing and retrieving data; a communication manager for receiving requests from the user and communicating with Web services; and the WSMX core, which is responsible for creating and tracking the execution semantics of each job in the system.

Developer and end user tools are on the periphery of WSMX. Various developer tools have already been presented that aid the design-time process

underlying the architecture. End user tools represent those tools that draw together application services according to a particular problem domain. A sketch of these tools will be given in Chapter 11. Next, we shall discuss a particularity of WSMX, which is not obvious from this sketch of the components. For the components to interact properly, a “glue” is needed that regulates the interplay of them: execution management.

Execution Management

Execution management has the role of coordinating the execution of the application services in a meaningful scenario. Such scenarios are called execution semantics, and they can be seen as low-level projections of the requirements of the application layer. The execution semantics define the way the application services have to work together to complete a useful scenario. Having the execution semantics as a separate concept in the architecture is the first step towards a highly decoupled system. That is, given a set of application services, when new execution semantics to be executed are defined, the whole architecture basically changes the way it offers a certain functionality, or even the functionality itself. As such, WSMX is a packaged environment capable of completing a “round trip” starting from accepting a user’s goal, and then performing discovery and selection, finding the most suitable Web service, mediating when necessary, invoking the service, and returning the results. But also, by using different execution semantics, WSMX can simply discover and return the services found to the user, allowing the requester itself to perform selection.

In WSMX, much of the process of finding and using Web services can be automated. Automatic discovery offers a great advancement over the current situation, allowing users to find services that can fulfill their requirements without spending large amounts of resources. Mediation and invocation also reduce the effort of manually interpreting interfaces, inputs, and outputs of Web services and switching data formats to fit the formats required by those Web services.

10.3.2 The Internet Reasoning Service IRS-III

IRS-III [33] is a framework and an implemented platform that acts as a broker mediating between the goals of a user or client and the available deployed Web services. Thus IRS is not a framework on its own, but uses WSMO as its ontology and follows the WSMO design principles. The conceptual model of IRS-III is based on five design principles:

- *Support capability based invocation.* IRS-III enables clients (human users or application programs) to invoke a Web service by simply specifying a concrete desired capability. IRS acts as a broker that finds, composes, and invokes appropriate Web services in order to fulfill the request.

- *Ease of use.* IRS interfaces are designed such that much of the complexity surrounding the creation of Semantic-Web-service-based applications is hidden. One-click-publishing is a corollary of the above design principle.
- *Agnostic to service implementation platform.* Within the design of IRS, there is no strong assumption about the underlying service implementation platform.
- *Connected to the external environment.* To support this, functions and relations can be defined in order to make calls to external systems, for example, invoking a Web service.
- *Inspectability.* In many parts of the life cycle of a software system, it is important that the developers are able to understand the design and behavior of the software being constructed. This principle is concerned with making the semantic descriptions accessible in a human-readable format. The key is that the content and form are easily understandable by the builders of Semantic Web service applications.

Many of the direct principles of IRS are application-focused, but from the first design principles, we can say that they largely follow the WSMO approach described in this book. The IRS-III ontology is based on the WSMO conceptual model; however, it has a number of differences. To achieve the goal of capability-based invocation, Web services are required to have input and output roles, and goals linked to Web services via mediators. Web services are linked to goals inherit the input and output roles of the goals. In WSMO, the mediation service slot of a mediator may point to a goal that declaratively describes the mapping. Goals in the context of a mediation service play a slightly different role in IRS-III. Rather than describing a mapping, goals are considered to have associated Web services and are therefore simply invoked.

IRS clients are assumed to be able to formulate their request as a goal instance. This means that it is only choreographies between IRS and the deployed Web services that are required. Choreography execution in IRS-III thus occurs from a client's perspective. That is to say, to carry out a Web service invocation, IRS executes a Web service client choreography, which sends the appropriate messages to the Web service deployed. In contrast, the current WSMO choreography describes all of the possible interactions that a Web service can have.

At the heart of the server of IRS-III is the WSMO library, where the WSMO definitions are stored using the representation language OCML [96]. The library is structured into knowledge models of WSMO goals, Web services, and mediators. The structures of each knowledge model are similar, but typically the applications consist of mediator models importing from relevant goal and Web service models. Following our design principle of inspectability, all information relevant to a Web service is stored explicitly within the library. Within WSMO, a Web service is associated with an interface that contains a separate orchestration and choreography. Orchestration specifies the control and dataflow of a Web service, which invokes other Web services (a composite

Web service). Choreography specifies how to communicate with a Web service. The choreography component communicates with an invocation module able to generate the required messages in SOAP format. A mediation handler provides functionality to interpret WSMO mediator descriptions, including running data mediation rules, invoking mediation services, and connecting multiple mediators together. Following from the principle of openness mentioned above the orchestration, choreography, and mediation components are themselves Semantic Web services.

10.4 Summary

In this chapter, we have presented the most relevant tools for enabling Semantic Web services using WSMO. Despite the fact that the first implementation efforts only started a little over a year ago, their intense interest has led to a diversified landscape of tools. While the infrastructure components can be regarded as quite stable higher-level components such as the mediation engine of WSMX are still experiencing a period of great evolution. Their use in various industrial and research projects is rapidly increasing their quality.

Furthermore, it should be noticed that most developments are being driven as community efforts under open source licenses, which allows early adopters to reuse and extend existing solutions. Given the dynamics of the subject matter the interested reader is encouraged to look at <http://tools.deri.org/wsm1>, which provides up-to-date information on the state of the available tools.

Applications of WSMO

A new technology can only be justified by successful application. Although WSMO is a relative new approach, there are already a few realized application scenarios. This chapter depicts some existing applications in the areas of e-commerce, e-government, and e-banking that have been developed mainly in joint academic and industrial efforts. We discuss the technical challenges in each area, and outline how semantically enabled Web service technology allows advanced support for these.

11.1 E-Commerce

E-commerce is a collective term for IT-based support for trading and bargaining between companies and end-consumers, as well as between enterprises. The former is commonly referred to as business-to-customer (B2C), and the latter as business-to-business (B2B) commerce. In both areas, technical support is evidently important, in order to ease processing of orders, provide sophisticated customer support, or manage industrial supply chains [114]. Innumerable technical solutions for both B2C and B2B commerce exist that have become an irreplaceable asset of today's business world – from simple automated cash machines offered by nearly every financial institution or electronic billing systems for accounting support, to automated just-in-time supply chain management systems in the automotive manufacturing industry.

The Internet provides a novel infrastructure for e-commerce that is worldwide accessible and easy to handle. Its growth and proliferation during the last decade has led to a tremendous development of new, Web-based e-commerce technologies. Nowadays, hardly any enterprise is without a professionally managed homepage that provides general information and, very often, also customer support. The dot-com hype in the late 1990s has given us numerous applications for B2C commerce, development suites for Web-based shopping and auction systems, and even novel business models for commerce on the

Web. Three thriving examples are Amazon (www.amazon.com) which provides Web-based distribution of consumer goods, Ebay (www.ebay.com) an auction system on the Internet, and travel-booking services such as Expedia (www.expedia.com). These are widely used B2C commerce solutions with remarkable market appraisal, last but not least due to the price advantages offered [78].

However, current Web-based commerce solutions suffer from the deficiencies of current Web technologies that have been outlined in Part I of this book. That is, automated processing of Web-content is awkward owing to unstructured information representation and non semantically defined terminologies. Because of this, existing e-commerce solutions deal with proprietary data structures and make tremendous efforts to integrate information from Web-applications with existing back-end systems for inventory or customer management. Although automated information interchange over the Internet is possible via SOAP and XML, the usage of Web service technologies is restricted to certain application scenarios with well-defined data and communication structures. In order to illustrate how Semantic Web service technologies, and especially WSMO, can be applied beneficially to overcome these deficiencies, we shall now present the modeling of the Amazon e-commerce service in WSMO in order to allow automated usage for B2C commerce, and then present a WSMO-based solution for application integration within the telecommunication sector as an example of a B2B commerce scenario.

11.1.1 B2C – The Amazon E-Commerce Web Service

As its contributing to the development of innovative e-commerce technologies based on Web services, Amazon provides a set of Web services that allows direct access to the Amazon technology platform (Amazon Web Services, see: <http://www.amazon.com>). Among the Web services, the Amazon e-commerce service (ECS) provides access to Amazon’s own e-commerce functionalities for use as a building block for e-commerce applications.

Overview of the ECS Web Service

ECS exposes Amazon’s product data and e-commerce functionality to help developers, web site owners, and merchants to leverage the data and functionality that Amazon uses to run its own e-commerce business. It offers the following functionalities: searching the Amazon catalog, looking up data for specific products, getting customer feedback for non-Amazon vendors, setting up shopping carts, and looking up customers’ wish lists and registries.

