

Dieter Fensel  
Federico Michele Facca  
Elena Simperl  
Ioan Toma

# Semantic Web Services

# Semantic Web Services



Dieter Fensel · Federico Michele Facca ·  
Elena Simperl · Ioan Toma

# Semantic Web Services



Prof. Dieter Fensel  
Dr. Federico Michele Facca  
Ioan Toma  
STI Innsbruck, ICT-Technologiepark  
University of Innsbruck  
Technikerstr. 21a  
6020 Innsbruck  
Austria  
[dieter.fensel@sti2.at](mailto:dieter.fensel@sti2.at)  
[federico.facca@sti2.at](mailto:federico.facca@sti2.at)  
[ioan.toma@sti2.at](mailto:ioan.toma@sti2.at)

Dr. Elena Simperl  
Institute AIFB, Building 11.40  
Karlsruhe Institute of Technology  
Englerstr. 11  
76128 Karlsruhe  
Germany  
[elena.simperl@kit.edu](mailto:elena.simperl@kit.edu)

ISBN 978-3-642-19192-3  
DOI 10.1007/978-3-642-19193-0  
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011927266

ACM Computing Classification (1998): H.3.5, D.2.12, I.2, J.1

© Springer-Verlag Berlin Heidelberg 2011

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 to prosecution under the German Copyright Law.

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.

### *Cover design: deblik*

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

# Preface

All students interested in researching the Internet are living in very interesting and exciting times. From the Internet's beginning in the 1960s up to the present day, the Internet has grown to become a vast platform where there are now 2 billion users of this global infrastructure.<sup>1</sup> Predictions are that this number will rise to over 4 billion with the advent of the Mobile Internet, providing access to the Internet via mobile devices including mobile phones. We are also witnessing a number of other trends which are driving the popular use of the Internet. Firstly, we have the Web, which now contains over 1 trillion resources with over 10 million added each day. At the end of 2009 alone, there were 234 million websites of which 47 million were added in the year. Secondly, we have the Web 2.0 phenomenon where the focus is on prosumers who play a dual role of consumer and producer. Enabling the general community to create and publish online material has facilitated the creation of content at unprecedented rates. For example, the current upload to Flickr is equivalent to 30 billion new photos per year and YouTube now serves over 1 billion videos per day. A final, general Internet phenomena, has been the rise of user generated applications on mobile devices as exemplified by Apple's iPhone and iPad Apps within the iTunes Store. The combination of public interest and a relatively simple creation and publishing process have resulted in over 200,000 Apps now being available for the iPhone and iPad and have led to Apple's share price surpassing Google's. For a researcher, especially in ICT, a great interest lies in making an impact on our present world in some way. Whilst this impact is usually seen in the economic or commercial arenas, it may also have important social implications. A particularly interesting phenomenon which has emerged over the last century is the transition of the Global Economy from the one based on manufacturing (and before that on agriculture) to the one based on services, sometimes known as the third sector. As defined in Wikipedia, a service is the non-material equivalent of a good where the provision of a service does not result in ownership which distinguishes a service from providing physical goods. From small beginnings at the turn of the twentieth century, services now dominate the Global Economy, accounting for 63% of the world's Gross

---

<sup>1</sup><http://www.internetworldstats.com/stats.htm>.

Domestic Product at \$37 trillion in 2009. In a fundamental respect, this textbook examines the link between the above mentioned through the inclusion of two main technologies: Web services and technologies associated with the Semantic Web. In a technical sense, Web services are computational components which can be invoked using standard Web protocols. More importantly, Web services can act as proxies for business services that can be used to deliver some functionality for clients or users which adds value. It is this second feature of Web services as online substitutes for business services that has caused tremendous interest in the corporate sphere. After initial interest, however, a number of problems emerged related to how online services could be found, invoked and composed. Over the past decade Semantic Web technology has been applied to address this through the delegation of certain portions of the above tasks to automated or semi-automated systems. This textbook examines the combination of Web services and the Semantic Web from a number of core ingredients, exploring the fundamental essence of its constituents. In particular, the Web services and the Semantic Web are broken down into a set of underlying principles and explained using simple examples. Connections to the real world and industrial deployment are outlined using a number of non-trivial use cases and through examination of the technologies supporting the commercial Web service search company Seekda. An associated website at [www.swsbook.org](http://www.swsbook.org) facilitates the inclusion and use of additional learning material. More generally, this book benefits from research experiences gained in three large European projects: DIP, SUPER and SOA4All which together have a combined budget of over 50 million euros. When combined with the extensive research track record of the authors, it is difficult to think of a more useful intellectual source for finding out about the Semantic Web Services area. If you are just embarking on a career as an ICT researcher, you are interested in services and/or semantics, or if you simply wish to catch up on the latest research in the Semantic Web area, then I thoroughly recommend that you read this book.

Knowledge Media Institute, The Open University, UK  
STI International, Austria

John Domingue

# Contents

## Part I Scientific and Technological Foundations of Semantic Web Services

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Web Science</b>	<b>9</b>
2.1	Motivation . . . . .	9
2.2	Technical Solution . . . . .	10
2.2.1	History of the Web . . . . .	10
2.2.2	Building the Web . . . . .	12
2.2.3	Web in Society . . . . .	16
2.2.4	Operationalizing the Web Science for a World of International Commerce . . . . .	18
2.2.5	Analyzing the Web . . . . .	21
2.3	Web 2.0 . . . . .	22
2.4	Conclusions . . . . .	22
	References . . . . .	23
<b>3</b>	<b>Service Science</b>	<b>25</b>
3.1	Motivation . . . . .	25
3.2	What Is a Service? . . . . .	26
3.3	Service Analysis, Design, Development and Testing . . . . .	27
3.4	Service Orchestration, Composition and Delivery . . . . .	29
3.5	Service Innovation . . . . .	30
3.6	Service Design Approach . . . . .	31
3.7	Service Pricing Method and Economics . . . . .	32
3.8	Service Quality Measurement . . . . .	32
3.9	Service Technologies . . . . .	33
3.10	Service Application . . . . .	34
3.11	Conclusions . . . . .	34
	References . . . . .	35

<b>4 Web Services . . . . .</b>	37
4.1 Motivation . . . . .	37
4.1.1 Service Oriented Computing (SOC) . . . . .	38
4.1.2 Service Oriented Architecture (SOA) . . . . .	39
4.2 Technical Solution . . . . .	40
4.2.1 Defining Web Services . . . . .	41
4.2.2 Web Service Technologies . . . . .	42
4.3 Illustration by a Larger Example . . . . .	55
4.4 Summary . . . . .	56
4.5 Exercises . . . . .	60
References . . . . .	64
<b>5 Web2.0 and RESTful Services . . . . .</b>	67
5.1 Motivation . . . . .	67
5.2 Technical Solution . . . . .	68
5.2.1 REST . . . . .	69
5.2.2 Describing RESTful Services . . . . .	69
5.2.3 Data Exchange for RESTful Services . . . . .	72
5.2.4 AJAX APIs . . . . .	77
5.2.5 Examples of RESTful Services . . . . .	78
5.3 Illustration by a Larger Example . . . . .	80
5.4 Summary . . . . .	83
5.5 Exercises . . . . .	84
References . . . . .	85
<b>6 Semantic Web . . . . .</b>	87
6.1 Motivation . . . . .	87
6.2 Technical Solution . . . . .	89
6.3 Extensions . . . . .	98
6.4 Summary . . . . .	101
6.5 Exercises . . . . .	102
References . . . . .	103
<b>Part II Web Service Modeling Ontology Approach</b>	
<b>7 Web Service Modeling Ontology . . . . .</b>	107
7.1 Motivation . . . . .	107
7.2 Technical Solution . . . . .	108
7.2.1 Ontologies . . . . .	110
7.2.2 Web Services . . . . .	113
7.2.3 Goals . . . . .	116
7.2.4 Mediators . . . . .	116
7.3 Extensions . . . . .	118
7.4 Illustration by a Larger Example . . . . .	119
7.4.1 Ontologies . . . . .	119
7.4.2 Goals . . . . .	120
7.4.3 Web Services . . . . .	120

7.4.4	Mediators . . . . .	124
7.5	Summary . . . . .	124
7.6	Exercises . . . . .	128
	References . . . . .	129
<b>8</b>	<b>The Web Service Modeling Language . . . . .</b>	<b>131</b>
8.1	Motivation . . . . .	131
8.1.1	Principles of WSMO . . . . .	131
8.1.2	Logics Families and Semantic Web Services . . . . .	132
8.2	Technical Solution . . . . .	134
8.2.1	WSML Language Variants . . . . .	134
8.2.2	WSML Basis . . . . .	136
8.2.3	Ontologies in WSML . . . . .	139
8.2.4	Web Services in WSML . . . . .	145
8.2.5	Goals in WSML . . . . .	149
8.2.6	Mediators in WSML . . . . .	152
8.2.7	Technologies for Using WSML . . . . .	152
8.3	Extensions . . . . .	153
8.4	Illustration by a Larger Example . . . . .	155
8.4.1	Travel Ontology . . . . .	155
8.4.2	Services . . . . .	157
8.4.3	Goal . . . . .	157
8.5	Summary . . . . .	158
8.6	Exercises . . . . .	158
	References . . . . .	160
<b>9</b>	<b>The Web Service Execution Environment . . . . .</b>	<b>163</b>
9.1	Motivation . . . . .	163
9.1.1	Service Orientation . . . . .	164
9.1.2	Execution Environment for Semantic Web Services . . . . .	168
9.1.3	Governing Principles . . . . .	168
9.2	Technical Solution . . . . .	169
9.2.1	SESA Vision . . . . .	169
9.2.2	SESA Middleware . . . . .	175
9.2.3	SESA Execution Semantics . . . . .	191
9.3	Illustration by a Larger Example . . . . .	200
9.3.1	Modeling of Business Services . . . . .	202
9.3.2	Execution of Services . . . . .	206
9.4	Possible Extensions . . . . .	210
9.4.1	Goal Subscription . . . . .	210
9.5	Summary . . . . .	213
9.6	Exercises . . . . .	213
	References . . . . .	215

## Part III Complementary Approaches for Web Service Modeling Ontology

<b>10</b>	<b>Triple Space Computing for Semantic Web Services</b>	219
10.1	Motivation	219
10.2	Technical Solution	221
10.2.1	Tuplespace Computing	221
10.2.2	Triple Space Computing	223
10.2.3	Triple Space Conceptual Models	224
10.2.4	Triple Space Architecture	229
10.2.5	Triple Space and Semantic Web Services	231
10.2.6	Triple Space and Semantic SOA	238
10.3	Illustration by a Larger Example	242
10.4	Summary	247
	References	248
<b>11</b>	<b>OWL-S and Other Approaches</b>	251
11.1	Motivation	251
11.2	OWL-S	252
11.2.1	Service Profile	252
11.2.2	Service Grounding	254
11.2.3	Service Model	256
11.2.4	An Extension to OWL-S	259
11.2.5	Tool Support	261
11.2.6	OWL-S Summary	263
11.3	METEOR-S	263
11.3.1	Semantic Annotation of Web services	264
11.3.2	Semantics-Based Discovery of Web Services	267
11.3.3	Composition of Web Services	268
11.3.4	METEOR-S Summary	269
11.4	IRS-III	270
11.4.1	Discovery, Selection and Mediation	273
11.4.2	Communication	274
11.4.3	Choreography and Orchestration	275
11.5	Summary	276
11.6	Exercises	277
	References	277
<b>12</b>	<b>Lightweight Semantic Web Service Descriptions</b>	279
12.1	Motivation	279
12.2	Technical Solution	280
12.2.1	SAWSDL	281
12.2.2	WSMO-Lite Service Semantics	283
12.2.3	WSMO-Lite in SAWSDL	288
12.2.4	WSMO-Lite for RESTful Services	289
12.3	Extensions	292
12.4	Summary	295

12.5 Exercises . . . . .	295
References . . . . .	295

## Part IV Real-World Adoption of Semantic Web Services

<b>13 What Are SWS Good for? DIP, SUPER, and SOA4All Use Cases . . . . .</b>	299
13.1 Motivation . . . . .	299
13.2 Data, Information, and Process Integration with Semantic Web Services (DIP) . . . . .	300
13.2.1 Motivation . . . . .	300
13.2.2 Technical Solution . . . . .	301
13.2.3 Use Cases . . . . .	302
13.3 Semantics Utilized for Process Management Within and Between Enterprises (SUPER) . . . . .	304
13.3.1 Motivation . . . . .	305
13.3.2 Technical Solution . . . . .	307
13.3.3 Use Cases . . . . .	311
13.4 Service Oriented Architectures for All (SOA4All) . . . . .	311
13.4.1 Motivation . . . . .	312
13.4.2 Technical Solution . . . . .	313
13.4.3 Use Cases . . . . .	318
13.5 Summary . . . . .	323
References . . . . .	323
<b>14 Seekda: The Business Point of View . . . . .</b>	325
14.1 Motivation . . . . .	325
14.2 Technical Solution . . . . .	326
14.2.1 Crawler . . . . .	327
14.2.2 Search Engine . . . . .	333
14.2.3 Bundle Configurator and Assistant . . . . .	333
14.3 Illustration by a Larger Example . . . . .	342
14.4 Summary . . . . .	349
14.5 Exercises . . . . .	351
References . . . . .	351
<b>Index . . . . .</b>	353



**Part I**

**Scientific and Technological Foundations**

**of Semantic Web Services**



# Chapter 1

## Introduction

“The Internet world as we know it today has undergone far-reaching and profound changes since it has become a critical communications infrastructure underpinning our economic performance and social welfare. With an estimate of 1.1 billion fixed users and over 3 billion mobile users world-wide already, the Internet of tomorrow is poised to become a fully pervasive infrastructure providing anywhere, anytime broadband connectivity.”<sup>1</sup>

Interoperability has always been an important topic in Computer Science. We began with the obscure ingredients of a simple network to exchange Morse code and punch cards which enabled a semi-automatic loom. The recipe evolved into an immensely powerful infrastructure that would serve as the basis of modern communication and collaboration. Several decades ago, the fundamental technologies that would spawn the Internet as we know it were still in their infancy. The past 30 years have brought these technologies to a mature level; communication, collaboration and system interoperability now depend upon “killer applications” such as email and the Mobile Web.

The ICT industry is witnessing a dramatic change, similar to the evolution of more traditional industries in the last century. The development of ICT solutions is now shifting from being a discipline mostly focussed upon productivity, to the one aiming at providing end-to-end customized services able to meet customers’ demands and bring distinctiveness to the solutions provided. This evolution is, to a large extent, due to the maturity which the ICT industry is reaching. Additionally, current trends such as the diversification of client hardware and communication infrastructure, the increasing availability of wireless and mobile access, as well as the ICT/Mobile/Media convergence, demonstrate the changes of scale and dynamicity that will be brought by the Future Internet and are furthermore making more evident the need to restructure the development and delivery chain to address future challenges and opportunities.

Still, ensuring interoperability in open, heterogeneous, and dynamically changing environments, such as the Internet, remains a major challenge for actual machine-to-machine integration. Applications are usually designed for a certain

---

<sup>1</sup> BLED Future Internet Manifesto, March 2008, available at <http://www.fi-bled.eu/>.

context; this context is then hardwired in terms of hidden assumptions at the design and implementation levels. As a consequence, significant problems in aligning data, processes, and protocols appear as soon as a specific piece of functionality is used within a different application context—this has been confirmed in numerous studies, which estimate a market for integration solutions in the order of a magnitude of over one trillion of dollars.<sup>2</sup> Technologies facilitating robust and scalable interoperability are a must for an economically based ICT within enterprises and a key component of the Internet of tomorrow.

In a business environment, it is the service that is important for a given customer, not the specific hardware or software components that are used to deliver this service. To a certain extent, the ICT industry responded to this challenge by moving from software to “serviceware” on the basis of the emerging concept of “Web services”. Currently, there is a paradigm shift in Computer Science: one generation ago, Computer Science learned to abstract from hardware to software; it is now in a stage of abstracting from software to serviceware in terms of Service Oriented Computing. At its core, this shift is concerned with the complexity of properly handling services in large-scale applications. The concept of service orientation is widely claimed to be the solution for the integration problem. However, it, in fact, implies that an additional step of abstraction in the design and development of ICT systems is still required in order to properly handle the complexity of managing the potentially very large numbers of available services.

In order for service-oriented technologies to scale, they will need to offer a significant degree of automation of service discovery, ranking and selection, composition, and invocation, as well as data-, protocol-, and process-mediation facilities. Manual integration is not feasible when aiming at sophisticated means for managing systems that are dynamically composed of numerous services, and automation can be achieved only if machine-processable descriptions of services are available. Moreover, semantic descriptions of these artifacts are necessary in order to establish and warrant interoperability that does not require the human user to either be present at all times of the process, or to manually hardwire certain integration solutions that will be rapidly outdated, or hardly reusable in an environment with such dynamic contexts. Such hardwired ad-hoc integrations only generate an additional burden for the next integration wave, leading to an exponentially growing problem. That is, semantics is required to achieve scalable and robust integration. The principled combination of semantic and service-oriented technologies has been motivated by the need to provide robust and scalable interoperability solutions in both closed and open IT environments. Semantic descriptions of services facilitate the flexible integration among applications within and beyond enterprise boundaries.

Several research efforts have addressed the formal semantic descriptions of computational artifacts in order to automate their use and reuse in software systems.

---

<sup>2</sup>Worldwide Enterprise Application Integration (EAI) Market. Venn Research Inc., March 2006.

Based on previous works in formal software specification and knowledge engineering, the latest development in this area is the concept of “Semantic Web Services” (SWS). The SWS approach is about describing services with metadata on the basis of domain ontologies as a means to enable their automatic location, execution, combination and usage. If services are described and annotated using machine-understandable semantics, a service requestor may specify the service needed in terms of the problem domain, or in other words, using business terminology. This is a major progress compared to the traditional technology, as to date the discovery of Web services is mostly based on signature matching or simply just on pre-defined addresses of the port to use. Further on, if the needs of the requestor cannot be fulfilled by a single service, semantic descriptions of service capabilities, and the way these services interact, can help to determine potential combinations of multiple services to accomplish the desired task.

The Semantic Web Services solutions described in this book, including service models, engineering methodologies, as well as development and execution environments, will have to be revised to reflect the requirements raised by these new scenarios. A large number of challenges are related not just to technology but also to business, governance and society. Future Semantic Web Services approaches will have to open up to adjacent disciplines such as service design and economics in order to overcome such challenges in a networked society of tomorrow.

With more and more data being published on the Web, companies are finding themselves in the position of having many critical business decisions rely solely upon a large amounts of public (linked) data. There are several sectors in which this can be clearly exemplified. In manufacturing, for instance, the business–customer relationship is no longer a one-way conversation where feedback can be controlled through careful treatment of a few traditional reviewers and opinion-makers. Through the Blogosphere, every Internet user is a potential reviewer and business’ attempted influence in this space is being carefully controlled. Instead, manufacturing businesses are turning more and more to eBusiness intelligence to track and react to such options, involving customers worldwide in design and production processes. In similar ways, sensor data increasingly informs business decisions in areas as diverse as tourism, manufacturing (e.g., clothing), and banking. Recent events in banking and World finance suggest that mining within such data sets could be of critical importance in predicting trends. A service-oriented technology platform supporting the realization of data-driven business applications as those afore mentioned will be one of the challenges for mastering large-scale data management in corporate environments. This will require the definition of new service models complying to the principles introduced by the Linked Data initiative, as well as new approaches to implement classical service tasks such as discovery, composition and mediation.

For Semantic Web Services to achieve industrial impact, it is essential for them to comply with and be based upon established technology standards. Many activities related to this area are affiliated to W3C, including several of the approaches presented in this book such as WSMO, OWL-S, SAWSDL, and more recently WSMO-Lite and MicroWSMO. Efforts to standardize the conceptual model under-

lying WSMO and OWL-S were undertaken in 2004/2005,<sup>3</sup> followed by SAWSDL in 2006/2007 in response to the need for more lightweight approaches building upon traditional WS-\* technology, providing a basic vocabulary to insert Semantic Web references within a WSDL service description.<sup>4</sup> The trend towards support for RESTful functionality had led to further W3C member submissions such as SA-REST, as well as WSMO-Lite,<sup>5</sup> and soon MicroWSMO and hRESTS.<sup>6</sup> WSMO-Lite builds on SAWSDL, which itself extends WSDL. When using WSMO-Lite, SAWSDL annotations can indicate more precisely the intent of the various other annotations. The same applies for MicroWSMO annotations. To include support for RESTful services, the SOA4All project (see Chap. 13) has developed MicroWSMO and hRESTS, two micro-formats supporting RDFa that are used to structure the HTML documentation of RESTful services so as to structure and organize the content of this documentation and facilitate machine-processability in a manner similar to WSDL and SAWSDL.

A second category of impact creation activities concentrates on the alignment of recent Semantic Web Services research with industrial initiatives, most prominently USDL (Unified Service Description Language).<sup>7</sup> Driven by SAP, USDL aims to provide a unified approach to describe and access business, as well IT services, taking into account the way such services are currently exposed and consumed in corporate environments—passing enterprise firewalls, within business networks, and as part of on-demand applications delivered over the cloud. In particular, the initiative puts emphasis on non-technical aspects of services, capturing the way consumers understand details of business operations, pricing, legal and other implications of using services. To increase its acceptance, the ultimate goal is to evolve USDL into a standard applicable to all types of services on the Internet, and integrating existing approaches, including Semantic Web Services frameworks. Future research will have to look into how this integration can be achieved in order to allow for an optimal leverage of semantic technologies to achieve seamless interoperability and scalability.

This book provides a comprehensive overview of Semantic Web Services in step with the actual practice in the field. It introduces the main socio-technological components that ground the Semantic Web Services vision (Web and service science, but also Web services in their SOAP and REST version) and several approaches realizing it, most notably the Web Service Modeling Framework WSMF consisting of the ontological service model WSMO, the WSML family of languages and the WSMX execution environment, as well as its recent resource-oriented extensions and underlying communication and coordination infrastructure. The real-world relevance is pinpointed through a series of case studies in large-scale R&D projects

<sup>3</sup>OWL-S: <http://www.w3.org/Submission/OWL-S>, WSMO: <http://www.w3.org/Submission/WSMO/>.

<sup>4</sup><http://www.w3.org/Submission/SA-REST/>.

<sup>5</sup><http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/>.

<sup>6</sup>[www.soa4all.org](http://www.soa4all.org).

<sup>7</sup><http://www.internet-of-services.com/>.

from the last years and a business-oriented proposition by the Semantic Web Services technology provider Seekda. Each chapter of the book is structured according to a pre-defined template, covering both theoretical and practical aspects, including walk-through examples and hands-on exercises. As such, the book is suitable for various audiences, as a state-of-the-art Semantic Web Services reference handling the full range of topics and technologies associated to Semantic Web Services, but also as an Computer Science textbook, primarily for an undergraduates course. The book has 14 chapters; the core Chaps. 2 to 13 require approximately a two-hour lecture each, thus the entire book can be taught during one semester. At the graduate level, the course could also include a selection of the complementary literature provided as further reading for each chapter. Additional learning material, including pointers to related education and training offers, is available at the book website at [www.swsbook.org](http://www.swsbook.org).

The chapters of this book provide a comprehensive overview of the current state-of-the-art of the formalisms, methods, tools and applications in the area of Semantic Web Services. They demonstrate that the achievements of the research community over the past years have resulted in technologies that are useful and applicable to the requirements of various business scenarios. Nevertheless, when compared to the more general IT landscape of the twenty first century, the reader will easily note that there are various significant developments and application scenarios which have not yet been considered in detail by existing Semantic Web Services solutions. In particular, the rapid dispersion of mobile and sensor computing technologies into numerous businesses, and as every-day life, the increasing importance of information analytics leveraging the huge volumes of data published over the Web, and the politically driven initiatives related to the design and development of principles, architectures and core components of the Internet of the future, all deserve further investigation.

The book is organized into four parts which, to some degree, can be read independently. The first part (Chaps. 2 to 6) is dedicated to the scientific and technological foundations of Semantic Web Services; readers familiar with the basics of Web services, World Wide Web or REST (Representational State Transfer) can start directly with the second part of the book (Chaps. 7 to 9), in which the main building blocks of the Web Service Modeling Framework are presented. The third part of the book (Chaps. 10 to 12) can be seen as complementary to the core WSM\* work. Chapter 10 introduces Triple Space computing, a communication and coordination paradigm for Web services based on the Web principle of persistent publish & read which promises to solve the autonomy issues known for message-based services infrastructures. Alternative Semantic Web Services approaches are analyzed in Chap. 11. The Web principle of persistent publish communication is also one of the themes of Chap. 12 which presents a model for lightweight service descriptions motivated by the raise of Web-based RESTful services. Finally, the fourth part of the book (Chaps. 13 and 14) discusses real-world adoption of Semantic Web Services. For a full comprehension of the case studies, it is advised that the reader studies Chaps. 7 to 9, and possibly also Chaps. 10 and 12, as these chapters introduce the

technologies validated through the case studies. Knowledge of the corresponding R&D projects is, however, not compulsory, though surely helpful. Exercises are given at the end of each chapter previously mentioned. They can be solved using the notions introduced in the associated chapter, but may involve the installation of additional software packages available on the Web.

# Chapter 2

## Web Science

**Abstract** Originally intended to be a simple way to share and interlink documents over the Internet, the Web has evolved into one of the most complex technologies currently available. This transformation took place in a short period of time, and in a rather chaotic manner, leading to the current lack of structure, security and confidentiality including billions of duplicate information pieces. Thus to overcome such issues, several key researchers contributing to the Web and related sciences are pointing out the need for a new emerging science, the so-called Web Science.

### 2.1 Motivation

The Web has evolved substantially since its creation, becoming one of the most complex technologies of the current time. Its evolution, initially driven primarily by researchers and specialists, is now strongly influenced by its billions of users. The Web Science is studying the Web transformation from its creation, to the present times, as well as its possible evolution in the future. Furthermore, Web Science delineates the directions in which it is expecting to further evolve. Web Science aims to answer two main questions, namely:

- How the Web functions, by analyzing the components that enable it to act as a decentralized information system [5].
- How Web traits arise and how they can be harnessed or held in check for the benefit of society.

As such, this science addresses the past, the present and the future of the Web. By analyzing the past, one can determine what makes the Web what it is today. By analyzing the trends and changes over the last few years, it can predict what the next steps will be. And finally, taking the predicted next steps into consideration, certain measures can be taken to enhance security, scalability, robustness or consistency.

This new discipline was formally launched in November 2006, when the Web Science Research Initiative (WSRI)<sup>1</sup> was founded. Since then, the initiative gathered researchers from 16 top universities, and it is leaded by the most prominent

---

<sup>1</sup><http://webscience.org/>.

researchers in the field (Tim Berners-Lee, Wendy Hall, Nigel Shadbolt, Danny Weitzner, and James Hendler).

Web Science is rapidly attracting researchers and practitioners from all over the world. Several universities are already providing extensive Web Science courses, covering subjects like the current and emerging Web concepts, technologies and Web-based software systems, social computing, networking and computational journalism.

Considering the vast area covered by Web Science, as well as the relatively short period of time since its foundation, this chapter is presenting an overview of its main activities and goals.

## 2.2 Technical Solution

As Web Science does not aim to solve a problem but rather tries to analyze an existing technology, we cannot really talk about a technical solution, but rather a technical approach for understanding the Web for developing this new, emerging science.

As for any emerging discipline, the Web Science curriculum is continuously evolving, with continuous growth in the number of areas of interest. There are, however, some fundamental research questions established as core elements of interest for Web Science by the Web Science Research Initiative. Such research questions cover:

- The history of the Web, how it transformed from a collection of documents to the Web we know today.
- Building the Web, the mechanisms and infrastructure needed.
- The influence of the Web in Society, as well as the influence of society over the Web.
- Operationalizing Web Science for a World of International Commerce.
- Analyzing the Web, its benefits and its drawbacks.

Numerous courses and publication are addressing the cited research areas. Among others, [8], [10] and [9] are analyzing the history of Web, W3C Recommendation [1] describes the architecture of the Web, [7] analyzes the influence of the Web in Society, while [5], [6] and [13] discuss the Web Science in general. The final objective is to determine its further evolution, long-term impact on the society, benefits and draw-backs.

In the following sections, we provide an overview of the topics introduced above. This overview is based on existing publications as well as ongoing research.

### 2.2.1 *History of the Web*

In the present day society, almost everyone has at least a basic notion of the Web origins and evolution. This section provides a short overview, based upon the World

Wide Web Consortium (W3C) history of the Web<sup>2</sup> and on the previously mentioned publications.

The history of the WWW can only be analyzed in conjunction with the main steps that led to its creation: the creation of the first network (the Advanced Research Projects Agency Network), its transition to the Internet, as well as the birth of the World Wide Web.

### **2.2.1.1 The Advanced Research Projects Agency Network (ARPANET)**

The first small network was created experimentally in 1965, when a dial-up telephone line was created to connect computers from Berkeley to the Massachusetts Institute of Technology (MIT). This was the first wide area network ever created. In the next two years, research in this direction evolved substantially, and the head of the Advanced Research Projects Agency (ARPA) computer research published the plans for ARPANET, a computer network system. The publication of the plans was the start of the collaboration between teams at ARPA, MIT, the National Physics Laboratory (UK) and Research ANd Development Corporation (RAND), which were previously working in isolation, towards the development of the first global network. Ideas from all these institutes were incorporated into the design of ARPANET. By 1969, it was already possible to connect computers between Stanford University and the University of California, Los Angeles (UCLA). By 1971, there were 23 host computers linked by ARPANET.

### **2.2.1.2 The Beginning of the Internet**

Since its inception, the aim of the Internet has been to provide a platform that would allow humans to exchange information. In the early 1970s, the success of the ARPANET led to the creation of a number of small, proprietary networks. Inside a network, it was possible to send messages from one computer to another (i.e., from one person to another), which is now one of the most common uses of the Internet—email. However, connecting these networks with each other required a common communication protocol.

The close collaboration between ARPA and Stanford University scientists led to the development of the first common language that allows different networks to communicate—the Transmission Control Protocol/Internet Protocol (TCP/IP).

This new development allowed humans to communicate asynchronously using computers, but this did not solve the problem of file sharing and information access. Even using File Transfer Protocol (FTP) servers, the first file transfer technology, people still needed to have previous knowledge of where certain information was available. Thus, some form of human communication was required in order to be able to transfer files.

---

<sup>2</sup><http://www.w3.org/History.html>.

### 2.2.1.3 The World Wide Web

The Internet was revolutionized in the early 1990s when the first program, called Gopher, allowing search on remote servers was developed at the University of Minnesota.<sup>3</sup> Gopher provided browsing facilities through hierarchies of application links, file, directories, graphics, etc. on a remote machine, plus facilities to search indexing servers.

However, this was only the first step towards the Web as we know it today. The true change and innovation was initiated by Enquire,<sup>4</sup> a small, easy to use information system developed by Tim Berners Lee. The World Wide Web (WWW) idea was further developed, which led to the creation of what we know today—an easy-to-use, easy-to-extend, easy-to-access global information system. The three pillars that supported the WWW were HyperText Transfer Protocol (HTTP, a standard protocol for retrieving hypertexts and other documents), Uniform (or Unique) Resource Identifier (URI, a globally unique identification system for every resource) and HyperText Markup Language (HTML, a format for creating and laying out human-readable, interlinked hypertext documents).

The first Web page was also created by T.B. Lee in 1990, and a copy of this page is still available online.<sup>5</sup>

The first International Conference on the World Wide Web (1994) gathered 380 participants from all over the world.<sup>6</sup> It laid the foundation for the creation of the W3C coalition.

## 2.2.2 *Building the Web*

As described in the previous section, the success of the early Web Architecture was based on three main building blocks, namely HTTP, URI and HTML. This section provides an overview of these main pillars, as well as of the Web Architecture as presented by the W3C recommendation [1].

### 2.2.2.1 HyperText Transfer Protocol (HTTP)

The Hypertext Transfer Protocol is an application-level protocol for distributed, collaborative, hypermedia information systems, which provides the means for a client-server interaction. Its development was coordinated by the World Wide Web Consortium and the Internet Engineering Task Force<sup>7</sup> (IETF). The pre-standard HTTP/1.1,

---

<sup>3</sup><http://gopherproject.org/Software/Gopher>.

<sup>4</sup><http://infomesh.net/2001/enquire/manual/>.

<sup>5</sup><http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html>.

<sup>6</sup><http://www94.web.cern.ch/WWW94/>.

<sup>7</sup><http://www.ietf.org/>.

defined in a series of Request for Comments,<sup>8</sup> (RFCs) was rapidly adopted in mid 1990s. By 1996, it was supported in Netscape 2.0, Netscape Navigator Gold 2.01, Mosaic 2.7, Lynx 2.5, and in Internet Explorer 3.0. The HTTP/1.1 standard was released in 1997.

For accessing the Web resources, HTTP defines eight methods: HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, and CONNECT. The most important are the request operations: GET, HEAD, PUT, DELETE, and POST. The GET method is triggered any time someone tries to access a Web resource via a browser. The URI associated to the GET defines the resource to be retrieved (i.e., its “address”). The HEAD method returns only the headers, i.e., the size of the remote resource and similar information (useful for testing); PUT is used to create a resource; DELETE, as the name suggests, is used for deleting a resource; the POST method is to update a resource.

There are two commonly adopted versions of HTTP: HTTP/1.0 and HTTP/1.1. Their definition can be found in RFCs 1945 (HTTP 1.0) and 2616 (HTTP 1.1).

### 2.2.2.2 Uniform Resource Identifiers (URIs)

Also initially defined in RFCs, the URIs are strings used to uniquely identify resources over the Internet. There are two types of URIs, namely the Uniform Resource Names (URNs) and Uniform Resource Locators (URLs). The distinction between these two is that an URN defines the identity of a resource, while an URL defines the way to find it.

A URI consists of four elements [10]:

<scheme> : / <authority><path>?<query>

where

Scheme denotes a protocol over which the respective resource is accessible  
 Authority shows the owning entity of the resource  
 Path provides the pointer to where the resource is available  
 Query is the query on the resource

Normally, one single URI should point to one single resource, and should remain unchanged over time.

What makes a cool URI? A cool URI is one which does not change.

What sorts of URI change? URIs don't change: people change them [4].

### 2.2.2.3 Hypertext for the Masses

Hypertext is text with references to other Web resources (hyperlinks). In other words, hypertext is a simple way to construct documents that reference other resources. The resources referred to are not limited to other documents, they can be

---

<sup>8</sup><http://www.rfc-editor.org/rfc.html>.

everything that is identified by a URI. There are currently two well known types of hypertext documents: static and dynamic hypertext. Static hypertext refers to static resources, which do not normally change their content or structure, and can be used, for example, to create catalogues or to cross-reference collection of data in documents. On the other hand, the more complex type of hypertext can develop very complex and dynamic systems of linking and cross-referencing.

The most famous implementation of hypertext is, of course, the World Wide Web.

A number of characteristics are considered important in characterizing a *good* hypertext [3]:

**Lots of documents** The hypertext's power mainly comes from the fact that it provides an easy and rapid access to large amounts of data. The data are linked to one another by hyperlinks.

**Lots of links** Every document can and should link to a multiple of other resources. It should present the reader with several links, offering a choice about where to go next. This is a different situation to the one where a document links only to another one, in which case we have a sequence of resources. By allowing multiple links, a hypertext allows the readers to select and follow the one which they are interested in.

**Range of Details** The main difference between a hypertext and a normal text is determined by the range of details it can offer. For normal text, adding more details means simply making the document bigger and more difficult to comprehend. For a hypertext, the details can be added by creating hyperlinks to other documents. In this way, readers can visit the content of the first hypertext, and then, if interested, further browse the details that interest them by following the associated links.

**Correct links** Many Web links are now pointing to unavailable resources. This is, of course, due to the nature of the Web, which changes continuously. A way of avoiding this situation would be to point only to resources under your control, but this will limit the advantages offered by the hypertext and by the Web, in general.

#### 2.2.2.4 Web Architecture

The Web Architecture was analyzed in several publications like [1] or [15]. In summary, the Web Architecture is two-tiered and characterized by a Web client that displays information content and a Web server that transfers information to the client [15].

The most important aspects that need to be considered regarding the Web Architecture are:

- The resources, i.e., what is represented.
- Identifiers, i.e., how the resources are identified over the Web (URI).
- Representation formalism, i.e., how the resources are represented, using what formalism and what representation language.

**Fig. 2.1** Resources, identifiers and representation

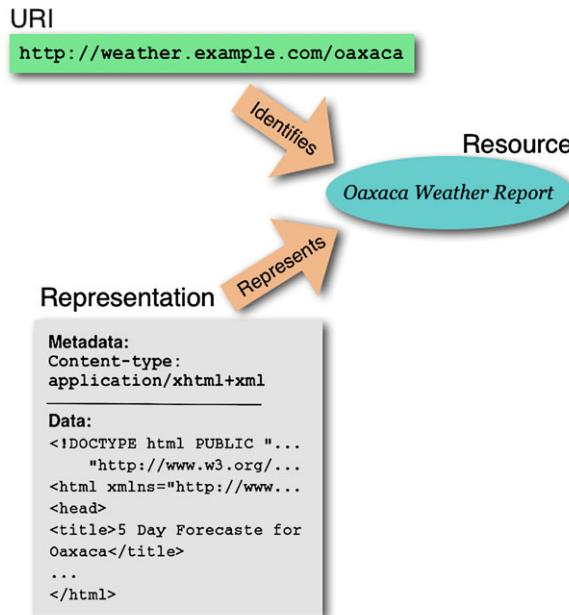


Figure 2.1 identifies the relations between resources, identifiers and representation (from [1]).