Through ECS, developers can access and deal with the following information: product data, including information about product availability and pricing for items in the Amazon catalog; content from customers such as reviews and wish lists, and “seller information, for instance general information and customer feedback about the wide range of vendors on the Amazon site. In

addition, developers can use ECS for managing shopping carts of products for purchases made through Amazon. This allows the receiving of commissions on sales that originate with another Web site or application. By use of this functionality, the technical handling of purchases is then completely transferred to Amazon (although purchasing directly through ECS is not supported). ECS is a read-only system, and so product data cannot be sent back to Amazon via ECS. The operations offered by ECS can be categorized as enquiries about operations for product data, manipulation of shopping cart, enquiries about customer content and seller information, and auxiliary operations.

The ECS Web service is offered as a WSDL description, i.e. the operations required for consumption with the datatypes expected, along with the technical endpoint, accompanied by a Web page with additional explanations in order to enable developers to utilize ECS correctly. Although this allows the ECS functionalities to be utilized and integrated within other B2C e-commerce applications, developers need to manually study the explanations on the Web page and inspect the WSDL description in detail. However, having a semantic description of the Web service would enable automated usage by the technologies presented in the preceding sections. Hence, the following depicts the semantic description in WSMO.

WSMO Description of the ECS Web Service

The semantic annotation of the WSMO description of the Amazon ECS Web service was achieved by analyzing its WSDL and XML Schema definitions and lifting them to an ontological description. We refer to [77] for the complete description, along with an explanation of how this was done. Because of space limitations, we shall not display the domain ontology, and restrict our examinations to the facilities for product search; the Amazon technology is very mature in these respects and hence there might be the most interesting aspects for incorporation into other B2C e-commerce solutions.

First, we study the functional description of ECS, modeled by a capability in WSMO, as shown in Listing 11.1. This defines the preconditions of the ECS Web service, capturing all its inputs. Here, however, we display only those related to product search. ECS can accept any of the inputs at any given point in time, so that the overall precondition is defined as a disjunction of all the possible inputs. The postconditions of a WSMO capability define the conditions that hold after the Web service has been executed. For the Amazon ECS – as well as for Web services in general – the postcondition depends on the precondition: a specific response can be provided only if the corresponding request has been provided as an input to the service. In consequence, the postconditions are defined as implication rules: if the request provided as input is of a specific type that satisfies the precondition, then the corresponding response will be given. The overall ECS postcondition is denoted by a conjunction of the postconditions for distinct requests. Because of space limitations, we display only the postconditions for search by author and keywords. The

only shared variable in the capability definition is `?request`. This is required in order to capture which postcondition will be achieved when a specific input is provided to the service.

```

capability amazonWSCapability
  sharedVariables {?request}
  precondition
    definedBy
      //A search by author, keywords, title , minPrice or maxPrice
      ?request memberOf am#itemSearchRequest or
      [...] .

  postcondition
    definedBy
      // The result for a search request by author is a container of products
      // such that all items in the container are amazonBooks have the requested author:
      (?request [
        author hasValue ?author
      ] memberOf am#searchBooks implies
      exists ?container
      ( ?container memberOf am#itemContainer and
        forall ?item
          ( ?container [
            items hasValue ?item
          ] memberOf am#itemContainer implies
            ?item [
              author hasValue ?author
            ] memberOf bk#amazonBook
          )
        ) ) and
      // The result for a search request by keyword is a container of products
      // such that all items in the container are amazonBooks have the requested keyword:
      (?request [
        keywords hasValue ?keyword
      ] memberOf am#searchBooks implies
      exists ?container
      ( ?container memberOf am#itemContainer and
        forall ?item
          ( ?container [
            items hasValue ?item
          ] implies
            ?item [
              keywords hasValue ?keyword
            ] memberOf bk#amazonBook
          )
        ) ) and [...] .

```

Listing 11.1. Capability of the Amazon e-commerce service

The second important part of the WSMO description of the Amazon ECS that we present here is the choreography interface. This defines the behavioral interface for consumption of the service by a client. In other words, it defines the communication structure supported by the Web service for consuming its functionality. As explained above, WSMO choreography interface definitions consist of a state signature that defines the ontological terminology and its usage for communication, and of guarded transitions that specify the dynamic aspects of the service's communication behavior. Listing 11.2 shows a selection from the ECS choreography interface. This corresponds to the product search functionalities that the above listing specified the postcondition for.

```

interface amazonWSInterface
  choreography
    stateSignature
      importsOntology _ "http://www.wsmo.org/ontologies/amazon/amazonOntology"
      in
        concept am#itemSearchRequest withGrounding
          _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
            wsdl.interfaceMessageReference (AWSECommerceServicePortType/ItemSearch/In)
          "
      out
        concept am#itemContainer withGrounding
          _ "http://webservices.amazon.com/AWSECommerceService/2005-03-23#
            wsdl.interfaceMessageReference (AWSECommerceServicePortType/ItemSearch/
              Out)"
      transitionRules
        // a search request for an item will trigger a response
        if (?ItemSearchRequest memberOf am#itemSearchRequest) then
          add(_# memberOf am#itemSearchResponse)
        endif

```

Listing 11.2. Choreography interface for Amazon e-commerce service

A developer who is considering utilizing the Amazon ECS Web service within his own e-commerce application initially obtains two aspects from this WSMO description. First, a precise description of what the Web service does (the information about the capability is not provided in WSDL, but merely in natural language in the accompanying explanations on the Web page), and secondly a concise description of how to consume the functionalities of ECS described by the choreography interface. If the developer can provide an appropriate request, then automatic use of ECS instead of manual integration is possible. Furthermore, if the developer chooses to define a WSMO goal for the e-commerce functionalities, then he/she is not limited to the Amazon ECS but can also use other e-commerce Web services that provide similar functionalities.

11.1.2 B2B – Enterprise Application Integration

B2B commerce is concerned with enabling seamless automated information interchange between enterprises. A significant problem for automated information interchange and processing within and between enterprises is the use of different terminologies, representation formats, and transfer protocols as well as potentially incompatible business processes, by entities that ought to interact. Starting from the Electronic Data Interface (EDI) and processing through more recent developments such as RosettaNet and ebXML, more than 200 standards, along with technical platforms, have been developed for specific purposes in B2B commerce. These developments must be integrated in order to facilitate seamless B2B commerce. This is the aim and challenge of enterprise application integration (EAI) technologies, where € 9 out of every € 10 that is invested in IT technologies are spent [22].

Web services have been invented as a novel EAI technology for tackling the integration problem. Actually, the initial motivation for Web services was

to develop a next-generation EAI technology by providing seamless access interfaces to computational facilities. Along with means for ad-hoc usage and combination and support for web-based information exchange, they shall serve as the basis for service-oriented architectures over the Web. This means that enterprises publish the access points and public processes of their functionalities as Web services, which are then invocable over the Web, possibly as composed or complex functionalities. However, as discussed in detail earlier, the initial Web Service Technology Stack allows the setting up of Web services for Web-based access to a computational facility with a standardized usage interface (i.e. the WSDL description). However, this does not address the problem of handling heterogeneities that hamper information interchange and automated interaction within and between businesses.

In order to illustrate how Semantic Web services and, especially, the WSMO framework and technologies can be used to provide adequate solutions to this problem, we shall describe an EAI problem in the telecommunications industry as presented in [34]. To stay competitive in the future market, many Internet service providers (ISPs) are extending their businesses by the wholesaling of mobile and landline telephone services and unbundled data lines. Least Cost Supply Management (LCSM) is a methodology that offers the possibility to select the best supplier in terms of price and quality for specific product bundles; therefore technologies for easy and flexible integration of suppliers that offer these services are needed.

Let us consider a virtual Internet service providers (VISP) that provides ISP bundles to customers on the basis of LCSM by detecting and aggregating offers from other providers. As in the real world, the parties involved might use different, incompatible systems. Therefore, a WSMO-enabled *VISP platform* mediates communication between the technical systems of the VISP, the suppliers, and the internal systems of the ISPs. Figure 11.1 gives an overview of the scenario for a customer requesting a broadband Internet package in the form of a flow-chart with the following central processing steps:¹

1. A customer buys a broadband Internet package from the VISP.
2. This package is recognized as a VISP product by the VISP enterprise resource planing (ERP) system.

¹ Some names and abbreviations used in Fig. 11.1 are the following:

- *Apache*. An open source Web server, see: www.apache.org.
- *SAP*. An enterprise resource planing (ERP) system provided by SAP, see: <http://www.sap.com/solutions>.
- *CCBS*. abbreviation for “Call Completion on Busy Subscriber”, a telecommunication management technology.
- *Biztalk*. an application integration platform provided by Microsoft, see: www.biztalk.org.
- *Websphere*. an application server provided by IBM, see www.ibm.com/software/websphere.