Furthermore, the document identifies a number of architectural principles, such as *Orthogonal Specifications*, *Extensibility*, *Error Handling*, and *Protocol-based Interoperability*.

**Orthogonal Specifications** The technologies used for representation, identification and interaction may evolve independently. If the specifications are orthogonal, the changes of one technology should not affect the others.

**Extensibility** The information on the Web, as well as the technologies used for representing and identifying it, is evolving over time. One of the Web architecture principles is to allow for the extensibility without affecting the interoperability.

**Error Handling** Every networked information system is subject to errors, and as such it should implement some form of mechanism for error handling. Typically, there are two important steps for error handling, namely error correction and error recovery. The first one means that an agent modifies one or more conditions so that it appears that the error never occurred (for example, some data may be retransmitted). The second means that error is not repaired, and an agent continues the execution, taking into consideration the appearance of the error.

**Protocol-based Interoperability** This last principle is referring to the fact that the interfaces are defined in terms of protocols. Using a certain protocol implies specifying the syntax, semantics, and sequencing constraints of the messages interchanged.

### 2.2.3 *Web in Society*

Since its creation, the Web has strongly influenced the society, and the society has influenced the Web. The WSRI identified several topics that should be addressed regarding the relationship between the Web and the society. Some of these topics address the economic influence of the Web, like e-Commerce, while the majority of them refer to how social life is influenced by the existence of the Web.

#### 2.2.3.1 E-Commerce

The e-Commerce (Electronic Commerce) refers to the selling and buying of goods over the Internet. These goods can be products (for example, buying a T-shirt online) or services (for example, consultancy or financial services).

Depending on the participants in an e-Commerce activity, we can distinguish between two types:

B2C e-Commerce activities conducted between businesses and consumers (business-to-consumers).

B2B e-Commerce activities conducted between businesses (business-to-business).

E-Commerce is a research area that has been investigated for more than 30 years, although the meaning has changed over the years. Initially e-Commerce was only about the facilitation of commercial transactions by adopting electronic transaction systems. In the late 1970s, however, two technologies were introduced that allowed businesses to electronically send commercial documents like purchase orders or invoices: the Electronic Data Interchange (EDI) and the Electronic Funds Transfer (EFT). Ten years later, new forms of e-Commerce, targeting the banking domain, were already widely in use (like credit cards or ATMs).

Online shopping, the most well-known form of e-Commerce of our time, was introduced in 1979. Two years later (in 1981) the first B2B online activity was recorded, followed in 1984 by the first B2C activity. The e-Commerce activities experienced a major expansion with the invention of the Web and the wide usage of the Internet. This lead to the need for security protocols and regulations, but by the end of 2000 most major companies were offering their products and services online.

#### 2.2.3.2 IP and Copyright

As the Internet is now widely used, and people all over the word are publishing the results of their work online, the Intellectual Property (IP) rights and the copyrights are becoming a most pressing issue.

The IP rights refer to the use of ideas or expressions, and represent temporal monopolies imposed by the state. They are usually imposed on non-rival goods which can be used simultaneously by multiple users (for example, they can be copied by multiple users).

Copyrights impose more strict restriction, as only the owner of the copyright can use that particular product. As in the IP's case, the good can also be an idea or an expression, but it can also include rival goods.

### 2.2.3.3 Co-Evolution of Society and Web

It is a well-known fact that society influences the Web and the content published on the Web, just as the Web influences the society.

Since its invention, the Web has evolved into an infrastructure developed by all its users. Every person that publishes something online is a contributor to the Web's development. Every person creating a personal profile (for example, on Facebook) brings his own contribution to the Web. Furthermore, there are users who publish information on the Web with the purpose of helping others to better understand certain concepts (Wikipedia is a good example of this).

Furthermore, the Internet is affecting our social life. Instead of calling our friends, we now "skype" them. Instead of asking somebody if he or she is available for a meeting, we search for their online calendar. This undoubtedly makes things easier and faster, but are there negative effects that should be considered?

Unfortunately, the answer is yes. On the one hand, we tend to retreat into some kind of "virtual reality". Numerous psychological studies are raising the alarm on the negative effects the Web has over our social life. Teenagers that grow up with the Internet tend to find face-to-face communication difficult, and are unable to deal with real-word situations. Instead, they prefer to retreat into the world of the Web where they can hide under some user names and live in their virtual reality.

Another negative effect is that there is no control over the information that is available. Most of the times when we try to access a so-called safe Web site, we are exposed to advertisement, or even pornographic materials. Adults are able to simply ignore them, but the effects on children and teenagers are devastating and beyond measurement. Furthermore, subscribing to a new email list or social network exposes you to unwanted emails, ranging from announcements that you have just inherited a huge amount of money from an unknown relative, to the more damaging viruses.

### 2.2.3.4 Globalization

The term globalization describes an ongoing process by which regional economies, societies and cultures have become integrated through a globally-spanning network of exchange.

The Web provides a robust support for globalization by allowing seamless communication between users all over the word. The impact of the Web and of the Internet on globalization is both positive and negative.

One of the positive effects, and probably the most noticeable, is in regards to the business sector. The Internet provides a new and challenging environment where the

rules of competition are constantly changing. This creates modernization and innovation on daily basis, and, ultimately, the end user is the one that benefits from this competition in terms of quality of products and services as well as in terms of cost. The way services are sold and delivered is highly influenced by the Internet, a network available all over the world. Due to its distributed nature, the communication between service providers and consumers is ‘at one click distance’, and this allows for client–business relationships which require very little effort from the client.

On the other hand, the negative effects also need to be taken into consideration. One unfortunate truth is that there are huge technological differences between countries, and the Web, in the majority of cases, is making this gap wider rather than reducing it. Developed countries will advance even faster thanks to easy access to novel information, while less developed ones do not have this benefit as they do not have the proper infrastructure to access the Web.

### **2.2.3.5 Collective Intelligence**

One of the main advantages of the Web is that it allows for the building of repositories of knowledge based upon the experience of multiple users. The most relevant example of this is Wikipedia where multiple users can input data based on their own expertise.

However, there are important aspects which hamper the potential benefit of the Web: How to integrate the inputs of multiple users? How to select the correct information out of the inputs received? One possibility, the one implemented by wikipedia, is simply to allow any user to provide input, with the risk of having other users contradicting their opinion. This, however, may lead to pages containing contradictory data, which is not really useful when trying to find information.

A different approach for managing collective intelligence is through a vote-based system. This approach assumes the probability that more people answering a question or providing information are correct than the probability that most people are wrong. A variant of this approach is to ask the users to rate certain information or products (for example, an article or an essay). If most of the people that rated the product are considering it valuable, the product is ranked accordingly. If most of the people rating it are considering the product of poor value, or even containing wrong information, the administrator may even consider removing it.

### **2.2.4 *Operationalizing the Web Science for a World of International Commerce***

As mentioned in the previous sections, the Web influences the society just as the society influences the Web. This section further analyzes several steps that need to be taken in order to operationalize Web Science for a world of international commerce. The previous sections also underlined how the end-user benefits from the

new challenges faced by the service providers, while this section will further focus upon the steps that need to be taken by service providers. WSRI identifies several topics relevant for this domain, such as the adopted business strategy, the regulation and security aspects, the online markets and the relationship between design and evolution.

#### **2.2.4.1 Business Strategy**

In developing a new online business, one must ask oneself a very important question: What makes this business different from others that offer similar products or services? Given the realities of globalization and the easy dissemination of ideas, it is hardly probable that one business identifies itself based on the offer. What makes the difference is, in most cases, the adopted business strategy.

There are many business strategies that can be followed, depending on the product offered, or on the clients targeted. For example, [14] points out that one key advantage of online commerce over ‘normal’ commerce is that a service deployed online is accessible to millions of users. This leads to the identification of the first requirement for a solid business strategy: manner of presentation of the information or the services, as well as easy access and easy payment are a must. Furthermore, [14] prescribes several requirements for an e-Commerce Web site. According to the author, it should provide:

- A ‘Unified Storefront’ that is a well-displayed showcase of products.
- A ‘Shopping Cart System’ where the client places the selected products.
- A system that integrates the online data transactions and e-Commerce payment solutions with the provider’s existing accounting system.
- A fine-tuned Search option to locate just the kind of product the client wants.

Apart from these specific Web features, the business strategy that governs an online business is similar to the business strategy that governs any other business. The only difference is that the client cannot manually browse the products, and as such, the user interface should be good enough to convince him or her to try out a particular online service or buy a product.

#### **2.2.4.2 Regulation and Security**

Regulation and security play an important role for the online commerce. Everybody can sell or buy something online, or even download products for free, but there are certain rules that need to be followed by both the service providers and consumers. Different countries have different rules regarding the e-Commerce, restricting both the type of products commercialized (for example, some countries may prohibit certain type of music) or the authorized sellers. This complements the IPs and copyright previously described, as, for example, you cannot multiply and sell a DVD for which you don’t own the copyright.

For most of us, however, of importance are the regulations protecting the clients. Whenever we perform an online transaction, we provide personal information, like address and credit card details. In this regard, the question of who controls the data cannot be ignored.

In response to this question, several data security standards have been created, such as the PCI security standard. There are also attempts from several governments to at least partially control the personal data. Unfortunately, even if rules and regulations exist, there are more and more accidents involving the loss of confidential, personal data (see, for example, details of how the un-encrypted data of 43,000 children were lost <http://news.zdnet.co.uk/security/>).

A second important aspect regarding security involves avoiding the spyware programs. The spywares are computer programs that extract data from the host computer and transmit it to a predefined address. Although there are several rules that prohibit the distribution of such programs, they still exist and every computer connected to the Internet is a possible target. Unfortunately, spywares do not limit themselves to our home-computers where we can implement whatever security measures we find. In order to avoid loss of data, very strict rules are imposed on the service providers which are manipulating the personal data of their customers.

#### 2.2.4.3 Online Markets

The online marketing, also known as Internet marketing or Web marketing, is the marketing of products or services over the Internet. It ties together creative and technical aspects of the Internet including, design, development, advertising, and sales.

There are several business models associated with the Internet market, such as:

- E-Commerce—as described in the previous sections, e-Commerce concerns the selling of goods directly to the consumers or to other businesses (B2C and B2B).
- Publishing—the sale of advertising.
- Lead-based Web sites—an organization that generates value by acquiring sales leads from its Web site.
- Affiliate marketing—this is the process in which a product or service developed by one person is sold by another active seller for a share of the profits.

A number of advantages of using Internet markets, from both the providers' and consumers' perspective, have been identified. First of all, it is relatively inexpensive, and allows businesses to reach a far wider audience than traditional markets with a reduced advertising cost. In addition, the consumers may buy products and services whenever they want, regardless of their geographical location. Second, the providers can easily make statistics regarding what category of people are buying which products, and use this data to optimize their business model. They can also determine what kind of advertisement will be more appropriate, in order to increase their sales.

### 2.2.5 Analyzing the Web

Analyzing the Web means extracting information about the Web and its users based on the Web site accessed, number of hits or profile of the users. These analyses are usually performed by a Web log analyzer, a program that parses the log of Web servers and determines who accesses it (the type of users), when, or in what way. They can also include measurements referring to a potential audience or voice sharing.

There is a typical set of parameters usually considered by a Web analyzer. These parameters range from information about the users to technical detail about the service. Among other parameters, the typical set includes:

- Number of visits and number of unique visitors.
- Visits duration and last visit.
- Authenticated users, and last authenticated visits.
- Most viewed, entry and exit pages.
- Files' type.
- Search engines, key phrases and keywords used to find the analyzed Web site.
- HTTP errors.

A comparative analysis of different sites may provide an overview of the general interest of the Web users, and may also lead to an educated guess regarding the future of the Web. Elon University conducted a survey on the future of the Internet.<sup>9</sup> The respondents were asked to envision the Internet in 2020; all of them were technology experts and social analysts, and their answers (taken from [2]) include the following quantitative results:

- About 77% said that the mobile computing device (the smartphone) with more significant computing power will be 2020s primary global Internet-connection platform.
- 64% favored the idea that 2020 user interfaces will offer advanced touch, talk and typing options and some added a fourth “T”—think.
- Nearly four out of five respondents (78%) said that the original Internet architecture will not be completely replaced by a next-generation net by 2020.
- Three out of five respondents (60%) disagreed with the idea that legislatures, courts, technology industries, and media companies will exercise effective intellectual property control by 2020.
- A majority—56%—agreed that in 2020 “few lines (will) divide professional from personal time, and that’s OK”.
- 56% said that while Web 2.0 is bringing some people closer, social tolerance will not be heightened by our new connections.
- 45% agreed and 44% disagreed with the notion that the greater transparency of people and institutions afforded by the Internet will heighten individual integrity and forgiveness.

---

<sup>9</sup> Available at <http://www.elon.edu/e-web/predictions/expertsurveys/2008survey/default.xhtml>.

- More than half (55%) agreed that many lives will be touched in 2020 by virtual worlds, mirror worlds, and augmented reality, while 45% disagreed or did not answer the question.

## 2.3 Web 2.0

As a straightforward consequence of having an exponentially growing number of users and of being part of day-to-day activities for personal and business benefits, the Web has already moved in new directions. We are now dealing with a Web that is mainly directed towards supporting collaborations in terms of Web applications, that gives its users the choice to interact and collaborate in a social media dialogue as creators of user-generated content. As illustrations of what Web 2.0 implies, we can consider social-networking sites, blogs, wikis, video-sharing sites, hosted services, Web applications and folksonomies.

The characteristics of Web 2.0 were summarized by [11] using the acronym SLATES (Search, Links, Authoring, Tags, Extensions, Signals):

Search provides support for keyword search for finding information.

Links connect information together.

Authoring every user has the ability to create content in a collaborative manner.

Tags allow the categorization of content by the users by adding tags.

Extensions imply that the Web is both an application platform as well as a document server.

Signals notify the users of changes in the content.

It is, however, important to note that Web 2.0 is based on ideas and principles, and not on particular technologies. Tim O'Reilly considers it as consisting of a so-called gravitational core [12], formed based on core-competencies, which links equally to technologies, ideas, and further development opportunities, as presented in Fig. 2.2.<sup>10</sup>

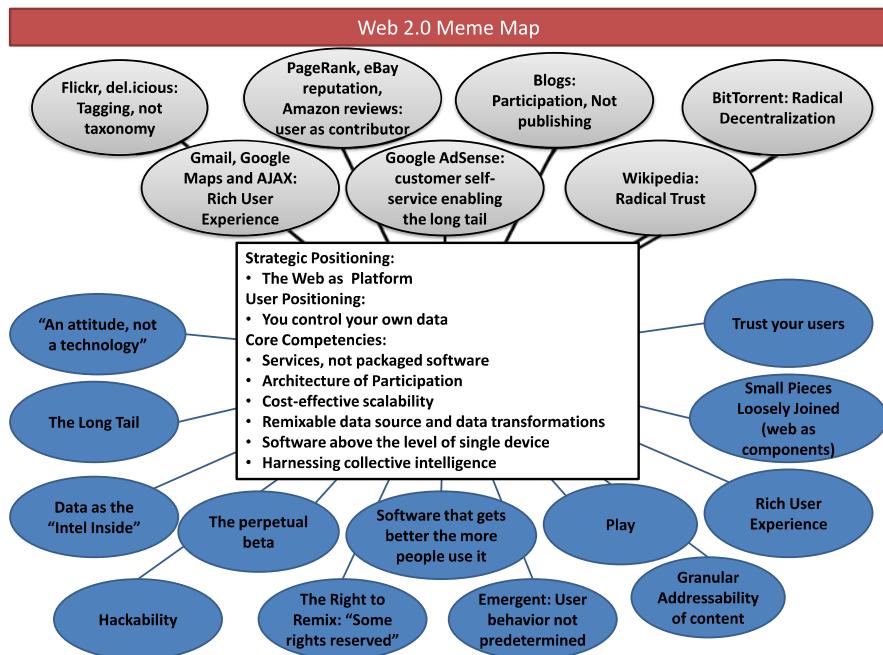
## 2.4 Conclusions

This chapter presents Web Science as the new emerging discipline which aims to investigate the past, the present and the future of the Web. This discipline is still not fully defined: a group of experts have so far only developed the curriculum that should be addressed. The areas range from the history of the Web to the analyses of the Web, and through the experts aim to determine the next phases in the evolution of the Web.

Every topic addressed in the curriculum was described in this chapter. However, the vastness of the domain does not allow for a detailed presentation of each of these

---

<sup>10</sup><http://oreilly.com/pub/a/web2/archive/what-is-web-20.html?page=1>.



**Fig. 2.2** Resources, identifiers and representation

topics. As such, the chapter is merely meant to initiate the reader to Web Science, and to present its basic concepts and ideas.

## References

1. Architecture of the World Wide Web, volume one. <http://www.w3.org/TR/webarch> (2004). <http://www.w3.org/TR/webarch>. Stand: 15.4.2009
2. Anderson, J.Q., Rainie, L.: The future of the Internet III. Tech. rep. (2008)
3. Baccala, B. (ed.): Connected: An Internet Encyclopedia. <http://www.freesoft.org/CIE/index.htm>
4. Berners-Lee, T.: Cool URIs don't change (1998). <http://www.w3.org/Provider/Style/URI.html>
5. Berners-Lee, T., Hall, W., Hendler, J.A., O'Hara, K., Shadbolt, N., Weitzner, D.J.: A framework for Web Science. Foundations and Trends in Web Science **1**(1), 1–130 (2006). doi:10.1561/1800000001
6. Berners-Lee, T., Hall, W., James, H.A.: A Framework for Web Science. Foundations and Trends(r) in Web Science. Now Publishers, Hanover (2006). <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1933019336>
7. Boyera, S.: How to achieve global impact? WWW Foundation (2009)
8. Gillies, J., Cailliau, R.: How the Web Was Born: The Story of the World Wide Web. Oxford University Press, Oxford (2000)
9. Herman, A., Swiss, T.: The World Wide Web and Contemporary Cultural Theory: Magic, Metaphor, Power. Routledge, London (2000). <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0415925029>

10. Lee, B.T., Fischetti, M.: Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor. Harper, San Francisco (1999)
11. McAfee, A.: Enterprise 2.0: the dawn of emergent collaboration. *MIT Sloan Management Review* **47**(3), 21–28 (2006)
12. O'Reilly, T.: What is Web 2.0, Design Patterns and Business Models for the Next Generation of Software. <http://oreilly.com/pub/a/web2/archive/what-is-web-20.html?page=1>
13. Shadbolt, N., Berners-Lee, T.: Web science: Studying the Internet to protect our future. *Scientific American Magazine* (2008)
14. Stewart, B.: E-business strategy vital for e-commerce Web solutions. <http://ebay123.50webs.com/e-business-strategy-vital-for-e-commerce-web-so.html>
15. Thompson, C., Hansen, G.: Current Web architecture. <http://www.objs.com/survey/WebArch.htm>

# Chapter 3

## Service Science

**Abstract** The goal of Service Science as a discipline is to promote service innovation and increase service productivity. It focuses on accommodating multiple clients and multiple providers' needs, as well as on the multi-phase business processes.

### 3.1 Motivation

Service Science represents an interdisciplinary discipline aiming to study, design, and implement service systems,<sup>1</sup> taking into consideration specific arrangements of people and technologies that take actions which provide value for others. It focuses upon innovation in the service sector, and it is proposed as an academic discipline and research area that complements other disciplines contributing to the knowledge of services—in this sense having common pillars with the Web Science previously described in Chap. 2.

Service Science is an interdisciplinary field, focusing on services as systems of interacting parts. A service should be seen as a complex system that involves people, technology, and business. All of this being considered, Service Science is based on a multitude of existing disciplines such as computer science, economics and human resources management.

The most well known initiative around Service Science is **Service Science, Management, and Engineering** by IBM.<sup>2</sup> HP is also showing an increasing interest in Service Science and created the Centre for Systems and Services Sciences,<sup>3</sup> a multi-institution initiative for developing and integrating the sciences that lead to the successful analysis, design and control of complex systems characterized by services requirements. Oracle Corp., one of the biggest software companies (which in 2007 had the third-largest software revenue according to [8]), created an industry consortium, the so-called **Service Research and Innovation Initiative**, focusing on establishing the service science as a key investment area by companies and governments as well as an academic discipline.

---

<sup>1</sup>[http://en.wikipedia.org/wiki/Service\\_Science,\\_Management\\_and\\_Engineering](http://en.wikipedia.org/wiki/Service_Science,_Management_and_Engineering).

<sup>2</sup><http://www.ibm.com/ibm/ideasfromibm/us/compsci/20080728/index.shtml>.

<sup>3</sup><http://www.hpl.hp.com/research/about/model.html>.

The European Service and Science Initiative (NESSI) network has established a Services Sciences and Systems Engineering Working Group<sup>4</sup> which also envisages a multi-disciplinary approach to address some of the challenges incurred in understanding services.

## 3.2 What Is a Service?

In the literature on the subject, we can find many definitions of a service:

Services are economic activities offered by one party to another, most commonly employing time-based performances to bring about desired results in recipients themselves or in objects or other assets for which purchasers have responsibility. In exchange for their money, time, and effort, service customers expect to obtain value from access to goods, labor, professional skills, facilities, networks, and systems; but they do not normally take ownership of any of the physical elements involved [9].

A service is a time-perishable, intangible experience performed for a customer acting in the role of a co-producer [3].

A service is the application of specialized competencies (knowledge and skills), through deeds, processes, and performances for the benefit of another entity or the entity itself.

In order to optimize services, improve access of customers to services and ensure an optimal price/optimal ratio, Service Science studies different aspects related to the services life-cycle.

**Service Science and IT** IT services (or e-Services) are services offered or deployed using IT infrastructure. As such, they are considered a sub-class of the more general services, and they are also the subject of the Service Science discipline. Studying e-Services represents a sub-domain of the service science, in general.

As such, a number of topics can be identified as relevant for IT Service Science:

- Service analysis, design, development and testing.
- Service orchestration, composition and delivery.
- Service innovation.
- Service design approach.
- Service pricing method and economics.
- Service quality measurement.
- Service technologies.
- Service applications.

The **service analysis, design, development and testing** represent the pre-deployment phases necessary for every service, be it an IT service or other. During these phases the market observations and feasibility studies are performed. A complete analysis of what the service and conditions with which it should comply is also

---

<sup>4</sup><http://www.nessi-europe.com/Nessi/Workinggroups/HorizontalWorkingGroups/ServicesSciences/tbid/244/Default.aspx>.

performed during this phase (requirement analysis). Based on the requirements, the service is designed, developed and finally tested before being deployed.

The **service orchestration, composition and delivery** refers to the combination of multiple services in order to achieve a common goal. These phases can again appear for both IT and non-IT services. In the IT domain, the problem, however, escalates to data or process heterogeneity issues, due to machine-to-machine communication.

**Service innovation** refers to several things, such as innovation in service products, service processes or service firms and companies. TEKES<sup>5</sup> provides the following definition for service innovation [7]: **Service innovation is a new or significantly improved service concept that is taken into practice. [...] A service innovation always includes replicable elements that can be identified and systematically reproduced in other cases or environments.**

**Service design approach** refers to the determination of different methodologies for developing different services, depending upon their functionality, deployment environment, users, and so on.

**Service pricing method and economics** addresses several aspects regarding the cost of using the service as well as payment methodology. For every service (both IT and non-IT) during this phase, the relationship between production cost and the price of using the service should be taken into consideration.

**Service quality measurements** refers to the quality of the service offered. There are several methodologies for assessing the quality, as further presented in this chapter.

**Service technologies** refers to the technologies used by the service. For the IT domain, this also refers to the technologies needed to invoke the service.

**Service applications** is primarily a term used for e-Services, and refers to the application domain of a service.

The following sections will analyze each of these topics, providing an overview of the main points of interest in Service Science.

### 3.3 Service Analysis, Design, Development and Testing

The first step in the development of a service, which is intensively studied by Service Science, is the analysis of the service that is to be provided. This phase should answer questions like: Which consumer groups are targeted? What are the customer's expectations? What are the challenges faced by the service? What is different/additional about this service compared with other services possessing a similar functionality?

The service provider has to take into consideration the needs of the customer so that the offered service:

---

<sup>5</sup><http://www.tekes.fi/en/community/Home/351/Home/473/>.

- Answers customer needs.
- Is priced reasonably.
- Has an aesthetic/attractive appearance.

Some other issues regarding e-Services are equally important from the customer perspective. Among these, the service designer has to consider a number of non-functional properties such as robustness, availability, facility of use, and security.

Based on these requirements, the designer provides the initial design of the service. There are several methodologies that can be considered, which will be further described in Sect. 3.6.

The design phase is followed by initial implementation efforts and testing. Usually, a set of pilot consumers are used for testing, and/or several test-cases are developed. Two crucial questions are answered during this phase:

- Does the service meet the business and technical requirements that guided its design and development?
- Is the service working as expected?

There are several testing topics that need to be considered when testing an e-Service, which are the same as the important topics when testing any software product:

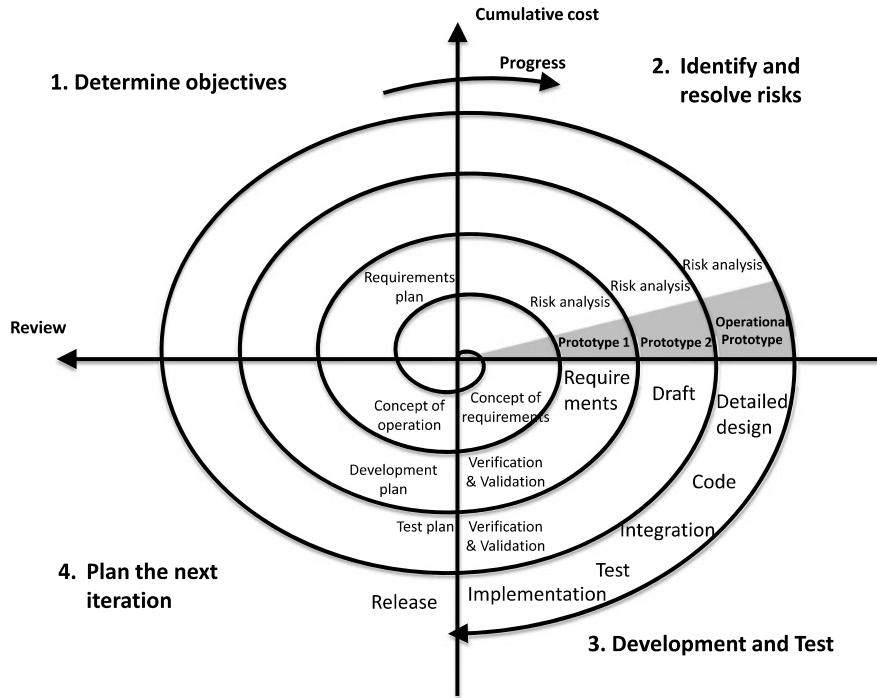
**Scope**—to detect existing failures to be corrected. This does not imply that all the failures can be detected, as testing cannot establish that a product functions properly under all conditions, only that it does not function properly under specific conditions.

**Functional vs non-functional testing**—for e-Services functional and non-functional requirements are equally important. The first set refers to the functionality of a service, while the second addresses questions like: How easy is it to use the service? Is the service robust and secure? (a mandatory feature for every business transaction), and so on.

**Defects and failures**—this particular aspect of the testing refers to the errors in the development processes which, in case of e-Services, can hamper the interaction with the consumer.

**Finding faults early**—as stated in [4], the sooner an error is identified, the cheaper it is to correct it. Testing a service before its release minimizes the risk of finding errors after its deployment, which potentially leads to retracting the service and dealing with dissatisfied customers.

There are also a number of testing approaches that should be taken into consideration, similar to the ones for testing software products: black box, white box or grey box testing. In the black-box case, the testers have absolutely no knowledge of the implementation details; in this case, the most appropriate testing method is by using pilot customers, interested only in what the services offers. The second type of testing, the white box, involves previous knowledge of how the service was developed; in this case, the appropriate testers should have deeper knowledge of the requirements (both functional and non-functional), as well as basic knowledge on the technologies and development methodology used. The third approach represents a combination of the previous two.



**Fig. 3.1** Spiral approach for software development

The results of the testing phase are fed back into the analysis and design phases; the entire process follows a spiral approach [1] which may be executed several times before the service is deployed (see Fig. 3.1).

## 3.4 Service Orchestration, Composition and Delivery

The service orchestration, composition and delivery refers to the deployment of a service, as itself or as part of a composition of services. In the second case, there are a number of heterogeneity issues that need to be taken into consideration, while the composition and orchestration of services represents a challenging research topic in itself.

This chapter further develops service delivery, disregarding its possible interactions with other services, but rather, considering only one e-Service and its customer.

The development and deployment of a service is not an activity that can begin from scratch, but rather reuses an existing infrastructure suitable for specific types of services. This infrastructure, known as **Service Delivery Platform**, is to be further adapted for given types of services. The adaptation in its turn can be used for every service conforming to the same pattern. Service Science studies the existing platforms, and prescribes the minimal set of components that should be implemented.

**Service Creation Environment** provides support for the creation of services, for example, in telecommunication domain, this environment should facilitate the rapid creation of new communication services.

**Execution Environment** must provide support for the actual execution of the service. It should be able to deal with confidential customer data (such as bank account information) as well as service specific data.

**Presence/Location Component** refers to both the virtual and the physical location of both the consumer and the service. In the case of e-Services, this aspect is not as important as in the case of traditional services (i.e., a customer in Germany may use a service located in the UK). What is of primary importance in the e-Services case is the context, i.e., the cultural and background information relevant to the two entities.

**Integration Component** takes into consideration the aspects related to the technologies used by the service as well as those used by the client. The integration should be seamless and transparent, at least from the perspective of clients. Ideally, every integration problem should be determined and solved during the testing phase as part of subsequent development efforts.

Service Oriented Architectures (SOAs) can be used as application integration technology within a service delivery platform. SOAs can also be used to standardize and reuse operational processes across service delivery platforms.

Two of the most widely used service delivery platforms are the HP Delivery Platform<sup>6</sup> and the Oracle Communication Service Delivery.<sup>7</sup>

### 3.5 Service Innovation

Service innovation refers to reusing existing services in order to provide novel, improved ones. It can address several aspects, depending on whether the point of view is from that of the service, the customer or the provider: **innovation in services**—new or improved service products; **innovation in service processes**—new or improved ways of designing and producing services (for example, technological improvements); **innovation in service firms, organizations, and industries**—innovations within the organization providing the service.

Any type of innovation applied should, however, consider all actors. For example, a technological innovation that may reduce the costs of the service from the provider's perspective should also be weighted against the effects it has on the consumer—is it making the service slower, insecure or not robust enough?

In [2], four dimensions of service innovation can be identified:

Innovation of the Service Concept it refers to a service concept that is new to its particular market. In many cases, the innovation of the service concept involves

---

<sup>6</sup><http://www.hp.com/hpinfo/newsroom/press/2007/071107xa.html>.

<sup>7</sup><http://www.oracle.com/industries/communications/oracle-communications-services-delivery.html>.

intangible characteristics of the service (for example, a new type of bank account or information service).

Innovation of the Client Interface particularly important for e-Services, this type of innovation refers to transformations of the interface between the service provider and its customers.

Innovation of the Service Delivery System it relates, but it is not limited to, the client interface. The innovation of the Service Delivery System includes any innovations of the service delivery platform.

The efforts from [2] are continued in [5], which provides a classification of the innovation techniques depending on the phase in the service lifecycle they affect.

Innovations associated with services production refer to innovation areas relevant for e-Services: technology, organization of internal processes, production and organization of industry.

Innovations associated with services product refer to innovation in the nature of the product and its features.

Innovations associated with services consumption address the delivery of the product, the role of the consumer as well as the organization of consumption.

Innovations associated with services markets refer to innovation in the organization of markets and in the marketing of the service.

From the service science perspective, innovation of services is one of the most important study topics. By innovation, new improved services can be created out of existing ones, at a minimum cost (in terms of both time and money). In addition, different innovations performed on the same service may lead to the creation of multiple services. For the IT services, innovation implies transformation in the service's components.

## 3.6 Service Design Approach

The service design is the activity of planning and organizing people, infrastructure, communication and material components of a service. The goal of this activity is not only to create the initial service, but also to improve its quality, the interaction between service providers and customers and the experience of customers.

The Service Design methodologies are used to organize people, infrastructure, communication and material components. The design of a service may involve re-organizing the activities performed by the service provider as well as the redesign of interfaces and interactions that customers use to contact the service provider (for e-Services this is referring to a Web site, user interface, blog, etc.).

Service design requires methods and tools to control all the elements of the design process, including the time and the interaction between actors. Three directions of service design are identified in [6]:

Actors identification of the actors involved in the definition of the service

Scenarios definition of possible service scenarios

Service representation of the service, using techniques that illustrate all the components of the service

The design is based on a set of functional and non-functional requirements for the service. The first step is the identification of the actors, i.e., which entities will play which role(s) in using the service (such as service provider, requester, and so on). The second step is for each of these actors to strictly identify the type of interactions they will have with the service, the expectations of those actors, as well as the inputs they may provide to the service. This is part of the scenarios definition, which should contain enough details for the service developers. These scenarios should cover a broad area of use cases, of service usage possibilities. The last step is the actual design of the service in terms of UML diagrams or any other modeling language.

### 3.7 Service Pricing Method and Economics

The service pricing and economics phase should strongly reflect the cost of developing and providing the service, as well as the benefits for the customer using the service.

There are two well known pricing methods for e-Services, namely Flat Fee Services Pricing and Rate Based Services Pricing, each of them having advantages and disadvantages from the perspective of both provider and customer.

**Flat Fee Services Pricing** is a fixed price for a specific service. This flat rate should be payed by the customer at regular intervals, established by a contract between the provider and the consumer, and the customer can use the service as much as she/he want for the same rate. A number of e-Services traditionally offer flat free service pricing, such as Internet service providers or telecommunication providers.

**Rate Based Services Pricing** is a variable rate, based on the principle “pay as you use”. There are three variants of rate based service pricing:

- Hourly Invoiced Services—billed purely based on the amount of time spent using them. A common example is again the Internet usage.
- Retained Services Packages—available in a variety of sizes to the customers needs. Usually the hourly rate is smaller as the size of the package increases.
- Monthly Retained Services—allow the customers to pay a monthly fee to receive a pre-set amount of time to be used within a month.

### 3.8 Service Quality Measurement

Quality of Service (QoS) refers to non-functional properties of a services which can be measured based on different parameters. Depending upon the type of service, different quality measures are taken into consideration. The most common ones, usually considered for e-Services, are listed below:

Availability qualitative aspect reflecting whether the service is ready for immediate use. It represents the probability that a service is available. A high probability means the that service is available most of the times (and 100% availability implies that the service is always available).

Accessibility qualitative aspect which can be represented as a probability, calculated as the success rate or chance of a successful service instantiation at a given point in time.

Integrity measures the success of the service in maintaining correct and consistent data.

Performance qualitative aspect measured in terms of throughput (the number of service requests served at a given time period) and latency (time between sending a request and receiving the response).

Reliability the degree to which it is capable of maintaining the service and service quality. For example, the number of failures per month or year represents a measure of reliability of an e-Service.

Regulatory measures how the service complies with a set of given rules or standards.

Security mainly refers to confidentiality and non-repudiation. Security aspects deal with, for example, in the case of e-Services, the authentication of the parties involved, encryption of messages, and access control.

## 3.9 Service Technologies

Service technology refers to the set of technologies needed for developing and providing a service. Considering the diversity of service types, there is no strict and well defined set of technologies that services should use. For e-Services, however, there are a number of key components that should be considered.

The first one is, as previously mentioned, the service delivery platform. However, the provider of a service may choose to develop it from scratch, not reusing any of the existing platforms.

One mandatory component for e-Services is a user interface by which the customer may access the service. This user interface should be as simple and user-friendly as possible, and should preferably assist the customer in using the service.

Another component of the e-Services should be available for allowing the customer to pay for the service he or she uses. Whether this is done by credit card, PayPal, standing orders, or any other payment options, it should not be restricted in any way. The only requirements is that a payment agreement be reached between the service provider and consumers.

Other components that may be part of the technologies used by the service are integrator, composer and orchestrator. These components are needed in case the service has a complex structure and involves cooperation with multiple services or multiple data formats.

### 3.10 Service Application

Service application refers to the ability to create and deploy a service. Like in the case of service technology, the diversity of services prohibits the existence of a strict set of service application methodologies or tool support. There are, however, a number of service application frameworks which are suitable for e-Services. Among these, the Microsoft Service Application Framework<sup>8</sup> is one of the most commonly used. It has a number of features that help the service providers in developing new services, such as:

- **Improved development experience**—it allows service developers to truly focus on the business logic of their applications, by providing support for writing code to configure a server that runs Internet Information Services (IIS), installing a Secure Sockets Layer (SSL) certificate, creating a virtual directory, managing credentials for a pool of application users, managing and caching distributed settings, tracking and load balancing endpoints, as well as several backup and restore tasks.
- **Improved integration with SharePoint**—these services plug their management UI into the SharePoint Service Management page, thereby providing a common experience for administrators. In addition, service developers can build their own administrative pages to manage their services and host them using SharePoint Central Administration.
- **Integration with Windows Communication Foundation**—the Windows Communication Foundation (WCF) service model addresses communication between client and service.
- **Round Robin Load Balancing**—SharePoint provides a round-robin load balancer implementation, which can be enhanced or replaced by third-party developers as necessary.
- **Claims Based Identity**—an external identity system is configured to give the application everything it needs to know about the user with each request, along with cryptographic assurance that the identity data received comes from a trusted source.
- **Backup and Restore**—the Service Application Framework allows for easy integration with the SharePoint backup/restore tool.

### 3.11 Conclusions

This chapter presents Service Science as the discipline which studies services, and, in particular, e-Services, relevant to the IT community. Service Science does not prescribe particular technologies or tools to be used for developing the services, nor does it restrict the ways a service is provided, but rather studies the common characteristics and the most well known technologies.

---

<sup>8</sup>[http://msdn.microsoft.com/en-us/library/ee537258\(office.14\).aspx](http://msdn.microsoft.com/en-us/library/ee537258(office.14).aspx).

Service Science is an interdisciplinary discipline, focusing on services as systems of interacting parts. A service should be seen as a complex system that involves people, technology, and business. This being considered, service science is based on a multitude of existing disciplines, such as computer science, economics and human resources management.

This chapter focuses on the IT services, i.e., services offered or deployed using IT infrastructure, and analyzes a number of topics relevant for this domain, from service analysis, design, development and testing (Sect. 3.3), to service applications (Sect. 3.10).

## References

1. Boehm, B.W.: A spiral model of software development and enhancement. Computer **21**(5), 61–72 (1988)
2. den Hertog, P.: Knowledge-intensive business services as co-producers of innovation. International Journal of Innovation Management **4**(4) (2000)
3. Fitzsimmons, M.J., Fitzsimmons, J.A.: Service Management: Operations, Strategy, Information Technology. McGraw Hill, New York (1997). 2nd Revised edition, October 1997
4. Kaner, C., Bach, J., Pettichord, B.: Lessons learned in software testing: a context-driven approach (2001)
5. Miles, I.: Patterns of innovation in service industries. IBM Systems Journal **47**(1), 115–128 (2008)
6. Morelli, N.: Developing new product service systems (pss): methodologies and operational tools. Journal of Cleaner Production **14**(17), 1495–1501 (2006)
7. Tekes: Finnish funding agency for technology and innovation. Tekes website (2007)
8. Verberne, B.: Software Top 100: Highlights. Software Top 100 website (2008)
9. Wirtz, J., Lovelock, C.H.: Services Marketing: People, Technology, Strategy. Prentice Hall, New York (2006)



# Chapter 4

## Web Services

**Abstract** One paradigm that changed academic and industrial perspective on how to realize Enterprise Application Integration (EAI) and build distributed applications is Web services. Their emergence is the result of an evolutionary process that started with the introduction of Remote Procedure Call (RPC) and continued with TP Monitors, Object Brokers and Message-Oriented Middleware. The driving concept and innovation behind Web services is the abstraction of functionalities as **services** hiding the technical realization in terms of hardware and software. In the end, it is the functionality that counts for a client and not the technicalities that realize this functionality. In this chapter, we provide an introduction to Web services. We look at the most important technologies from the Web services technologies landscape. First, the theoretical aspects of these technologies are described including their concepts, definitions and mechanisms. Second, each of the technologies are exemplified by means of practical examples. Finally, we discuss extensions of Web services and provide a set of exercises that should be solved by the reader interested in fixing the concepts and notion presented in this chapter.

### 4.1 Motivation

The Web as we know it is an enormous success. Currently, there are trillions of pages available on the Web, and this number is rapidly increasing. The success of the Web is due, in principle, to the underlying open, scalable approach and technologies. A closer look at the current Web shows that it is rather a **static** more than a **dynamic** system. It is static in the sense that the applications available on the Web cannot be easily reused in different contexts and scenarios. The Web was actually designed for interactions among applications and humans, enabling information sharing. It provides basic support for business-to-customer (B2C) e-Commerce, and very limited support for business-to-business (B2B) integration and Enterprise Application Integration (EAI). Interoperability is a major challenge that the Web and Web technologies do not support and do not address directly. In this context, the idea of Web services was introduced as the technology that enables interoperability and integration of business functionalities.

The ultimate vision is that a program tasked to achieve a result can use Web services as support for its computation or processing. The program can discover Web

services and invoke them fully automated. Hence, it becomes a service requester. If the Web services have a cost attached, the program knows when to search for cheaper services as well as all possible payment methods. Furthermore, the program might be able to mediate any differences between its specific needs and a Web service that almost fits.

Web services connect computers and devices with each other using the Internet to exchange data and combine data in new ways. They can be defined as software objects that can be assembled over the Internet using standard protocols to perform functions or execute business processes. The key to Web services is on-the-fly software creation through the use of loosely coupled, reusable software components. This has fundamental implications in both technical and business terms. Software can be delivered and paid for as fluid streams of services, as opposed to packaged products. It is possible to achieve automatic, ad-hoc interoperability between systems to accomplish business tasks. Business services can be completely decentralized and distributed over the Internet and accessed by a wide variety of communications devices. Businesses can be released from the burden of complex, slow and expensive software integration and focus instead on the value of their offerings and mission critical tasks. The Internet will then become a global common platform where organizations and individuals communicate with each other to carry out various commercial activities and provide value-added services. The barriers to providing new offerings and entering new markets will be lowered to enable access for small and medium-sized enterprises. The dynamic enterprise and dynamic value chains become achievable and may even be mandatory in a competitive environment.

The design of enterprise information systems has gone through several changes in the recent years. In order to respond to the requirements of enterprises for flexibility and dynamism, the traditional monolithic applications have become substituted by smaller, compressible units of functionality known as services. Information systems must be re-tailored to fit this paradigm, with new applications developed as services, and legacy systems must be updated in order to expose service interfaces. The drive is towards a design of information systems which adopt paradigms of Service Oriented Computing (SOC) together with the SOA implementation and relevant Web service technologies.

#### ***4.1.1 Service Oriented Computing (SOC)***

Service Oriented Computing (SOC) is a new computing paradigm that utilizes services as the fundamental elements for the development of rapid, low-cost and easy integrable enterprise applications [14, 16]. One of the main goals of SOC is to enable the development of networks of the integrated and collaborative applications, regardless of both the platform where applications or services run (e.g., the operating system) and programming languages used to develop them. In this paradigm, services are autonomous, self-describing and platform-independent computational

entities, which provide a uniform and ubiquitous access to information for a wide range of computing devices (such as desktops, PDAs, or cellular phones) and software programs across different platforms. Any piece of code and any application component deployed on a system can be reused and transformed into a network-available service. Since these services are based on the service orientation paradigm, they can then be easily described, published, discovered, and dynamically assembled for developing distributed and interoperable systems.

The main goal of SOC is to facilitate the integration of newly built and legacy applications, which exist both within and across organizational boundaries. SOC must overcome and resolve heterogeneous conflicts due to different platforms, programming languages, security firewalls, etc. The basic idea behind this orientation is to allow applications which were developed independently (using different languages, technologies, or platforms) to be exposed as services and then interconnect them exploiting the Web infrastructure with respective standards such as HTTP, XML, SOAP, WSDL and even some complex service orchestration standards like BPEL.

#### **4.1.2 Service Oriented Architecture (SOA)**

The service oriented paradigm of computation can be abstractly implemented by the system architecture called Service Oriented Architecture (SOA) [5, 8]. The purpose of this architecture is to address the requirements of loosely coupled, standard-based, and protocol-independent distributed computing, mapping enterprise information systems isomorphically to the overall business process flow [15]. This attempt is considered to be the latest development of a long series of advancements in software engineering addressing the reuse of software components.

Historically, the first major step of this evolution has been the development of the concept of **function**. Using functions, a program is decomposed into smaller sub-programs and writing code is focused on the idea of the application programming interface (API). An API, practically, represents the contract to which a software component has to commit. The second major step was the development of the concept of **object**. An object is a basic building block which contains both data and functions within a single encapsulated unit. With the object-oriented paradigm, the notions of classes, inheritance and polymorphism are introduced. In this way, classes can be viewed as a lattice. The concept of a **service** becomes the next evolution step introduced with the advent of SOC and its SOA implementation.

The following Fig. 4.1 illustrates the **Web Services Programming Model** which consists of three components: service consumers, service providers and service registrars. Ignoring the detailed techniques for the connection of the three components, this model also represents the SOA basic model. A service registrar (also called a service broker) acts as an intermediary between the provider and the consumer. A service provider simply publishes the service. A service consumer tries to find services using the registrar; if it finds the desired service, it can set up a contract with the provider in order to consume this service.

**Fig. 4.1** Web services programming model



The fundamental logical view of services in SOA is based on the division of service description (frequently called interface) and service implementation [15]. Service interface defines the identity of a service and its invocation logistics. Service implementation implements the internal operations that the service is designated to do. Based on this division, service providers and services consumers are loosely coupled. Furthermore, the services can be significantly reused and adapted according to certain requirements. Because service interfaces are platform independent and implementation is transparent for the service consumers, a client from any communication device using any computational platform, operating system and any programming language should be capable of using the service. The two facets of the service are distinct; they are designed and maintained as distinct items, though their existence is highly interrelated.

Based on the service autonomy and the clean separation of service interfaces from internal implementation, SOA provides a more flexible architecture that unifies business processes by modularizing large applications into services. Furthermore, enterprise-wide or even cross-enterprise applications can be realized by means of services development, integration, and adaptation.

The remainder of this chapter focuses on the technical solutions available to implement the Service Oriented Architecture and Web services vision.

## 4.2 Technical Solution

Web services are “modular, self-describing, self-contained applications that are accessible over the Internet” [11]. They can be seen as an effort to build a distributed platform on top of the WWW, by adding a new level of functionality to the current Web. A set of Web service technologies, most of them standardized by various standardization bodies (e.g., OASIS, W3C), provide the means for communication and integration of Web services-based applications. Three of these technologies form the foundation of most Web services based systems, namely: (i) WSDL [9]—an XML description language for Web services, (ii) SOAP [19]—an XML based message format to exchange arbitrary XML data between services and clients, and (iii)

UDDI [4]—data model and API for Web service registries. Before we investigate the actual technologies that are currently used to implement Web services, let us first define the notion of a service.

### 4.2.1 Defining Web Services

As has been pointed out in [17], the notion of a service is semantically overloaded. Different communities have a different understanding of what a service is. For example, in the business community a service is seen as a business activity that often results in intangible outcomes or benefits [1], while in Computer Science the terms service and Web service are often regarded as interchangeable to name a software entity accessible over the Internet. In the remainder of this section, we define the high level terminology used in this chapter. More precisely, we make a clear distinction between the notions of a **service** and a **Web service**.

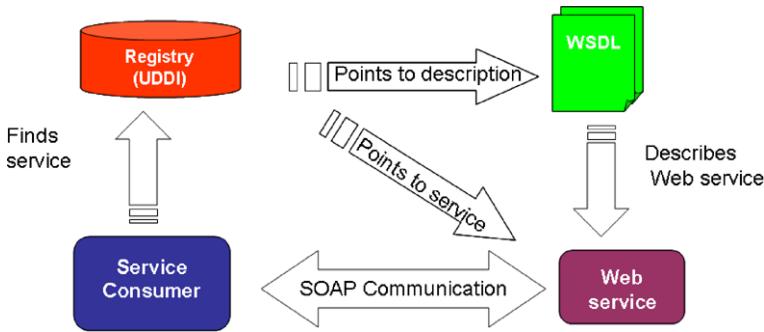
**Service** A service is defined in [17] as a provision of value in some domain. As an example, let us consider a user who wants to book a flight from Vienna to Innsbruck on a given date. The appropriate service provides a flight ticket with the specified constraints. Such provision is independent of how the supplier and the provider interact, i.e., it does not matter whether the requester goes to an airline tickets office or uses the airline Web site to book the flight. We understand the term service in the sense given in [17], that is, as a provision of value.

**Web Service** A Web service is defined in [17] as a computational entity accessible over the Internet (using Web service standards and protocols). Re-analyzing the 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. As such, the Web service is an electronic means to request a service, but not the service itself. We understand the term Web service in this sense, i.e., as a means to request a service, described using agreed standards.

#### 4.2.1.1 Service Properties

There are three different types of properties that can be identified for services and Web services, namely: (i) **functional**, (ii) **behavioral** and (iii) **non-functional** properties. They can be briefly defined as follows:

- The **functional** description contains the formal specification of what a service can do.
- The **behavioral** description concerns how the functionality of the service can be achieved in terms of the interaction with the service and the functionality required from other Web services.



**Fig. 4.2** Web services usage process

- The **non-functional descriptions** capture constraints over the previous two types of properties [10].

For example, considering a train booking service, invoking its functionality (booking a train ticket) might be constrained by using a secure connection (**security** as non-functional property) or by actually performing the invocation of the services at a certain point in time (**temporal availability** as a non-functional property).

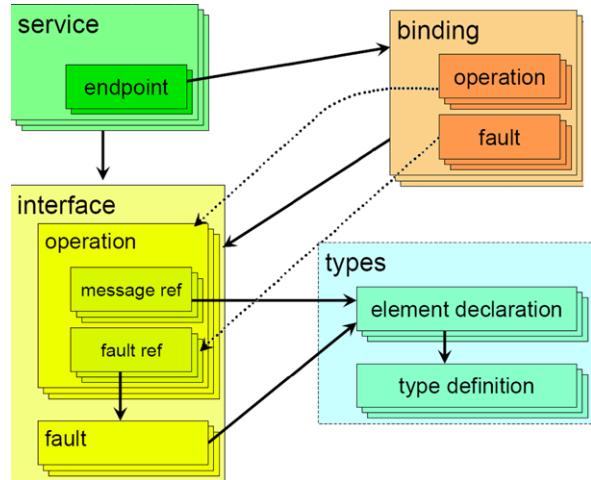
### 4.2.2 Web Service Technologies

The overall Web services usage process illustrated at the technical level is depicted in Fig. 4.2.

The service providers register and advertise their services with an UDDI registry by providing details such as contact information of the service provider, pointers to services and their WSDL descriptions, etc. Once the registry is populated with services related information, service consumers, which in most cases are developers of service-oriented applications, could pragmatically access the registry. UDDI queries are issued by the service consumers and potentially the UDDI registry positively answers these queries. If a relevant service is found, the registry provides the service consumer with the information needed to access the service. However, the service consumer still needs to figure out how to actually access and invoke the service, information usually available in the WSDL description of the service. This introduces a big overhead and generates problems when distributed applications are developed. Keeping human intervention (i.e., software developer intervention) at minimum and automating service related tasks could overcome the previous mentioned problem.

#### 4.2.2.1 Web Service Description Language (WSDL)

WSDL [9] is an XML based description language for Web services. A service is defined as a collection of network **endpoints** or **ports**. Two levels of service description are identified in WSDL. First, an abstract interface level provides details about

**Fig. 4.3** WSDL elements

types, operations and interfaces of the service. Second, concrete network protocols and endpoints specify how and where the service can be accessed. The remainder of this section describes in detail each of the WSDL elements, using the WSDL version 2.0 (see Listing 4.1).

At the abstract interface level, **types** are defined in terms of XML Schemas. Types are used to define **messages** which represent data being exchanged between service providers and service invokers. Messages are grouped in **operations**, each operation consisting of one or more messages. Operations specify as well the order in which the messages are sent or received, following predefined message exchange patterns (i.e., “In-Out”, “Out-Only”, “Robust-In-Only”, “In-Optional-Out”). Finally, the **interface** groups together more operations. The graphical representation of the WSDL structure is represented in Fig. 4.3.

In WSDL, an operation represents a simple exchange of messages that follows a specific message exchange pattern (MEP). The simplest of MEPs, “In-Only”, allows a single application message to be sent to the service, and “Out-Only” symmetrically allows a single message to be sent by the service to its client. Somewhat more useful is the “Robust-In-Only” MEP, that also allows a single incoming application message but in case there is a problem with it, the service may reply with a fault message. Perhaps the most common MEP is “In-Out”, which allows an incoming application message followed either by an outgoing application message or an outgoing fault message. Finally, an interesting MEP commonly used in messaging systems is “In-Optional-Out” where a single incoming application message may (but does not have to) be followed either by a fault outgoing message or by a normal outgoing message, which in turn may be followed by an incoming fault message (i.e., the client may indicate to the service a problem with its reply).

In order to communicate with a Web service described by an abstract interface, a client must know how the XML messages are serialized on the network and where exactly they should be sent. In WSDL, on-the-wire message serialization is de-

```

<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
    xmlns:tns="http://www.example.com/wsdl20sample"
    xmlns:whttp="http://www.w3.org/ns/wsdl/http"
    xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
    targetNamespace="http://www.example.com/wsdl20sample">

<!-- Abstract types --&gt;
&lt;types&gt;
    &lt;xss:schema xmlns="http://www.example.com/wsdl20sample"
        xmlns:xss="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.example.com/wsdl20sample"&gt;
        &lt;xss:element name="request"&gt; ... &lt;/xss:element&gt;
        &lt;xss:element name="response"&gt; ... &lt;/xss:element&gt;
    &lt;/xss:schema&gt;
&lt;/types&gt;

<!-- Abstract interfaces --&gt;
&lt;interface name="Interface1"&gt;
    &lt;fault name="Error1" element="tns:response"/&gt;
    &lt;operation name="Opp1" pattern="http://www.w3.org/ns/wsdl/in-out"&gt;
        &lt;input messageLabel="Msg1" element="tns:request"/&gt;
        &lt;output messageLabel="Msg2" element="tns:response"/&gt;
    &lt;/operation&gt;
&lt;/interface&gt;

<!-- Concrete Binding Over HTTP --&gt;
&lt;binding name="HttpBinding" interface="tns:Interface1"
    type="http://www.w3.org/ns/wsdl/http"&gt;
    &lt;operation ref="tns:Opp1" whttp:method="GET"/&gt;
&lt;/binding&gt;

<!-- Concrete Binding with SOAP--&gt;
&lt;binding name="SoapBinding" interface="tns:Interface1"
    type="http://www.w3.org/ns/wsdl/soap"
    wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
    wsoap:mePDefault="http://www.w3.org/2003/05/soap/meP/request-response"&gt;
    &lt;operation ref="tns:Opp1" /&gt;
&lt;/binding&gt;

<!-- Web service offering endpoints for both the bindings--&gt;
&lt;service name="Service1" interface="tns:Interface1"&gt;
    &lt;endpoint name="HttpEndpoint" binding="tns:HttpBinding" address="http://www.example.com/rest"/&gt;
    &lt;endpoint name="SoapEndpoint" binding="tns:SoapBinding" address="http://www.example.com/soap"/&gt;
&lt;/service&gt;
&lt;/description&gt;
</pre>

```

**Listing 4.1** WSDL 2.0 Skeleton example

scribed in a binding and then a service construct enumerates a number of concrete endpoint addresses.

A binding generally follows the structure of an interface and specifies the necessary serialization details. The WSDL specification contains two predefined binding specifications, one for SOAP (over HTTP) and one for plain HTTP. These bindings specify how an abstract XML message is embedded inside a SOAP message envelope or in an HTTP message, and how the message exchange patterns are realized in SOAP or HTTP. Due to extensive use of defaults, simple bindings only need to specify very few parameters, as in the example below. A notable exception to defaulting in binding are faults, as in SOAP every fault must have a so-called fault code

with two main options, Sender or Receiver, indicating who has a problem. There is no reasonable default possible for the fault code.

Bindings seldom need to contain details specific to a single actual physical service, therefore in many cases they can be as reusable as interfaces, and equivalent services by different providers only need to specify the different endpoints, sharing the interface and binding descriptions.

The **service** construct in WSDL represents a single physical Web service that implements a single interface. The Web service can be accessible at multiple endpoints, each potentially with a different binding, for example, one endpoint using an optimized messaging protocol with no data encryption for the secure environment of an intranet and a second endpoint using SOAP over HTTP for access from the Internet.

#### 4.2.2.2 Simple Object Access Protocol (SOAP)

SOAP [19] is an XML based message format to exchange arbitrary XML data. It uses mostly HTTP/HTTPS as an underlying communication protocol. SOAP was designed to address transport and messaging for large distributed environments. It defines how to organize information using XML so that it can be exchanged between machines. However, SOAP is not aware of the semantics of the messages being transmitted. From a communication point of view, SOAP is a stateless and one-way communication protocol. The communication between two parties needs to be encoded in SOAP and any complex communication pattern between the parties involved needs to be implemented by the underlying system. There are three important aspects that define together the SOAP protocol: (i) the message structure, (ii) the processing model, and (iii) the protocol bindings.

**Message Structure** At the top level a SOAP message consists of an **envelope** element where the application encodes whatever information needs to be sent. The envelope element defines the start and the end of the message. An **envelope** element contains a body that is mandatory and a **header** element that is optional. The **header** element contains any optional attributes of the message, either at an intermediary point or at the ultimate end point. The **body** element contains call and response information, in other words, the XML data comprising the message being sent or received. Additionally, a SOAP message that is generated on the service side might contain a **fault** element in case of failure. Listing 4.2 contains a SOAP skeleton showing the relation between the elements described above.

To illustrate how the SOAP communication between two parties is encoded, let us consider a simple example in which a client application wants to interact with a weather service. The service is expecting as input the city name and in return it provides the current temperature in that city. The SOAP request is illustrated in Listing 4.3 and the associated response in case no problems occurred during the invocation is illustrated in Listing 4.4.

```
<?xml version=1.0?> <soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
  ...
  </soap:Header>

  <soap:Body>
  ...
  <soap:Fault>
  ...
  </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

**Listing 4.2** SOAP skeleton

POST /InStock HTTP/1.1 Host: www.example.org Content-Type: application/soap+xml; charset=utf-8 Content-Length: nnn

```
<?xml version="1.0"?> <soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/weather">
    <m:GetTemperature>
      <m:CityName>Innsbruck</m:CityName>
    </m:GetTemperature>
  </soap:Body>

</soap:Envelope>
```

**Listing 4.3** SOAP request

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

```
<?xml version="1.0"?> <soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/services/weather">
    <m:GetTemperatureResponse>
      <m:Temperature>10</m:Temperature>
    </m:GetTemperatureResponse>
  </soap:Body>

</soap:Envelope>
```

**Listing 4.4** SOAP replay

**Processing Model** The SOAP processing model defines the rules for processing a SOAP message. More precisely, a distributed processing model is proposed, in which the SOAP message originates at an initial SOAP sender and is sent to an ultimate SOAP receiver via zero or more SOAP intermediaries. The header information of a SOAP message plays an important role in the overall SOAP processing model,

defining what part of the message is allowed to be processed by whom. Parties involved in the processing model are called SOAP nodes. SOAP defines three roles for SOAP nodes, namely: **none**, **next** and **ultimateReceiver**. Within a SOAP message any header block that was assigned to a **none** role should not be processed by any SOAP nodes. A header block that was assigned to a **next** role should be processed by each SOAP node along the path, excluding the initial node. A header block that was assigned to a **ultimateReceiver** role must be processed only by the recipient of the message. The processing of the header block by various SOAP nodes along the path is done by each individual node depending on its role, and it is mandatory if the **mustUnderstand** flag is on. If this flag is not set, SOAP nodes which cannot understand the header blocks targeted to them could simply ignore the header blocks.

**Protocol Bindings** To specify how the SOAP message is going to be transported over the network, the **protocol binding** part of SOAP is used. SOAP enables exchange of SOAP messages using a variety of underlying protocols. A common used protocol for SOAP binding is HTTP, but other protocols can be used as well, e.g., SMTP. Three transport aspects are described as part of a SOAP binding, namely: (i) **addressing** that specifies what an endpoint address looks like, (ii) **serialization** that specifies how to put XML message in on-the-wire bits and bytes, and (iii) **connection** that specifies how to send the bits to an endpoint.

#### 4.2.2.3 Universal Description, Discovery and Integration Protocol (UDDI)

UDDI [4], on the one hand, provides a mechanism for service providers to register their Web services and, on the other hand, for service consumers to find them. UDDI defines a standard **data model** and **API** for Web services registries. The **data model** is used to store into registry information about businesses or service providers, service offered by these businesses as well as technical information about these services in form of WSDL descriptions. There are five data structure types defined as part of the data model:

- The **businessEntity** describes a business or service provider entity in terms of name, description and type of business performed. For defining the business type, UDDI offers the possibility to attach business categories from predefined classification hierarchies such as UNSPSC<sup>1</sup> using the **categoryBag** element and a set of identifiers using the **identifierBag** element. A **businessEntity** is usually providing one or more services that are linked with the **businessEntity** via the **businessService** data structure.

Listing 4.5 exemplifies how a **businessEntity** is represented for an imaginary **Weather Company**. Such a company provides, for example, a weather service with which a client interacts by exchanging SOAP messages as exemplified in Sect. 4.2.2.2.

---

<sup>1</sup><http://www.unspsc.org/>.

```
<businessEntity businessKey="uuid:EAE4D5A8-CFF4-4501-55AD-E702126866A0"
    operator="http://www.example.org/weather"
    authorizedName="John Doe">
    <name>Weather Company</name>
    <description>
        We provide you with weather information
    </description>
    <contacts>
        <contact useType="general info">
            <description>General Information</description>
            <personName>John Doe</personName>
            <phone> +(43) 444-1111 </phone>
            <email>jdoe@weather.com</email>
        </contact>
    </contacts>
    <businessServices>
    ...
    </businessServices>
</businessEntity>
```

**Listing 4.5** UDDI businessEntity—XML representation

```
<businessServices>
    <businessService serviceKey="uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
        businessKey="uuid:BA744ED0-3EE7-91D3-DC23-11E035229C64">
        <name>WeatherWebService</name>
        <description>A weather Web service</description>
        <bindingTemplates>
        ...
        </bindingTemplates>
        <categoryBag />
    </businessService>
</businessServices>
```

**Listing 4.6** UDDI businessService—XML representation

- The **businessService** contains information about the service provided by a businessEntity, including name, description and information on how to access the services as part of the **bindingTemplate** element.

An example of **businessService** element describing the Weather service introduced above is presented in Listing 4.6

- The **bindingTemplate** provides detailed information about a service including an entry-point address for accessing the service.

An example of **bindingTemplate** element associated with the Weather service is presented in Listing 4.7.

- The **tModel** is the part of the data model intensively used for search. It collects the information uniquely identifying the service specification.

An example of **tModel** element associated with the Weather service is presented in Listing 4.8.

An UDDI registry can be accessed using the **UDDI Registry API**. This API provides programmatic means to publish services, access and query the registry. The methods of the API are grouped in six sets of APIs:

```

<bindingTemplate
    bindingKey="uuid:D6F1B765-BDB3-4837-828D-8284301E5A2A"
    serviceKey="uuid:C0E6D5A8-3E16-4f01-99DA-035229685A40">
    <description xml:lang="en">SOAP binding for
weather service</description>
    <accessPoint
URLType="http">http://www.example.org/weather:80/soap</accessPoint>
    <tModelInstanceDetails>
        <tModelInstanceInfo
tModelKey="uuid:EB1B645F-CF2F-491f-811A-4868705F5904" />
    </tModelInstanceDetails>
</bindingTemplate>
</bindingTemplates>

```

**Listing 4.7** UDDI bindingTemplate—XML representation

```

<tModel tModelKey="uuid:xyz987...">
    operator="http://www.example.org/weather"
    authorizedName="John Doe">
    <name>WeatherInterface Port Type</name>
    <description>
        An interface for the Weather service
    </description>
    <overviewDoc>
        <overviewURL>
            http://www.example.org/services/weather.wsdl
        </overviewURL>
    </overviewDoc>
</tModel>

```

**Listing 4.8** UDDI tModel—XML representation

- The **UDDI Inquiry API** contains methods to query the registry (e.g., **find\_service**, **find\_business**, **find\_binding**, etc.) and methods to retrieve detailed information about specific entities (e.g., **get\_serviceDetail**, **get\_bindingDetail**, etc.)
- The **UDDI Publishers API** contains methods to add, modify and delete entities in the registry. Methods such as **save\_business**, **save\_service**, **save\_tModel**, etc. are used to add or modify UDDI registry entities, while methods such as **delete\_business**, **delete\_service**, **delete\_tModel**, etc. are used to remove entities from the registry.
- The **UDDI Security API** contains methods to get and discard authentication tokens that are used in communication with the registry (**get\_authToken**, **discard\_authToken**, etc.)
- The **UDDI Custody and Ownership Transfer API** contains methods that allow the transfer of ownership and structures from one publisher to another. Methods such as **get\_trnasferToken** and **transfer\_custody** are part of this API.
- The **UDDI Subscription API** contains methods that enable the monitoring of changes in a registry. The following methods are part of the subscription API:**get\_subscriptionResult**, **get\_subscriptions**, **save\_subscriptions**, etc.
- The **UDDI Replication API** contains methods that enable the replication of information between registries, so that different registries can be kept synchronized.

#### 4.2.2.4 Other Web Services Standards

The rest of this section provides an overview of some of the other Web services related standards, namely **WS-Addressing** for communicating addressing information between services, **WS-Policy** for specifying policies regarding quality of service, both of a user and a service side, **WS-Security** for describing security constraints and requirements when interacting with services and **ebXML** as an alternative registry solution.

#### 4.2.2.5 WS-Addressing

WS-Addressing [6] is a specification of transport-neutral mechanisms that allow Web services to communicate addressing information. The approach addresses some of the drawbacks of SOAP, such as the lack of a standard way to specify where a message is going, how to return a response, or where to report an error. WS-Addressing introduces two important constructs: **endpoint references** and **message information headers**. The endpoint reference provides pointers to where the Web service messages can be targeted. It consists of an address (a URI) and a set of reference properties that uniquely identify the target instance. The message information header conveys end-to-end message characteristics including addressing for source and destination endpoints as well as message identity. One problem tackled by WS-Addressing is how to direct requests coming from a specific client to a specific instance of the service handling the interaction with that client.

An example of an **endpoint reference** in WS-Addressing is provided in Listing 4.9.

In this example, the **wsa:Address** element contains the address that represents the location of the service and identifies it. The following element **wsa:Reference Properties** contains the resource to which the service refers and the **wsa:Reference Parameters** specifies additional information about the resource(s) used by the service that facilitates the interaction.

The example in Listing 4.10 demonstrates the usage of **message information header** as part of the SOAP message following WS-Addressing specification.

```
<wsa:EndpointReference>
    <wsa:Address>
        http://www.example.com/services/weather
    </wsa:Address>
    <wsa:ReferenceProperties>
        <tns:resourceID>InnsbruckData42</tns:resourceID>
    </wsa:ReferenceProperties>
    <wsa:ReferenceParameters>
        <tns:expires>32000</tns:expires>
    </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

**Listing 4.9** WS-Addressing endpoint reference

```

<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <S:Header>
    <wsa:MessageID>
      uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
    </wsa:MessageID>
    <wsa:From>
      <wsa:Address>http://examples.org/client1</wsa:Address>
    </wsa:From>
    <wsa:ReplyTo>
      <wsa:Address>http://examples.org/client1</wsa:Address>
    </wsa:ReplyTo>
    <wsa:To>http://examples.org/weather</wsa:To>
    <wsa:Action>http://examples/weather/GetTemperature</wsa:Action>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>

```

**Listing 4.10** WS-Addressing example

#### 4.2.2.6 WS-Policy

The Web Services Policy Framework (WS-Policy) [2] is a specification that allows advertising service providers to describe policies of Web services (on security, quality of service, etc.) and service consumers to specify their policy requirements.

The key feature of WS-Policy is its flexible and extensible grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web services-based system. In this context, WS-Policy defines a base set of constructs that can be used and extended by other Web services specifications to describe a broad range of service requirements and capabilities.

At the core of WS-Policy stands the concept of policy expression which contains domain-specific, Web service policy information. WS-Policy further defines a core set of constructs to indicate how choices and/or combinations of domain-specific policy assertions apply in a Web services environment. In the following, we shortly describe the concepts for a policy expression and the way they are related. A policy expression is an XML Infoset representation of a policy.

At the abstract level, a policy is a potentially empty collection of policy alternatives. Alternatives are not ordered. They may differ significantly in terms of the behaviors they indicate, as well as conversely, they may be very similar. In either case, the value or suitability of an alternative is generally a function of the semantics of assertions within the alternative, and it is therefore beyond the scope of WS-Policy.

**Policy Alternative** A policy alternative is a logical construct which represents a potentially empty collection of policy assertions. The vocabulary of a policy alternative is the set of all assertion types within the alternative. The vocabulary of a policy is the set of all assertion types used in all the policy alternatives in the policy.

**Policy Assertion** A policy assertion identifies a behavior that is a requirement (or capability) of a policy subject. Assertions indicate domain-specific (e.g., security,

```
<wsp:Policy>
<wsp:ExactlyOne>
<wsse:SecurityToken>
<wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
</wsse:SecurityToken>
<wsse:SecurityToken>
<wsse:TokenType>wsse:X509v3</wsse:TokenType>
</wsse:SecurityToken>
</wsp:ExactlyOne>
</wsp:Policy>
```

**Listing 4.11** WS-Policy example

transactions) semantics and are expected to be defined in separate, domain-specific specifications. Assertions are strongly typed by the domain authors that define them.

Additionally, WS-Policy defines the following terms:

- **Policy Assertion Type**

A policy assertion type represents a class of policy assertions and implies a schema for the assertion and assertion-specific semantics.

- **Policy Assertion Parameter**

A policy assertion parameter qualifies the behavior indicated by a policy assertion.

- **Policy Vocabulary**

The policy vocabulary of a policy is the set of all policy assertion types used in the policy.

- **Policy Subject**

A policy subject is an entity (e.g., an endpoint, message, resource, interaction) with which a policy can be associated.

- **Policy Scope**

A policy scope is a collection of policy subjects to which a policy may apply.

- **Policy Attachment**

A policy attachment is a mechanism for associating policy with one or more policy scopes.

Listing 4.11 provides an example of a policy requiring the use of Kerberos or X509 security tokens.

Syntactically, WS-Policy offers several ways of expressing policies, but all are equivalent to a single set of alternatives containing only atomic assertions. WS-Policy Framework is accompanied by further specifications:

- **WS Policy Attachment** [3] specifies two general-purpose mechanisms for associating Policies with one or more policy subjects.
- **Web Services Security Policy Language** [12] specifies policy assertions regarding the use of Web Services Security [13].
- **WS Policy Assertions** [7] defines several general-use policy assertions.

WS-Policy is meant for expressing the capabilities, requirements, and general characteristics of entities in an XML Web services-based system [2]. WSDL de-

scriptions of Web services are an example of the entities to which policies are attached even while they would fit the above definition; this split indicates that not all capabilities, requirements and general characteristics qualify as policies.

#### 4.2.2.7 WS-Security

As mentioned in Sect. 4.1, Web services technologies were introduced to support interoperability and integration at enterprises level. Solutions at this level are not completed without considering features such as security of transactions. To address the security requirements in the Web services world, a new standard, called WS-Security, was introduced in 2004 by OASIS.<sup>2</sup> WS-Security [13] is a communications protocol providing the means for applying security to Web services. The protocol contains specifications on how integrity and confidentiality can be enforced on Web services messaging.

More precisely, WS-Security is an extension of SOAP to support secure communication between Web services. From a implementation point of view, WS-Security defines a SOAP header block, called **Security**, where security information should be included. As part of the same header block, WS-Security constructs can be used to specify which of the SOAP nodes is supposed to process the security information.

WS-Security defines a set of security tokens that can be attached to messages. The set of security tokens include: (i) **UsernameToken** used to provide user name and password information, (ii) **BinarySecurityToken** used to include security certificates and keys, and (iii) **XML Token** used to include rules and processes for specific XML-based security token formats. An example of **WS-Security** specification, taken from the WS-Security specification document, is provided in Listing 4.12.

In the above example, the **wsse:Security** element contains security information for the intended recipient. A custom token is associated with the message. The **ds:Signature** element is used to specify a digital signature that ensures the integrity of the signed elements. Within the **ds:Signature** element one can specify what is being signed and the type of canonicalization being used. The elements being signed and the way to digest them are specified using the **ds:Reference**. In this case only, the message body is signed. The signature value is declared in the **ds:SignatureValue** element. The **ds:KeyInfo** element provides information, partial or complete, as to where to find the security token associated with this signature. The rest of the message contains the body of the message.

#### 4.2.2.8 ebXML

An ebXML registry<sup>3</sup> is an information system that securely manages any content type and the standardized metadata that describes it. It provides a set of services

---

<sup>2</sup><http://www.oasis-open.org/home/index.php>.