3. The VISP sends an order to its ISP (e.g. an ebXML document).
4. An adapter transforms this message into a WSMO goal.
5. The VISP platform selects an appropriate supplier using the LCSM methodology and invokes the respective Web services.
6. After successful account installation, the ISP creates an account for the customer along with ordering information.
7. The ISP monitors the whole process; when the order process is finished the VISP platform sends a confirmation back to the VISP.
8. The VISP ERP system processes the order.
9. The VISP sends a confirmation to the customer.

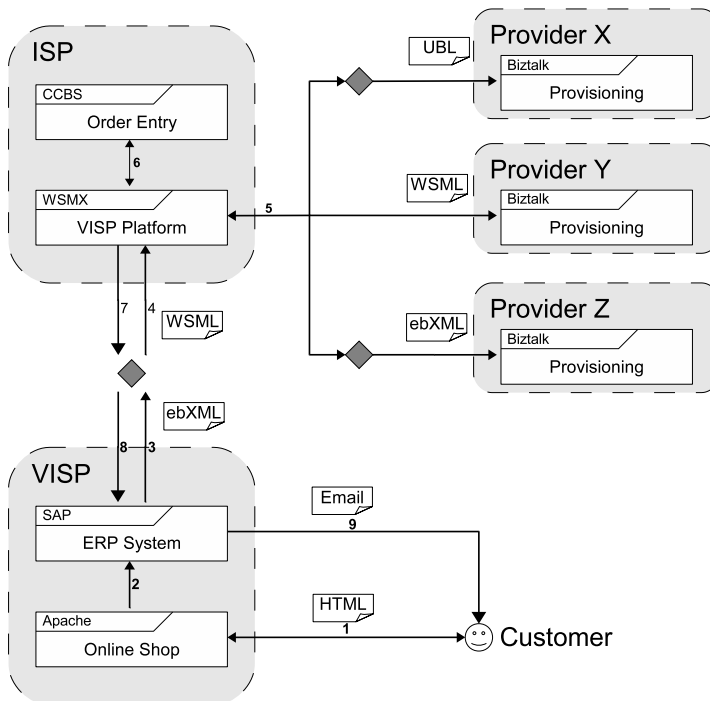


Fig. 11.1. Overview of a VISP

All of the parties involved in this scenario are, most likely, in different locations, and therefore all information interchange and communication takes place over the Web. The Web services of the ISP and the suppliers carry WSMO descriptions similar to that of the Amazon ECS Web services discussed earlier. The parties involved may very well use different ontologies within their service descriptions or might have incompatible business processes. However, on the basis of semantic descriptions, the VISP platform is

able to resolve a request as in the scenario outlined by applying semantically enabled technologies for Web service usage.

For the purpose of this example, we shall omit the provision of the WSMO element definitions but shall discuss how the VISIP platform achieves its functionality by applying the techniques for discovery, composition, mediation, and execution described in detail in Chapter 9. For discovery, the ISP aims to detect the cheapest offer among all suppliers that satisfies the customer's request. Therefore, it creates a WSMO goal containing an axiom stating that the product to be delivered must have the lowest price among the applicable products. This allows to run a discovery engine that finds the Web service of the supplier that offers the requested product at the cheapest rate. Moreover, the choreography interface descriptions, along with a grounding to an executable Web service communication technology, allow the ISP to automatically invoke and execute the suppliers' Web services.

If no supplier can provide the requested product as a whole but it can be provided as an aggregate by combining the offers from several suppliers, the VISIP platform can compose the respective Web services and provide the result as a new, virtually created product bundle to the customer. Imagine that providing such a composed product bundle requires the Web services of provider X and provider Y. However, several types of heterogeneity that hamper the necessary interaction of the Web services might occur: X might use UBL² for describing its Web services, while Y might use WSML; X might define the customer name as a string, while Y might define it as consisting of a given name, middle initial, and surname. Furthermore, it might occur that the composition of X and Y defines an interleaved process, meaning that X requires some input from the ISP that can only be given after the ISP has received a certain output from Y. Therefore, a WW mediator \mathcal{M} can be utilized that provides the mediation facilities for handling these mismatches. In this case, \mathcal{M} would incorporate two OO mediators – one for translating between UBL and WSML, and one that resolves terminology mismatches between the ontologies – and would utilize a process mediator that allows to resolve the communication mismatches between X and Y.

In conclusion, we observe that Semantic Web services allow one to handle the EAI problem by lifting information and process descriptions to a semantic level. A specific characteristic of WSMO is that its four top-level concepts correspond to the core elements of a Semantic Web service based on an EAI platform and define them in a concise and unambiguous manner. Ontologies provide the semantically formalized terminology, WSMO Web services semantically annotate and execute computational functionalities, WSMO goals precisely specify client objectives in a machine-processable way, and WSMO mediators provide an infrastructure and techniques for handling and resolving potential heterogeneities that would prevent successful interaction of entities.

² OASIS Universal Business Language, see <http://docs.oasis-open.org/ubl/prd-UBL-2.0>.

11.2 E-Government

In comparison with e-commerce, the area of e-government is a new application area for Web-based IT solutions; it is concerned with providing an advanced public infrastructure and governmental services on the Internet [17]. Commencing with Web portals for countries, cities or aspects of public interest, e-government has recently gained official interest. Remarkable effort have been initiated by the United States government (see <http://www.whitehouse.gov/omb/egov>). Aiming at a novel level of citizen support via a completely Web-based infrastructure, including registration of citizens and businesses, handling of immigration and visas, public health-care, and other services related public administration; [80] gives an overview of best practice and existing efforts in e-government.

Although the problem of widely used, incompatible standards is not as prevalent as in B2B commerce, legacy data and systems do exist and should be utilized in novel e-government solutions. Hence, technology developers are applying widely the Semantic Web and Web service technologies in order to prevent problems arising this. The first aspect is concerned with the use of ontologies as the data model for allowing advanced processing, maintenance, and usage of information within e-government resources, by applying ontology technologies developed for the Semantic Web [63]. The second aspect is concerned with the use of Web services as a basic building block for dynamic, more complex e-government services [120]. Naturally, such techniques lack support for automated Web service usage as is needed for dynamic access, composition, and execution of computational facilities over the Web.

In consequence, [57] has proposed an architecture for e-government applications that consists of four layers, as displayed in Fig. 11.2. The lowest layer comprises systems and information resources that provide general-purpose data or services useful for e-government. The second layer provides standard access and usage interfaces (Web services) to the first layer. The third layer provides the semantic annotation for the second layer in order to allow automated handling and execution of e-government services, and the fourth layer denotes actual e-government applications that work on the third-layer technologies. Evidently, the semantically enabled technologies for discovery, composition, mediation, and execution of Web services presented in this book are allocated at the semantics layer of this architecture.

A prototype e-government application that realizes this architecture provides an emergency planning and notification system for Essex, a county located to the northeast of London in the UK. The first-layer resources use geographical information systems (GIS) that are provided as Web services (second layer) and have a WSMO Web service description (third layer). The application (fourth layer) utilizes these to provide a decision support system that assists emergency authorities such as emergency planners, the police, the ambulance service, and the army in gathering information related to any particular type of event, quicker and more accurately.

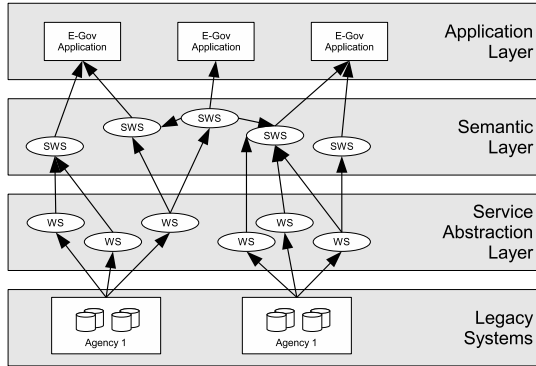


Fig. 11.2. Semantic Web services in e-government

As is explicitly denoted in Fig. 11.2, Semantic Web services provide the semantic layer for automated Web service usage. Similar to the usage within B2B commerce applications, the benefit of the WSMO framework is that it not only provides a means for semantically describing Web services, but also defines the other core elements of e-government applications based on Semantic Web services. While WSMO mediators allow the handling of potential heterogeneities that hamper interoperability, the notion of WSMO goals allows one to specify the end user requirements in an explicit, unambiguous, and machine-readable manner. For instance, the Stansted Airport flight management can define a goal of being notified when a storm is arriving that might impair landing conditions in the airport; such essential information can be seamlessly integrated into the flight management system.