<sup>3</sup>[www.ebxml.org](http://www.ebxml.org).

```

<?xml version="1.0" encoding="utf-8"?>
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
  xmlns:ds="...">
  <S11:Header>
    <wsse:Security xmlns:wsse="...">
      <xxx:CustomToken wsu:Id="MyID"
        xmlns:xxx="http://fabrikam123/token">
        FHUIORv...
      </xxx:CustomToken>
      <ds:Signature>
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
          <ds:Reference URI="#MsgBody">
            <ds:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>LyLsFOPi4wPU...</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>Djbchm5gK...</ds:SignatureValue>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#MyID"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
      </wsse:Security>
    </S11:Header>
    <S11:Body wsu:Id="MsgBody">
      <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payments">
        QQQ
      </tru:StockSymbol>
    </S11:Body>
  </S11:Envelope>

```

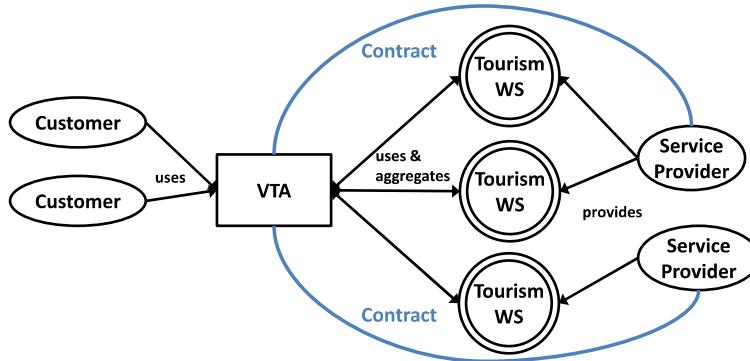
**Listing 4.12** WS-Security example

that enable sharing of content and metadata between organizational entities in a federated environment. An ebXML Registry may be deployed within an application server, a Web server, or some other service container. The registry may be available to clients as a public, semi-public, or private Web site. The ebXML Registry thus provides a stable store where submitted information is made persistent. Such information is used to facilitate business to business relationships and transactions.

In this context, submitted content for an ebXML Registry includes, but is not limited to, XML schema and documents, process descriptions, ebXML core components, context descriptions, UML models, information about organizations, and software components.

The ebXML Registry Information Model (RIM) specification defines the types of metadata and content that can be stored in an ebXML Registry. The companion document ebXML Registry Services and Protocols (RS) defines the services provided by an ebXML Registry and the protocols used by clients of the registry to interact with these services.

According to the RIM specification, an ebXML Registry is capable of storing any type of electronic content such as XML documents, text documents, images,



**Fig. 4.4** Virtual travel agency scenario overview

sound and video. Instances of such content are referred to as a Repository Items. Repository Items are stored in a content repository provided by the ebXML Registry. In addition to the Repository Items, an ebXML Registry is also capable of storing standardized metadata that may be used to further describe the Repository Items. Instances of such metadata are referred to as a Registry Objects, or one of its sub-types. Registry Objects are stored in the registry provided by the ebXML Registry.

Although a few industry groups have endorsed ebXML Registry, the vendor community has essentially ignored this standard.

### 4.3 Illustration by a Larger Example

To illustrate how the Web service technologies described in the previous sections can be used to implement a distributed application following SOA principles, let us consider the following scenario based on the scenario introduced in [18].

Imagine a “Virtual Traveling Agency” (VTA) which is an end user service providing eTourism services to customers. These services can cover all kinds of information services concerned with tourism—from information about events and sights in an area to services that support online booking of flights, hotels, car rentals, etc. Such VTAs are already in existence; currently those portals are accessible via html sites. The partners of the VTAs are integrated via conventional B2B integration. In a nutshell, the VTA application provides the following functionality: the customers use the VTA service as the entry point for their requests. These end-user services are aggregated by the VTA by invoking and combining Web services offered by several tourism service providers. To facilitate this, there can be a so-called “umbrella” contract between the service providers and the VTA for regulating usage and allowance of the Web services. Figure 4.4 gives an overview of Virtual Travel Agency scenario.

As depicted in Fig. 4.4, there are three actors involved in the overall scenario: (i) the **customer** representing the end-user that requests a service provider by the VTA, (ii) the **tourism service provider** representing a commercial company that provides specific tourism services, and (iii) the **VTA** representing the entity that provides tourism services to customers by aggregating the separate services provided by the single Service Providers.

Let us consider an imaginary hotel company, called “The Blue Hotel”, that allows its clients to check rooms availability using the “BlueHotelService” Web service. The WSDL representation of this service is available in Listing 4.13.

Once the “BlueHotelService” is set-up, the Blue Hotel company needs to register the service with an UDDI registry. In this way, potential clients can query the registry and find relevant information about the service. As described in Sect. 4.2.2.3, the UDDI registry contains a set of data structures dedicated to “BlueHotelService”, once this service is registered with the registry.

The **businessEntity** for the “Blue Hotel” company is depicted in Listing 4.14.

Listing 4.15 contains the **businessService** description for the “BlueHotelService” available in the UDDI registry.

The **bindingTemplate** element associated with the Blue Hotel service is presented in Listing 4.16.

The **tModel** element associated with the Blue Hotel service is presented in Listing 4.17.

In the following listings, we show how the client can interact with the “BlueHotelService”. A potential client for this scenario is the VTA application. The communication between the VTA application and the Blue Hotel Service is done using SOAP messages. A simple interaction between VTA and the Blue Hotel Service includes issuing a **request** for checking the availability of a specific type of room for a given time period interval and a **response** generated by the Blue Hotel Service containing information about the room type, rate type and the actual rate for the room. Listing 4.18 contains the request of the VTA application and Listing 4.19 contains the response of the Blue Hotel Service.

## 4.4 Summary

This chapter discussed the basics of Web service technologies. It started first by introducing the paradigms that are gaining more and more momentum in the academia and industry, namely Service Oriented Computing and Service Oriented Architectures. They are centered around the notion of service and are a new way of designing distributed applications, supporting interoperability and integration on a large scale. We have first introduced definitions of concepts and notions used in the Web service domain. One important distinction we made was between a **service** that represents a provision of value in some domain, and a **Web service**, that is, a computation entity accessible over the Internet. We further discussed in details the three core pillar technologies for Web services: WSDL, SOAP and UDDI.

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://www.bluehotel.com/wsdl/BlueHotelService"
    xmlns:tns= "http://www.bluehotel.com/wsdl/BlueHotelService"
    xmlns:bhns = "http://www.bluehotel.com/schemas/BlueHotelService"
    xmlns:wssoap= "http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsdlx= "http://www.w3.org/ns/wsdl-extensions">

</documentation>
This document describes the Blue Hotel Web service.
</documentation>

<types>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.bluehotel.com/schemas/BlueHotelService"
    xmlns="http://www.bluehotel.com/schemas/BlueHotelService">

<xs:element name="checkAvailability" type="tCheckAvailability"/>
<xs:complexType>
<xs:sequence>
<xs:element name="checkInDate" type="xs:date"/>
<xs:element name="checkOutDate" type="xs:date"/>
<xs:element name="roomType" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="checkAvailabilityResponse"
type="tCheckAvailabilityResponse"/>
<xs:complexType>
<xs:sequence>
<xs:element name="roomType" type="xs:string"/>
<xs:element name="rateType" type="xs:string"/>
<xs:element name="rate" type="xs:double"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="invalidDataError" type="xs:string"/>

</xs:schema>
</types>

<interface name = "BlueServiceInterface" >

<fault name = "invalidDataFault"
      element = "bhns:invalidDataError"/>
```

**Listing 4.13** WSDL description of BlueHotelService

As presented in the chapter, the language used to describe Web services is WSDL, including the type systems used in interface definitions, the messages exchanges by a service, the operations it exposes, and the binding to the actual protocol to be used for invocation. SOAP, on the other hand, can be seen as the communication/messaging protocol for Web services. SOAP specifies a standard way to encode messages transmitted between Web services. It can use different transport protocols, such as HTTP or SMTP, that are specified as bindings. Finally, we have discussed UDDI as the discovery approach for Web services. UDDI registry

```

<operation name="opCheckAvailability"
    pattern="http://www.w3.org/ns/wsdl/in-out"
    style="http://www.w3.org/ns/wsdl/style/iri"
    wsdlx:safe = "true">
    <input messageLabel="In"
        element="bhns:checkAvailability" />
    <output messageLabel="Out"
        element="bhns:checkAvailabilityResponse" />
    <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
</operation>

</interface>

<binding name="BlueServiceSOAPBinding"
    interface="tns:BlueServiceInterface"
    type="http://www.w3.org/ns/wsdl/soap"
    wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP"/>

    <fault ref="tns:invalidDataFault"
        wsoap:code="soap:Sender"/>

    <operation ref="tns:opCheckAvailability"
        wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>

</binding>

<service name="BlueService"
    interface="tns:BlueServiceInterface">

    <endpoint name="reservationEndpoint"
        binding="tns:BlueServiceSOAPBinding"
        address = "http://www.bluehotel.com/BlueService"/>
</service>
</description>

```

**Listing 4.13** (Continued)

```

<businessEntity businessKey="uuid:3AE4D5A8-CFF4-4501-55AD-E702126866A0"
    operator="http://www.bluehotel.com/"
    authorizedName="George Blue">
    <name>Blue Hotel</name>
    <description>
        The Blue Hotel, one of the best in town
    </description>
    <contacts>
        <contact useType="general info">
            <description>General Information</description>
            <personName>George Blue</personName>
            <phone> +(43) 555-222 </phone>
            <email>gblue@bluehotel.com</email>
        </contact>
    </contacts>
    <businessServices>
        ...
    </businessServices>
</businessEntity>

```

**Listing 4.14** Blue Hotel UDDI businessEntity—XML representation

```
<businessServices>
<businessService serviceKey="uuid:36F1B765-BDB3-4837-828D-8284301E5A2A"
    businessKey="uuid:3A744ED0-3EE7-91D3-DC23-11E035229C64">
    <name>BlueHotelService</name>
    <description>The Blue Hotel Web service</description>
    <bindingTemplates>
        ...
    </bindingTemplates>
    <categoryBag />
</businessService>
</businessServices>
```

**Listing 4.15** Blue Hotel UDDI businessService—XML representation

```
<bindingTemplate
    bindingKey="uuid:36F1B765-BDB3-4837-828D-8284301E5A2A"
    serviceKey="uuid:40E6D5A8-3E16-4f01-99DA-035229685A40">
    <description xml:lang="en">SOAP binding for
    Blue Hotel service</description>
    <accessPoint
        URLType="http">http://www.bluehotel.com/hotel:80/soap</accessPoint>
        <tModelInstanceDetails>
            <tModelInstanceInfo
                tModelKey="uuid:AE1B645F-CF2F-491f-811A-4868705F5904" />
            </tModelInstanceDetails>
        </bindingTemplate>
    </bindingTemplates>
```

**Listing 4.16** Blue Hotel UDDI bindingTemplate—XML representation

```
<tModel tModelKey="uddi:WE1B6Q5F-CF2F-491f-811A-4868705F5904"
    operator="http://www.bluehotel.com/hotel"
    authorizedName="George Blue">
    <name>BlueHotelInterface Port Type</name>
    <description>
        An interface for the Blue Hotel service
    </description>
    <overviewDoc>
        <overviewURL>
            http://www.bluehotel.com/services/BlueHotelService.wsdl
        </overviewURL>
    </overviewDoc>
</tModel>
```

**Listing 4.17** Blue Hotel UDDI tModel—XML representation

defines the structure of a Web services registry and APIs to access, publish, query and modify registry entities.

The last part of the chapter presented other important Web services standards that are used to describe and handle various other aspects of Web services, including WS-Addressing for communicating addressing information between services, WS-Policy for specifying policies regarding quality of service, both on the user and service side, WS-Security for describing security constraints and requirements when interacting with services and ebXML as an alternative registry solution.

```
POST /InStock HTTP/1.1 Host: www.example.org Content-Type: application/soap+xml; charset=utf-8 Content-Length: nnn

<?xml version="1.0"?> <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope" soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:bhns= "http://www.bluehotel.com/wsdl/BlueHotelService">
<bhns:checkAvailability>
<bhns:checkInDate>2009-03-24</bhns:checkInDate>
<bhns:checkOutDate>2009-03-30</bhns:checkOutDate>
<bhns:roomType>Single</bhns:roomType>
</bhns:checkAvailability>
</soap:Body>

</soap:Envelope>
```

**Listing 4.18** SOAP request

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

<?xml version="1.0"?> <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope" soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:bhns= "http://www.bluehotel.com/wsdl/BlueHotelService">
<bhns:checkAvailabilityResponse>
<bhns:roomType>Single</bhns:roomType>
<bhns:rateType>Discount</bhns:rateType>
<bhns:rate>150.50</bhns:rate>
</bhns:checkAvailabilityResponse>
</soap:Body>

</soap:Envelope>
```

**Listing 4.19** SOAP replay

## 4.5 Exercises

**Exercise 1** Complete the WSDL file from Listing 4.20 in a way that the operations it provides correspond to the following procedure calls:

- float add(float x, int y)
- boolean lessThan(int a, int b)
- String trim(String s)

The service is available at the following endpoint address: <http://www.swsbook.com/theService>

**Exercise 2** Given the WSDL file depicted in Listing 4.21 (<http://www.webservicex.net/stockquote.asmx?WSDL>) for a Stock Quote Web service, design a SOAP request and a SOAP response for the GetQuote operation in the StockQuoteSoap port type.

**Exercise 3** Consider a bus transportation company as one company that provides tourism services as part of the VTA scenario introduced in Sect. 4.3. Design a

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CompService" targetNamespace="urn:Foo"
  xmlns:tns="urn:Foo"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"/>
  <types>
    <message name="|\_\_|">
      <part name="parameter_1" type="xsd:integer"/>
      <part name="parameter_2" type="_____"/>
    </message>
    <message name="_____">
      <part name="par_1" type="_____"/>
      <part name="par_2" type="_____"/>
    </message>
    <message name="_____">
      <part name="echoRequestIn" type="_____"/>
    </message>
    <message name="_____">
      <part name="resultdiv" type="_____"/>
    </message>
    <message name="_____">
      <part name="resultmul" type="_____"/>
    </message>
    <message name="_____">
      <part name="resulthecho" type="_____"/>
    </message>
<portType name="CompIF">
  <operation name="div">
    parameterOrder="|\_\_|"
      <input message="_____"/>
      <output message="_____"/>
    </operation>
    <operation name="mul" parameterOrder="_____">
      <input message="_____"/>
      <output message="_____"/>
    </operation>
    <operation name="echo">
      <input message="_____"/>
      <output message="_____"/>
    </operation>
  </portType>
<binding name="ComplFBinding" type="tns:CompIF">
  <operation name="mul">
    <input>
      <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
use="encoded" namespace="urn:Foo"/>
      </input>
      <output>
        <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
use="encoded" namespace="urn:Foo"/>
        </output>
        <soap:operation soapAction="" />
      </operation>
      {!dots} <!-- other bindings skipped for space reasons --> {!dots}
    </operation>
    <operation>
      {!dots}
    </operation>
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"

```

**Listing 4.20** Web service—WSDL

```

style="rpc"/>
</binding>
<service name="CompService">
  <port name="ComplIFPort" binding="tns:ComplIFBinding">
<soap:address location="_____"/>
  </port>
</service>
</definitions>
```

**Listing 4.20** (Continued)

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns="http://www.webserviceX.NET"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  targetNamespace="http://www.webserviceX.NET"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
      targetNamespace="http://www.webserviceX.NET">
      <s:element name="GetQuote">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="symbol" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetQuoteResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetQuoteResult"
              type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
  </wsdl:types>
  <wsdl:message name="GetQuoteSoapIn">
    <wsdl:part name="parameters" element="tns:GetQuote" />
  </wsdl:message>
  <wsdl:message name="GetQuoteSoapOut">
    <wsdl:part name="parameters" element="tns:GetQuoteResponse" />
  </wsdl:message>
  <wsdl:message name="GetQuoteHttpGetIn">
    <wsdl:part name="symbol" type="s:string" />
  </wsdl:message>
  <wsdl:message name="GetQuoteHttpGetOut">
    <wsdl:part name="Body" element="tns:string" />
  </wsdl:message>
  <wsdl:message name="GetQuoteHttpPostIn">
    <wsdl:part name="symbol" type="s:string" />
  </wsdl:message>
  <wsdl:message name="GetQuoteHttpPostOut">
    <wsdl:part name="Body" element="tns:string" />
  </wsdl:message>
```

**Listing 4.21** Stock Quote Web service—WSDL

```
<wsdl:portType name="StockQuoteSoap">
  <wsdl:operation name="GetQuote">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get Stock quote for a
company
Symbol</documentation>
  <wsdl:input message="tns:GetQuoteSoapIn" />
  <wsdl:output message="tns:GetQuoteSoapOut" />
</wsdl:operation>
</wsdl:portType>
<wsdl:portType name="StockQuoteHttpGet">
  <wsdl:operation name="GetQuote">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get Stock quote for a
company
Symbol</documentation>
  <wsdl:input message="tns:GetQuoteHttpGetIn" />
  <wsdl:output message="tns:GetQuoteHttpGetOut" />
</wsdl:operation>
</wsdl:portType>
<wsdl:portType name="StockQuoteHttpPost">
  <wsdl:operation name="GetQuote">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get Stock quote for a
company
Symbol</documentation>
  <wsdl:input message="tns:GetQuoteHttpPostIn" />
  <wsdl:output message="tns:GetQuoteHttpPostOut" />
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="StockQuoteSoap" type="tns:StockQuoteSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
  <wsdl:operation name="GetQuote">
    <soap:operation soapAction="http://www.webserviceX.NET/GetQuote"
style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="StockQuoteHttpGet" type="tns:StockQuoteHttpGet">
  <http:binding verb="GET" />
  <wsdl:operation name="GetQuote">
    <http:operation location="/GetQuote" />
    <wsdl:input>
      <http:urlEncoded />
    </wsdl:input>
    <wsdl:output>
      <mime:mimeXml part="Body" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="StockQuoteHttpPost" type="tns:StockQuoteHttpPost">
  <http:binding verb="POST" />
  <wsdl:operation name="GetQuote">
    <http:operation location="/GetQuote" />
    <wsdl:input>
      <mime:content type="application/x-www-form-urlencoded" />
    </wsdl:input>
    <wsdl:output>
```

**Listing 4.21** (Continued)

```

<mime:mimeXml part="Body" />
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="StockQuote">
  <wsdl:port name="StockQuoteSoap" binding="tns:StockQuoteSoap">
    <soap:address location="http://www.webservicex.net/stockquote.asmx" />
  </wsdl:port>
  <wsdl:port name="StockQuoteHttpGet" binding="tns:StockQuoteHttpGet">
    <http:address location="http://www.webservicex.net/stockquote.asmx" />
  </wsdl:port>
  <wsdl:port name="StockQuoteHttpPost" binding="tns:StockQuoteHttpPost">
    <http:address location="http://www.webservicex.net/stockquote.asmx" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

**Listing 4.21** (Continued)

WSDL Web service for one of the services this company provides and explain all its elements. The service must contain at least two operations of different type. The binding to SOAP should be used. All the messages must have complex types.

**Exercise 4** Create the XML representations of at least two of the UDDI elements businessEntity, businessService, bindingTemplate, and tModel for a bus transportation company and its bus service. Additionally, explain why UDDI can be described as being both a name service and a directory service, mentioning the types of enquiries that can be made.

## References

1. Baida, Z., Gordijn, J., Omelichenko, B., Akkermans, H.: A shared service terminology for online service provisioning. In: Proceedings of the Sixth International Conference on Electronic Commerce (ICEC04) (2004)
2. Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Langworthy, D., Nadalin, A., Nagaratnam, N., Prafullchandra, H., von Riegen, C., Roth, D., Schlimmer, J., Sharp, C., Shewchuk, J., Vedamuthu, A., Yalcinalp, U., Orchard, D.: Web services policy 1.2—framework (ws-policy). Technical report (2006). <http://www.w3.org/Submission/WS-Policy/>
3. Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Maruyama, H., Nadalin, A., Orchard, D., Prafullchandra, H., von Riegen, C., Roth, D., Schlimmer, J., Sharp, C., Shewchuk, J., Vedamuthu, A., Yalcinalp, U.: Web services policy 1.2 attachment (ws-policyattachment) (2006). <http://www.w3.org/Submission/WS-PolicyAttachment/>
4. Bellwood, T.: UDDI version 2.04 API specification. (2002). Available from <http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm>
5. Booth, D., Hugo Haas, W., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web service architecture. W3C, Working Notes (2003/2004). <http://www.w3.org/TR/ws-arch/>
6. Box, D., Christensen, E., Curbera, F., Ferguson, D., Frey, J., Hadley, M., Kaler, C., Langworthy, D., Leymann, F., Lovering, B., Lucco, S., Millet, S., Mukhi, N., Nottingham,

- M., Orchard, D., Shewchuk, J., Sindambiwe, E., Storey, T., Weerawarana, S., Winkler, S.: Web services addressing (ws-addressing) (2004). <http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/>
7. Box, D., Hondo, M., Kaler, C., Maruyama, H., Nadalin, A., Nagaratnam, N., Patrick, P., von Riegen, C., Shewchuk, J.: Web services policy assertions language (ws-policyassertions) (2003). <http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policyassertions.asp>
  8. Burbeck, S.: The tao of e-business services (2000). <http://www-128.ibm.com/developerworks/library/ws-tao/>
  9. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web services description language (WSDL) 1.1 (2001). <http://www.w3.org/TR/wsdl>
  10. Chung, L.: Non-functional requirements for information systems design. In: Proceedings of the 3rd International Conference on Advanced Information Systems Engineering—CAiSE'91, 7–11 April 1991, Trondheim, Norway. LNCS, pp. 5–30. Springer, Berlin (1991)
  11. Curbera, F., Nagy, W.A., Weerawarana, S.: Web services: why and how. In: OOPSLA 2001 Workshop on Object-Oriented Web Services, ACM, New York (2001)
  12. Della-Libera, G., Gudgin, M., Hallam-Baker, P., Hondo, M., Granqvist, H., Kaler, C., Maruyama, H., McIntosh, M., Nadalin, A., Nagaratnam, N., Philpott, R., Prafullchandra, H., Shewchuk, J., Walter, D., Zolfonoon, R.: Web services security policy language (ws-security-policy) (2002). <http://msdn.microsoft.com/ws/2002/12/ws-security-policy/>
  13. Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R.: Web services security: soap message security 1.0 (2004). <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-messagessecurity-1.0.pdf>
  14. Papazoglou, M.P.: Service-oriented computing: concepts, characteristics and directions. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003 (2003)
  15. Papazoglou, M.P., van den Heuvel, W.-J.: Service oriented architectures: approaches, technologies and research issues. VLDB Journal (2007)
  16. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing research roadmap (2006)
  17. Preist, C.: A conceptual architecture for semantic web services. In: Proceedings of the International Semantic Web Conference 2004 (ISWC 2004) (2004)
  18. Stollberg, M., Lausen, H., Keller, U., Zaremba, M., Haller, A., Fensel, D., Kifer, M.: D3.3 v0.1 WSMO use case virtual travel agency. Working Draft D3.3v0.1, WSMO (2004). Available from <http://www.wsmo.org/2004/d3/d3.3/v0.1/>
  19. W3C: SOAP Version 1.2 Part 0: Primer (2003)



# Chapter 5

## Web2.0 and RESTful Services

**Abstract** In this chapter, we discuss technologies that can be used to develop and deploy services according to the core principles of the World Wide Web. These new type of services are known as RESTful services. First, we introduce the theoretical aspects of these technologies, including the main concepts, definitions and mechanisms. Second, we illustrate each of them by means of practical examples. In addition, we provide an analytic comparison of RESTful and traditional WSDL-based services which have been discussed in the previous chapter.

### 5.1 Motivation

The central components of the Web services technology stack, as it has been introduced in the previous chapter, are WSDL [3] as an XML description language for Web services, SOAP [12] as an XML-based message format to exchange arbitrary XML data between services and clients, and UDDI [1] as a data model and API for Web service registries. The technical landscape around Web services is, however, much more scattered—spanning according to recent surveys [13] more than 80 Web services standards, technologies and specifications. This diversity is naturally related to a high learning curve, which is required to get familiar with the overall area, identify the most important developments, and stay up-to-date with the latest advances. Independently of these adoption issues, the WS-\* technology stack does not have much in common with the Web, despite the fact that their very name suggests something else [5]. In other words, Web services, as they are used in most cases, do not necessarily follow the same design principles and do not rely on the same core formats and protocols that have lead to the undeniable success of the Web as a global medium for communication. In a nutshell, communication on the Web is conceived as persistent publication—an information provider makes information available by publishing it online, and this information is then accessible to the intended readers. In contrast, Web services are frequently geared towards targeted messaging, with obvious consequences in terms of scalability. Two other aspects in which Web and Web services differ are the usage of URIs and the style of interaction [7, 15]. When sending and receiving SOAP messages, the content being exchanged is hidden in the body of the message, and not addressed as an explicit Web resource by its own URI. Consequently, all Web machinery involving caching and security checks is disabled,

since its use would require parsing and understanding of all possible XML dialects that could be used to write a SOAP message. In turn, referring to the content via an explicit URI in an HTTP request would allow the content of a message to be treated just as any other Web resource. Moving along to the second aspect, the Web service technology stack is oriented towards modeling stateful resources, in contradiction to the REST (REpresentational State Transfer) architectural principles [6] underlying the World Wide Web. Due to its stateless design, application integration and servers for this architecture are easy to build. Every HTTP request for a URI should retrieve the same content independent of what has happened before in other sessions, or in the history of the current session. This allows the use of thin servers that do not need to store, manage and retrieve the earlier session history to process the current session. When a stateful conversation is required, this should be explicitly modeled by different URIs. Applying these principles to Web services-based communication means that there should not be a single URI for a Web service, or hidden ways to model and exchange state information; instead, each potential state of a Web service should be explicitly addressable by a different URI.

These basic ideas have lead to the emergence of a new approach to design and build services artifacts, called RESTful services. Closely exploiting Web technologies and the REST architectural style, RESTful services make use of URI in a Web-compliant way—URIs are used to identify resources providing a global addressing space. In addition, the objects exchanged by the interacting services are resources published on the Web, manipulated using a fix set of operations that are classical Web methods: GET, DELETE, POST, and PUT. By embracing the publishing principle of the Web, RESTful services are implementing stateless interaction providing decoupling in time, space and reference. As a positive side-effect, the scope of the technological landscape involved is considerably narrower, including standards such as HTTP, URI, MIME, XML, which are familiar to a larger base of potential developers. This lightweight infrastructure, where services can be built with minimal tooling, has potentially a lower barrier for adoption than it is the case for WSDL-based services. Following the same line of reasoning, the interaction and integration of a RESTful service is easier than in case of WS-\* technologies due to the lower learning curve for clients and lower support demand for providers.

## 5.2 Technical Solution

In this section, we give an overview of the technologies that are used to realize RESTful services. We start in Sect. 5.2.1 by describing REST, the set of principles and the architecture style used to build large-scale distributed systems such as the World Wide Web. Section 5.2.2 contains an overview of existing approaches to describe RESTful services. More precisely, we discuss the Web Application Description Language and textual descriptions for RESTful services. Section 5.2.3 introduces the major approaches for data exchange between RESTful services including JSON, a lightweight computer data interchange format and XML-based solutions. Finally, in Sect. 5.2.4, we describe AJAX APIs, a RESTful technology that can be used to access RESTful services.

### 5.2.1 REST

REpresentational State Transfer (REST) [6] is a key design idiom that embraces a stateless client–server architecture. REST was originally introduced as an architectural style for building large-scale distributed hypermedia systems. This architectural style is a rather abstract entity, whose principles have been used to explain the excellent scalability of the HTTP 1.0 protocol and have also constrained the design of its following version, HTTP 1.1. The term REST very often is used in conjunction with HTTP. Even though REST is mostly used in conjunction with Web, REST is not really about the Web. REST is rather a distillation of the way the Web already works, a set of principles that can be applied for building scalable large distributed systems.

Conceptually, REST conceives resources as first-class objects identified by URIs. Resources represent the concepts that ground the conversation between a client and a server. According to this paradigm, services are seen as well as resources available on the Web. They are manipulated through their representations. Clients can request a specific representation of the concept from the representations the server makes available. More precisely, in case of services, clients that want to use a service access a particular representation of the resources representing the service by transferring application content using a small, globally-defined set of remote methods. These methods describe the action to be performed on resources. The methods used are the well-known HTTP methods for creating, reading, updating and deleting resources: GET, POST, DELETE, and PUT. If resources can be seen as concepts of the conversation, the methods that we mentioned above are the verbs that the developer can use to describe the necessary actions to be performed including creation, reading, updating, and deletion (CRUD).

One key principle of REST is stateless interaction between participants in a conversation. A state in this case means the state of the application/session. The state is maintained as part of the content transferred from client to server/service and back. This allows services to continue the conversation from the point it was left off. The stateless approach allows to implement a characteristic that is often desirable in distributed systems, namely decoupling, i.e., decoupling in time in this case.

### 5.2.2 Describing RESTful Services

Before we introduce existing approaches to describe RESTful services, let us first define what RESTful services are. In a nutshell, a RESTful service is a service implemented using HTTP and REST principles. More precisely, a RESTful service is a collection of Web resources that are interlinked and that can be manipulated using standard HTTP methods (e.g., POST, GET, PUT, or DELETE). An important feature of RESTful services is that they are data-centric rather than functionality-centric as WS-\* Web services are. One may notice the similarity in terms of principles and technologies used between RESTful services and Web applications. However, there

are some clear differences between the two concepts. The most important one is that RESTful services are intended for machine consumption, i.e., users of the services are other services or applications, while Web applications are intended for human consumption. RESTful services differ as well from WS-\* services. The key difference is that RESTful services follow Web principles and use more Web resources. As mentioned before, a RESTful service is a collection of Web resources. These resources are identified by URIs of the form **baseURI/ID**. The ID can be any unique identifier. For example, if a RESTful Web service representing a collection of rooms offered by a hotel might have the URI <http://example.com/resources/hotel>. If the service uses the room number as the ID then a particular room might be present in the collection as <http://example.com/resources/hotel/123>.

Although the number of RESTful services is increasing, there is currently no standard approach to describe this kind of services. If, for WS-\* Web services, the Web Service Description Language is the de facto standard, in the RESTful service domain all approaches to describe services are still in an early stage. To date, most RESTful services are described using free text. More structured approaches such as the Web Application Description Language (WADL) are currently emerging. In the rest of this section, we give an overview of these approaches.

### 5.2.2.1 Web Application Description Language (WADL)

The Web Application Description Language (WADL) [11] is an XML-based language for the description of RESTful services. It is designed to provide a machine-processable protocol description format for use with HTTP-based Web applications, especially those using XML to communicate. So far, WADL has been poorly adopted.

A WADL document is defined using the following elements:

**Application** is a top level element that contains the overall description of the service. It might contain grammars, resources, method, representation and fault elements.

**Grammars** acts as a container for definitions of any XML structures exchanged during execution of the protocol described by the WADL document. Using the sub-element **include** one or more structures can be included.

**Resources** acts as a container for the resources provided by the Web application.

**Resource** describes a single resource provided by the Web application. Each resource is identified by an URI, and the associated resources parent element. It can contain the following sub-elements: **path\_variable** that is used to parameterize the identifiers of the parent resource, zero or more **method** elements, and zero or more **resource** elements.

**Method** describes the input to and output from an HTTP protocol method that may be applied to a resource. A method element might have two child elements: a request element that describes the input to be included when applying an HTTP method to a resource, and a response element that describes the output that results from performing an HTTP method on a resource. A request element might contain query variable elements.

```

<?xml version="1.0"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://research.sun.com/wadl/2006/10 wadl.xsd"
  xmlns:tns="urn:yahoo:yn"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:yn="urn:yahoo:yn"
  xmlns:ya="urn:yahoo:api"
  xmlns="http://research.sun.com/wadl/2006/10">
  <grammars>
    <include
      href="NewsSearchResponse.xsd"/>
    <include
      href="Error.xsd"/>
  </grammars>
  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
    <resource path="newsSearch">
      <method name="GET" id="search">
        <request>
          <param name="appid" type="xsd:string"
            style="query" required="true"/>
          <param name="query" type="xsd:string"
            style="query" required="true"/>
          <param name="type" style="query" default="all">
            <option value="all"/>
            <option value="any"/>
            <option value="phrase"/>
          </param>
          <param name="results" style="query" type="xsd:int" default="10"/>
          <param name="start" style="query" type="xsd:int" default="1"/>
          <param name="sort" style="query" default="rank">
            <option value="rank"/>
            <option value="date"/>
          </param>
          <param name="language" style="query" type="xsd:string"/>
        </request>
        <response>
          <representation mediaType="application/xml"
            element="yn:ResultSet"/>
          <fault status="400" mediaType="application/xml"
            element="ya:Error"/>
        </response>
      </method>
    </resource>
  </resources>
</application>

```

**Listing 5.1** WADL example

**Representation** describes a representation of the state of a resource, and can either be declared globally as a child of the application element, embedded locally as a child of a request or response element, or referenced externally.

**Fault** is similar to a representation element in structure, but differs in that it denotes an error condition.

Listing 5.1 contains the WADL description of the Yahoo New Search [14].

As described earlier, the structure of a WADL description begins with an application description and defines the XML namespaces used elsewhere in the service description (Lines 1–8). It is followed then by a definition of the XML grammar used by the service (Lines 9–14). In this case, **NewsSearchResponse.xsd** and **Error.xsd** are included in the description. The actual Yahoo News Search Web service

is described in the rest of the document. The service has a ‘search’ GET method that requires a set of parameters (e.g., the Application ID, the query, etc.). The method returns two possible outputs.

### 5.2.2.2 Textual Descriptions of RESTful Services

The descriptions of most of the RESTful services on the Web are provided as textual information encoded in HTML pages. A typical example is the Flickr RESTful services. In this case, a set of HTML pages describe the REST API, the RESTful service provided by Flickr. Listing 5.2 contains the Flickr documentation for one of the Flickr API methods, namely **flickr.photos.getRecent**. This method returns a list of the latest public photos uploaded to Flickr.

The full description of all Flickr methods is available at [8]. The textual description of the service offers valuable information for the software developer that builds application interacting with the RESTful service. Furthermore, the textual description available in HTML can be annotated with lightweight semantic annotations that could enable more accurate search/discovery of RESTful services.

### 5.2.3 Data Exchange for RESTful Services

Data exchange is one important aspect in any communication between distinct applications. Just as any other kind of service, RESTful services can be seen as applications or software entities that can be accessed over the Internet. In this particular case, data exchange can be done in various ways. In the following, we introduce existing approaches to exchange data to and from RESTful services. Two major approaches are discussed, namely JSON, a lightweight computer data interchange format (Sect. 5.2.3.1), and XML, a general language for sharing structured data between information systems and implicitly (Sect. 5.2.3.2).

#### 5.2.3.1 JSON

JSON [4], short for JavaScript Object Notation, is a lightweight computer data interchange format, easy for humans to read and write, and, at the same time, easy for machines to parse and generate. Another key feature of JSON is that it is a completely language-independent. Programming in JSON does not raise any challenge for programmers experienced with the C family of languages. There are various reasons for preferring JSON to XML as data exchange format instead of XML, given the widespread adoption of the latter. First, data entities exchanged with JSON are typed, while XML data is typeless; some of the built-in data types available in JSON are: string, number, array and boolean. XML data, on the other hand, are all strings. Second, JSON is lighter and faster than XML as on-the-wire data format. Third,

### Authentication

This method does not require authentication.

#### Arguments

##### api\_key (Required)

Your API application key. See here for more details.

##### extras (Optional)

A comma-delimited list of extra information to fetch for each returned record. Currently supported fields are: license, date\_upload, date\_taken, owner\_name, icon\_server, original\_format, last\_update, geo, tags, machine\_tags, o\_dims, views, media.

##### per\_page (Optional)

Number of photos to return per page. If this argument is omitted, it defaults to 100. The maximum allowed value is 500.

##### page (Optional)

The page of results to return. If this argument is omitted, it defaults to 1.

### Example Response

This method returns the standard photo list xml:

```
<photos page="2" pages="89" perpage="10" total="881">
    <photo id="2636" owner="47058503995@N01"
        secret="a123456" server="2" title="test_04"
        ispublic="1" isfriend="0" isfamily="0" />
    <photo id="2635" owner="47058503995@N01"
        secret="b123456" server="2" title="test_03"
        ispublic="0" isfriend="1" isfamily="1" />
    <photo id="2633" owner="47058503995@N01"
        secret="c123456" server="2" title="test_01"
        ispublic="1" isfriend="0" isfamily="0" />
    <photo id="2610" owner="12037949754@N01"
        secret="d123456" server="2" title="00_tall"
        ispublic="1" isfriend="0" isfamily="0" />
</photos>
```

To map <photo> elements to urls, please read the url documentation.

#### Error Codes

1: bad value for jump\_to, must be valid photo id.

100: Invalid API Key

The API key passed was not valid or has expired.

105: Service currently unavailable

The requested service is temporarily unavailable.

111: Format "xxx" not found

The requested response format was not found.

112: Method "xxx" not found

The requested method was not found.

114: Invalid SOAP envelope

The SOAP envelope send in the request could not be parsed.

115: Invalid XML-RPC Method Call

The XML-RPC request document could not be parsed.

### Listing 5.2 Flickr RESTful service documentation

JSON integrates natively with JavaScript, a very popular programming language used to develop applications on the client side; consequently, JSON objects can be directly interpreted in JavaScript code, while XML data needs to be parsed and

```
{
var weatherInnsbruck = {
    "city" : "Innsbruck",
    "date" : "2009-04-01",
    "temperature" : 15
};
}
```

**Listing 5.3** JSON object notation example

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<weatherInnbruck>
<city>Innsbruck</city>
<date>2009-04-01</date>
<temperature>15</temperature>
</weatherInnbruck>
```

**Listing 5.4** JSON object notation represented in XML

assigned to variables through the tedious usage of DOM APIs (Document Object Model APIs, see the next section on XML).

JSON is built on two structures: (i) a collection of name/value pairs. In various languages, this is realized as an object, record, structure, dictionary, hash table, keyed list, or associative array; and (ii) an ordered list of values. In most languages, this is realized as an array, vector, list, or sequence. The notations used to represent the structures in JSON are conceived to be very intuitive. For example, an object notation is an unordered set of name/value pairs. A JSON object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma). An example of JSON object is provided in Listing 5.3.

This object has three properties or name/value pairs. The name is a string in our example, city, date and temperature. The value can be any JavaScript object. In our example, the values are ‘Innsbruck’, ‘2009-04-01’ and 15. The first two values are strings, but the temperature is a number. The reader might ask why the date is represented as a string. The answer is simple. One of the sore points of JSON is the lack of a date/time literal. This is mainly due to the absence of a date/time literal in JavaScript, too. Most applications using JSON as a data format use either a string or a number to express date and time values. If a string is used, you can generally expect it to be in the ISO 8601 format.

The corresponding XML representation of the weather Innsbruck JSON object is provided in the following listing (Listing 5.4).

A more complex example of JSON object is available in Listing 5.5. In this example, we define an object called **weatherAustria**. The object has three properties representing three regions of Austria. The values of each property are an array holding JSON objects, one for each city from the region. Each of these objects has a structure identical to the structure of the object introduced in the previous listing (Listing 5.3).

The same text expressed as XML is provided in Listing 5.6.

```
var weatherAustria = {  
    "weatherTyrol": [  
        { "city" : "Innsbruck",  
          "date" : "2009-04-01",  
          "temperature" : 15},  
  
        { "city" : "Kufstein",  
          "date" : "2009-04-01",  
          "temperature" : 16},  
  
        { "city" : "Schwaz",  
          "date" : "2009-04-01",  
          "temperature" : 13}  
    ],  
  
    "weatherVienna":  
        { "city" : "Vienna",  
          "date" : "2009-04-01",  
          "temperature" : 16},  
  
    "weatherUpperAustria": [  
        { "city" : "Linz",  
          "date" : "2009-04-01",  
          "temperature" : 14},  
  
        { "city" : "Welz",  
          "date" : "2009-04-01",  
          "temperature" : 15}  
    ]  
}
```

**Listing 5.5** JSON example expressed as text

In order to access data from a JSON object, one uses the dot notation. Taking as example the **weatherInnsbruck** JSON object introduced in Listing 5.3, **weatherInnsbruck.city** returns the value of the property city of this object. Property values of complex objects are accessed in the same way. For example, the temperature value for Innsbruck in the **weatherAustria** JSON object is delivered by the expression **weatherAustria.weatherTyrol[0].temperature**.

As discussed earlier, JSON is a text-based data exchange format. Information received and sent to/from the client and server side is formatted as text. One can transform the textual JSON representation into a JSON object using the **eval()** method, which invokes the JavaScript compiler. Since JSON is a proper subset of JavaScript, the compiler will correctly parse the text and produce an object structure. The conversion from the object representation to the text representation of a JSON information is achieved using the **toJSONString()** method applied to the JSON object.

To summarize, JSON is a lightweight, text-based data exchange format based on a subset of the literal notation from the JavaScript programming language. It is used in scenarios where a JavaScript implementation is available to one or both interacting parties, for example, in the case of Ajax-style Web applications. A major advantage of JSON is that there is virtually no learning curve for developers familiar with JavaScript and the C-like languages. At present, the support for JSON is on the increase, as more and more libraries for working with JSON across all major platforms, and frameworks are being developed.

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<weatherAustria>
  <weatherTyrol>
    <city>Innsbruck</city>
    <date>"2009-04-01"</date>
    <temperature>15</temperature>
  </weatherTyrol>
  <weatherTyrol>
    <city>Kufstein</city>
    <date>"2009-04-01"</date>
    <temperature>16</temperature>
  </weatherTyrol>
  <weatherTyrol>
    <city>Kufstein</city>
    <date>"2009-04-01"</date>
    <temperature>16</temperature>
  </weatherTyrol>
  <weatherInnabruck>
  <weatherVienna>
    <city>Vienna</city>
    <date>"2009-04-01"</date>
    <temperature>16</temperature>
  </weatherVienna>
  <weatherUpperAustria>
    <city>Linz</city>
    <date>"2009-04-01"</date>
    <temperature>14</temperature>
  </weatherUpperAustria>
  <weatherUpperAustria>
    <city>Wels</city>
    <date>"2009-04-01"</date>
    <temperature>15</temperature>
  </weatherUpperAustria>
</weatherAustria>
```

**Listing 5.6** JSON example expressed as XML

### 5.2.3.2 XML

XML [2], short for EXtensible Markup Language, is a general-purpose specification for creating custom mark-up languages. It was designed as a language for sharing structured data on the net between information systems, for encoding documents and for serializing data. XML began as a simplified subset of the SGML, the Standard Generalized Markup Language. SGML is the language from which HTML stems. XML allows the user to define his/her own tags. With arbitrary SGML tags, one can begin to indicate the meaning of particular data directly within the document source—this was exactly what was required to successfully share structured data across different information systems (in contrast with HTML which can be considered relatively unstructured/less-rich due to its lack of expressivity, i.e., well-defined content). XML is simply an SGML subset tailored for the World Wide Web, which imposes certain restrictions, such as closed tags and case sensitive names, while still allowing for user-defined tags. User-defined tag names and attributes in XML allow for easier machine-processing and at the same time XML provides a common, structured format which allows machines to share data—a standard representational format for documents beyond HTML including typical office documents

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<weatherVienna>
  <city>Vienna</city>
  <date>"2009-04-01"</date>
  <temperature>17</temperature>
</weatherVienna>
```

**Listing 5.7** XML response of RESTful weather service

such as calendars and spreadsheet (which previously had no widely used standardized interoperable formats).

Just like WS-\* Web services, RESTful Web services can use XML as a standard format for service input and output. Unlike the former, RESTful Web services do not require data being transmitted to be wrapped as part of protocol specific elements (i.e., SOAP messages, in the case of WS-\* services). In this way, there is no overhead of parsing and handling complex message structures; the information is directly exposed to the client's parser.

XML-formatted messages are using parsing API such as DOM (Document Object Model) or SAX (Simple API for XML). Support for DOM and SAX is available in almost all programming languages, making it easy for clients to consume XML messages. An example of an XML message representing the response of a RESTful weather service is available in Listing 5.7.

There are several major concerns with respect to the usage XML for data exchange between RESTful services and its clients. First, XML introduces a certain overhead due to its mark-up nature. Second, there are computational costs associated with parsing an XML message, and using DOM for the navigation of the XML tree model, none of which occur for JSON. Thus, JSON is recommended to be used for passing data to/from a front-end, usually lightweight front-end. The usage of XML as data exchange format when communicating with RESTful services is preferred in other situations, for instance, when rich clients or other servers are involved.

### 5.2.4 AJAX APIs

Asynchronous JavaScript and XML, or shortly AJAX [9], is a family of Web development technologies used to realize interactive, rich Web applications. One major advantage of AJAX-based applications is that they can retrieve data from the server asynchronously in the background, without interfering with the display and behavior of the existing Web page. With AJAX one can create better, faster, and more user-friendly Web applications by increasing the interactive animation on Web pages [9].

Several technological building blocks form the foundation of the AJAX approach. First, AJAX uses XHTML and CSS for standards-based presentation. Data interchange and manipulation is based on XML and XSLT. AJAX makes extensive use of the XMLHttpRequest object to asynchronously retrieve data from the server. Using this object, the JavaScript code can communicate directly with the server,

trading data without reloading the entire Web page. Finally, to bind everything together, AJAX relies on JavaScript as the underlying programming language.

In the following, we briefly outline the line of arguments that shows the benefits of AJAX against standard approach to implement Web application. To understand this, let us first recap how a standard Web application works. In the traditional model of a Web application, the user triggers through the interface an HTTP request to a Web server. The Web server processes the request, generates an HTML Web page and sends it back to the client. On the server side, the first step, namely processing of the request, might take some time due, in general, to the complex business logic being executed. The drawback in this interaction is that while the Web server processes the request the client is left waiting, and the user experience is thus suboptimal. AJAX tries to specifically address this part of the overall interaction process. It introduces a third entity between the client and the server, namely the AJAX engine. Having more parties in the interaction does not have a negative effect on the interaction times, but rather offers the user a more dynamic interaction experience. This engine is responsible for both rendering the interface the user sees, and communicating with the server on the user's behalf. The communication is done asynchronously independently of the communication with the server.

### 5.2.5 Examples of RESTful Services

In this section, we describe some existing RESTful services that are available on the Web: the Atom Publishing Protocol and the Flickr API.

#### 5.2.5.1 Atom Publishing Protocol

The Atom Publishing Protocol (AtomPub) [10] is an application-level protocol for publishing and editing Web resources. It uses HTTP and XML as an underlying transfer protocol and an exchange format, respectively. The data being exchanged is specified in a XML based language called Atom Syndication Format. The specification Atom Syndication Format describes two kinds of Atom Documents: Atom Feed Documents and Atom Entry Documents. An Atom Feed Document has as root element the atom:feed element, and is a representation of an Atom feed, including metadata about the feed, and some or all of the entries associated with it. An Atom Entry Document has as root element atom:entry and represents exactly one Atom entry, outside of the context of an Atom feed.

An example of a document in the Atom Syndication Format is available in Listing 5.8.

The Atom Publishing Protocol supports the creation of Web Resources and provides a means to interact with and manage collections of resources; discover and describe collections; and interact with and manage individual resources. A high-level overview of the AtomPub service is available in Fig. 5.1.

```

<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">

<title>Example Feed</title>
<subtitle>A subtitle.</subtitle>
<link href="http://example.org/feed/" rel="self"/>
<link href="http://example.org/">
<updated>2009-04-05T12:45:02Z</updated>
<author>
  <name>Ioan Toma</name>
  <email>ioan.toma@sti2.at</email>
</author>
<id>urn:uuid:60a76c80-d399-11d9-b91c-0003939e0af6</id>

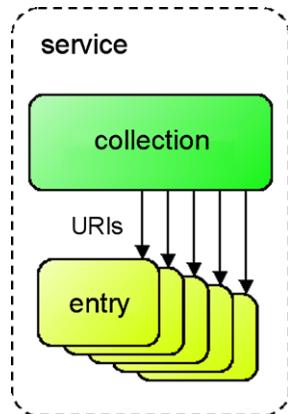
<entry>
  <title>Atom-Powered Robots Run Amok</title>
  <link href="http://example.org/2003/12/13/atom03"/>
  <id>urn:uuid:34c5f6d5-df18-e2cc-1112-80da344efa6a</id>
  <updated>2009-04-05T12:45:02Z</updated>
  <summary>RESTful services are great.</summary>
</entry>

</feed>

```

**Listing 5.8** Atom syndication format example

**Fig. 5.1** AtomPub RESTful service—high level overview



As illustrated in Fig. 5.1, Atom Publishing Protocol distinguishes between two types of resources: collections and entries. The first elements represent collections of the later elements. An AtomPub RESTful service should provide methods to manage both collections and entries. For collections the GET HTTP method is used to list the entries in a collection, while POST HTTP method is used to add an entry in the collection. In the case of entry resources, the GET HTTP method is used to retrieve a particular entry, PUT HTTP method is used to replace and update an entry and finally DELETE HTTP method is used to implement the delete operation for entries.

```
<rsp stat="ok">
<user id="9YYYY6246@N00" nsid="9YYYY6246@N00">
<username>ioantoma</username>
</user>
</rsp>
```

**Listing 5.9** XML response of Flickr service

### 5.2.5.2 Flickr API

Flickr<sup>1</sup> is a photo-sharing Web site that allows Internet users to upload and browse photos, and add tags, comments, and annotations to these. This functionality is programmatically exposed as WS-\* and RESTful services. In a nutshell, the Flickr RESTful service is a collection of operations or methods that enable a complete management of photos and the information related to them. Some of the supported methods are **flickr.photos.addTags**, **flickr.photos.delete**, **flickr.contacts.getList**, **flickr.photos.comments.editComment**. As mentioned in Sect. 5.2.2.2, the Flickr API methods that form the Flickr RESTful service are described in textual form.

Flickr uses the HTTP methods GET and POST to implement most of the methods from the API it provides. The general way to invoke any method of the API is to access <http://api.flickr.com/services/rest/?method=method&parameters>, where the last part of the URI **method&parameters** is replaced by a specific method name and the associated parameters. For example, calling the Flickr method **flickr.people.findByUsername** with the required parameter having the value **ioantomam** will produce the result available in Listing 5.9.

One important aspect when using the Flickr RESTful service is authentication. Users should be authenticated using the Authentication API provided by Flickr. Once logged-in the user receives an authentication token and signature that are used to invoke any Flickr method. In general, most ‘REST-like APIs’ have proprietary authentication mechanisms; they define additional mandatory API keys for this purpose. A closer look at the Flickr RESTful service reveals that the service does not follow all RESTful services principles:

- The client must know all methods beforehand.
- Non-standard authentication, tokens in URIs.
- URIs with method name identify verbs (operations), not nouns (data). As an example, instead of “DELETE URI <photo>” the Flickr API has “POST nothing to URI <delete/photo>”.

## 5.3 Illustration by a Larger Example

To illustrate how RESTful services described in the previous sections can be used to implement a distributed application, we select the same scenario that was introduced in Chap. 4. We remind the reader that, in the previously mentioned scenario,

---

<sup>1</sup>[www.flickr.com](http://www.flickr.com).

```

<?xml version="1.0"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://research.sun.com/wadl/wadl.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:bhns = "http://www.bluehotel.com/schemas/BlueHotelService"
  xmlns="http://research.sun.com/wadl">
  <grammars>
    <include href="http://www.bluehotel.com/schemas/BlueHotelService.xsd"/>
  </grammars>

  <resources base="http://www.blueHotel.com/BlueService/">
    <resource uri="checkAvailability"> 18
      <method href="checkAvailability"/></resource>
    </resources>

    <method name="GET" id="checkAvailability">
      <request>
        <query_variable name="checkInDate" type="xsd:date" required="true"/>
        <query_variable name="checkOutDate" type="xsd:date" required="true"/>
        <query_variable name="roomType" type="xsd:string" required="true"/>
        <query_variable name="results" type="bhns:Response"/>
      </request>

      <response>
        <representation mediaType="application/xml" element="bhns:Response"/>
        <fault id="SearchError" status="400" mediaType="application/xml"
          element="bhns:Error"/>
      </response>
    </method>
  </application>

```

**Listing 5.10** WADL description of BlueHotelService

a ‘Virtual travel agency’ (VTA) is providing to end users eTourism services. The agency acts as a middle man entity aggregating various tourism services provided by other tourism providers such as booking of flights, hotels, rental cars, etc. Such tourism service providers could be the imaginary hotel company, called ‘The Blue Hotel’, introduced in the previous chapter. The Blue Hotel allows its clients to check for rooms availability using the ‘BlueHotelService’ Web service.

The WADL representation of this service is available in Listing 5.10.

Data exchanged between the client, in this case the VTA application, and the RESTful service, in this case the BlueHotelService, can be represented using JSON as explained in Sect. 5.2.3.1. A simple JSON object sent by the BlueHotelService if requested to provide information about the availability of single rooms in a given time interval is listed in Listing 5.11.

The corresponding XML representation of the **blueHotelSingleRoom** JSON object is provided in the following listing (Listing 5.12).

A more complex JSON object might, for instance, include an array of simple JSON objects as the one just discussed. Listing 5.13 shows an object containing the information for all types of rooms during the same period of time.

Listing 5.14 shows the same, only in XML.

```
{
var blueHotelSingleRoom = {
    "startDate" : "2009-03-24",
    "endDate" : "2009-03-30",
    "roomType" : "Single"
    "rateType" : "Discount"
    "rate" : 150.50
};
}
```

**Listing 5.11** Text representation of a simple JSON object exchanged by the BlueHotel RESTful service

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<blueHotelSingleRoom>
<startDate>"2009-03-24"</startDate>
<endDate>"2009-03-30"</endDate>
<roomType>"Single"</roomType>
<rateType>"Discount"</rateType>
<rate>150.50</rate>
</blueHotelSingleRoom>
```

**Listing 5.12** XML representation of a simple JSON object exchanged by the BlueHotel RESTful service

```
var blueHotelRoom = {
    "singleDiscount" : {
        "startDate" : "2009-03-24",
        "endDate" : "2009-03-30",
        "roomType" : "Single"
        "rateType" : "Discount"
        "rate" : 150.50},
    "singleNoDiscount" : {
        "startDate" : "2009-03-24",
        "endDate" : "2009-03-30",
        "roomType" : "Single"
        "rateType" : "No Discount"
        "rate" : 170.00},
    "doubleDiscount" : {
        "startDate" : "2009-03-24",
        "endDate" : "2009-03-30",
        "roomType" : "Double"
        "rateType" : "Discount"
        "rate" : 200.00},
    "doubleNoDiscount" : {
        "startDate" : "2009-03-24",
        "endDate" : "2009-03-30",
        "roomType" : "Double"
        "rateType" : "No Discount"
        "rate" : 220.00}
}
```

**Listing 5.13** Text representation of a complex JSON object exchanged by the BlueHotel RESTful service

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<blueHotelRoom>
  <singleDiscount>
    <startDate>2009-03-24</startDate>
    <endDate>2009-03-30</endDate>
    <roomType>Single</roomType>
    <rateType>Discount</rateType>
    <rate>150.50</rate>
  </singleDiscount>
  <singleNoDiscount>
    <startDate>2009-03-24</startDate>
    <endDate>2009-03-30</endDate>
    <roomType>Single</roomType>
    <rateType>No Discount</rateType>
    <rate>170.50</rate>
  </singleNoDiscount>
  <doubleDiscount>
    <startDate>2009-03-24</startDate>
    <endDate>2009-03-30</endDate>
    <roomType>Double</roomType>
    <rateType>Discount</rateType>
    <rate>200.00</rate>
  </doubleDiscount>
  <doubleNoDiscount>
    <startDate>2009-03-24</startDate>
    <endDate>2009-03-30</endDate>
    <roomType>Double</roomType>
    <rateType>No Discount</rateType>
    <rate>220.00</rate>
  </doubleNoDiscount>
</blueHotelRoom>
```

**Listing 5.14** XML representation of a complex JSON object exchanged by the BlueHotel RESTful service

## 5.4 Summary

In this chapter, we gave an overview of an alternative approach to implement functionality, namely RESTful services. We started by introducing the design principles and core architecture style for RESTful services. Known as REST, this architecture style and set of principles is the very same that turned the Web into a successful large-scale distributed system as we know it today. Following this line of reasoning, we have introduced, explained and exemplified some of the most popular approaches to describing and annotating RESTful functionality. The Web Application Description Language, shortly WADL, is one of these. We have seen that this language provides elements to describe various aspects of RESTful services. For example, its elements capture the resources provided by the application, the methods that may be applied to a resource, representations of the resources, and error conditions. In terms of data exchange, we have discussed JSON and XML as alternative approaches. JSON is easy-to-use, reduces communication overhead, and is natively integrated with JavaScript. XML brings in some advantages as well: extensibility and support in all popular programming languages. In the last part of the chapter, we have discussed one RESTful services technology used to implement applications that are interacting with RESTful services, namely AJAX. It is based on

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<Rockbands>
<Rockband>
<Name>Beatles</Name>
<Country>England</Country>
<YearFormed>1959</YearFormed>
<Members>
<Member>Paul</Member>
<Member>John</Member>
<Member>George</Member>
<Member>Ringo</Member>
</Members>
</Rockband>
<Rockband>
<Name>Rolling Stones</Name>
<Country>England</Country>
<YearFormed>1962</YearFormed>
<Members>
<Member>Mick</Member>
<Member>Keith</Member>
<Member>Charlie</Member>
<Member>Bill</Member>
</Members>
</Rockband>
<Rockband>
<Name>Queen</Name>
<Country>England</Country>
<YearFormed>1970</YearFormed>
<Members>
<Member>Freddie</Member>
<Member>Brian</Member>
<Member>Roger</Member>
<Member>John</Member>
</Members>
</Rockband>
<Rockband>
<Name>Pink Floyd</Name>
<Country>England</Country>
<YearFormed>1965</YearFormed>
<Members>
<Member>Nick</Member>
<Member>Syd</Member>
<Member>David</Member>
<Member>Roger</Member>
<Member>Richard</Member>
</Members>
</Rockband>
</Rockbands>
```

**Listing 5.15** Rock band music info—XML representations

JavaScript programming for browsers providing reach interaction experience with RESTful services for Web users.

## 5.5 Exercises

**Exercise 1** Create a WADL file that corresponds to a business service providing the following functionality:

- Create a new order
  - public PurchaseOrderStatus acceptPO(PurchaseOrder order)
- Retrieve an existing order
  - public PurchaseOrder retrievePO(String orderID)
- Modify an existing order
  - public PurchaseOrder updatePO(PurchaseOrder order)
- Cancel an order already submitted
  - public void cancelPO(String orderID)

The service should be available at the following endpoint-address: <http://www.swsbook.com/theRESTfulService>.

**Exercise 2** Consider a bus transportation company being a company that provides tourism services as part of the VTA scenario introduced in Sect. 5.3. Design a WADL description for one of the services this company provides and explain all its elements.

**Exercise 3** Given the XML representation of a data object exchanged with a RESTful service that provides music information in Listing 5.15 create the corresponding JSON textual representation.

**Exercise 4** Create a JSON representation, both text and XML of the data exchanged between the VTA application and the bus transportation company service. The VTA is the client, the bus transportation company is the service in this particular interaction. The information returned by the bus transportation company includes: the start and end city of the trip, the price for the ticket and a list of departure hours. The bus transportation company offers trips from multiple cities in Austria.

## References

1. Bellwood, T.: UDDI version 2.04 API specification (2002). Available from <http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm>
2. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible markup language (xml) 1.0 (fourth edition)—origin and goals. Working draft, World Wide Web Consortium (2006). Available from <http://www.w3.org/TR/xml/>
3. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. (2001). <http://www.w3.org/TR/wsdl>
4. Crockford, D.: Json—javascript object notation. Technical report, json.org (2006). <http://www.json.org/>
5. Fensel, D.: Triple-space computing: Semantic Web Services based on persistent publication of information. In: Proc. of IFIP Int. Conf. on Intelligence in Communication Systems, pp. 43–53 (2004)
6. Fielding, R.T.: Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine, Irvine, California (2000). <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
7. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Transactions on Internet Technology 2(2), 115–150 (2002). doi:[10.1145/514183.514185](https://doi.org/10.1145/514183.514185)

8. Flickr: Flickr services. Technical report, Flickr.com (2009). Available from <http://www.flickr.com/services/api/>
9. Garrett, J.J.: Ajax: A new approach to web applications. Technical report, AdaptivePath.com (2005). Available from <http://www.adaptivepath.com/ideas/essays/archives/000385.php>
10. Gregorio, J., de Hora, B.: The atom publishing protocol. Rfc 5023, Internet Draft draft-ietf-atompub-protocol-17 (2007). Available from <http://tools.ietf.org/html/rfc5023>
11. Hadley, M.J.: Web application description language (wadl). Working draft, Sun Microsystems Inc. (2006). Available from <https://wadl.dev.java.net/wadl20061109.pdf>
12. W3C: SOAP Version 1.2 Part 0: Primer (2003)
13. Wikipedia. List of Web service specifications. [http://en.wikipedia.org/wiki/WS-\\*](http://en.wikipedia.org/wiki/WS-*)
14. Yahoo!: Yahoo! web APIs. Technical report, yahoo.com (2005). <http://developer.yahoo.net/>
15. zur Muehlen, M., Nickerson, J.V., Swenson, K.D.: Developing web services choreography standards—the case of rest vs. soap. *Decision Support Systems* **40**(1), 9–29 (2005)

# Chapter 6

## Semantic Web

**Abstract** This chapter gives an overview of the Semantic Web which aims to enable automatic retrieval, extraction and integration of information on the World Wide Web (WWW). Event though the current Web has become a big success, it lacks a proper support to Web users when it comes to finding, extracting and combining information. The main obstacle is that, at present, the meaning of Web content is not machine-accessible. The Semantic Web extends the current Web providing machine processable semantics to Web resources. This chapter is built around the Semantic Web layered architecture introduced by its inventor, Tim Berners-Lee. Each layer and language of the Semantic Web is described in terms of its major concepts and core aspects. As part of the lower layers of the Web architecture, the chapter introduces Unicode: the Uniform Resource Identifier (URI), the Extensible Markup Language (XML), the Document Type Definition (DTD), and the XML Schema. The chapter then discusses languages that are built on top of the syntactic layers introduced previously, namely the Resource Description Framework (RDF), RDF Schema (RDFS), SPARQL, the Web Ontology Language (OWL), its new version OWL2, and the Rule Interchange Format (RIF). The chapter then briefly reports on the other layers of the Semantic Web architecture and discusses the latest development of the Web and Semantic Web, know as Linked Open Data.

### 6.1 Motivation

The current Web, or the World Wide Web (WWW), as we know it today is a big success. It was introduced more than 15 years ago as a means for global information sharing. Web development started even earlier with the creation of the Internet as a means to interweave computers. Starting as a small solution for some thousand users, the Web grew exponentially counting presently more that 1 trillion pages.<sup>1</sup> In a nutshell, “the Web is a system of interlinked documents that run over the Internet. With a Web browser, a user views Web pages that may contain text, images, and other multimedia and navigates between them using hyperlinks”.<sup>2</sup> Two of the

---

<sup>1</sup><http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html>.

<sup>2</sup>[http://en.wikipedia.org/wiki/World\\_Wide\\_Web](http://en.wikipedia.org/wiki/World_Wide_Web).

most representative applications that contributed to the big success of the Web are Netscape and Google.<sup>3</sup> The Netscape browser was the first to provide an intuitive and simple graphic interface that allows users to easily navigate the Web. It immediately attracted a large community, making it attractive for others to present their information on the Web. The other big Web success story is Google, the popular search engine that is associated with the mega growth of the Web. Google indexes more than 1 trillion pages, making it possible to access information instantly on a global scale.

The success of the Web is based on three simple principles:

- A simple and uniform addressing schema to identify information chunks.
- A simple and uniform representation formalism to structure information chunks allowing browsers to render them.
- A simple and uniform protocol to access information chunks.

The addressing scheme used on the Web is based on Uniform Resource Identifiers (URIs). URIs are pointers to resources on the Web to which request methods can be applied to generate potentially different responses. Resources are described using a simple and uniform representation language called HyperText Markup Language (HTML), which provides a set of pre-defined tags to annotate information sources and to define interlinks between them using URIs. Finally, the Hypertext Transfer Protocol (HTTP) is a simple request/response protocol that can be used to access resources on the Web. HTTP integrates nicely with the other two basic components, namely URIs and HTML. HTTP relies on the URI naming mechanism and provides a way to publish and retrieve resources described using HTML.

In the recent years, a new phenomenon called Web2.0 has become very popular. Web 2.0 sites such as Facebook,<sup>4</sup> MySpace,<sup>5</sup> Delicious,<sup>6</sup> YouTube,<sup>7</sup> and LinkedIn<sup>8</sup> have grown tremendously over the past few years, garnering increased media and popular attention. The result of this increased awareness is that the Web has become more powerfully socially linked now than ever before. Even though there are very minor differences between Web1.0 and Web2.0 in terms of technologies used, there are differences in emphasis between the two. With the Web 1.0 technology, a significant amount of software skills and investment in software was necessary to publish information. Web 2.0 technology changed this dramatically. The four major breakthroughs of Web 2.0 are:

- Blurring the distinction between content consumers and content providers.
- Moving from media for individuals towards media for communities.
- Blurring the distinction between service consumers and service providers.

---

<sup>3</sup>[www.google.com](http://www.google.com).

<sup>4</sup>[www.facebook.com](http://www.facebook.com).

<sup>5</sup><http://www.myspace.com/>.

<sup>6</sup><http://delicious.com/>.

<sup>7</sup><http://www.youtube.com/>.

<sup>8</sup><http://www.linkedin.com>.

- Integrating human and machine computing in a new way.

With the Web 2.0 technology the billions of people on the Web not only consume but produce much of its content. In this way, the Web has become a true instant communication channel of mankind on a global scale.

With all the advances of the Web1.0 and Web2.0 technologies, the current Web still has a set of major limitations. The most visible is related to finding relevant information. Finding information on Web1.0 and Web2.0 is based on keyword search which has a limited recall and precision. This is due, in principle, to the ambiguity of the written natural language. For example, documents containing synonyms of the query keywords will not be retrieved by a naive keyword-based search engine even though they are relevant and should be included in the answer. Another shortcoming of the keyword search is caused by homonyms from natural language. For example, searching information on “Jaguar” will bring up pages containing information about both “Jaguar” (the car brand) and “Jaguar” (the animal) even though the user is interested only in one of them. Other factors that have a negative influence on the recall and precision of keyword search used on the current Web are different spelling variants and spelling mistakes. Moreover, the use of different native languages make the cross-language retrieval of relevant document difficult. Last but not least, current search engines provide no means to specify the relationship between a resource and a term (e.g., sell/buy).

Finding relevant information is not the only task where current approaches could be significantly improved. Extracting relevant information from Web pages, is another one. One-size-fits-all automatic solution for extracting information from Web pages is not possible due to different formats and different syntaxes. Even on the very same Web page, extracting relevant information proves to be a challenging task. Current approaches for information extraction are based on wrapper technology. The extraction of information from Web sites can be implemented using XML based standards such as XSLT.<sup>9</sup> Usually, the extracted information is stored as structured data in XML format or databases. Using wrappers does not really scale because the actual extraction of information depends on the Web site format and layout.

Last but not least, another task where current Web solutions are limited, is in combining and reusing information on the Web. In a typical scenario, the same information is searched for in different digital libraries. For example, when planning to travel from Innsbruck to Rome, one could consider different alternatives such as traveling by train, airplane or bus. Information about all these alternatives comes from different Web sites. Furthermore, it is often the case that a combination of partial solutions is needed to fulfill the initial request.

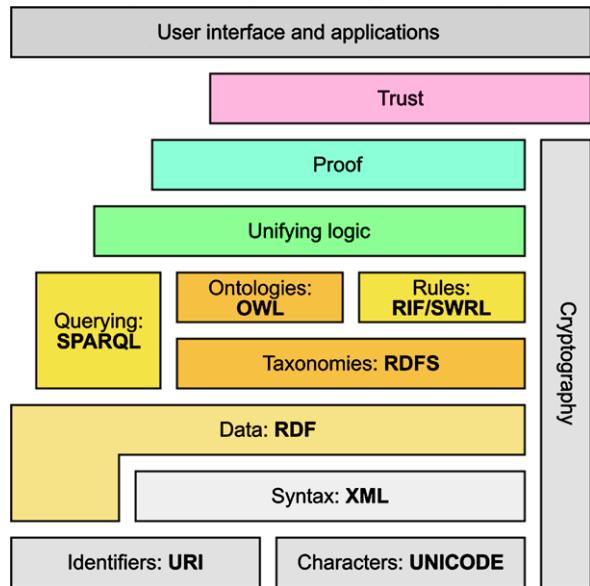
## 6.2 Technical Solution

To address the limitations of the current Web discussed in Sect. 6.1, a solution based on formally describing and using the meaning of information and services has been

---

<sup>9</sup><http://www.w3.org/TR/xslt>.

**Fig. 6.1** Semantic Web architecture



proposed. The solution, called the Semantic Web, enables machines to understand and potentially satisfy user requests, by processing the meaning or “semantics” of information. The Semantic Web improves the current Web by increasing automation and accuracy in various aspects such as search, information extraction and information integration. As envisioned initially by Tim Berners-Lee [4], “the Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation”. According to [7], “the explicit representation of the semantics of data, accompanied with schema definitions, will enable a Web that provides a qualitatively new level of service”. The Semantic Web complements the current Web with machine processability. Thus computers or machines become entities that extract and interpret information, rather than being just devices to post and render information for human users.

Figure 6.1 shows the so-called Semantic Web Layer Cake that illustrates the Semantic Web architecture.

In the lower part of the architecture, we find three building blocks that can be used to encode text (Unicode), to identify resources on the Web (URIs) and to structure and exchange information (XML). **Unicode**<sup>10</sup> is the ISO standard that defines the international character set. It is an extension of the American Standard Code for Information Interchange (ASCII) that is limited to only 128 characters (a–z, A–Z, 0–9, punctuation) and possesses a larger set of characters including all letters and signs from all human languages. In total, Unicode defines 100.000 characters having a clearly defined process for adding new ones. One distinct feature of Unicode is the

<sup>10</sup><http://en.wikipedia.org/wiki/Unicode>.

potential to define extension code pages, where each code page is an interpretation of the text not a property of it. Unicode can be implemented by different character encodings. UTF-8 is one possible encoding for Unicode, being also the most used Unicode encoding on the Web.

The **Uniform Resource Identifier (URI)** is another standard part of the foundational layers of the Semantic Web. URIs are used to identify resources on the Web. Depending on their purpose, URIs can be classified in two classes: Uniform Resource Names (URNs) and Uniform Resource Locators (URLs). A Uniform Resource Name (URN) defines an item identity, while a Uniform Resource Locator (URL) provides a method for finding it. The syntax for defining a URI is:

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

where the **schema** is used to distinguish between different types of URI (e.g., http, ftp, etc.), the **authority** is normally identified with a server, the **path** is identified with a directory or a file on the server, the **query** adds extra parameters, and **fragid** identifies a secondary resource.

Also part of the lower level of the architecture, the **Extensible Markup Language (XML)**<sup>11</sup> has been developed by the W3C with the intention of providing a language that can be used to define user specific document types. The development of XML was motivated by a set of deficiencies of existing markup language such as the inflexibility of HTML or the complexity of SGML (note that XML itself is a subset of SGML). XML has quickly become the predominant standard for exchanging structured and semistructured data. XML can be seen as a very flexible approach that allows markup information about the logical structure of a document to be mixed with the actual content of the document. Structural information in XML is defined in terms of elements and attributes. An XML element has a name or label and a content, where the content can be itself another XML element or set of XML elements. An element name or label together with markup delimiters is called a tag. An attribute is a name–value pair inside the opening tag of an element, allowing the inclusion of additional information as part of tags.

To make the structure of XML documents accessible to machines, two other standards were developed, namely Document Type Definition (DTD)<sup>12</sup> and XML schema.<sup>13</sup> Using DTD or XML Schema one can define a set of constraints on the content allowed to occur as part of XML elements or attributes. Compliance with the XML syntactic rules makes an XML document well-formed. If an XML document is well-formed and conforms with a DTD or XML schema specification, it is called a valid XML document. Document Type Definitions (DTD) specify grammars for XML documents. However, DTDs provide limited support for specifying structure and also are not XML documents themselves. XML Schema was introduced to overcome these limitations. Besides using the XML syntax and providing a richer support for defining XML structure, XML schema has two other advantages, namely

---

<sup>11</sup><http://www.w3.org/standards/xml/>.

<sup>12</sup>[http://en.wikipedia.org/wiki/Document\\_Type\\_Definition](http://en.wikipedia.org/wiki/Document_Type_Definition).

<sup>13</sup><http://www.w3.org/TR/xmlschema-1/>.



**Fig. 6.2** RDF statement

its reusability and extended set of default datatypes. Regarding reusability, XML Schema can import and extend the terminology provided by other XML Schemas. In this context, XML Namespaces play a major role. XML Namespaces are used to qualify unique names in XML documents.

**Resource Description Framework (RDF)** [13] is a simple, yet powerful data model and language for describing Web resources. RDF introduces three fundamental concepts, namely resources, properties and statements. A resource represents the entity we want to make statements about. A resource may be anything denoted by a URI. A property in RDF defines the relationship between subject resources and object resources. Using RDF, one can make statements about Web resources in the form of triples (**subject, predicate, object**). Statements are RDF triples, where the subject and predicate are resources and the object can be a resource or a literal. An example of a RDF statement is available in Fig. 6.2. A literal can be either a plain literal, i.e., any text, or a type literal, where as types the XML Schema datatypes are recommended. Using the RDF model, triples can be combined in large graph structures. The nodes of an RDF graph are resources denoted by URIs, blank nodes, which are nodes without URIs, or literals. From the logic point of view, an RDF graph can be seen as a collection of facts where each triple represents a ground fact if the edges are URIs or literals, or existential quantified binary predicates, where existential qualified variables correspond to blank nodes.

Note that RDF properties are only binary predicates that relate subjects to objects. Information that usually would be expressed using  $n$ -ary predicates cannot be expressed by means of a single RDF statement but rather a set of statements that correspond to a set of binary predicates. Different interconnected triples lead to a more complex graphic model. Basically, an RDF document is a directly connected graph where the nodes are subjects or objects of the triples, the labels are the predicates and the edges are the RDF statements. The RDF graph representation of a set of RDF statements is useful especially for humans. For machines, other RDF representations are more useful especially when machine interoperability is the goal. Being understandable by machines also implies being interoperable and exchangeable across distributed and heterogeneous systems and applications. For this purpose, standardized serialization mechanisms are defined, some of them published as W3C recommendations, for converting the abstract RDF graph into a concrete exchange format. The most known serializations are: RDF/XML, Notation 3 (N3),<sup>14</sup>

---

<sup>14</sup>[http://en.wikipedia.org/wiki/Notation\\_3](http://en.wikipedia.org/wiki/Notation_3).

Turtle.<sup>15</sup> and N-Triples.<sup>16</sup> The RDF/XML serialization is machine-processable and XML compliant, enabling the easy exchange of information between very different types of systems and applications. Notation 3 (N3) was designed with human-readability in mind, being more compact and readable than the RDF/XML serialization. N-Triple is a minimal, plain text serialization format for RDF. Finally, Turtle (Terse RDF Triple Language) is a superset of N-Triples format providing much richer format.

RDF defines a set of complex data structures, i.e., containers and collections. An RDF container is an open set of resources or literals that contain unspecified numbers of elements and possibly duplicates. RDF defines three types of containers:

- **rdf:Bag** defines an unordered set of resources or literals.
- **rdf:Seq** defines an ordered set of resources or literals.
- **rdf:Alt** defines a set of alternatives, i.e., only one value can be selected.

An RDF collection is a closed set of resources or literals. By contrast to RDF containers, an RDF collection might contain duplicate elements. Additionally, RDF introduces the notion of a list **rdf:List**. The property **rdf:first** points to the first member of the list and a property **rdf:rest** recursively points to sublist containing the remaining members or to **rdf:nil** if the end of the list is reached; **rdf:nil** represents an empty list. To represent a complex data structure in RDF, blank nodes can be used to indirectly attach to a resource a consistent set of properties.

As mentioned previously, RDF can be used to create statements about Web resources. Furthermore, using RDF one can define statements about other RDF statements, a mechanism that is known as **reification**. Statements are regarded as resources identified using URIs from which other statements can be created. Reification statements can be used to define belief or trust in other statements. It can be used to deduce a level of confidence or degree of usefulness for each statement. Reification is a powerful mechanism that offers a high flexibility in modeling, but simultaneously creates problems when layering other languages on top of RDF. The major cause of this problem is that by using RDF reification, one can make arbitrary statements over statements and there is consequently no restriction of mixing the data and metadata levels. For example, the statement (**rdfs:Class rdf:type rdfs:Class**) states that class is an instance of a class, mixing the data and metadata levels. In general, the use of RDF reification is discouraged as the semantics of reification are unclear and as reified statements are quite cumbersome.

RDF not only provides a graph model to publish and link data on the Web, it also provides the foundational shared data model on which other capabilities are built: querying (SPARQL<sup>17</sup> is built on top of RDF), embedding (RDFa,<sup>18</sup> GRDDL<sup>19</sup> rely on the RDF model), and reasoning (RDFS and OWL are defined on top of RDF).

---

<sup>15</sup>[http://en.wikipedia.org/wiki/Turtle\\_syntax](http://en.wikipedia.org/wiki/Turtle_syntax).

<sup>16</sup><http://en.wikipedia.org/wiki/N-Triples>.

<sup>17</sup><http://www.w3.org/TR/rdf-sparql-query/>.

<sup>18</sup><http://www.w3.org/TR/xhtml-rdfa-primer/>.

<sup>19</sup><http://www.w3.org/2004/01/rdfh/spec>.

**RDF schema (RDFS)** [5] extends RDF by introducing means to model classes, properties, hierarchy of classes and properties, and simple domain and range restrictions. RDFS provides the following language constructs: **rdfs:Class**, **rdfs:subClassOf**, **rdfs:subPropertyOf**, **rdfs:domain**, and **rdfs:range**. It is a lightweight ontology language that can be used to create taxonomy like ontologies. The notion of the resource is introduced in RDFS using the **rdfs:Resource** construct. All resources are implicit instances of **rdfs:Resource**. RDFS also introduces the notion of class (**rdfs:Class**) that can be used to type resources. A class describes a set of resources, classes being resources themselves. To define relations between resources, RDFS provides the language constructs needed to define properties, i.e., **rdfs:domain** to specify the classes associated with a property, **rdfs:range** to specify the type of the property value, and **subPropertyOf** to define hierarchies of properties. The semantics of RDFS is based on sets and set operators (union intersection and inclusion). RDF already provides the membership declaration property **rdf:type** that allows one to capture in a graph structure which resource belongs to which class (a kind of set) and which couples of resources belong to which relation (another kind of set). RDFS provides the vocabulary to describe the relations between these sets; relations between classes, relations between properties and between classes and properties. Mixing the data and metadata levels is inherited by RDFS from RDF. There is no restriction on changing the semantics of the ontology modeling constructs, which makes layering of logical languages such as OWL on top of RDF/RDFS a difficult task. Another issue that hampers the proper layering and furthermore the integration of Semantic Web languages is the different assumptions these languages adhere to, i.e., closed world assumption or open world assumption. In a closed world, any missing information is considered as false, while in an open world missing information is treated as unknown. Furthermore, in languages or systems that adhere to the closed world assumption, the schema information behaves as constraints over the data. While RDF adheres to open world assumption, RDFS is agnostic in this respect, leaving at the application level the decision under which assumption the information should be interpreted. The RDFS approach on open vs. closed world assumption is more appropriate, allowing different logical languages to be more easily layered on top of RDFS than on top of RDF.

The **SPARQL Protocol and RDF Query Language (SPARQL)** [19] is the de-facto standard used to query RDF data. While RDF and the RDF Schema provide a model for representing Semantic Web data and for structuring semantic data using simple hierarchies of classes and properties, respectively, the SPARQL language and protocol provide the means to express queries and retrieve information from across diverse Semantic Web data sources. The natural question is why we need a new query language, given that XML query languages can be used to query over the XML serialization of an RDF graph. The need for a new language is motivated by the different data models and semantics at the level of XML and RDF, respectively. More precisely, the existing XML query languages, such as XPath<sup>20</sup> and

---

<sup>20</sup><http://www.w3.org/TR/xpath/>.