11.3 E-Banking

Yet another application area for Semantic Web services and WSMO is e-banking, i.e. Web-based IT solutions for processing, handling, and monitoring financial transactions. The financial sector has always been an early adapter to innovative technologies in order to manage the vast amount of information that needs to be dealt with on a daily basis. Today's financial business use IT to a very high extent, starting from customer management in private customer bank branches, through credit card business and online banking, to professional day-trading technology such as Bloomberg Professional, a system used by investment brokers all over the world.

Because of the monetary value of the information interchanged, security of data and transactions and system stability are the primary and the critical factors for success of IT solutions for the banking business. However, [25] observes certain shortcomings of the existing technologies used in the financial sector that can be overcome, or at the very least, addressed, with semantic technology:

- *Intra-institutional.* Banks and other financial institutions usually consist of several interrelated departments that need to interact in order to process daily requests. Each department usually has its own structure and processing procedure, which means heterogeneities are likely to arise and hamper successful and effective interaction. Here, semantic technologies can be used for advanced information management based on ontologies. Furthermore, if each department were to publish its processing functionality as a Web service, service-oriented architectures for managing the processing procedure of the overall institution could be envisioned.
- *Inter-institutional.* Financial institutions usually need to cooperate in order to provide their facilities. For instance, money transfers between different banks and exchanging information about the financial standing of customers, are regular and important procedures in the banking business. As most interacting institutions will be in different locations, and since heterogeneities in terminology, data formats, and business processes are more or less inevitable, a service-oriented architecture based on Semantic Web service technology appears to be a suitable means to provide adequate support.
- *Customer support.* A critical factor for the success and marketability of banks is sophisticated, high-quality support for their customers – be they private persons, businesses, or other financial institutions. Here, semantic technologies can be applied in order to develop advanced customer support technologies.

The first two aspects relate directly to B2B integration as previously discussed in Section 11.1.2. In the following, we illustrate the third aspect where banks can provide new, interesting applications for advanced customer support enabled by Semantic Web services.

Most banks are organized as commercial businesses. Hence, efficiency with respect to resources and costs, and nondisclosure of information and processing procedures are inevitable requirements for IT. Web services can be recognized as an appropriate technology, as they merely require to define the public process for consuming a functionality while internal aspects of how this functionality is achieved can be kept unpublished. Therefore, we can assume that convincing banks and other financial-service providers that publishing their functionality as a Web service is possible, as it does not contradict their primary interests as commercial businesses.

Having semantic descriptions of the functionalities provided allows the development of new, interesting applications that provide added value to customers. The following describes two of such applications that use WSMO and are currently under development within the DIP project (see dip.semanticweb.org):

- *Mortgage comparison.* A major area of financial business is the provision, selling, and management of mortgages. This is a compelling business: the provider (i.e. the bank) will earn money by selling a mortgage contract,

and the client is naturally interested in the cheapest offer that suites his needs. Usually, clients consider several offers and compare these in detail with respect to the monetary amount in question. If several banks provide their mortgage offers via semantically described Web services, we can easily imagine an application that compares the offers and automatically selects the one which is most suitable for the customer's needs. Here, the WSMO framework assists in creating client requests in the form of formally specified goals, and WSMO mediators allow possible mismatches on the data and process level to be handled.

- *Stock market client.* Similarly to the business of mortgages, stock market support for private and business customers is an important area of operation of financial institutions. Stock market brokers are interested in technical solutions that automate customer support to the highest extent possible in order process as many transactions as possible. Some brokers allow trading in specific stock options only; therefore, the brokerage available differs slightly between providers, as do other conditions. As in the first scenario, we can imagine an end-customer client that observes the stock market and finds the most appropriate provider for handling a trade request on the basis of Semantic Web service descriptions provided by stock market brokers.

11.4 Summary

This chapter has examined examples of applications in order to illustrate and demonstrate the benefits attainable with Semantic Web service technology. We observe that the benefits for these applications provided by new technologies are similar in many application areas. In a nutshell, the essential ingredients are as follows:

1. *Ontologies* provide formalized terminology and domain knowledge definitions, which can serve as the basis for advanced, automated information organization, management, and processing.
2. *Web services* provide a means for making computational facilities accessible and usable over the Internet via standardized interfaces.
3. *Semantic Web services* provide semantically enabled technologies for automated Web service discovery, and composition, and execution of semantic annotation of Web services.
4. The *WSMO Framework* allows Semantic Web services to be realized and incorporates two additional top-level notions that denote core components of Web-service-based systems: *Goals* for semantically specifying client requests, and *Mediators* for handling potential mismatches that would hamper successful interaction.

Conclusion and Outlook

In the course of this book we have illustrated how to enable Semantic Web services by means of the Web Service Modeling Ontology approach. WSMO, together with its formal language WSML and the reference implementation WSMX, is a comprehensive framework that integrates two complementary and revolutionary technical advances, Web services and the Semantic Web, into a single computing architecture.

We shall briefly recapitulate the content of this book and outline the vision of Semantic Web services. In this final chapter, we give the reader a concise overview of the ongoing research in the area of Semantic Web services and of the corresponding efforts of international standardization bodies. The story does not end here, but with these final remarks we want to give the reader an opportunity to keep up effectively with the trends and developments within the Semantic Web and Semantic Web services communities.

Finally, we conclude the book with a critical outlook on technologies that offer alternatives to or are complementary to Semantic Web services.

12.1 Semantic Web Services Using WSMO

While the overall intuition behind Web services and service-oriented architectures is widely acknowledged for its potential to revolutionize the world of computing, success depends on resolving three fundamental challenges that SOAs alone do not address, namely, the automation of searching, integration and mediation. The WSMO framework addresses the semantic gap which hampers this automation by providing the following building blocks: a framework for semantic annotation of services, means for grounding semantic specifications in each component of the SOA framework, and semantically enabled solutions for the components of the overall framework.

In Part II, we outlined the vision of semantically enabled services contained in our understanding of the Web Service Modeling Ontology. We outlined the

concepts, language and usage tasks of a semantically enabled service architecture, aimed towards a future of dynamic collaboration of distributed services.

We face the fact that a fully fledged Semantic Web service framework will still take some time to be completely realized in terms of standards, products, and (what we currently call) production systems. So far – as we have outlined in Part III – there already exist several proof-of-concept applications and tools; however, these are still the product of research efforts and of specific needs in niche markets. Nevertheless, these efforts demonstrate in the small the real potential of Semantic Web services. Adoption in terms of standardization efforts will hopefully be the next step, and the first step in the direction of widespread exploitation.

12.1.1 Future Work on WSMO/L/X

The Web Service Modeling Ontology in its current state formalizes all the main principles that underlie our work and is now very mature. That said, we do not expect fundamental changes to the conceptual model anymore and believe that it is in a suitable shape to tackle all of the main problems that we want to address in such a framework. The effort in terms of the concrete language of WSML and in terms of tools is still more dynamic. Future work on WSML will include the application of the language to various use cases and the improvement of WSML tools, such as editors and reasoners (see Chapter 10). From a more theoretical point of view, considering language development, the semantics of WSML-Full has not yet been defined completely; we are currently investigating several nonmonotonic logics, such as autoepistemic and default logic and their combinability with purely classical logics in order to close this gap. There are approaches which combine expressive description logics with nonmonotonic logic programming without requiring the expressiveness of WSML-Full (e.g. [35, 118]). Incorporating such approaches into WSML is a matter of ongoing research. We are also currently working on defining the operational semantics for Web service capabilities and interface descriptions. Such an operational semantics is necessary for the automation of several Web-service-related tasks, such as discovery (see Section 9.2) and composition (see Section 9.4).

12.2 Standardization Efforts

In Part I of this book we discussed standard specifications that provide the basis for Semantic Web services. We made the criticism that until now these standards have not addressed the semantic gap between heterogenous entities interacting in a service-oriented architecture. To this end, it can be expected that a whole set of possible competing proposals for standards will soon be endorsed by several standardization bodies. As outlined throughout this book,

different aspects of Semantic Web service technologies present themselves at different levels of maturity at present.

The results relating to standards for ontology languages are quite mature, although some aspects, such as a Web enabled rule extension, have not yet been finalized.

The work on the overall framework for Semantic Web services is not as mature yet. In summer 2005, the W3C organized a workshop on *Frameworks for Semantics in Web Services*¹ to clarify the various views of the community and explore current chances for standardization. Over 80 participants from industry and academia presented their proposals, but despite many conceptual overlaps, one intention of the workshop was not achieved: The community was still far from agreement on the details of the technologies to be deployed, and therefore not ready for a rapid move towards standardization. Still, the strong industrial participation and the active contribution from global players showed once more that the problems that Semantic Web services are trying to solve are key aspects of today's IT business.

Although activities towards standardizing an overall framework for Semantic Web services have been stalled by the W3C for the moment, standardization organizations such as the Organization for the Advancement of Structured Information Standards (OASIS)² and also W3C itself have in the meantime established several working groups or technical committees to develop and standardize particular aspects of Semantic Web service technologies, as presented in this book.