```
PREFIX ex: <http://example.org/> .
SELECT ?name
WHERE
{
  ex:john, ex:hasName, ?name .
}
```

**Listing 6.1** Simple SPARQL query

XQuery,<sup>21</sup> can be used to formulate queries dependent on the XML data structure that is queried. For example, the information encoded in the RDF triple (**ex:john**, **ex:hasName**, “**John Smith**”) can be syntactically represented in XML in multiple ways, and there consequently exist multiple query representations to search for John’s name, depending on the XML structure used to encode the information. By using RDF, we abstract from the multitude of syntactic representations in XML, and focus on the data and its meaning. The SPARQL query language was developed for the RDF layer of the Semantic Web architecture (see Fig. 6.1). The query language has been developed without considering the other core languages of the Semantic Web, namely RDFS, OWL and RIF.

The SPARQL is a matching graph pattern language. It defines a set of graph patterns, the simplest being the triple pattern. A triple pattern is like a normal RDF triple but with the possibility of a variable instead of an RDF term in the subject, predicate, or object positions. An example of a simple SPARQL query is provided in Listing 6.1.

SPARQL introduces the notion of **variable binding**, that is, a pair (**variable**, **RDF term**), where **variable** is a query variable of interest indicated by ? or \$ and the **RDF term** is the value assigned to the **variable** after the query has been executed. Similar to the namespace mechanism used for writing RDF/XML, SPARQL allows the definition of prefixes for namespaces. Prefixes are used inside a query to increase its readability. SPARQL introduced a set of constructs and clauses that could be part of a query. The SELECT clause identifies the variables to appear in the query results and the WHERE clause provides the basic graph pattern to match against the data graph. Additional constructs are FROM that is used to refer to the name of the graph to be queried, LIMIT to put an upper bound on the number of solutions returned, DISTINCT to eliminate duplicate solutions, REDUCED to permit but not to enforce duplicate solutions to be eliminated, OFFSET to specify that the solutions generated should start after the specified number of solutions, ORDER BY to establish the order of a solution sequence, etc.

More complex graph patterns can be formed by combining smaller patterns in various ways. SPARQL introduces four complex graph patterns, namely Group Graph Patterns, Optional Graph Patterns, Alternative Graph Patterns, and Patterns on Named Graphs. A Group Graph Pattern is a set of graph patterns that must all match. An Optional Graph Pattern defines additional patterns that may extend the solution. An Alternative Graph Pattern consists of two or more possible patterns

---

<sup>21</sup><http://www.w3.org/TR/xquery/>.

that are tried when executing the query. Finally, in the case of Patterns on Named Graphs, patterns are matched against named graphs.

*Graph patterns as part of SPARQL queries.* SPARQL has four query forms. These query forms use the solutions from pattern matching to form result sets or RDF graphs. The four query forms supported by SPARQL are the classical SELECT query which returns all, or a subset of, the variables bound in a query pattern match, CONSTRUCT which returns an RDF graph constructed by substituting variables in a set of triple templates, ASK which returns a boolean indicating whether a query pattern matches or not, and DESCRIBE which returns an RDF graph that describes the resources found.

The SPARQL language 1.0 discussed in this chapter does not provide support for aggregate functions such as counting, numerical min/max/average. It also lacks support for expressing subqueries, i.e., queries inside queries, negation as failure like in relation databases, project expressions, etc. The SPARQL Working Group is working towards defining a new version of SPARQL, i.e., SPARQL 1.1<sup>22</sup> that addresses the above mentioned limitations of SPARQL 1.0.

The backbone of the Semantic Web are **ontologies** [6]. An ontology, as defined in [9], is a “formal, explicit specification of a shared conceptualization”. The “specification” part of the previous definition indicates that an ontology is a concrete form of representing a conceptualization of a domain. An ontology is “formal” because it can be processed by machines represented using a formal language. Furthermore, an ontology is “explicit” because the knowledge encoded is explicitly specified. Finally, the “share” part of the definition previously quoted refers to an agreed consensus among the parties involved in the creation of the ontology. In a nutshell, an ontology is defined in terms of **concepts**, **properties** or **relations**, **axioms**, and **individuals**. Concepts are the core elements of the conceptualization corresponding to entities of the domain being described. Properties and relations are used to describe interconnections between concepts. Individuals are instantiations of the ontological concepts.

A set of formalisms and languages inspired by the work done in Knowledge Representation (e.g., Description Logics [1], Logic Programming [12, 14]) were created to formalize knowledge on the Semantic Web. Two of these formalisms, namely RDF(S) and OWL, were standardized by the W3C consortium.

OWL<sup>23</sup> which stands for **Web Ontology Language**, is actually the standard language for representing knowledge on the Web. This language was designed to be used by applications that need to process the content of information on the Web instead of just presenting information to human users [16]. Using OWL one can explicitly represent the meaning of terms in vocabularies and the relationships between those terms. In the Semantic Web stack architecture introduced in Fig. 6.1, OWL is part of the ontology language layer and is placed above the RDF and XML layers. Using OWL, one can express much richer relationships than with RDF(S), thus yielding the benefit of enhanced inference capabilities.

---

<sup>22</sup><http://www.w3.org/TR/sparql11-query/>.

<sup>23</sup><http://www.w3.org/2004/OWL/>.

OWL is based on Description Logics and has its origin in DAML+OIL ontology language [10], incorporating lessons learned from the design and application of DAML+OIL which is a combination of OIL [8] and DAML-ONTO [15] ontology languages.

When designing a language, there is always a trade-off between expressivity of the language and tractability of inference that can be done using that language. With OWL, this trade-off has been partly addressed through a layered approach. There are three OWL sub-languages: **OWL Lite**, **OWL DL** and **OWL Full**. Starting from the lower layer, **OWL Lite**, up to the upper layer, **OWL Full**, each sub-language is enriched in syntax and semantic.

OWL Full is layered syntactically and semantically on top of RDF(S) providing maximum expressivity, but with no guarantee of computational behavior. OWL DL restricts OWL Full to the part which offers completeness (a guarantee that an answer can and will be found) and decidability (a guarantee that the computation will end in a finite time). OWL Lite restrict OWL DL further to a language that can be used to express classification hierarchy and simple constraints. OWL Lite can be seen as a variant of the *SHIF(D)* description logic language, whereas OWL DL is a variant of the *SHOIN(D)* language [11]. The *SHIF(D)* language allows complex class descriptions, including conjunction, disjunction, negation, existential and universal value restrictions for roles, role hierarchies, transitive roles, inverse roles, a restricted form of cardinality restrictions (cardinality 0 or 1) and (limited) support for concrete domains. *SHOIN(D)* adds support for individuals in class descriptions and arbitrary cardinality constraints.

There are two major syntaxes for OWL. The most prominent one is based on RDF/XML syntax [2]. The second syntax, called abstract syntax [18] was developed for OWL DL and is similar to the syntax used in many Description Logics reasoners. The abstract syntax can only be used for the DL dialect of OWL. The RDF/XML syntax is more complex than the abstract syntax and not as intuitive to use.

The semantic of OWL assigns meaning to all variants of OWL. OWL has a direct model-theoretic semantics<sup>24</sup> which is based on DAML+OIL model-theoretic semantics. There are two ways of describing the semantic: as an extension of RDF(S) model theory and as a direct model-theoretic semantics of OWL.

**OWL2** [17] is the result of the latest development of the Web Ontology Language. A number of shortcomings were identified with respect to OWL 1, especially with the OWL Lite variant. OWL Lite has been defined as a much too expressive Description Logic. This hampered implementation of Lite reasoning based on existing semantic repository technology and also made layering rules on top of this language unfeasible. OWL 2 defines actually three new sublanguages: OWL2EL, OWL2QL and OWL2RL profiles. OWL2EL is particularly suitable for applications employing ontologies that define very large numbers of classes and/or properties. It provides polynomial time algorithms for all the standard reasoning tasks of description logic. OWL2QL enables efficient query answering over large instance populations, and OWL2RL restricts the expressiveness towards rule languages. This is currently the

---

<sup>24</sup><http://www.w3.org/TR/owl-semantics/>.

pathway that most industrial semantic repository developers follow and together with OWL2QL will probably define the most important Semantic Web representation languages from a technological point of view. The OWL2QL profile is designed so that sound and complete query answering is in LOGSPACE with respect to the size of the data (assertions), while providing many of the main features necessary to express conceptual models such as UML class diagrams and ER diagrams. It is designed so that data (assertions) stored in a standard relational database system can be queried through an ontology via a simple rewriting mechanism, i.e., by rewriting the query into an SQL query that is then answered by the RDBMS system, without any changes to the data.

The **Rule Interchange Format (RIF)**<sup>25</sup> is the W3C Recommendation that defines a framework to exchange rule-based languages on the Web. Like OWL, RIF defines a set of languages covering various aspects of the rule layer of the Semantic Web. More precisely RIF includes three dialects, a Core dialect which is extended into a Basic Logic Dialect (BLD) and Production Rule Dialect (PRD). RIF-BLD define a simple logic-oriented rule language, RIF-PRD captures most of the aspects of the production rule systems, and RIF-Core is the intersection between these languages. Essentially, RIF covers rule languages with different semantics, i.e., logic based rule languages with a declarative interpretation, and production based rule languages. The former usually have a declarative semantics in terms of a variation of a minimal Herbrand model and are an alternative model for databases called deductive databases. The latter usually only have an operation semantics and are used to express dynamic aspects of processes. They are much closer to being a kind of programming language based on a blackboard architecture and event triggers. Finally, these three dialects are complemented by The Framework for Logic Dialects (RIF-FLD) as a means to define new RIF dialects.

### 6.3 Extensions

**Linked Data** is a relatively new initiative that can be seen as a sub-topic and an extension of Semantic Web. It is used to “describe a method of exposing, sharing, and connecting data via dereferenceable URIs on the Web”.<sup>26</sup> The vision of Linked Data is primarily to publish structured data in RDF format. The focus is shifted from the ontological level and inference which are central for the classic Semantic Web, to provide simple principles and methods of publishing and linking data on the Web.

Introduced in [3] by the inventor of the Web, Sir Tim Berners-Lee, Linked Data is defined as RDF graphs, published so that they can be navigated across servers by following the links that interconnect these graphs. There are four simple design principles that need to be followed when publishing data as part of Linked Data:

- Using URIs as names for things.

---

<sup>25</sup>[http://www.w3.org/2005/rules/wiki/RIF\\_Working\\_Group](http://www.w3.org/2005/rules/wiki/RIF_Working_Group).

<sup>26</sup>[http://en.wikipedia.org/wiki/Linked\\_Data](http://en.wikipedia.org/wiki/Linked_Data).

- Using HTTP URIs so that people can look up those names.
- When someone looks up a URI, providing useful RDF information.
- Including RDF statements that link to other URIs so that they can discover related things one should follow.

Implementation of these principles has the potential to change the Web of documents as we know it today, into a Web of data. The Web of Data is envisioned as a global database consisting of objects and their descriptions in which objects are linked with each other with a high degree of object structure, with explicit semantics for links and content designed for humans and machines.

One community project that is realizing the idea of Linked Data and the Web of Data is the Linked Open Data Project.<sup>27</sup> Its goal is to extend the Web with a data commons by publishing various open data sets as RDF on the Web and by setting RDF links between data items from different data sources. Since the beginning of 2007, Linking Open Data project (LOD) has been growing continuously. In October 2007, datasets consisted of over two billion RDF triples, which were interlinked by over two million RDF links. By May 2009, this had grown to 4.2 billion RDF triples, interlinked by around 142 million RDF links. Currently the LOD cloud contains 203 data sets, 25 billion RDF triples and more than 400 million RDF links. Figure 6.3 depicts the Linking Open Data cloud as available in September 2010.

The central dataset of the LOD is DBpedia,<sup>28</sup> an effort to extract structured information from Wikipedia and make this information available on the Web as RDF. DBpedia is referred to within multiple datasets of the LOD cloud providing a high level of connectivity. Other linked LOD datasets are: Wordnet, the RDF representation of the most popular lexical knowledge base, Geonames<sup>29</sup> a worldwide geographical database, UniProt<sup>30</sup> the largest integrated database with protein and gene-related information, OpenCyc<sup>31</sup> the most popular upper-level and general ontology and many more.

As mentioned earlier regarding the publishing of data as part of LOD, four principles should be followed. A simple algorithm compliant with these principles would include the following simple steps:

- Select the relevant vocabularies and reuse existing vocabularies to increase the value of the data set and to increase interoperability.
- Partition the RDF data graph into data pages.
- Assign a URI to each data page.
- Create HTML variants for each data page to allow rendering of pages in browsers.
- Assign a URI to each entity (cf. “Cool URIs for the Semantic Web”).
- Add page metadata such as publisher, license, topics, etc., and link sugar.

---

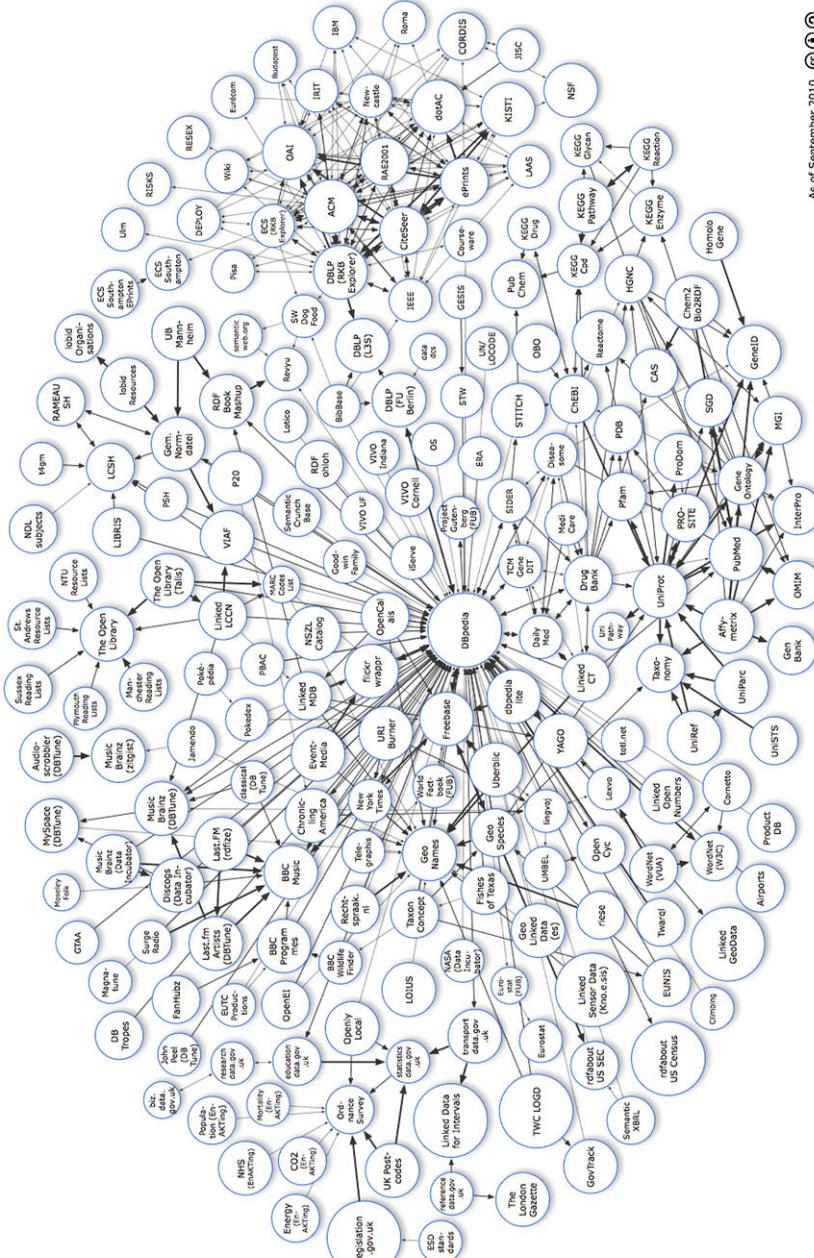
<sup>27</sup><http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData>.

<sup>28</sup><http://dbpedia.org/>.

<sup>29</sup>[www.geonames.org/](http://www.geonames.org/).

<sup>30</sup>[www.uniprot.org/](http://www.uniprot.org/).

<sup>31</sup><http://www.opencyc.org/>.



**Figure 6.3** Linking Open Data cloud, September 2010. Figure from <http://richard.cyganik.de/2007/10/lod/>

- Add a Semantic Sitemap to allow crawlers to find the data set or SPARQL end points to access the data set.

To support the development of LOD, various vocabularies and tools have been developed. voiD,<sup>32</sup> the “Vocabulary of Interlinked Datasets” is just such an approach that provides the terminology needed to describe LOD datasets. Using voiD one can specify what a dataset is about (topic, technical details), how and under what conditions it can be accessed, how the dataset is interlinked with other datasets, qualitative aspects of the dataset such as interlinking with other dataset as well as quantitative statistics such as the number of links, resources, etc. Other approaches and tools exist for bringing legacy data to the Web of Data and to make use of LOD data, i.e., to search, browse and mashup Linked Data.

To conclude, the Web of Data is a promising approach that extends the classical Semantic Web by transforming the Web from a Web of documents into a Web of data. The focus is on publishing data as RDF and on making links between datasets. A promising and interesting future direction to explore with respect to Linked Data is providing a service layer abstraction on top of the LOD layer.

## 6.4 Summary

This chapter provided a short overview of a challenging and continuously growing research area, called the Semantic Web, which aims to develop methods and technologies that enable processing, querying and integrating information on the Web in a transparent and automatic fashion. First, the success factors of the current Web as well as its limitations were discussed. In particular, finding, extracting and combining information are major limitations of the current Web. The solution proposed by the Semantic Web is to provide formal descriptions or machine-readable metadata that would enable automated agents and other software to access the Web more intelligently. First, the so-called Semantic Web Layer Cake that illustrates the Semantic Web architecture was introduced. The components and languages of the architecture introducing the basic concepts and aspects of each of language were discussed in more details. The foundation layer of the Semantic Web, which includes technologies and languages fundamental to Web itself, was introduced. Unicode, the Uniform Resource Identifier (URI), the Extensible Markup Language (XML), the Document Type Definition (DTD) and the XML Schema are also part of this large layer. The languages that are part of the core of Semantic Web were presented in details, including the Resource Description Framework (RDF) and RDF Schema (RDFS) as data model and simple ontology language, SPARQL as the standard query language for RDF data, the Web Ontology Language (OWL) version 1 and 2 and the Rule Interchange Format (RIF). The chapter concludes with a discussion on extensions of the current and the Semantic Web towards a larger, more open and interconnected Web of data, known as Linked Open Data.

---

<sup>32</sup><http://semanticweb.org/wiki/VoiD>.

```

<rdf:RDF
  xmlns:j.0="http://xmlns.com/foaf/0.1/"
  xmlns:j.1="http://www.deri.org/person#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <rdf:Description rdf:about="http://www.dieter.fensel">
    <j.0:depiction>http://www.deri.at/images/members/dieter_fensel.jpg</j.0:depiction>
    <j.1:workStreetAddress>Technikerstrasse 21a</j.1:workStreetAddress>
    <j.0:givenname></j.0:givenname>
    <j.0:mbox>dieter.fensel@st2.at</j.0:mbox>
    <j.0:homepage>http://www.fensel.com</j.0:homepage>
    <j.1:middle_name></j.1:middle_name>
    <j.1:workOrganization>University of Innsbruck</j.1:workOrganization>
    <j.1:workDepartment>Institute of Computer Science</j.1:workDepartment>
    <j.1:workPostalCode>A6020</j.1:workPostalCode>
    <j.1:workFaxNumber>+43 512 507 9872</j.1:workFaxNumber>
    <j.1:workCity>Innsbruck</j.1:workCity>
    <j.0:name>Univ. Prof. Dr. Dieter Fensel</j.0:name>
    <j.1:workPhoneNumber>+43 512 507 6485, 6488</j.1:workPhoneNumber>
    <j.1:workCountry>Austria</j.1:workCountry>
    <j.0:title></j.0:title>
    <j.0:family_name>Fensel</j.0:family_name>
  </rdf:Description>
</rdf:RDF>

```

**Listing 6.2** RDF annotation representing a person

## 6.5 Exercises

**Exercise 1** Please, answer these questions to the best of your knowledge.

1. What novelty and advantages does the Semantic Web bring as compared to Web 1.0 and Web 2.0?
2. What are the conceptual differences between the data model of XML and the data model of RDF? Compare the features and application areas of supporting APIs for XML and RDF.
3. Compare query languages XPATH and SRARQL. What are the main concepts and similarities, what are the main differences? Can you recognize relations to Relational Database Query languages such as SQL?
4. What does OWL add on top of RDF/RDFS? In what sense is OWL DL more restrictive than RDFS?

**Exercise 2** Find an OWL ontology, for instance, using Swoogle Semantic Web search engine (<http://swoogle.umbc.edu.>), and explain at least five different concepts and properties in the ontology by using the relations and axioms you can find in the ontology itself. Pick an ontology which uses some constructs not expressible in RDFS.

**Exercise 3** Clarify in which ways the following RDF annotation representing a Person (Listing 6.2) does not comply with the FOAF schema. What are the changes that need to be done to make this annotation compliant with the FOAF specification?

**Exercise 4** Answer the questions about SPARQL queries.

- Given the following data and SPARQL query, what result will the query deliver?

Data:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Anna" ;
foaf:mbox <mailto:anna@example.net> .
_:b foaf:name "loan" .
```

Query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name WHERE {
?node foaf:name ?name .
?node foaf:mbox ?mbox .
}
```

- Given the following data and SPARQL query, what result will the query deliver?

Data:

```
@prefix dc10: <http://purl.org/dc/elements/1.0/> .
@prefix dc11: <http://purl.org/dc/elements/1.1/> .
_:a dc10:title "SPARQL Query Language Tutorial" .
_:b dc11:title "SPARQL Query Language (2nd ed)" .
_:c dc10:title "SPARQL" .
_:c dc11:title "SPARQL" .
```

Query:

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?title
WHERE { { ?book dc10:title ?title } UNION
{ ?book dc11:title ?title } }
```

- What does the following SPARQL query perform?

```
CONSTRUCT { ?friend pim:fullName ?name .
?friend foaf:mbox ?mbox }
WHERE { ?person foaf:knows ?friend .
?friend foaf:given ?name .
?friend foaf:mbox ?mbox .
FILTER regex(?person, "Anna") .}
```

## References

- Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
- Beckett, D.: RDF/XML syntax specification (revised). Recommendation 10 February 2004, W3C (2003)
- Berners-Lee, T.: Design issues: linked data. Technical rep. (2006). Available at <http://www.w3.org/DesignIssues/LinkedData.html>

4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. *Scientific American* **284**(5), 34–43 (2001)
5. Brickley, D., Guha, R.V.: RDF vocabulary description language 1.0: RDF schema. W3C Recommendation. Technical rep. (2004). Available at <http://www.w3.org/TR/rdf-schema/>
6. Fensel, D.: *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer, Berlin (2003)
7. Fensel, D., Musen, M.A.: The Semantic Web: A brain for humankind. *IEEE Intelligent Systems* **16**(2), 24–25 (2001)
8. Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F.: OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems* **16**(2) (2001)
9. Gruber, T.R.: A translation approach to portable ontology specifications. *Knowledge Acquisition* **5**(2), 199–220 (1993)
10. Horrocks, I., van Harmelen, F.: Reference description of the DAML+OIL (March 2001) ontology markup language. Technical report, DAML (2001). <http://www.daml.org/2001/03/reference.html>
11. Horrocks, I., Patel-Schneider, P.F.: Reducing OWL entailment to description logic satisfiability. In: Proc. of the 2003 International Semantic Web Conference (ISWC 2003), Sanibel Island, Florida (2003). <http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/HoPa03b.pdf>
12. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. *Journal of the ACM* **42**(4), 741–843 (1995)
13. Klyne, G., Carroll, J.J.: Resource description framework (RDF): concepts and abstract syntax. W3C Recommendation. Technical rep. (2004). Available at <http://www.w3.org/TR/rdf-concepts/>
14. Lloyd, J.W.: *Foundations of Logic Programming*, 2nd edn. Springer, Berlin (1987)
15. McGuinness, D.L.: Ontologies come of age. In: Fensel, D., Hendler, J., Lieberman, H., Wahlster, W. (eds.) *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, pp. 171–194. MIT Press, Cambridge (2003). Chap. 6
16. McGuinness, D.L., van Harmelen, F.: Owl web ontology language overview (February 2004). Technical report, W3C (2004). <http://www.w3.org/TR/owl-features/>
17. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 web ontology language profiles. Recommendation 27 October 2009, W3C (2009)
18. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL web ontology language semantics and abstract syntax. Recommendation 10 February 2004, W3C (2004)
19. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation. Technical rep. (2008). Available at <http://www.w3.org/TR/rdf-sparql-query/>

**Part II**

**Web Service Modeling Ontology Approach**



# Chapter 7

## Web Service Modeling Ontology

**Abstract** This chapter presents the Web Service Modeling Ontology, an ontology for describing various aspects related to Semantic Web Services. WSMO is the major initiative in the area of Semantic Web Services in Europe. The starting point of WSMO was the Web Service Modeling Framework (WSMF) (Fensel and Bussler in *Electronic Commerce Research and Applications* 1(2):113–137, 2002), which was refined and extended, resulting in (i) a formal ontology, the Web Service Modeling Ontology (WSMO), (ii) a formal family of languages, the Web Service Modeling Language (WSML), and (iii) an execution environment, the Web Service Modeling Environment (WSMX). The four main elements of WSMF have been adopted and refined in WSMO, resulting in four main elements for describing Semantic Web Services: (i) ontologies that provide the terminology used by other elements, (ii) goals that define the problems that should be solved by Web services, (iii) Web services descriptions that define various aspects of a Web service, and (iv) mediators which facilitate the resolution of interoperability problems. WSMO is a key technology needed for the successful realization of Semantic Web Services and Semantically Enabled Service-oriented Architectures. It provides a conceptual model for Semantic Web Services that combines the design principles of the Web, the Semantic Web and distributed service oriented computing in order to provide a clean modeling solution for various aspects related to Semantic Web Services.

### 7.1 Motivation

Current Web service technologies around SOAP [11], WSDL [3], and UDDI [1] operate at the syntactic level. Therefore, although they support interoperability between the many diverse application development platforms that exist today through common standards, they still require human interaction to a large extent: the human programmer has to manually search for appropriate Web services in order to combine them in a useful manner. Having a human programmer in the loop that needs to take care of various service related tasks limits the scalability of service-oriented solutions and greatly curtails the added economic value of envisioned with the advent of Web services [4]. In order to automate tasks such as Web service discovery, composition and execution, semantic descriptions of Web services are required. Semantic Web Services are building on top of Web services technology by describing

various aspects of services using explicit, machine-understandable semantics that enables a certain degree of automation for various service related tasks. In a nutshell, work in the area of Semantic Web [2] is being applied to Web services in order to keep the intervention of the human user to a minimum. WSMO provides a fully-fledged framework for Semantic Web Services. It includes a conceptual model, a formal language with a formal syntax and semantics based on different logics in order to support different levels of logical expressiveness, and an execution environment that is able to discover, select, compose and invoke Semantic Web Services modeled and formalized in WSMO. This chapter introduces one of the most important approaches to Semantic Web Services, namely the Web Service Modeling Ontology.

## 7.2 Technical Solution

The WSMO approach to Semantic Web Services includes: (i) the Web Service Modeling Ontology (WSMO), a conceptual model for Semantic Web Services, (ii) the Web Service Modeling Language (WSML), a language providing a formal syntax and semantics for WSMO, and (iii) the Web Service Execution Environment (WSMX), an execution environment for WSMO descriptions formalized in WSML. The rest of this section describes in detail the WSMO conceptual model. The other two building blocks of the WSMO approach, namely the Web Service Modeling Language (WSML) and the Web Service Execution Environment (WSMX), will be described in Chaps. 8 and 9, respectively.

WSMO [7] provides ontological specifications for the core elements of Semantic Web Services being based on the following design principles:

- **Web Compliance**

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

- **Ontology Based**

Ontologies are used as the data model throughout WSMO, meaning that all resource descriptions as well as all data interchanged during service usage are based on ontologies. 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 regarding 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 heterogeneities that naturally arise in open environments. Heterogeneities can occur in terms of data, underlying ontology, protocol, and 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, operate in specific contexts which will not be the same as for available Web services. For example, a user may wish to book a holiday according to preferences for 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 available services.

- **Description versus Implementation**

WSMO distinguishes between the descriptions of Semantic Web Services elements (description) and executable technologies (implementation). The former requires a concise and 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.

- **Execution Semantics**

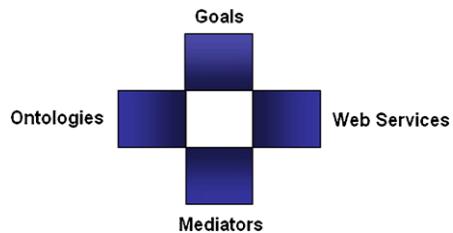
In order to verify the WSMO specification, the formal execution semantics of reference implementations such as WSMX as well as other WSMO-enabled platforms provide the technical realization of WSMO. This principle serves as a means to precisely define the functionality and behavior of the systems that are WSMO-compliant.

- **Service versus Web service**

A Web service is a computational entity that is able to achieve a user goal by invocation. A service, in contrast, is the actual value provided by this invocation. WSMO provides the 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.

Based on these design principles, the WSMO meta-model has four top level elements which can be seen in Fig. 7.1, namely Ontologies, Web Services, Goals, and Mediators. To effectively describe Semantic Web Services, we need to understand each of these four elements. WSMO makes use of the Meta-Object Facility (MOF) [6] specification to define its meta-model. The MOF provides a language and framework for specifying technology neutral meta-models. The benefit of using MOF is that the model and the languages that are ultimately used to describe Semantic Web Services are separated from one another. This separation gives significantly more freedom than with competing approaches such as OWL-S (see Chap. 11). In WSMO, every element can be annotated using the WSMO annotations. One of the most used standards for annotations is the Dublin Core (DC) Metadata Set [12]. In the following sections, we use the MOF specification to describe the four top level elements of the WSMO metamodel and their child elements.

**Fig. 7.1** WSMO top level elements



```

Class ontology
hasAnnotations type annotations
importsOntology type ontology
usesMediator type ooMediator
hasConcept type concept
hasRelation type relation
hasFunction type function
hasInstance type instance
hasRelationInstance type relationInstance
hasAxiom type axiom
  
```

**Listing 7.1** Ontology definition

### 7.2.1 *Ontologies*

Ontologies in WSMO provide the terminology used across the descriptions of all other WSMO elements. Ontologies are crucial to the success of Semantic Web Services, as they provide the means by which enhanced information processing becomes possible and complex interoperability problems can be solved. WSMO is an epistemological model in that it is general enough to intuitively capture existing languages used for describing ontologies. In Listing 7.1 a WSMO ontology is described using the MOF notation.

#### 7.2.1.1 **Annotations**

It is possible to add annotations onto all WSMO elements. They include aspects such as the creator, the creation date, the version, etc. The elements defined by the Dublin Core Metadata Initiative [12] are taken as a starting point. Dublin Core is a set of attributes that define a standard for cross-domain information resource description. WSMO uses IRIs for identification of elements. This is also true of annotations which are specified as key value pairs, where the key is an IRI identifying the property, and the value is another element.

#### 7.2.1.2 **Imported Ontologies**

All top levels elements within the WSMO meta-model may use the **importsOntology** statement to import ontologies that contain the relevant concepts needed to

```

Class concept
hasAnnotations type annotations
hasSuperConcept type concept
hasAttribute type attribute
hasDefinition type logicalExpression multiplicity = single-valued

Class attribute
hasAnnotations type annotations
hasRange type concept multiplicity = single-valued

```

**Listing 7.2** Concept definition

build a description, for example, a WSMO Web service description will import those WSMO Ontologies that contain concepts needed to describe the service in question.

### 7.2.1.3 Mediators for Importing Ontologies

Of course, it may not always be possible to directly import a given ontology, as mismatches between statements in the importing and imported ontology or between any two given imported ontologies may exist. In this case, a mediator is required, and the ontology is imported via this mediator. This mediator deals with the alignment, merging and transformation of the imported ontology so that existing heterogeneity issues that may arise are fixed. Like the **importsOntology** statement, the **usesMediator** statement may be used on all top level elements within the WSMO meta-model.

### 7.2.1.4 Concepts

Concepts represent the basic agreed upon terminology for a given domain. A concept is made up of a set of attributes, where each attribute has a name and a type. Using the MOF notation, the definition of a concept is depicted in Listing 7.2.

A concept can also be a sub-concept of one or more direct super-concepts using the **hasSuperConcept** statement. Sub-concepts inherit the signatures of all its super-concepts, for example, all attributes defined for a given concept will also hold for any of its sub-concepts.

Using the **hasAttribute** statement a set of attributes on the concept can be specified. The MOF definition of an attribute is also specified in Listing 7.2. The range of an attribute can be constrained to a given concept, meaning that all instances of the concept that specify a value for this attribute must conform to this range restriction.

Furthermore, a concept can be further refined by adding a logical expression through the **hasDefinition** statement. This logical expression can be used to express additional constraints on the concept than cannot be captured through the **hasAttribute** or **hasSuperConcept** statement.

```

Class relation
hasAnnotations type annotations
hasSuperRelation type relation
hasParameter type parameter
hasDefinition type logicalExpression multiplicity = single-valued

Class parameter
hasAnnotations type annotations
hasDomain type concept multiplicity = single-valued

```

**Listing 7.3** Relation definition

```

Class function sub-Class relation
hasRange type concept multiplicity = single-valued

```

**Listing 7.4** Function definition

### 7.2.1.5 Relations and Functions

Relations and functions define connection between several concepts. The arity of a relation is not restricted, thus a relation is able to model dependencies between two or more concepts. The MOF definition of a relation is presented in Listing 7.3.

Each relation can have zero or more super-relations. A relation inherits the signature of the super relation along with any associated constraints. Similarly to attributes for concepts, a relation can define a set of parameters, which may be a named or an ordered unnamed set. It is possible to define the domain of each of the parameters, where the domain specifies the allowed values that can be placed in this slot of the relation when it is instantiated. WSMO Ontologies can also have functions, specified with the **hasFunction** statement. Functions, as described in Listing 7.4, are special relations with a unary range, specified along with the parameters (the domain). Functions can be used to represent built in predicates of common data types. The semantics of a relation or function can be captured within a logical expression specified with the **hasDefinition** statement.

### 7.2.1.6 Instances of Concepts and Relations

As can be seen in Listing 7.5, it is possible to instantiate both concepts and relations defined within a WSMO Ontology. When instantiating a concept or relation, values are assigned to the attributes or parameters of the concept or relation being instantiated, where the type of the value being assigned conforms to the range of the attribute or the domain of the parameter. Instances may be defined explicitly within the ontology, however, in general a link to an external store of instances will be given when a large number of instances exist.

```

Class instance
hasAnnotations type annotations
hasType type concept
hasAttributeValues type attributeValue

Class relationInstance
hasAnnotations type annotations
hasType type relation
hasParameterValue type parameterValue

```

**Listing 7.5** Instance definitions

```

Class axiom sub-Class wsmoElement
hasDefinition type logicalExpression

```

**Listing 7.6** Axiom definition

```

Class service
hasAnnotations type annotations
hasNonFunctionalProperties type nonFunctionalProperties
importsOntology type ontology
usesMediator type {ooMediator, wwMediator}
hasCapability type capability multiplicity = single-valued
hasInterface type interface

```

**Listing 7.7** Web service definition

### 7.2.1.7 Axioms

An axiom, as showed in Listing 7.6, is a logical expression together with its non-functional properties. Axioms provide a mechanism for adding arbitrary logical expressions to an ontology, and can be used to refine concepts, relation or function definitions, to add arbitrary axiomatic domain knowledge, or to express constraints.

### 7.2.2 Web Services

Web services are computational entities that provide some functionality that has an associate value in a given domain. A WSMO Web service is a formal description of the Web services functionality, in terms of a capability, and the method to interact with it, in terms of an interface. A formal description of a WSMO Web service using the MOF notation given in Listing 7.7.

The ontologies that are required in order to define the service can be imported via the **importsOntology** and **usesMediator** statements. The **usesMediator** statement may also be used with a **wwMediator** (see Sect. 7.2.4.4), in cases where process or protocol heterogeneity issues need to be resolved.

```
Class nonFunctionalProperty
  hasAnnotations type annotations
  hasDefinition type axiom
```

**Listing 7.8** Non-functional property definition

```
Class capability
  hasAnnotations type annotations
  hasNonFunctionalProperties type nonFunctionalProperties
  importsOntology type ontology
  usesMediator type ooMediator
  hasPrecondition type axiom
  hasAssumption type axiom
  hasPostcondition type axiom
  hasEffect type axiom
```

**Listing 7.9** Capability definition

### 7.2.2.1 Non-Functional Properties for Web Services

A service can specify one or more non-functional properties. Listing 7.8 contains the MOF definition of a non-functional property in WSMO.

In a nutshell, the non-functional properties captured constraints over the functional and behavioral properties of the service. Typical examples of non-functional properties include temporal and locative availability of the service, security, rights, obligations, reliability, performance, etc. Non-functional properties are not specific to Web services only, but other WSMO elements descriptions (i.e., goal, mediators, capabilities and interfaces) can contain non-functional properties descriptions.

### 7.2.2.2 Capabilities

The capability of the service describes the real value of the service and is described in MOF in Listing 7.9.

The capability of a service contains a set of axioms that describe the state of the world before and after the execution of the service. Using the **hasPrecondition** and **hasPostcondition** statements, it is possible to make axiomatic statements about the expected inputs and outputs of the service, i.e., what information must be available for the service to be executed and what information will be available after the service has been executed. The **hasAssumption** and **hasEffect** statements can be used to state the assumed state of the world prior to execution and the guaranteed state of the world afterwards. The capability of a service can be used by a requester for discovery purposes, i.e., to determine if the functionality of the service meets the requesters functional needs.

```
Class interface
hasAnnotations type annotations
hasNonFunctionalProperties type nonFunctionalProperties
importsOntology type ontology
usesMediator type ooMediator
hasChoreography type choreography
hasOrchestration type orchestration
```

**Listing 7.10** Interface definition

```
Class choreography
hasAnnotations type annotations
hasStateSignature type stateSignature
hasState type state
hasTransitionRules type transitionRules
```

**Listing 7.11** Choreography definition

### 7.2.2.3 Interfaces

The interface of a service describes how the functionality can be achieved. It is described in MOF as depicted in Listing 7.10.

The interface of a service provides a dual view on the operational competence of the service. Using the **hasChoreography** statement, a decomposition of the capability in terms of interaction with the service is provided, while using the **hasOrchestration** statement the capability can be decomposed in terms of the functionality required from other services in order to realize this service. This corresponds to the distinction between communication and cooperation. The choreography describes how to interact with a service, while the orchestration describes how the overall function of the service is realized through cooperation with other services (see Listing 7.11). The interface of a service is presented in a machine processable manner, allowing for software to automatically determine the behavior of the service and to reason on it.

Both choreography and orchestration are defined using the same formalism, based on Abstract State Machines [5]. The choreography of a service is defined in MOF as follows.

The most important parts of the definition are the state signature and the transition rules. The state signature defines the state ontology used by the service together with the definition of the types of modes the concepts and relations may have which describes the service's and the requestor's rights over the instances. The transition rules express changes of states by changing the set of instances [8].

Applying different transition rules, the choreography evolves from the initial state which technically is the same as the state signature, if not otherwise specified, to the final state, going through several intermediate states. In the final state, no further updates based on transition rules can be applied.

The orchestration has a similar definition as the choreography, with the main difference being that the choreography considers two participants in a conversation, while the orchestration is a description of the cooperation of multiple participants.

```
Class goal
hasAnnotations type annotations
hasNonFunctionalProperties type nonFunctionalProperties
importsOntology type ontology
usesMediator type {ooMediator, wwMediator}
requestsCapability type capability multiplicity = single-valued
requestsInterface type interface
```

**Listing 7.12** WSMO goal class

```
Class mediator
hasAnnotations type annotations
hasNonFunctionalProperties type nonFunctionalProperties
importsOntology type ontology
hasSource type {ontology, goal, webService, mediator}
hasTarget type {ontology, goal, webService, mediator}
hasMediationService type {webService, goal, wwMediator}
```

**Listing 7.13** Mediator definition

### 7.2.3 Goals

A goal specifies the objectives that a client may have when consulting a Web service, describing aspects related to user desires with respect to the requested functionality and behavior. Ontologies are used as the semantically defined terminology for goal specification. Goals model the user's view in the Web service usage process, and therefore are a separate top-level entity in WSMO. The goal class is described in Listing 7.12.

### 7.2.4 Mediators

Mediators describe elements that resolve interoperability problems between different elements, e.g., between two ontologies or two services. Mediators are a core element of WSMO and aim to resolve heterogeneity problems at the data, process and protocol levels. The definition of a mediator using the MOF notation is depicted in Listing 7.13.

Like all WSMO elements, a mediator can define a set of annotations using the **hasAnnotations** statement. Furthermore, the terminology needed from other ontologies to define this mediator can be imported using the **importsOntology** statement. The source component of a mediator defines the resources for which the heterogeneities are resolved, while the target component defines the resources that receive these mediated source components. A mediation service can be used to define the facility applied for performing the mediation using the **hasMediationService** statement. This service may explicitly link to a Web service description or may link to a Goal describing the functionality needed, which can then be resolved to a Web service at runtime using service discovery. There are four types of mediators within the WSMO metamodel.

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

**Listing 7.14** ooMediator definition

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

**Listing 7.15** ggMediator definition

#### 7.2.4.1 Ontology-to-Ontology Mediators

Ontology-to-Ontology Mediators (ooMediators), as defined in Listing 7.14, provide a mechanism by which mismatches between two or more ontologies can be resolved. The source component of an ooMediator is an ontology or another ooMediator, and the source defines the resources for which mismatches will be resolved by the mediator. The target of an ooMediator can be an ontology, goal, Web service or a mediator, and the target defines the target component which will receive the results of the mediation.

As described in Sect. 7.2.1, ooMediators are used across all WSMO top-level elements within the **usesMediator** statement in order to import the terminology required by a resource description whenever there are mismatches between the ontologies to be used.

#### 7.2.4.2 Goal-to-Goal Mediators

Goal-to-Goal Mediators (ggMediators), as defined in Listing 7.15, connect goals to another, allowing for relationships between different goals to be specified. For instance, a ggMediator can be used to specify that one goal is equivalent to another goal, and that the source goal is a refinement of the target.

The source of a ggMediator can be a goal or another ggMediator, which is also true for the target of a ggMediator. By specifying that a ggMediator is the source or target, one can build chains of mediators.

#### 7.2.4.3 Web Service-to-Goal Mediators

Web Service-to-Goal Mediators (wgMediators), defined in Listing 7.16, or Goal-to-Web Service Mediators, provide a mechanism for expressing relationships between Web services and goals. Primarily, wgMediators are used to pre-link services to goals, or to cache the results of previously performed discovery actions. For example, a wgMediator may be used to express that a given Goal can be totally or partially fulfilled by a given Web service.

```
Class wgMediator sub-Class mediator
  usesMediator type ooMediator
  hasSource type {webService, goal, wgMediator, ggMediator}
  hasTarget type {webService, goal, ggMediator, wgMediator}
```

**Listing 7.16** wgMediator definition

```
Class wwMediator sub-Class mediator
  usesMediator type ooMediator
  hasSource type {webService, wwMediator}
  hasTarget type {webService, wwMediator}
```

**Listing 7.17** wwMediator definition

#### 7.2.4.4 Web Service-to-Web Service Mediators

Web Service-to-Web Service Mediators (wwMediators), defined in Listing 7.17, provide a mechanism for expressing relationships between Web services. These relationships could include stating that two Web services provide equivalent functionality, a group of Web services provides equivalent functionality to another Web service, or that one Web service is a refinement of another.

wwMediators can also be used to establish interoperability between Web services in cases where they would otherwise be not interoperable, i.e., a wwMediator could be used to specify a mediation between the choreography of two Web services, where mediation can involve the data, protocol or process levels.

### 7.3 Extensions

In this section, we briefly introduce a number of extensions for modeling services that were inspired by the WSMO approach. One characteristic that is common to all classical approaches for Semantic Web Services, including WSMO, is the top-down modeling. However, top-down modeling decouples the Semantic Web Service model from the actual syntactic description which contains where the service is located and how exactly it can be invoked. Another approach that gained more momentum is the bottom-up approach for modeling that is adopted by a number of extensions for modeling services.

Current extensions to WSMO conceptual model include those that are following the bottom-up path such as WSMO-Lite, a lightweight ontology for service semantics, Semantic Annotations for WSDL (SAWSDL) that provides an semantic annotation mechanisms allowing the specification of pointers to semantic concepts from within a syntactic description of a WSDL service and MicroWSMO that basically has the same purpose as SAWSDL but for RESTful services. SAWSDL and MicroWSMO build on top of syntactic representations formalisms, namely WSDL and hRESTS. The approaches mentioned above are known as lightweight for services given the fact that they “add a bit of semantics” on top of existing non-semantic

standards. More details about each of the mentioned lightweight approaches for Semantic Web Services are provided in Chap. 12. In the rest of this section, we briefly summarize the main features of the lightweight approaches.

WSMO-Lite is a simple, yet powerful ontology that can be used to model all important aspects of a service. It distinguishes between four type of semantics, i.e., functional, non-functional, behavioral, and information model. Furthermore, WSMO-Lite provides an RDF Schema ontology to express the four types of semantics mentioned previously. The WSMO-Lite ontology can be used to semantically annotate both WSDL-based and RESTful-based services. In the former case, the approach used to provide hooks for attaching semantics is SAWSDL, while the later is MicroWSMO.

SAWSDL is a set of extensions for WSDL.<sup>1</sup> The two major constructs provided by SAWSDL are **model references** that can be used to point to semantic concepts and **schema mappings** that can be used to specify data transformations between the XML data structure of messages and the associated semantic model. SAWSDL by itself, however, does not specify any actual types of semantics.

MicroWSMO can be seen as a direct realization of SAWSDL annotations over hRESTS. As SAWSDL, MicroWSMO provides a set of elements that can be used to point to semantic descriptions. These include **model**, **lowering** and **lifting** which can be associated with the underlying hREST description.

## 7.4 Illustration by a Larger Example

In this section, we use the “Virtual Traveling Agency” scenario introduced in Chap. 4 to exemplify how services can be semantically annotated using the WSMO approach. In this scenario, a customer uses the Virtual Traveling Agency (VTA) application to access eTourism services. The reminder of this section illustrates how a concrete service offers online train booking tickets, as well as how related ontologies, goals, and mediators can be modeled using WSMO. We use the Web Service Modeling Language (WSML) to represent all of these elements. The modeling example presented here is partially based on content available in [9].

### 7.4.1 *Ontologies*

We use the following domain ontologies: **International Train Ticket**, **Temporal**, **Locative**, **Price**, **Quality**, **Provider**, **Measures**.

Six of the ontologies mentioned above, namely **Temporal**, **Locative**, **Price**, **Quality**, **Provider**, **Measures**, are generic ontologies containing non-functional properties models. These ontologies have been defined in [10].

---

<sup>1</sup>Web Services Description Language, <http://www.w3.org/TR/wsdl20/>.

The “International Train Ticket” ontology defines a train trip and the surrounding concepts shown in Listing 7.18. The definition of the ontology is based on the travel itinerary ontology from the DAML ontology library.<sup>2</sup> This ontology captures information about travel itineraries for trips by plane. Our ontology reuses the itinerary and flight concepts and adapts them to define train trips, also introducing new concepts such as the train station. The International Train Ticket ontology also makes use of the person ontology<sup>3</sup> which defines a subset of vCard. The person concept is used to define the passenger information for an itinerary.

### 7.4.2 Goals

Goals denote what a user wants as the result of a Web service. To exemplify how goals are modeled in WSMO, we define in Listing 7.19 a generic goal for buying a ticket for any kind of trip. Please note that in WSMO, goals can be defined on different levels of granularity. Through ggMediators, new, more specific goals can be created out of generic existing goals. Generic goals can also be thought of as being pre-defined in a specific application context, from which concrete Goals can be generated.

### 7.4.3 Web Services

For exemplification, let us consider an end-user service (a service with which the user interacts) for purchasing international train tickets offered by the Austrian national train operator OEBB; this Web service can be composed of other Web services, each for the search and purchasing facility of international train tickets. This setting allows for the correct modeling of all notions of a WSMO Web service description: a capability of the end-user service and its choreography for user-service interaction, as well as the orchestration which incorporates the aggregated Web services.

A Web service capability in WSMO is described by pre- and postconditions, assumptions and effects, as defined in [7]. Listing 7.20 shows the OEBB Web service. The capability of the OEBB Web service includes:

- **Precondition**

This input has to include information about the buyer, a train trip that a ticket is to be purchased for, and information on the passenger for whom the ticket shall be valid. To be a valid input, the following restrictions are defined for the trip: the start and end locations are restricted to stations in Austria or Germany;

---

<sup>2</sup><http://www.daml.org/ontologies/ontologies.html>.

<sup>3</sup><http://daml.umbc.edu/ontologies/ittalks/person>.

```
wsmiVariant _"http://www.wsmo.org/wsmi/wsmi—syntax/wsmi—rule"

namespace { _"http://www.wsmo.org/ontologies/trainConnection",
  dc _"http://purl.org/dc/elements/1.1#",
  prs _"http://www.wsmo.org/2004/d3/d3.3/v0.1/20041008/resources/owlPersonMediator.wsmi"
  xsd _"http://www.w3.org/2001/XMLSchema#",
  wsmi _"http://www.wsmo.org/wsmi/wsmi—syntax#",
  loc _"http://www.wsmo.org/ontologies/nfp/locativeNFPOntology#",
  temp _"http://www.wsmo.org/ontologies/nfp/temporalNFPOntology#",
  qua _"http://www.wsmo.org/ontologies/nfp/qualityNFPOntology#",
  po _"http://www.wsmo.org/ontologies/nfp/providerNFPOntology#",
  prc _"http://www.wsmo.org/ontologies/nfp/priceNFPOntology#",
  mes _"http://www.wsmo.org/ontologies/nfp/measuresNFPOntology#"
}

ontology _"http://www.wsmo.org/ontologies/trainConnection"
  annotations
    dc#title hasValue "International Train Ticket Ontology"
    dc#creator hasValue "STI Innsbruck"
    dc#subject hasValues {"Train", "Itinerary", "Train Connection", "Ticket"}
    dc#description hasValue "Domain Ontology International Train Ticket"
    dc#format hasValue "text/html"
    dc#identifier hasValue _"http://www.wsmo.org/ontologies/trainConnection"
    dc:contributor hasValues {"Michael Stollberg", "Ruben Lara",
      "Holger Lausen", "Ioan Toma"}
    dc#language hasValue "en-US"
    wsmi#version hasValue "Revision : 2.0"
  endAnnotations

  importsOntology {_ "http://www.wsmo.org/ontologies/nfp/locativeNFPOntology",
    _ "http://www.wsmo.org/ontologies/nfp/temporalNFPOntology"}

  usedMediators
  {_ "http://www.wsmo.org/2004/d3/d3.3/v0.1/20041008/resources/owlPersonMediator.wsmi"}

  concept station subConceptOf loc#GeoLocation
    annotations
      dc#description hasValue "Train station"
    endAnnotations
    code ofType _string
      annotations
        dc#description hasValue "Code of the station"
      endAnnotations
    borderToCountry ofType loc#Border
      annotations
        dc#description hasValue "For stations located at the border"
      endAnnotations

  concept ticket
    annotations
      dc#description hasValue "a ticket for an itinerary"
    endAnnotations
    itinerary ofType itinerary
    provider ofType po#Provider
    price ofType prc#AbsoultePrice

  concept itinerary
    annotations
      dc#description hasValue "An itinerary between two locations"
    endAnnotations
    passenger ofType prs#person
```

**Listing 7.18** Domain ontology “International Train Ticket”

```

annotations
dc#description hasValue "prs#person is a subset of vCard (http://www.ietf.org/rfc/
rfc2425.txt)"
endAnnotations
recordLocatorNumber ofType _string
trip ofType trip

concept trip
start ofType loc#LocativeEntity
end ofType loc#LocativeEntity
via ofType loc#RouteSpecification
departure ofType temp#TimePoint
arrival ofType temp#TimePoint
duration ofType temp#TimeInterval
distance ofType mes#Distance

concept trainTrip subConceptOf trip
annotations
dc#description hasValue "A train trip"
endAnnotations
start ofType station
end ofType station
via ofType station
seat ofType _string
train ofType _string
class ofType _string

axiom departureBeforeArrival
annotations
dc#description hasValue "Integrity Constraint: departure has to be before arrival"
endAnnotations
definedBy
!— ?T[
    departure hasValue ?D,
    arrival hasValue ?A
]memberOf trip
and ?A <= ?D.

axiom startNotEqualEnd
annotations
dc#description hasValue "Integrity Constraint: the start and end of a trip have to be
different"
endAnnotations
definedBy
!— ?T[
    start hasValue ?Start,
    end hasValue ?End
]memberOf trip
and ?Start = ?End.

```

**Listing 7.18** (Continued)

the departure date for the trip has to be after the current date; and the payment method of the buyer has to be a credit card.

- **Assumption**

The credit card submitted as input has to be valid (not expired).

- **Postcondition**

The service returns a purchase for train tickets valid in Austria and Germany by the OEBB as the provider, with payment by credit card only.

- **Effect**

```

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"

namespace {
    _"http://www.wsmo.org/example/VTA/goals/GeneralTrainTrip#",
    dc _"http://purl.org/dc/elements/1.1#",
    tc _"http://www.wsmo.org/ontologies/trainConnection#",
    temp _"http://www.wsmo.org/ontologies/nfp/temporalNFPontology#",
    loc _"http://www.wsmo.org/ontologies/nfp/locativeNFPontology#",
    pay _"http://www.wsmo.org/ontologies/nfp/paymentNFPontology#",
    xsd _"http://www.w3.org/2001/XMLSchema#",
    wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
    po _"http://www.wsmo.org/ontologies/purchase#"}
}

goal "http://www.wsmo.org/example/VTA/goals/GeneralTrainTrip.wsml"

annotations
    dc#title hasValue "Buying a ticket online"
    dc#subject hasValues {"Tickets", "Online Ticket Booking", "trip"}
    dc#creator hasValue "STI Innsbruck"
    dc#description hasValue "Express the goal of buying a ticket for a trip"
    dc#contributor hasValues {"Michael Stollberg", "Ruben Lara", "Holger Lausen", "Ioan Toma"}
    dc#format hasValue "text/html"
    dc#language hasValue "en-US"
    dc#coverage hasValues {tc#austria, tc#germany}
    dc#version hasValue "Revision : 2.0"
endAnnotations

importsOntology {_"http://www.wsmo.org/ontologies/trainConnection#",
    _"http://www.wsmo.org/ontologies/nfp/locativeNFPontology#",
    _"http://www.wsmo.org/ontologies/purchase#"}

postcondition
    axiom purchasingTicketForTrip
        annotations
            dc#description hasValue "This goal expresses the general desire of buying a ticket
                for
                any kind of itinerary ."
        endAnnotations
        definedBy
            exists ?Purchase, ?Purchaseorder, ?Buyer, ?Product, ?PaymentMethod, ?Ticket,
                ?Itinerary, ?Passenger, ?Trip
            (?Purchase memberOf po#purchase[
                po#purchaseorder hasValue ?Purchaseorder,
                po#buyer hasValue ?Buyer
            ] and
            ?Buyer memberOf po#buyer and
            ?Purchaseorder memberOf po#purchaseOrder[
                po#product hasValues {?Product},
                po#payment hasValue ?PaymentMethod
            ] and
            ?PaymentMethod memberOf po#paymentMethod and
            ?Product memberOf po#product[
                po#item hasValues {?Ticket}
            ] and
            ?Ticket memberOf tc#ticket[
                tc#itinerary hasValue ?Itinerary
            ] and
            ?Itinerary memberOf tc#itinerary[
                tc#passenger hasValue ?Passenger,
                tc#trip hasValue ?Trip
            ]
        }
    
```

**Listing 7.19** Goal—buying a ticket online

```

] and
?Passenger memberOf tc#Person and
?Trip memberOf tc#trip) .

effect
axiom havingTradeForTrip
annotations
    dc#description hasValue "The goal effect is to get the purchased ticket delivered
        to the buyer."
endAnnotations
definedBy
exists ?Delivery, ?Product, ?Buyer, ?Ticket
(
?Delivery memberOf po#delivery[
    po#deliveryItem hasValues {?Product},
    po#receiver hasValue ?Buyer
] and
?Product memberOf po#product[
    po#item hasValues {?Ticket}
] and
?Buyer memberOf po#buyer and
?Ticket memberOf tc#ticket
) .

```

**Listing 7.19** (Continued)

The sold ticket is delivered to the buyer's shipping address, either by a drop ship carrier or via online delivery.

#### 7.4.4 Mediators

As mentioned in Sect. 7.2.4, mediators play a significant role in the WSMO modeling as they are responsible for solving the heterogeneity problem. WSMO defines four types of mediators, namely ooMediators, ggMediators, wgMediators and ww-Mediators. Listing 7.21 contains a description of an ooMediator that imports other ontologies or ooMediators into WSMO entities and resolves possible terminology mismatches. If no mismatch has to be resolved, the syntactical simplification “importOntologies” can be used.

## 7.5 Summary

Semantic Web Services are one of the most promising research directions to improve the integration of applications within and across enterprise boundaries. In this context, this chapter provided an overview of one of the most important approaches to Semantic Web Services, namely the Web Service Modeling Ontology. We argue that, in order for Semantic Web Services to succeed, a fully fledged framework needs to be provided: starting with a conceptual model, continuing with a formal

```

wsmlVariant _"http://www.wsmo.org/wsml/wsml--syntax/wsml--rule"

namespace {
    _"http://www.wsmo.org/example/VTA/services/OEBBService.wsml#",
    dc _"http://purl.org/dc/elements/1.1#",
    tc _"http://www.wsmo.org/ontologies/trainConnection#",
    temp _"http://www.wsmo.org/ontologies/nfp/temporalNFPontology#",
    loc _"http://www.wsmo.org/ontologies/nfp/locativeNFPontology#",
    pay _"http://www.wsmo.org/ontologies/nfp/paymentNFPontology#",
    xsd _"http://www.w3.org/2001/XMLSchema#",
    wsml _"http://www.wsmo.org/wsml/wsml--syntax#",
    po _"http://www.wsmo.org/ontologies/purchase#"
}

webservice
"http://www.wsmo.org/example/VTA/services/OEBBService.wsml"

annotations
dc#title hasValue "OEBB Online Ticket Booking Web Service"
dc#creator hasValue "STI Innsbruck"
dc#description hasValue "web service for booking online train tickets for Austria and Germany"
dc#contributor hasValues {"Michael Stollberg", "Ruben Lara", "Holger Lausen", "Ioan Toma"}
dc#format hasValue "text/html"
dc#language hasValue "en-US"
dc#coverage hasValues {tc#austria, tc#germany}
dc#version hasValue "Revision : 2.0"
endAnnotations

importsOntology {_"http://www.wsmo.org/ontologies/trainConnection#",
_"http://www.wsmo.org/ontologies/nfp/temporalNFPontology#",
_"http://www.wsmo.org/ontologies/nfp/locativeNFPontology#",
_"http://www.wsmo.org/ontologies/nfp/paymentNFPontology#",
_"http://www.wsmo.org/ontologies/purchase#"}

capability oebbWSCapability
precondition
    axiom oebbWSprecondition
        annotations
            dc#description hasValue "The oebbWSprecondition puts the following conditions ont
                the input: it has to include a buyer with a billTo and a shipTo address, and credit card as a paymentMethod, and trip with the start-and end-location have to be in Austria or in Germany, and the departure date has to be later than the current date."
        endAnnotations
definedBy
forAll ?Buyer, ?BuyerBilltoAddress, ?BuyerShiptoAddress, ?BuyerPaymentMethod,
?Trip, ?Start, ?End, ?Departure (
    ?Buyer memberOf po:buyer[
        po#billToAddress hasValue ?BuyerBilltoAddress,
        po#shipToAddress hasValue ?BuyerShiptoAddress,
        po#hasPaymentMethod hasValues {?BuyerPaymentMethod}
    ] and
    ?BuyerBilltoAddress memberOf loc#Address and
    ?BuyerShiptoAddress memberOf loc#Address and
    ?BuyerPaymentMethod memberOf pay#CreditCard and
    ?Trip memberOf tc#trainTrip[
        tc#start hasValue ?Start,
        tc#end hasValue ?End,
        tc#departure hasValue ?Departure
]
)

```

**Listing 7.20** OEBB web service for booking online train tickets for Austria and Germany

```

] and
(?Start.locatedIn = austria or ?Start.locatedIn = germany) and
(?End.locatedIn = austria or ?End.locatedIn = germany) and
temp#after(?Departure,currentDate)
).

postcondition
axiom oeBBWSpostcondition
annotations
    dc#description hasValue "the output of the service is a purchase for a
        ticket for train trips wherefore
        the start – and end – location have to be in Austria or in Germany and
        the departure date has to be later than the current Date."
endAnnotations
definedBy
forAll ?Purchase, ?Seller, ?OEBBContactInformation, ?Purchaseorder,
    ?PaymentMethod, ?Product, ?Ticket,
    ?Itinerary , ?Trip, ?Start, ?End
(
    ?Purchase memberOf po#purchase[
        purchaseorder hasValue ?Purchaseorder,
        seller hasValue ?Seller
    ] and
    ?Seller memberOf po#seller[
        contactInformation hasValue ?OEBBContactInformation,
        acceptsPaymentMethod hasValues {?PaymentMethod}
    ] and
    ?OEBBContactInformation memberOf po#contactInformation[
        name hasValue "Oesterreichische Bundesbahn",
        emailAddress hasValue "office@oebb.at",
        physicalAddress hasValue ?OEBBAddress
    ] and
    ?OEBBAddress memberOf loc#Address[
        streetAddress hasValue "Hauptfrachtenbahnhof 4",
        city hasValue innsbruck,
        country hasValue austria
    ] and
    ?Purchaseorder memberOf po#purchaseOrder[
        product hasValues {?Product},
        payment hasValue ?PaymentMethod
    ] and
    ?PaymentMethod memberOf pay#CreditCard and
    ?Product memberOf po#product[
        item hasValues {?Ticket}
    ] and
    ?Ticket memberOf tc#ticket[
        itinerary hasValue ?Itinerary
    ] and
    ?Itinerary memberOf tc#itinerary[
        trip hasValue ?Trip
    ] and
    ?Trip memberOf tc#trainTrip[
        start hasValue ?Start,
        end hasValue ?End,
        departure hasValue ?Departure
    ] and
    (?Start.locatedIn = austria or ?Start.locatedIn = germany) and
    (?End.locatedIn = austria or ?End.locatedIn = germany) and
    temp#after(?Departure,currentDate)
).

```

**Listing 7.20** (Continued)

```

effect
axiom oebbWSeffect
  annotations
    dc:description hasValue "the sold ticket is delivered to the buyer via a
      drop ship carrier or via email."
  endAnnotations
  definedBy
    forAll ?Delivery, ?Product, ?Buyer, ?BuyerShipToAddress, ?Seller,
      ?OEBBContactInformation, ?OEBBAddress
    (
      ((?Delivery memberOf po:dropShipDelivery[
        deliveryItem hasValues {?Product},
        receiver hasValue ?Buyer,
        sender hasValue ?Seller,
        carrier hasValue PostAt
      ]) or
      (?Delivery memberOf po#onlineDelivery[
        deliveryItem hasValues {?Product},
        receiver hasValue ?Buyer,
        onlineDeliveryMethod hasValue "email"
      ]) and
      ?Product memberOf po#product[
        item hasValues {?Ticket}
      ] and
      ?Buyer memberOf po#buyer[
        shipToAddress hasValue ?BuyerShipToAddress
      ] and
      ?BuyerShipToAddress memberOf loc#Address and
      ?Seller memberOf po#seller[
        contactInformation hasValue ?OEBBContactInformation
      ] and
      ?OEBBContactInformation memberOf po#contactInformation[
        name hasValue "Oesterreichische Bundesbahn",
        emailAddress hasValue "office@oebb.at",
        physicalAddress hasValue ?OEBBAddress
      ] and
      ?OEBBAddress memberOf loc#Address[
        streetAddress hasValue "Hauptfrachtenbahnhof 4",
        city hasValue innsbruck,
        country hasValue austria
      ]
    ).
  
```

```

interface oebbWSInterface
  annotations
    dc:description hasValue "describes the Interface of Web service"
  endAnnotations
  choreography ***
  orchestration ***

```

**Listing 7.20** (Continued)

language to provides formal syntax and semantics (based on different logics in order to provide different levels of logical expressiveness) for the conceptual model, and ending with an execution environment that glues together all components that use the language for performing various tasks that would eventually enable automation of the service. The WSMO approach tackles, in a unifying manner, all the aspects of such a framework, and potentially provides the conceptual basis and the technical means for realizing Semantic Web Services: it defines a conceptual model

```

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"

namespace {
    _"http://www.wsmo.org/example/VTA/mediators/owlAddressMediator#",
    dc _"http://purl.org/dc/elements/1.1#",
    loc _"http://www.wsmo.org/ontologies/nfp/locativeNFPOntology#",
    wsml _"http://www.wsmo.org/wsml/wsml-syntax#"
}

ooMediator
"http://www.wsmo.org/example/VTA/mediators/owlAddressMediator.wsml"

annotations
    dc#title hasValue "OO Mediator importing the OWL Factbook ontology to WSM"
    dc#creator hasValue "STI Innsbruck"
    dc#description hasValue "Mediator to import an OWL address ontology into a WSM locations
        ontology"
    dc#contributor hasValues {"Michael Stollberg", "Ruben Lara", "Holger Lausen", "Ioan Toma"}
    dc#format hasValue "text/html"
    dc#language hasValue "en-US"
    dc#version hasValue "Revision : 2.0"
endAnnotations

importsOntology {_ "http://www.wsmo.org/ontologies/trainConnection#",
    _ "http://www.wsmo.org/ontologies/nfp/temporalNFPOntology#",
    _ "http://www.wsmo.org/ontologies/nfp/locativeNFPOntology#",
    _ "http://www.wsmo.org/ontologies/nfp/paymentNFPOntology#",
    _ "http://www.wsmo.org/ontologies/purchase#"}

source _"http://daml.umbc.edu/ontologies/italks/address"

target _"http://www.wsmo.org/ontologies/nfp/locativeNFPOntology"

useService
_"http://138.232.65.151:8080/TranslatorService/OWL2WSML/"

```

**Listing 7.21** OO-Mediator “importing the OWL address ontology to the location ontology”

(WSMO) for defining the basic concepts of Semantic Web Services, a formal family of languages (WSML) (see Chap. 8) which provides a formal syntax and semantics for WSMO by offering different variants based on different logics in order to provide different levels of logical expressiveness (thus allowing different trade-offs between expressivity and computability), and an execution environment (WSMX) (see Chap. 9) which provides a reference implementation for WSMO and interoperability of Semantic Web Services. This chapter has focused on the WSMO conceptual model. The following two chapters discuss the other constituents of the approach, namely the formal family of languages, WSML, and the execution environment, WSMX. To conclude, the clean conceptual model and the design principles of the Web that underly the approach, make WSMO a key contribution towards realizing the vision of Semantic Web Services and Semantically Enabled Service-oriented Architectures.

## 7.6 Exercises

**Exercise 1** Extend the WSMO ontology available in Listing 7.18 with other elements that capture knowledge about bus transportation. You should add at least

one example of concept, relation, function, concept instance, relation instance, and axiom.

**Exercise 2** Consider a bus transportation company as one company that provides tourism services as part of the VTA scenario. Create a WSMO description for the bus transportation service. The resulting description should include description of the capability, including pre-conditions, post-conditions, assumptions, and effects. Explain in your own words what the difference between these elements is using your example.

**Exercise 3** Discuss differences between a service and a Web service in the WMSO approach. Give an example of service and one of Web service using in the hotel booking sub-domain of the VTA scenario.