For instance, most of the ongoing architectural effort on the Web Service Execution Environment (WSMX) has in the meantime moved to the *Semantic Execution Environment Technical Committee* (SEE TC) of OASIS. For the individual WSMX components outlined in this book, there is currently major progress and we have probably not been able to cover all the latest developments. Still, we can predict that we shall be able to choose between several alternative industrial strength implementations and strategies for every task and component within a few years.

While the SEE TC of OASIS focuses on the development of a reference execution environment for a semantically enhanced service-oriented architecture, several other groups are working on other important related aspects.

Within OASIS itself, various committees are working mainly on architecture related issues:

The OASIS *SOA Reference Model* (RM) Technical Committee is chartered to develop a reference model for serviceoriented architectures. The OASIS *Electronic Business Service-Oriented Architecture* (ebSOA) Technical Committee focuses on continuing work on the ebXML technical architecture to bring it into a more up-to-date form that takes into account subsequent re-

¹ <http://www.w3.org/2005/01/ws-swsf-cfp.html>.

² <http://www.oasis-open.org>.

leases of the ebXML³ specifications and other work on Web services and service-oriented architectures. The OASIS *SOA Adoption Blueprints* Technical Committee has recognized the shortage of clear, illustrative examples of working implementations of SOAs based on real-world needs and requirements that can be used as references for best practice and is working in this direction. Finally, the OASIS *Web Services Resource Framework* (WSRF) Technical Committee is defining a generic, open framework for modeling and accessing stateful resources using Web services. This includes mechanisms to describe views of the state of a Web service (or any other Web resource), to support management of this state through properties associated with the Web service, and to describe how these mechanisms are extensible to groups of Web services.

The W3C is not addressing issues specifically related to service-oriented architectures, but the results of several W3C working groups are crucial to the realization of Semantic Web services in that they focus on specific aspects on Web services and the Semantic Web.

The *Semantic Annotations for WSDL* Working Group aims to add minimal semantic extensions to WSDL. WSDL itself only specifies a way to describe the abstract functionalities of a service and concretely how and where to invoke it. The WSDL 2.0 specification, however, does not include semantics in service descriptions, and thus two services can have similar descriptions but totally different meanings or, vice versa, similar meanings with completely different descriptions. The objective of the Semantic Annotations for WSDL Working Group is to develop a mechanism to enable semantic annotation of WSDL descriptions. This mechanism will take advantage of the WSDL 2.0 extension mechanisms to build simple, generic support for semantics in Web services.

The *Rule Interchange Format* Working Group is chartered to specify a format for rules, such that they can be used across diverse systems. It aims to create a common format for static and dynamic business rules. This format (or language) will function as an interlingua into which established and new rule languages can be mapped, allowing rules written for one application to be published, shared, and reused in other applications and other rule engines. This is essentially along the same lines as the WSML language. One motivation for creating WSML was that at the time of starting the effort there was no rule language that fulfilled our requirements of (1) a clear layering, and (2) declarative semantics based on well established and proven formalism that (3) is compliant with Web technologies such as URIs, XML, and RDF.

Summarizing, while SOAs are widely accepted as the next generation of computing to which most software vendors are committed, standards are still evolving. From 2000 to 2005, SOA standard proposals grew enormously in number and complexity, with few reference technologies. Some standards already exist, while others are scheduled for development and release up to 2012

³ <http://www.ebxml.org>.

within OASIS and W3C. A review of the hundreds of standards related to Web services and SOAs is beyond the scope of this final chapter. However, we have noted the most important ones. At the end of 2005, Web service and SOA standards efforts were being commenced, as vendors recognize that without reasonable, effective, and agreed standards, SOA technology will not progress and customers will neither purchase nor deploy SOA solutions.

12.3 Industrial Collaboration

Even when a so-called standard is in place, this is only a prerequisite for the success of a technology, and not a guarantee by any means. The eventual criterion for success that Semantic Web service technology has to face will be industrial adoption. While scalability and precision are the foci of current industrial efforts related to Web services and service-oriented architectures, the more fundamental goal of enabling Semantic Web services is at the core of our research interests, as outlined in this book.

The challenge for the research and industrial communities over the next few years will be to collaboratively realize the concepts described in this book. This challenge involves two ongoing and complementary paradigm shifts in computing: (1) the movement to service-orientation, and (2) the use of semantic technologies and ontologies in industrial-scale infrastructures and applications. Achieving such a goal will require collaboration not only within the research community, but also among the global players in industry. To achieve what we consider true realization – defined not in terms of purely research prototypes but in terms of industrial-scale production applications – collaboration between the research and industrial communities is essential.

This will require research to understand the state and nature of the relevant industrial problems, products, and solutions. It will require industry to understand the relevant challenges and opportunities to which research can contribute. It will require researchers to collaborate with industry so that research results can be achieved and then integrated into industrial solutions. Among other things, work has to be done on industrial-standard ontologies that annotate the most common business concepts and standards. Examples of such common concepts are ordering and billing in various domains such as manufacturing, financial services, healthcare, inventory, and tourism. The hope is that as soon as we have reached a critical mass of semantic annotations, the effort required to adapt concepts to a new domain or specific business will be significantly reduced, and benefits will outweigh the cost of creating the necessary descriptions for Semantic Web services. Low entry barriers and high benefits are a key prerequisite for any new technology.

This challenge of collaboration has been recognized in the European Union. The European Commission is funding several large integrated projects and strategic research projects that aim to facilitate development in precisely this direction.

12.4 Alternatives to Classical Web Services

Even if we have all appropriate semantic annotations in place and assume that these are widely provided and used, there still remains an open question: Are current Web service technologies the right means to achieve “Semantic Web services”? Or, in other words how much “Web” is there in Web services? Current technologies designed around WSDL and SOAP are not necessarily the only starting point towards semantically enabled Web services [79].

A core paradigm of the Web is information exchange via persistent publication, i.e. one party publishes a piece of information on the Web, and any other party who knows the location of the resource can retrieve and process the information at any later point in time, without the need for synchronization with the original publisher. This functionality has significantly contributed to the scalability of the Web, since it reduced the amount of interaction between the sender and the recipient.

Current Web service technologies ignore this particularity to a large extent, the “Web” components used are mainly the communication layer (HTTP) and the message format (XML), and otherwise, Web service technology per se is a tightly coupled message exchange. This is similar to the situation in the pre-Web age, where one had to send an email message to a scientific colleague, asking for a specific paper or piece of information. In the current Web, used as an infrastructure for humans, this pattern of communication has been widely replaced by persistent publication and asynchronous retrieval. The shift from information dissemination based on message exchange has not only made the Web scale tremendously, it has also sped up the dissemination process. When we compare Web services with this important principle of the Web, it becomes evident that Web services per se do not follow this core idea. The idea of publishing data and accessing it asynchronously is lacking. Instead of publishing the information on the basis of a global, persistent URI, Web services establish (1) stateful conversations based on (2) the hidden content of messages.

The negative effect of such distributed applications that communicate via message exchange is that they require a strong coupling in terms of reference and time. This means that traditional Web services require that the sender and receiver of data (1) maintain a connection at the same time, (2) agree on the data format, and (3) know each other and share a common representation. Therefore, the communication has to be directed to a particular service and is synchronous as long as neither party implements asynchronous communication.

The above line of argument is in alignment with the worries expressed by the REST⁴ community [42, 136]. The two major criticisms of Web services are about the improper usage of URIs and the violation of the stateless ar-

⁴ REST stands for Representational State Transfer, a term coined by Roy Fielding in his Ph.D. dissertation [42].

chitecture of the Web. It is one of the basic design principles of the Web and the REST architecture not to provide stateful protocols and resources.

In practice, this means that when SOAP messages are sent and received, the content of the information is hidden in the body and not addressed as an explicit Web resource with its own URI. Referring to the content transmitted via an explicit URI in an HTTP request would allow the content of a message to be treated like any other Web resource.

A possible approach to address these problems might be to look for alternative middleware paradigms to the current Web service technologies. As it turns out, *Tuple Space computing* [47] which can be seen as a paradigm for exchanging data between processes that is very similar to the paradigms of the Web, shows many of the desired characteristics.

Tuple Spaces have been introduced in parallel-programming languages such as Linda to implement communication between parallel processes [47]. Instead of sending messages forward and backward, a simple communication means is provided. Processes can write, delete, and read tuples from a persistent space. Tuple Space computing has very strong advantages; it decouples three orthogonal dimensions involved in information exchange: reference, time, and space.

This strong decoupling of all three relevant dimensions has obvious design advantages for defining reusable, distributed, heterogeneous, and quickly changing communicating applications. Moreover, a service paradigm based on Tuple Space computing also revisits the Web paradigm: information is persistently written to a globally accessible space where other processes can smoothly access it without starting a cascade of message exchanges.

In fact the Web and Tuple Spaces have many things in common. They are both scalable information spaces for persistent publication and information dissemination that (in principle) do not rely on replication. However, traditional Tuple Spaces provide a flat and simple data model that does not provide any nesting or linking of data. Hence, tuples with the same number of fields and the same field order but different semantics cannot be distinguished.

In order to make Tuple Spaces usable for Semantic Web services, it would thus be an appealing idea to move Tuple Space computing in the direction of the Semantic Web, by adopting the Semantic Web data model (RDF) and allowing semantics-aware matching rules, which we could call (borrowing from RDF triples as the underlying data model) “Triple Spaces”.

Note that these ideas do not make Semantic Web services based on traditional Web services obsolete. More sophisticated technologies are still needed in the Semantic Web. Tuple spaces can help to decouple communication; however, they do not provide answers to data and information heterogeneities up front. That is still a task for the Semantic Web, as it provides standards to represent machine-processable semantics of data.

We emphasize also that Triple Space computing is more a complement to Semantic Web services based on “traditional” Web service technologies than something aimed at replacement of these technologies. Nevertheless, we

wanted to point out this alternative explicitly, in order to give a complete picture of ongoing research efforts towards making the vision of Semantic Web services come true. Research efforts towards using Triple Space computing to provide an alternative Semantic Web middleware are currently being undertaken in several ongoing research projects.⁵ Preliminary ideas and results related to Triple Space computing are available, for instance, in [23, 39, 79].

It is our goal that the Web Service Modeling Ontology will provide the foundations that will eventually cater for groundings to the best-suited and most widely adopted service technologies, such that WSMO will be able to provide the semantic descriptions in any of these technologies, be they “pure” Web services, Triple Spaces, or another emerging technology.

⁵ TripCom – Triple Space Communication: <http://www.tripcom.org>,
TSC – Triple Space Computing: <http://tsc.deri.at>.

References

1. R. Akkiraju et al. *Web Service Semantics: WSDL-S*. W3C Member Submission, 7 November 2005. Available from <http://www.w3.org/Submission/WSDL-S/>.
2. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer, Berlin/Heidelberg, 2004.
3. S. Anderson et al. *Web Services Trust Language (WS-Trust)*. Technical report, Actional, BEA, Computer Associates, IBM, Layer 7, Microsoft, Oblix, Open Network Ping Identity, Reactivity, RSA Security Verisign, February 2005.
4. T. Andrews et al. *Business Process Execution Language for Web Services*, version 1.1. Technical report, IBM, May 2003. <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>.
5. F. Baader et al., editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
6. Z. Baida, J. Gordijn, B. Omelayenko, and H. Akkermans. A shared service terminology for online service provisioning. In *ICEC '04: Proceedings of the 6th International Conference on Electronic commerce*, pages 1–10, Delft, The Netherlands, 2004.
7. S. Bajaj et al. *Web Services Policy Attachment (WS-PolicyAttachment)*. Technical report, BEA, IBM, Microsoft, SAP, Sonic Software, Verisign, September 2004.
8. S. Bajaj et al. *Web Services Policy Framework (WS-Policy)*. Technical report, BEA, IBM, Microsoft, SAP, Sonic Software, Verisign, September 2004.
9. S. Battle et al. *Semantic Web Services Framework (SWSF)*.
10. T. Bellwood et al. *UDDI version 3.0*, July 2002. Available from <http://uddi.org/pubs/uddi-v3.00-published-20020719.htm>.
11. B. Benatallah, M.-S. Hacid, C. Rey, and F. Toumani. Request rewriting-based web service discovery. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, volume 2870 of *LNCS*, pages 242–257. Springer, Berlin/Heidelberg, 2003.
12. T. Berners-Lee and M. Fischetti. *Weaving the Web*. Harper, San Francisco, 1999.
13. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

14. S. Bhiri et al. An orchestration and business process ontology. DIP Deliverable D3.4, January 2005.
15. A. D. Birell and B. J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 1(2):39–59, 1984.
16. P. V. Biron and A. Malhotra, editors. *XML Schema Part 2: Datatypes*. W3C Recommendation, 28 October 2004. Available from <http://www.w3.org/TR/xmlschema-2/>.
17. M. Boehlen, J. Gamper, W. Polasek, and M. A. Wimmer. *E-Government: Towards Electronic Democracy*. Springer, Berlin/Heidelberg, 2005.
18. A. Borgida et al. Classic: A structural data model for objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 59–67, 1989.
19. R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
20. D. Brickley and R. V. Guha, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, 10 February 2004. Available from <http://www.w3.org/TR/rdf-schema/>.
21. D. Brickley and L. Miller. FOAF Vocabulary Specification. <http://xmlns.com/foaf/0.1/>, 2004.
22. C. Bussler. *B2B Integration: Concepts and Architecture*. Springer, Berlin/Heidelberg, 2003.
23. C. Bussler. A minimal triple space computing architecture. In *Proceedings of the 2nd WSMO Implementation Workshop (WIW)*, Innsbruck, Austria, 2005.
24. L. Cabral and J. Domingue. Mediation of semantic web services in IRS-III. In *Proc. Workshop on Mediation in Semantic Web Services (MEDIATE 2005), in conjunction with ICSOC 2005*, 2005.
25. P. Castells, B. Foncillas, R. Lara, M. Rico, and J. L. Alonso. Semantic Web technologies for economic and financial information management. In *Proceedings of the First European Semantic Web Symposium, ESWS 2004, Heraklion, Crete, Greece*, 2004.
26. W. Chen, M. Kifer, and D. S. Warren. HiLog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, 1993.
27. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, editors. *Web Services Description Language (WSDL) 1.1*. March 2001. <http://www.w3.org/TR/wsdl>.
28. E. Cimpian and A. Mocan. WSMX Process Mediation Based on Choreographies. In *Proceedings of the 1st International Workshop on Web Service Choreography and Orchestration for Business Process Management, at BPM 2005, Nancy, France*, 2005.
29. W. J. Clancey. Heuristic classification. *Artificial Intelligence*, 27(3):289–350, 1985.
30. J. de Bruijn, editor. *The Web Service Modeling Language (WSML)*. WSML Final Draft, 5 October 2005. Available from <http://www.wsmo.org/TR/d16/d16.1/v0.21/>.
31. J. de Bruijn, A. Polleres, R. Lara, and D. Fensel. OWL DL vs. OWL Flight: Conceptual modeling and reasoning on the Semantic Web. In *Proceedings of the 14th International World Wide Web Conference (WWW2005)*, Chiba, Japan, 2005.
32. M. Dean and G. Schreiber, editors. *OWL Web Ontology Language Reference*.

33. J. Domingue, L. Cabral, F. Hakimpour, D. Sell, and E. Motta. IRS-III: A platform and infrastructure for creating WSMO-based Semantic Web services. In *Proceedings of the Workshop on WSMO Implementations (WIW 2004)*, Frankfurt, Germany, 2004.
34. A. Duke, M. Richardson, S. Watkins, and M. Roberts. Towards B2B integration in telecommunications with Semantic Web services. In *Proceedings of the 2nd European Semantic Web Conference, ESWC 2005, Heraklion, Crete*, pages 710–724, 2005.
35. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. In D. Dubois, C. Welty, and M.-A. Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*. AAAI Press, Menlo Park, CA, 2004.
36. A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: a tool for collaborative ontology construction. In *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1996.
37. C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.
38. D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, 2nd edition*. Springer, Berlin/Heidelberg, 2003.
39. D. Fensel. Triple-space computing: Semantic Web services based on persistent publication of information. In *Proceedings of the IFIP Int'l Conference on Intelligence in Communication Systems, Bangkok Thailand*, pages 43–53, 2004.
40. D. Fensel, J. Angele, S. Decker, M. Erdmann, H.-P. Schnurr, R. Studer, and A. Witt. Lessons learned from applying AI to the web. *International Journal of Cooperative Information Systems*, 9(4):361–382, 2000.
41. D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.
42. R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, CA, 2000.
43. I. O. for Standardization (ISO). *Code for the Representation of Names of Languages*, ISO 639:1988 (E/F). First edition, 1988.
44. I. O. for Standardization (ISO). *Representation of Dates and Times*, ISO 8601:2000. 2nd edition, 2004.
45. A. V. Gelder, K. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
46. M. Genesereth. KIF. knowledge interchange format: Draft proposed American national standard (dpANS). Technical Report NCITS.T2/98-004, ANSI KIF Ad Hoc Group, 1998. Available from <http://logic.stanford.edu/kif/dpans.html>.
47. D. Gerlerner. *Mirrorworlds*. Oxford University Press, New York, 1992.
48. J. González-Castillo, D. Trastour, and C. Bartolini. Description logics for matchmaking of services. In *KI-2001 Workshop on Applications of Description Logics*, September 2001.
49. S. Grimm, B. Motik, and C. Preist. Variance in e-business service discovery. In *Proceedings of the ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications*, Hiroshima, Japan, Nov. 2004.
50. B. N. Grosz, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of the 12th*

- International Conference on the World Wide Web (WWW2003)*, Budapest, Hungary, 2003.
51. B. N. Groszof, Y. Labrou, and H. Y. Chan. A declarative approach to business rules in contracts: Courteous logic programs in XML. In M. P. Wellman, editor, *1st ACM Conf. on Electronic Commerce (EC-99)*, pages 68–77, Denver, CO, 1999. ACM Press.
 52. O. M. Group. UML 1.4.2 specification. Technical Report ISO/IEC 19501, ISO, 2005.
 53. T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.
 54. M. Grüninger. A guide to the ontology of the process specification language. In R. Studer and S. Staab, editors, *Handbook on Ontologies in Information Systems*. Springer, Berlin/Heidelberg, 2003.
 55. M. Grüninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *Proceedings of IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, Canada, 1995.
 56. N. Guarino. Formal ontology and information systems. In *Formal Ontology in Information Systems, Proceedings of FOIS'98*, pages 3–15, Trento, Italy, June 1998. IOS Press.
 57. A. Gugliotta, L. Cabral, J. Domingue, V. Roberto, M. Rowlatt, and R. Davies. A Semantic Web service-based architecture for the interoperability of e-government services. In *Proceedings of the 5th International Conference on Web Engineering (ICWE 2005)*, Sydney, Australia, 2005.
 58. Y. Gurevich. *Evolving Algebras 1993: Lipari Guide*, pages 9–36. Oxford University Press, New York, 1995.
 59. F. Halsall. *Computer Networking and the Internet, 5th edition*. Pearson Education, Harlow, 2005.
 60. P. Hayes. *RDF Semantics*. W3C Recommendation, 10 February 2004. Available from <http://www.w3.org/TR/rdf-mt/>.
 61. H. He, H. Haas, and D. Orchard. *Web Services Architecture Usage Scenarios*. W3C Working Group Note, 11 February 2004. Available from <http://www.w3.org/TR/ws-arch-scenarios/>.
 62. J. Heflin and J. Hendler. Dynamic ontology on the web. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, pages 443–449. AAAI Press, Menlo Park, CA, 2000.
 63. R. Hodgson and D. Allemang. Semantic Technology for E-Government. In J. Cardoso and A. Sheth, editors, *Semantic Web Services, Processes and Applications*, pages 291–310. Springer, Berlin/Heidelberg, 2005.
 64. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof, and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission, 21 May 2004. Available from <http://www.w3.org/Submission/SWRL/>.
 65. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
 66. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic location of services. In *Proceedings of the 2nd European Semantic Web Conference (ESWC 2005)*, Crete, Greece, 2005.

67. U. Keller, H. Lausen, and M. Stollberg. On the semantics of functional descriptions of Web services. In *Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, Montenegro, 2006.
68. S. N. Khoshafian and G. P. Copeland. Object identity. In *Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '86)*, pages 406–416, 1986.
69. M. Kifer. Rules and ontologies in F-Logic. In N. Eisinger and J. Maluszyński, editors, *Reasoning Web: First International Summer School 2005, Msida, Malta*, volume 3564 of *LNCS*, pages 22–34. Springer, Berlin/Heidelberg, 2005.
70. M. Kifer, J. de Bruijn, H. Boley, and D. Fensel. A realistic architecture for the Semantic Web. In *Proceedings of the International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML-2005)*, Ireland, Galway, November 2005.
71. M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A logical framework for Web service discovery. In *ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications*, 2004.
72. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.
73. A. Kiryakov, D. Ognyanov, and V. Kirov. A framework for representing ontologies consisting of several thousand concepts definitions. DIP Deliverable D2.2, Ontotext Lab, 2004.
74. M. C. A. Klein, J. Broekstra, D. Fensel, F. van Harmelen, and I. Horrocks. Ontologies and schema languages on the web. In D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, pages 95–139. MIT Press, Cambridge, MA, 2003.
75. G. Klyne and J. J. Carroll, editors. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation, 10 February 2004. Available from <http://www.w3.org/TR/rdf-concepts/>.
76. J. Kopecký, D. Roman, M. Moran, and D. Fensel. Semantic Web Services Grounding. In *International Conference on Internet and Web Applications and Services (ICIW 2006)*, page 127, Guadeloupe, French Caribbean, Feb. 2006.
77. J. Kopecky, D. Roman, and J. Scicluna, editors. *WSMO Use Case: Amazon E-commerce Service*. WSMO Final Draft, 5 December 2005. Available from <http://www.wsmo.org/TR/d3/d3.4/v0.1/>.
78. S. Korper and J. Ellis. *The E-Commerce Book: Building the E-Empire, 2nd edition*. Morgan Kaufmann, San Francisco, 2000.
79. R. Krummenacher, M. Hepp, A. Polleres, C. Bussler, and D. Fensel. WWW or What is Wrong with Web Services. In W. Löwe and J.-P. Martin-Flatin, editors, *Proceedings of the 3rd European Conference on Web Services (ECOWS 2005)*, pages 235–243, Växjö, Sweden, Nov. 2005. IEEE Computer Society, Los Alamitos, CA.
80. B. D. Lanvin. *E-Government Handbook*. Center for Democracy and Technology and infoDev, 2004. Available from <http://www.cdt.org/egov/handbook/>.
81. R. Lara, A. Polleres, H. Lausen, D. Roman, J. de Bruijn, and D. Fensel. *A Conceptual Comparison between WSMO and OWL-S*. WSMO Final Draft, 12 April 2005. Available from <http://www.wsmo.org/TR/d4/d4.1/v0.1/>.
82. D. B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

83. F. Leymann. Web services flow language (WSFL 1.0). <http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>, 2001.
84. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web*, Budapest, Hungary, May 2003.
85. J. W. Lloyd. *Foundations of Logic Programming*, 2nd edition. Springer, Berlin/Heidelberg, 1987.
86. A. Malhotra, J. Melon, and N. Walsh, editors. *XQuery 1.0 and XPath 2.0 Functions and Operators*. W3C Candidate Recommendation, 8 June 2006 .Available from <http://www.w3.org/TR/xpath-functions/>.
87. D. Martin et al. *OWL-S: Semantic Markup for Web Services*. W3C Member Submission, 22 November 2004. Available from <http://www.w3.org/Submission/OWL-S/>.
88. D. McDermott. DRS: A set of conventions for representing logical languages in RDF. January 2004. Available from <http://www.daml.org/services/owl-s/1.1B/DRSGuide.pdf>.
89. D. L. McGuinness. Ontologies come of age. In D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, chapter 6, pages 171–194. MIT Press, Cambridge, MA, 2003.
90. S. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46–53, 2001.
91. R. Meersman. Semantic ontology tools in IS design. In *Proceedings of the ISMIS'99 Conference*, Warsaw, Poland, June 1999.
92. N. Mitra, editor. *SOAP Version 1.2 Part 0: Primer*. W3C Recommendation, 24 June 2003. Available from <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>.
93. A. Mocan, editor. *WSMX Data Mediation*. WSMX Working Draft, 11 October 2005. Available from <http://www.wsmo.org/TR/d13/d13.3/v0.2/>.
94. A. Mocan, E. Cimpian, M. Stollberg, F. Scharffe, and J. Scicluna, editors. *WSMO Mediators*. WSMO Final Draft, 21 December 2005. Available from <http://www.wsmo.org/TR/d29/>.
95. M. Moran and A. Mocan. Towards Translating between XML and WSMO based on mappings between XML Schema and an equivalent WSMO Ontology. In *Proceedings of the 2nd WSMO Implementation Workshop (WIW)*, Innsbruck, Austria, 2005.
96. E. Motta. An overview of the OCML modelling language. In *8th Workshop on Knowledge Engineering: Methods and Languages (KEML 98)*, Karlsruhe, Germany, 1998.
97. A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo, editors. *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*. OASIS, March 2004.
98. S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the 11th International World Wide Web Conference (WWW2002)*, Honolulu, Hawaii, May 2002.
99. B. Norton. Dataflow for orchestration in WSMO. WSMO Working Draft D15.1, 2006.
100. B. Norton, S. Foster, and A. Hughes. A compositional semantics for OWL-S. In *Proceedings of the 2nd International Workshop on Web Services and Formal*

- Methods (WS-FM 05)*, number 3670 in LNCS. Springer, Berlin/Heidelberg, Sept 2005.
101. B. Norton and C. Pedrinaci. 3-Level Service Composition and Cashew: A Model for Orchestration and Choreography in Semantic Web Services. In *Proceedings of 2nd International Workshop on Agents, Web Services and Ontologies Merging (AWeSOME'06)*, Oct 2006.
 102. N. Noy. Semantic integration: a survey of ontology-based approaches. *ACM SIGMOD Record*, 33(4):65–70, 2004.
 103. Object Management Group Inc. (OMG). Meta Object Facility (MOF) Specification v1.4, 2002.
 104. B. Omelayenko and D. Fensel. An Analysis of Integration Problems of XML-Based Catalogs for B2B Electronic Commerce. In *Proceedings of the 9th IFIP 2.6 Working Conference on Database Semantics (DS-9)*, pages 232–246, April 2001.
 105. J. O’Sullivan, D. Edmond, and A. T. Hofstede. What is a service? towards accurate description of non-functional properties. *Distributed and Parallel Databases*, 12(2–3):117–133, 2002.
 106. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In I. Horrocks and J. Hendler, editors, *The Semantic Web - ISWC 2002: First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002. Proceedings*, volume 2342 of LNCS, pages 333–347. Springer, Berlin/Heidelberg, 2002.
 107. M. Paolucci, N. Srinivasan, and K. Sycara. Expressing WSMO Mediators in OWL-S. In *Proceedings of the ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications*, Hiroshima, Japan, Nov. 2004.
 108. P. F. Patel-Schneider and D. Fensel. Layering the semantic web: Problems and directions. In I. Horrocks and J. Hendler, editors, *The Semantic Web - ISWC 2002: First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002. Proceedings*, volume 2342 of LNCS, pages 16–29. Springer, Berlin/Heidelberg, 2002.
 109. P. F. Patel-Schneider, P. Hayes, and I. Horrocks, editors. *OWL Web Ontology Language Semantics and Abstract Syntax*. W3C Recommendation, 10 February 2004. Available from <http://www.w3.org/TR/owl-semantics/>.
 110. A. Patil, S. Oundhakar, A. Sheth, and K. Verma. METEOR-S Web service Annotation Framework. In *Proceedings of the 13th International World Wide Web Conference (WWW2004)*, 2004.
 111. C. Preist. A conceptual architecture for semantic web services. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *The Semantic Web – ISWC 2004: Third International Semantic Web Conference, Hiroshima, Japan*, volume 3298 of LNCS, pages 395–409. Springer, Berlin/Heidelberg, Nov. 2004.
 112. S. Rajesh and D. Arulazi. Quality of service for Web services – demystification, limitations, and best practices.
 113. R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge, MA, 2001.
 114. J. Reynolds. *The Complete E-Commerce Book: Design, Build, and Maintain a Successful Web-Based Business*. CMP Books, Gilroy, CA, 2004.
 115. R. Richardson and A. F. Smeaton. Using WordNet in a knowledge-based approach to information retrieval. Technical Report CA-0395, School of Computer Applications, Dublin City University, Dublin, Ireland, 1995.

116. D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.
117. D. Roman, H. Lausen, and U. Keller, editors. *Web Service Modeling Ontology (WSMO)*. WSMO Final Draft, 13 April 2005. Available from <http://www.wsmo.org/TR/d2/v1.2/>.
118. R. Rosati. *DL + log*: Tight integration of description logics and disjunctive datalog. In P. Doherty, J. Mylopoulos, and C. Welty, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 10th International Conference (KR-06)*. AAAI Press, Menlo Park, CA, 2006.
119. F. Scharffe and J. de Bruijn. A language to specify mappings between ontologies. In *Proc. of the Internet Based Systems IEEE Conference (SITIS05)*, 2005.
120. S. K. Sharma and J. N. D. Gupta. Web Services Architecture for M-Government: Issues and Challenges. *International Journal of Electronic Government*, 1(4):462–474, 2004.
121. M. Stollberg, E. Cimpian, and D. Fensel. Mediating capabilities with delta-relations. In *Proceedings of the First International Workshop on Mediation in Semantic Web Services, Amsterdam, the Netherlands*, 2005.
122. M. Stollberg, E. Cimpian, A. Mocan, and D. Fensel. A Semantic Web Mediation Architecture. In *Proceedings of the 1st Canadian Semantic Web Working Symposium (CSWWS 2006), Quebec*, 2006.
123. M. Stollberg, U. Keller, and D. Fensel. Partner and service discovery for collaboration establishment with Semantic Web services. In *Proceedings of the 3rd International Conference on Web Services*, Orlando, Florida, July 2005.
124. M. Stollberg, H. Lausen, R. Lara, U. Keller, M. Zaremba, A. Haller, D. Fensel, and M. Kifer. *WSMO Use Case "Virtual Travel Agency"*. WSMO Working Draft, 19 November 2004. Available from <http://www.wsmo.org/TR/d3/d3.3/v0.1/>.
125. R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1–2):161–197, 1998.
126. K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5:173–203, 2002.
127. A. Tanenbaum. *Computer Networks, 4th edition*. Prentice Hall, Upper Saddle River, NJ, 2004.
128. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, editors. *XML Schema Part 1: Structures, 2nd Edition*. W3C Recommendation, 28 October 2004. Available from <http://www.w3.org/TR/xmlschema-1/>.
129. M. Uschold, M. King, S. Morale, and Y. Zorgios. The enterprise ontology. *The Knowledge Engineering Review*, 13(1):31–89, 1998.
130. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, June 2003.
131. K. Verma, A. Mocan, M. Zaremba, A. Sheth, J. Miller, and C. Bussler. Linking semantics web service efforts - integrating WSMX and METEOR-S. In *2nd International Workshop on Semantic and Dynamic Web Processes (SDWP 2005)*, July 2005.

132. K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management, Special Issue on Universal Global Integration*, 6(1):17–39, 2005.
133. S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Furgeson. *Web Services Platform Architecture*. Prentice Hall, Upper Saddle River, NJ, 2005.
134. S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. Dublin Core metadata for resource discovery. RFC 2413, IETF, September 1998.
135. G. Wiederhold. Mediators in the architecture of the future information systems. *Computer*, 25(3):38–49, 1994.
136. M. zur Muehlen, J. V. Nickerson, and K. D. Swenson. Developing Web services choreography standards - the case of REST vs. SOAP. *Decision Support Systems*, 40(1):9–29, 2004.

Index

- abstract state machines (ASMs), 71
- BP4WS, 48
- business-to-business (B2B), 161
- business-to-customer (B2C), 158

- capability, 69
- choreography, 70
- composition, 132
- control flow, 133

- data flow, 134
- data mediation, 126
- discovery, 115
- document type definition, 17
- DOM, 150
- DRS, 102

- e-banking, 166
- e-commerce, 157
- e-government, 165
- eclipse platform, 145
- eXtensible Markup Language (XML),
14

- F-Logic, 35

- goal, 74
- grounding
 - behavioral, 139
 - conceptual, 135
 - data, 136

- heuristic classification, 118

- HyperText Markup Language (HTML),
12
- HyperText Transfer Protocol (HTTP),
11

- interface, 69
- Internet Reasoning Service (IRS), 154

- Knowledge Interchange Format (KIF),
30, 102

- matchmaking, 123
- mediation, 124
- Meta-Object Facility MOF, 60
- middleware, 40

- nonfunctional properties, 64

- OASIS, 171
- ontology, 25, 63
- ontology alignment, 126
- ontology mapping, 126
- ontology merging, 126
- orchestration, 133
- OWL, 26, 33
- OWL-S, 101

- process mediation, 127

- reasoner, 142
- Remote Procedure Call (RPC), 40
- Resource Description Framework
(RDF), 31
- REST, 174

- Semantic Web, 25
- Semantic Web language layer cake, 26
- Semantic Web Rule Language (SWRL), 34
- Semantic Web Services Framework (SWSF), 104
- Semantic Web Services Language (SWSL), 105
- service-oriented architecture (SOA), 39
- SOAP, 42
- state signature, 72

- transition rule, 73
- triple spaces, 175

- UDDI, 46
- Unique Resource Identifiers (URI), 10

- Virtual Travel Agency, 113

- W3C, 9
- Web Ontology Language (OWL), 26, 33

- Web Service, 37, 67
- Web Service Definition Language (WSDL), 44
- World Wide Web, 7
- WSDL-S, 107
- WSML, 83
 - API, 142
 - atom, 91
 - logical expression, 90
 - molecule, 91
 - reasoner, 142
 - syntax, 86
- WSMO Studio, 149
- WSMO4J, 142
- WSMT, 146
- WSMX, 151

- XML, 17
- XML namespaces, 18
- XML Schema, 20
- XPath, 21
- XSLT, 21