## References

1. Bellwood, T., Clément, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y.L., Januszewski, K., Lee, S., McKee, B., Munter, J., von Riegen, C.: UDDI Version 3.0 (2002)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. *Scientific American* **284**(5), 34–43 (2001). <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&ref=sciam>
3. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. (2001). <http://www.w3.org/TR/wsdl>
4. Fensel, D., Bussler, C.: The Web Service Modeling Framework (WSMF). *Electronic Commerce Research and Applications* **1**(2), 113–137 (2002)
5. Gurevich, Y.: Evolving Algebras 1993: Lipari guide. In: Börger, E. (ed.) *Specification and Validation Methods*, pp. 9–37. Oxford University Press, London (1994). <http://citeseer.ist.psu.edu/gurevich93evolving.html>
6. Object Management Group Inc. (OMG): Meta Object Facility (MOF) Specification v1.4. (2002). <http://www.omg.org/technology/documents/formal/mof.htm>
7. Roman, D., Lausen, H. (eds.): U.K.: Web Service Modeling Ontology (WSMO). Working Draft D2v1.2, WSMO, (2005). Available from <http://www.wsмо.org/TR/d2/v1.2/>
8. Roman, D., Scicluna, J., Nitzsche, J.: Ontology-based choreography. Deliverable D14v1.0., WSMO (2007). Available from <http://www.wsмо.org/TR/d14/v1.0/>
9. Stollberg, M., Lausen, H., Keller, U., Zaremba, M., Haller, A., Fensel, D., Kifer, M.: D3.3 v0.1 WSMO use case virtual travel agency. Working Draft D3.3v0.1, WSMO (2004). Available from <http://www.wsмо.org/2004/d3/d3.3/v0.1/>
10. Toma, I., Foxvog, D.: Non-functional properties in Web services. Working draft, Digital Enterprise Research Institute (DERI), August (2006). Available from <http://www.wsмо.org/TR/d28/d28.4/v0.1/>
11. W3C: SOAP Version 1.2 Part 0: Primer (2003)
12. Weibel, S., Kunze, J., Lagoze, C., Wolf, M.: Dublin Core metadata for resource discovery. RFC 2413, IETF (1998)



# Chapter 8

## The Web Service Modeling Language

**Abstract** Several languages have been developed to describe ontologies, e.g., OWL, KIF, OIL, F-Logic. They have different characteristics and different underlying logics that make them more suitable to certain types of use, e.g., OWL is perfect for describing hierarchies. In some cases, they have been used to describe Semantic Web Services, e.g., OWL-S is an ontology expressed in OWL to describe Semantic Web Services, but they have not been designed with this purpose in mind. In this chapter, we provide an introduction to the Web Service Modeling Language (WSML), a language for the specification of different aspects of Semantic Web Services. First, we discuss the motivation behind the introduction of WSML, then we analyze the WSML language in detail providing examples for each of its constructs. This analysis is followed by a large example showing a fully fledged set of Semantic Web Services descriptions based on WSML. Finally, we provide a set of exercises to practice WSML.

### 8.1 Motivation

In the previous chapters, we have discussed the importance of Semantic Web Services to enable automate tasks, such as discovery, mediation, selection, composition, and invocation of services. To achieve this vision, providing support for Web services description is fundamental. Specifically, in Chap. 7, we introduced the Web Service Modeling Ontology (WSMO). WSMO identifies the conceptual elements required for such descriptions, thereby providing a framework for Web service description. Nevertheless, nothing was said about a concrete formal language to support such description and the properties it should possess. In this section, we discuss these requirements in light of the WSMO framework, current available ontology languages and logic frameworks.

#### 8.1.1 Principles of WSMO

The Web Service Modeling Ontology WSMO [13, 29], introduced in detail in Chap. 7, provides a conceptual model for the description of Web services. A language to support a conceptual framework needs to reflect its fundamental principles.

In particular, for the discussion in this chapter, the most relevant WSMO principles are: **Web Compliance**, **Ontology-based**, **Separation of requester and provider roles**, and **Strict decoupling of resources**. All of the principles were extensively explained in the previous chapter.

Following this prior discussion, WSMO further identifies four main top-level elements: **ontologies**, **Web services**, **goals**, and **mediators**.

A language that implements WSMO should support the formal description of the aforementioned elements, while simultaneously respecting the WSMO fundamental principles. Furthermore, in order to properly support WSMO concepts of service and goals, a language needs to be able to describe both functional and behavioral, as well as non-functional aspects of every element.

### **8.1.2 Logics Families and Semantic Web Services**

The reference language proposed by W3C to define ontologies, the Web Ontology Language (OWL) [22], is based on Description Logics (DLs) [1]. Description Logics is a family of logic-based knowledge representation formalisms that corresponds to a 2-variable decidable fragment of First Order Logic.

Practical experience in building applications has revealed several shortcomings of OWL, and hence Description Logics for modeling knowledge. In particular, some of the most prominent features missing in OWL are [25]:

- **Higher Relational Expressivity.** While OWL allows describing the concepts through a rich set of primitives, its capability to express knowledge about roles is very reduced, and does not correspond to the sets of primitives required by real world applications. Roles in Description Logic (DL) can be described only as pairs since it is a 2-variable based logic, while in many applications more complex roles require more complex definitions. For example, the knowledge that “an uncle of a person is a male sibling of one of the parents of that person” cannot be formalized in DLs.
- **Polyadic Predicates.** OWL supports only unary and binary predicates, while in many real world applications, n-ary predicates are required. For example, the predicate “the individual X bought the book Y from the seller Z” is a 3-ary predicate. This predicate cannot be directly modeled in DLs. One possible solution is to create a fourth term K, representing the predicate and linking the other terms to it. (i.e., “the purchase K has buyer X, seller Z and object Y”).
- **Close-World Reasoning.** OWL and DLs are based on the Open-World Assumption (OWA), i.e., what is not currently known to be true is undecided (neither true nor false). This assumption often collides with the needs of applications for which the Closed-World Assumption (i.e., what is not currently known to be true is false) is more appropriate. Let us consider a relation that represents the train connection between two cities and instances of this relation only for the actual existing connections; in the first case, when there is no instance of this relation between two cities, we cannot say whether the two cities are connected or not; in the second case, we are sure that the two cities are not connected by any train.

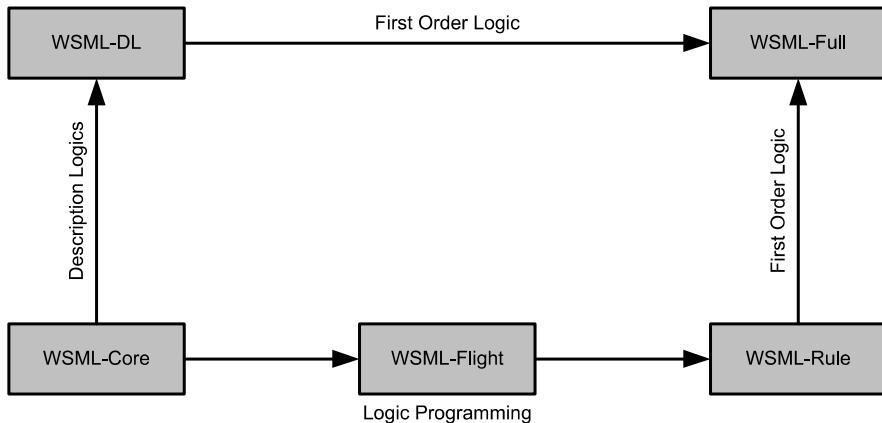
- **Integrity Constraints.** Integrity constraints cannot be easily expressed in OWL. This is due the fact that OWL adopts the Open-World Assumption and does **not** adopt the Unique Names Assumption (UNA) (i.e., two resources with different identifiers are different resources). For example, if we consider a formalization where we define the concept “flight seat” that has a property called “hasPassenger” with only one value of type passenger, then if we have two people (e.g., Mary and John) assigned to the same seat by mistake, in OWL this means that Mary and John are the same individual.
- **Modeling Exceptions.** In many practical situations, the exceptions modeling is crucial. For example, most birds are able to fly, but exceptionally some of them do not have this ability. Such a domain cannot be modeled in OWL: the axioms Bird and CanFly, CannotFly and Bird, and CanFly or CannotFly make the concept CanFly unsatisfiable.

The above list provides some insight into the limitations of OWL. Beside these limitations, we can also observe that in many situations, in order to model the domain of an application, we need to be able to express complex rules. OWL by itself does not provide any rule definition support. In the context of Web services, rules are important to enable the formal definition of services functionalities like “this service can be invoked only if the credit card is valid, and a credit card is valid if it is not expired and its monthly maximum credit amount was not reached”. To provide rule support for OWL, the Semantic Web Rule Language (SWRL) [17] was defined. SWRL is a simple extension of OWL with material (first-order) implication and due to the straightforward way in which the rules are integrated with OWL, it is trivially undecidable (close-world reasoning). Furthermore, SWRL cannot address tasks such as integrity constraints, since it was designed as a first-order language.

A proper combination of Description Logics with Logic Programming, enables the resolution of these problematics. At the same time, it constitutes a useful formal language paradigms for ontology description and knowledge representation on the Semantic Web [19]. The formal properties, as well as reasoning algorithms, for both paradigms have been thoroughly investigated in the respective research communities (Description Logic and Logic Programming), and efficient reasoning implementations are available for both paradigms. A formal language for WSMO should leverage the research that has been done in both areas, and the implementations that are available, by catering for these language paradigms.

The difference in the expressiveness and underlying assumptions should be overcome by defining a means for interaction between descriptions. On the one hand, it is desirable to use a common subset of both paradigms for such interaction (cf. [14]) so that it is not necessary to compromise on computational properties and so that existing implementations may be used. On the other hand, using a common subset requires compromising on expressiveness, which is not desirable in many situations; a common superset would include the expressiveness of both paradigms, but would require compromising on computational properties such as decidability (cf. [21]).

In the next section, we illustrate, based upon the previous discussion, how the WSML language was designed.



**Fig. 8.1** The WSML variants [15]

## 8.2 Technical Solution

A formal description of Web services, as well as user goals, has three major characteristics: static background knowledge in the form of ontologies, the functional description of the service (suitable for discovery and high-level composition), and the behavioral description of the service (suitable for selection, mediation, composition, and invocation). In this chapter, we present a language framework addressing all three aspects—the Web Service Modeling Language (WSML) that constitutes the formal grounding of the Web Service Modeling Ontology introduced in Chap. 7.

In order to address the problem of ontology description, the Web Service Modeling Language incorporates the Description Logic and Logic Programming formal language paradigms and the RDF Schema and OWL Semantic Web ontology languages. With regards to the functional description of services, WSML relies upon a flexible framework based on Abstract State Spaces, which can be combined with a number of logical languages. Finally, WSML addresses the problem of behavioral description with a flexible expressive language that has its conceptual roots in Abstract State Machines.

In the following paragraphs, we discuss these aspects in more details.

### 8.2.1 WSML Language Variants

In Sect. 8.1.2, we discussed the need for a formalism based on both Description Logic (DL) and Logic Programming (LP) paradigms. WSML responds to this need by incorporating a number of different language **variants** corresponding to DL and LP.

Figure 8.1 shows the WSML language variants and their inter-relationships. The variants differ in logical expressiveness and underlying language; they allow users

to make a trade-off between the expressiveness of a variant and the complexity of reasoning for ontology modeling on a per-application basis. Additionally, different variants are more suitable for different kinds of reasoning and representation.

- **WSML-Core** is at the intersection between the Description Logic  $\mathcal{SHIQ}$  and Horn Logic, also known as Description Logic Programs [14], and it has the lowest expressive power of the WSML variants. Every WSML-Core description is also a WSML-DL and WSML-Flight/Rule description. The main features of the language are the support for modeling classes, attributes, binary relations and instances. Furthermore, the language supports class hierarchies, as well as relation hierarchies. WSML-Core provides support for datatypes and datatype predicates.
- **WSML-DL** is the Description Logic variant of WSML. It captures the Description Logic  $\mathcal{SHIQ(D)}$ , which corresponds to a large part of (the DL species of) OWL [10]. This variant allows for interoperability with OWL DL ontologies: WSML-DL ontologies, goals, and Web services may import OWL DL ontologies.
- **WSML-Flight** is the least expressive of the two LP-based variants of WSML. Compared to WSML-Core, it adds features such as meta-modeling, constraints, and nonmonotonic negation. WSML-Flight is based on a Logic Programming variant of F-Logic [20] and is semantically equivalent to Datalog with inequality and (locally) stratified negation [28]. This variant allows for interoperability with RDF graphs and RDFS ontologies: WSML-Flight ontologies, goals, and Web services may import RDF graphs and RDFS ontologies.
- **WSML-Rule** extends WSML-Flight with further features from Logic Programming, namely the use of function symbols, unsafe rules, and unstratified negation, in order to increase the language expressibly.
- **WSML-Full** unifies WSML-DL and WSML-Rule. A definition of the semantics for WSML-Full, generalizing WSML-DL and WSML-Rule, is proposed in [6, 7]. However, as the combination of Description Logics and nonmonotonic logic programs is still an open research issue—some work has been done on the topic [4, 5, 8, 12, 24, 25, 30], but there is no consensus on which is the most appropriate semantics for the combination. Therefore, the current version of WSML does not define a semantics for WSML-Full, but instead outlines a number of properties the semantics should exhibit.

WSML has two alternative layering paths: WSML-Core  $\Rightarrow$  WSML-DL  $\Rightarrow$  WSML-Full and WSML-Core  $\Rightarrow$  WSML-Flight  $\Rightarrow$  WSML-Rule  $\Rightarrow$  WSML-Full. For both layering paths, WSML-Core and WSML-Full mark the least and most expressive layers. The two layering paths are to a certain extent disjoint, namely the inter-operation between WSML-DL, on the one hand, and WSML-Flight and -Rule, on the other, is only possible through a common subset (WSML-Core) or through a very expressive superset (WSML-Full). More details on WSML layerings can be found in [9].

In the next sections, we provide an overview of the WSML language through an introduction of its syntax and vocabulary. In particular, we discuss WSML by presenting its user friendly syntax, that presents WSML in terms of a normative surface syntax with keywords based on WSMO. A detailed discussion of the other syntaxes of WSML, namely XML and RDF, is presented in [9].

### 8.2.2 WSM Basis

As already introduced, WSM aims at modeling both conceptual elements and related logical rules. To provide proper support for both modeling aspects, the WSM syntax is composed by two parts: the **conceptual syntax** and the **logical expression syntax**.

The conceptual syntax reflects the WSMO conceptual model and it is independent from the particular WSM variant. The WSM conceptual syntax is shared between all variants, with the exception of some restrictions that apply to the modeling of ontologies in WSM-Core and WSM-DL. It allows for the modeling of ontologies, Web services, goals and mediators. The logical expression syntax provides access to the full expressive power of the language underlying the particular variant. The basic entry points for logical expressions in the conceptual syntax are the axioms in ontologies, the assumptions, preconditions, postconditions, and effects in capability descriptions, and the transition rules in choreography descriptions.

WSM is a “Web language” and according to this design principle adopts Web standards in its syntax. In particular, aside from including an XML and RDF syntax, WSM adopts IRIs [11] for the identification of objects and resources, uses the XML schema datatypes [2] for typing concrete data values, and reuses the XQuery [16] comparators and functions for corresponding built-in predicates.

#### 8.2.2.1 IRIs and WSM

WSM distinguishes between three types of identifiers: IRIs, compact IRIs (i.e., IRIs based on namespaces) and data values. Compact IRIs are specific to the surface syntax, and do not appear in the XML and RDF serialization.

An **IRI** (Internationalized Resource Identifier) [11] uniquely identifies a resource (e.g., concepts, relations, individual) in a Web-compliant way. IRI is a standard evolution of the popular URI standard, and constitutes one of the building blocks of the Web architecture [18]. In the surface syntax IRIs are delimited using an underscore and double quote ‘\_”, and a double quote “”, for example, \_“<http://www.wsmo.org/wsm/wsm-syntax#>”.

An IRI can be abbreviated to a compact IRI, which is of the form **prefix#localname** to improve readability. In case the prefix and separator ‘prefix#’ are omitted, the name belongs to the default namespace. Both the namespace prefixes and the default namespace are specified in the prologue of a WSM document (see Listing 8.1 for an example). The header also includes the definition of the WSM flavor adopted by the WSM document.

A WSM compact IRI corresponds to the use of Qualified Names (QNames) in RDF which is slightly different from QName in XML. While in XML a QName is merely an abbreviation for an IRI, in RDF and WSM it is a tuple <namespaceURI, localname>.

WSM enables the use of anonymous identifiers. An anonymous identifier represents an IRI which is meant to be globally unique. Unnumbered anonymous IDs

are denoted with ‘\_#’ and each occurrence denotes a new anonymous ID. Different occurrences of ‘\_#’ are unrelated. Numbered anonymous IDs are denoted with ‘\_#n’ where n stands for an integer denoting the number of the anonymous ID. Anonymous identifiers are disallowed for the following elements: the top-level elements ontology, goal, webService, mediator, capability, interface, choreography and orchestration.

### 8.2.2.2 WSML Datatypes

Datatypes in WSML are inherited from XML. They include strings, integers, decimals, or structured datatypes (e.g., dates and XML content). Strings are Unicode character sequences delimited with double quotes (e.g., "Dieter Fensel"), where quotes are escaped using the backslash character ‘\’; integers are sequences of digits, optionally preceded with the minus symbol (e.g., -31 0 5 1889); decimals consist of two sequences of digits separated with a point, optionally preceded with the minus symbol (e.g., -42.34 -1.47 537.341 8768.1).

To represent structured data values using strings, integers, and decimals, WSML defines constructor functions, called **datatype wrappers**; the names of these wrappers correspond to the IRIs of the corresponding (XML schema) datatypes. For example, the date “March 30th, 2009” is represented as: xsd#date(2009,3,30). In WSML, datatypes can be instantiated by using the syntax shortcut \_ in front of the datatype name (e.g., \_date(2009,3,30)).

Strings, integers, and decimals correspond to values of the respective XML schema datatypes [2] string, integer, and decimal. Furthermore, the datatypes recommended for use in WSML are the XML schema datatypes, plus the RDF datatype rdf#XMLLiteral.

### 8.2.2.3 WSML Prologue

Each WSML document contains a section with all the common elements between the different variants and different specifications. In particular, the prologue specifies the WSML variant adopted by the document (e.g., Flight, DL) and the namespace definitions for the document. Variants are specified with the keyword **wsmlVariant**, while namespaces are specified with the keyword **namespace**. The following listing presents a WSML prologue:

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
namespace { _"http://example.org/ontologies/Weather#",
  loc _"http://example.org/ontology/Location#",
  dc _"http://purl.org/dc/elements/1.1/",
  xsd _"http://www.w3.org/2001/XMLSchema#",
  wsml _"http://www.wsmo.org/wsml/wsml-syntax#" }
```

In the listing, the WSML variant adopted is WSML-Flight, Table 8.1 reports all the IRIs for the different variants.

The namespace without Qualified Name constitutes the default namespace of the document.

**Table 8.1** IRIs for the different WSML variants

WSML Variant	IRI
WSML-Core	<a href="http://www.wsmo.org/wsml/wsml-syntax/wsml-core">http://www.wsmo.org/wsml/wsml-syntax/wsml-core</a>
WSML-Flight	<a href="http://www.wsmo.org/wsml/wsml-syntax/wsml-flight">http://www.wsmo.org/wsml/wsml-syntax/wsml-flight</a>
WSML-Rule	<a href="http://www.wsmo.org/wsml/wsml-syntax/wsml-rule">http://www.wsmo.org/wsml/wsml-syntax/wsml-rule</a>
WSML-DL	<a href="http://www.wsmo.org/wsml/wsml-syntax/wsml-dl">http://www.wsmo.org/wsml/wsml-syntax/wsml-dl</a>
WSML-Full	<a href="http://www.wsmo.org/wsml/wsml-syntax/wsml-full">http://www.wsmo.org/wsml/wsml-syntax/wsml-full</a>

### 8.2.2.4 WSML Header

The WSML Header consists of the common parts among any WSML specification. All WSML specifications have in common the possibility of having annotations to provide metadata about them, ontology imports to refer to ontologies defined in other WSML descriptions (or other Semantic Web based languages, e.g., OWL and RDF Schema), use mediators in order to enable interoperability with other WSML specifications.

**Annotations** Annotations are used to describe metadata such as natural language names for WSML elements, data about the authors, and more elaborate natural language descriptions. An annotation consists of a property–value pair, and multiple annotations are grouped into annotation blocks which start with the keyword **annotations** and end with the keyword **endAnnotations**. Such annotation blocks always occur immediately after the definition of an element (e.g., Web service, ontology, concept, axiom) and the annotations in the block apply to the **identifier** of the element.

Consider the following description:

```
webService _"http://example.org/services/Weather"
  annotations
    dc#creator hasValue "Federico M. Facca"
    dc#description hasValue "Describes a web service for the weather forecast"
    wsml#version hasValue "1.2"
    dc#title hasValue "Weather Service"
    dc#date hasValue _date(2009,3,30)
      dc#format hasValue "text/plain"
      dc#language hasValue "en-US"
  endAnnotations
```

It is the description of a Web service that is identified by the IRI <http://example.org/services//Weather>. The description contains several annotations, namely a creator, a description, a version, a title, a date, a format, and a language. For annotations, WSML recommends the adoption of Dublin Core properties [34] (e.g., title, creator). Other properties can also be adopted, for example, WSML defines a version property to identify the version of a document.

**Import Ontologies** Any WSMO element may import ontologies in order to reuse the terminologies in the corresponding descriptions. The ontology imports in

WSML are declared using the **importsOntology** keyword, followed by a list of IRIs identifying the ontologies to be imported. The imported ontologies may be WSML, RDF Schema, or OWL DL ontologies.

The following listing shows how an ontology about locations can be imported in the weather forecast Web service.

```
webService _"http://example.org/services/Weather"
importsOntology {"http://example.org/ontology/Location"}
```

**Import Mediators** Mediators ensure interoperability between different WSML elements (ontologies, goal, Web services). A mediator can be imported in a WSML element by using the keyword **usesMediator**. The types of mediators which can be used are constrained by the type of specification. An ontology allows for the use of different mediators than, for example, a goal or a Web service (see Sect. 8.2.6 for more details).

The following listing shows how an ontology about weather can use an ontology to ontology mediator to resolve issues on measurement units heterogeneity.

```
webService _"http://example.org/ontology/Weather"
usesMediator _"http://example.org/ooMetricUnitsToImperialUnits"
```

### 8.2.3 *Ontologies in WSML*

An ontology in WSML consists of the elements **concept**, **relation**, **instance**, **relationInstance** and **axiom**. Additionally, an ontology may have annotations and may import other ontologies. We start the description of WSML ontologies with an example which demonstrates the elements of an ontology, in Listing 8.1, and describe the individual elements in more detail below.

The prologue of the listing illustrates how the WSML variant is declared and how the default namespace and namespace prefixes are declared.

In the following section, we discuss the elements that compose an ontology in WSML in more detail.

#### 8.2.3.1 Concepts

Concepts (sometimes also known as ‘classes’) play a central role in ontologies: they form the basic terminology of the domain of discourse. A concept is described by the attributes associated with it. The annotations, as well as the attribute definitions, are grouped together in one syntactical construction, as can be seen in the example concept `WeatherObservation` in Listing 8.1.

Attribute definitions can take two forms, namely **constraining** (using **ofType**) and **inferring** (using **impliesType**) attribute definitions. Constraining attribute definitions

```

wsmVariant _"http://www.wsmo.org/wsmi/wsmi-syntax/wsml-flight"
namespace { _"http://example.org/ontology/Weather#",
    loc _"http://example.org/ontology/Location#",
    dc _"http://purl.org/dc/elements/1.1#",
    units _"http://example.org/ontology/MetricUnits#"
}

ontology _"http://example.org/ontology/Weather"
importsOntology {
    _"http://example.org/ontology/Location",
    _"http://example.org/ontology/MetricUnits"
}

concept WeatherObservation
annotations
    dc:description hasValue "This concept describes weather observations"
endAnnotations
hasLocation ofType (1 1) loc#Location
hasTime impliesType (1 1) _dateTime
hasTemperature ofType (1 1) TemperatureObservation
hasWind ofType (1 1) WindObservation
hasPressure ofType (1 1) PressureObservation
hasCloud ofType (0 *) CloudObservation
hasPrecipitation ofType (0 1) PrecipitationObservation

concept Observation

concept TemperatureObservation subConceptOf Observation
hasMeasure ofType (1 1) _decimal
hasUnit impliesType (0 1) units#TemperatureUnit

concept WindObservation subConceptOf Observation
hasMeasure ofType (1 1) _decimal
hasUnit impliesType (0 1) units#SpeedUnit
hasDirection ofType (1 1) _decimal

concept PressureObservation subConceptOf Observation
hasMeasure ofType (1 1) _decimal
hasUnit impliesType (0 1) units#SpeedUnit

concept CloudObservation subConceptOf Observation
hasType impliesType (1 1) CloudType
hasAmount ofType (1 1) _decimal
hasHeight ofType (1 1) _decimal

concept CloudType

concept PrecipitationObservation subConceptOf Observation

instance WeatherObservationExample memberOf WeatherObservation
hasLocation hasValue loc#Innsbruck
hasTime hasValue _dateTime(2009,3,30,10,20,0,000)
hasTemperature hasValue TemperatureObservationExample
hasWind hasValue WindObservationExample
hasPressure hasValue PressureObservationExample

instance TemperatureObservationExample memberOf TemperatureObservation
hasMeasure hasValue _decimal(14.0)
hasUnit hasValue units#Celsius

instance WindObservationExample memberOf WindObservation
hasMeasure hasValue _decimal(14.0)
hasUnit hasValue units#Knots
hasDirection hasValue _decimal(30.0)

instance PressureObservationExample memberOf PressureObservation
hasMeasure hasValue _decimal(1020.0)

```

**Listing 8.1** An example WSML ontology

defines a typing constraint on the values for this attribute, similar to integrity constraints in databases; inferring attribute definitions implies that the type of the values for the attribute is inferred from the attribute definition, similar to range restrictions on properties in RDFS and universal value restrictions in OWL.<sup>1</sup>

It is possible to declare several types for a single attribute; these are interpreted conjunctively which means that every type is applicable. If the type of the attribute is unknown, the list maybe empty. Each attribute definition may have a number of associated features, namely, transitivity, symmetry, reflexivity, and the inverse of an attribute, as well as minimal and maximal cardinality constraints.

Constraining attribute definitions, as well as cardinality constraints, require closed-world reasoning and are thus not allowed in WSML-Core and WSML-DL. As opposed to features of roles in Description Logics, attribute features such as transitivity, symmetry, reflexivity and inverse attributes are local to a concept in WSML. Thus, none of these features may be used in WSML-Core and WSML-DL.

### 8.2.3.2 Relations

Different WSML variants offer a different level of support for modeling relations. As such, WSML-Core and WSML-DL do not allow using relations; however, unary and binary relations in Description Logics correspond to concepts and attributes in WSML. Relations 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 or a (possibly empty) list of concept identifiers. The usage of **ofType** and **impliesType** correspond with the usage in attribute definitions. Namely, parameter definitions with the **ofType** keyword are used to check the type of parameter values, whereas parameter definitions with the **impliesType** keyword are used to infer concept membership of parameter values.

The following is an example of a relation.

```
relation hasTemperatureObservation(ofType loc#Location, ofType _decimal, ofType _dateTime)
  annotations
    dc:description hasValue "Relation between Location, Temperature and time of observation."
  endAnnotations
```

### 8.2.3.3 Instances

An instance may be member of zero or more concepts and may have a number of attribute values associated with it. Note that the specification of concept membership is optional and the attributes used in the instance specification do not necessarily have to occur in the associated concept definition. Consequently, WSML instances

---

<sup>1</sup>The distinction between inferring and constraining attribute definitions is explained in more detail in [3, Sect. 2].

can be used to represent semi-structured data, since without concept membership and constraints on the use of attributes, instances form a directed labeled graph. Because of the possibility to capture semi-structured data, most RDF graphs can be represented as WSML instance data, and vice versa.

### 8.2.3.4 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, but also to add arbitrary axiomatic domain knowledge and to express constraints. The following is an example of an axiom that defines the relation hasTemperatureObservation (indicating the temperature measurement at a certain point in time at a given location) as part of the domain knowledge of the ontology presented in Listing 8.1.

```
axiom hasTemperatureObservationAxiom
  annotations
    dc#relation hasValue hasTemperatureObservation
    dc#description hasValue "Defines the hasTemperatureObservation relation as the relation between location
      and the temperatures observed at that location."
  endAnnotations
  definedBy
    hasTemperatureObservation(?x,?y,?z) :- ?k memberOf WeatherObservation
      and ?k[hasLocation hasValue ?x, hasTime hasValue ?z, hasTemperature hasValue ?w]
      and ?w[hasMeasure hasValue ?y].
```

The syntax of logical expressions is explained in more detail in the next subsection.

### 8.2.3.5 Logical Expressions

WSML logical expressions are used in axioms of ontologies and in Web service capabilities, as well as in transition rules of Web service choreographies and non-functional properties. In this section, we first explain the general logical expression syntax, which encompasses all WSML variants, and then review the restrictions on this general syntax for each of the variants.

WSML logical expression syntax adopts a first-order logic style: it has constants, function symbols, variables, predicates and a set of logical connectives (i.e., negation, disjunction, conjunction, implication, and quantifiers). WSML also includes a number of connectives specifically for the Logic Programming based variants (i.e., negation-as-failure, LP-implication and database-style integrity constraints). Finally, all the keywords defined to model ontologies (e.g., **memberOf**, **subConceptOf**, ...) can be used to create a logical expression.

Logical expressions may include variables as well. In WSML, variables are identified by a question mark, followed by an arbitrary number of alphanumeric characters, e.g., `?x`, `?name`, `?123`. Free variables in WSML (i.e., variables that 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).

Exceptions to this rule are: variables included in the **sharedVariables** construct and variables included in a logical expression occurring in a non-functional property where they act as parameters.

Terms are either identifiers (i.e., IRIs or data values), variables, or constructed terms of the form  $f(t_1, \dots, t_n)$ . An atom is a predicate symbol with a number of terms as arguments, e.g.,  $p(t_1, \dots, t_n)$ . In addition, WSML has a special kind of atoms, called **molecules**, which are used to capture information about concepts, instances, attributes and attribute values. WSML molecules are:

- An **isa** molecule is a concept membership molecule of the form  $t_1 \text{ memberOf } t_2$  or a subconcept molecule of the form  $t_1 \text{ subConceptOf } t_2$ , where  $t_1$  and  $t_2$  are terms.
- An **object** molecule is an attribute value expressions  $t_1[t_2 \text{ hasValue } t_3]$ , a constraining attribute signature expression  $t_1[t_2 \text{ ofType } t_3]$ , or an inferring attribute signature expression  $t_1[t_2 \text{ ofType } t_3]$ , where  $t_1$ ,  $t_2$ , and  $t_3$  are terms.

WSML first-order connectives include:

- The unary negation operator **neg**.
- The binary operators for conjunction **and**.
- Disjunction **or**.
- Right implication **implies**.
- Left implication **impliedBy**.
- Dual implication **equivalent**.

Variables may be universally quantified using **forall** or existentially quantified using **exists**. First-order formulae are obtained by combining atoms using the mentioned connectives. Examples of first-order formulae in WSML are [15]:

```
//every person has a father
forall ?x (?x memberOf Person implies exists ?y (?x[father hasValue ?y])).

//Man and Woman are disjoint.
?x memberOf Man implies neg ?x memberOf Woman
```

Logic Programming based WSML variants include: the negation-as-failure symbol **naf** in front of atomic formulas, 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 only allowed to occur in the body of a Logic Programming rule or an integrity constraint. The following logical connectives are allowed in the head of a Logic Programming rule: **and**, **implies**, **impliedBy**, and **equivalent**. The following connectives are allowed in the body of a rule (or constraint): **and**, **or**, and **naf**. The logical expressions previously expressed in first-order formulae can be also represented with LP rules and database constraints [15]:

```
//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.
```

**Table 8.2** Features in WSML variants

Feature	Core	DL	Flight	Rule	Full
Classical Negation ( <b>neg</b> )	–	X	–	–	X
Existential Quantification	–	X	–	–	X
(Head) Disjunction	–	X	–	–	X
<i>n</i> -ary relations	–	–	X	X	X
Meta Modeling	–	–	X	X	X
Default Negation ( <b>naf</b> )	–	–	X	X	X
LP implication	–	–	X	X	X
Integrity Constraints	–	–	X	X	X
Function Symbols	–	–	–	X	X
Unsafe Rules	–	–	–	X	X

In the previous paragraphs, we discussed the general syntax of WSML logical expressions. Each variant poses some restriction on the use of specific connectives and on the usage of modeling constructs.

- WSML-Core corresponds with the intersection of Description Logic and Horn Logic. WSML-Core disallows the use of the equality symbol, disjunction in the head of a rule and existentially quantified variables in the head of a rule. As a DL fragment, it disallows the use of function symbols, restricts the arity of predicates to unary and binary, and prohibits chaining variables over predicates.
- WSML-DL relaxes WSML-Core constraints toward complete support of DL *SHIQ*. WSML-DL does not allow the use of function symbols, restricts the arity of predicates to unary and binary, and prohibits chaining variables over predicates. WSML-DL allows classical negation, and disjunction and existential quantification in the heads of implications.
- WSML-Flight extends WSML-Core toward Horn Logic. WSML-Flight allows variables in place of instance, concept and attribute identifiers, relations of arbitrary arity. Finally, 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-Rule extends WSML-Flight with function symbols and unsafe rules.
- WSML-Full is equivalent to the general logical expression syntax of WSML and allows the full expressiveness of all other WSML variants as conjunction of the expressive power of Description Logic and Horn Logic. In particular, WSML-Full adds to WSML-DL full first-order modeling: *n*-ary predicates, function symbols and chaining variables over predicates. Furthermore, WSML-Full adds to WSML-DL non-monotonic negation. With respect to the WSML-Rule variant, WSML-Full adds disjunction, classical negation, multiple model semantics, and the equality operator.

Table 8.2 mentions a number of language features and indicates in which variant the feature may be used, illustrating the differences between each variant.

```

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace { namespace { _"http://example.org/ontologies/Weather#",
    loc _"http://example.org/ontology/Location#",
    dc _"http://purl.org/dc/elements/1.1#",
    wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
    xsd _"http://www.w3.org/2001/XMLSchema#"
}

webService _"http://example.org/service/WeatherUSA"
annotations
    dc#creator hasValue "Federico M. Facca"
    dc#description hasValue "Describes an web service for the weather forecast"
    wsml#version hasValue "1.2"
    dc#title hasValue "Weather Service"
    dc#date hasValue _date(2009,3,30)
        dc#format hasValue "text/plain"
        dc#language hasValue "en-US"
endAnnotations

importsOntology {
    _"http://example.org/ontology/Location",
    _"http://example.org/ontology/Weather"
}

capability
...
interface
...
nonFunctionalProperty
...

```

**Listing 8.2** An example Web service description

### 8.2.4 Web Services in WSM

WSML based Web service descriptions formalize the functionality, behavior, and other aspects of Web services. A Web service description consists of a capability, which describes the functionality, one or more interfaces, which describe the possible ways of interacting with the service, and non-functional properties, which describe non-functional aspects of the service. Listing 8.2 shows the structure of a simple Web service for weather forecasts.

#### 8.2.4.1 Capabilities

Capabilities define functionality provided by services and restrictions over them. In WSM, capabilities can be formalized in two ways: (i) as concepts of a task ontology and (ii) as a detailed state-based description of the functionality. Capabilities of the type (i) are also referred to as **set-based capabilities**; capabilities of the type (ii) are also referred to as **state-based capabilities**.

A set-based capability simply consists of the **capability** keyword followed by an IRI identifying a concept in a task ontology. Such a task ontology is typically im-

```

capability
  sharedVariables {?location}
precondition
  annotations
    dc:description hasValue "The input is of a location and it must be in the US."
  endAnnotations
definedBy
  ?location[hasCountry hasValue loc#US] memberOf loc#Location.
postcondition
  annotations
    dc:description hasValue "The weather forecast returned corresponds to the input location."
  endAnnotations
definedBy
  ?forecast[hasLocation hasValue ?location] memberOf WeatherForecast.

```

**Listing 8.3** An example of service capability

ported in the Web service to which the capability belongs. In the case of the Web service for weather forecasts, a set-based capability can be:

```

webService _"http://example.org/service/WeatherUSA"
importsOntology {
  _"http://example.org/ontology/Location",
  _"http://example.org/ontology/Weather"
}

capability _"http://example.org/ontology/WeatherTasks#ProvideWeatherForecast"

```

A state-based capability consists of a set of axioms defining four different types of capabilities: preconditions, assumptions, postconditions and effects. Preconditions and assumptions describe the state before the execution of a Web service. While preconditions describe conditions over the information space—that is, conditions over the input, e.g., the location given as input is in the US—assumptions describe conditions over the state of world that cannot be verified by the requester of the service, but which might explain the failure of the service, e.g., there must exist a weather forecast for the given location. Postconditions describe the relation between the input and the output, e.g., the weather forecast returned corresponds to the input location. In this sense, they describe the information state after execution of the service. Effects describe changes caused by the service beyond the inputs and outputs, e.g., the user credit is reduced by the amount of the cost of the service invocation.

Listing 8.3 describes the capability of the simple Web service for adding items to a shopping cart: given a shopping cart identifier and a number of items, the items are added to the shopping cart with this identifier. The **sharedVariables** construct is used to identify variables that are shared between the pre- and postconditions and between the assumptions and effects. Shared variables can be used for to refer to the input (?cartId and ?item) or pass variables between the pre- and postcondition or assumption and effect (e.g., ?number).

### 8.2.4.2 Interfaces and Choreographies

Interfaces describe how to interact with a service from the requester point-of-view (**choreography**) and how the service interacts with other services and goals it needs to fulfill its capability (**orchestration**). Orchestration descriptions are external to WSML; WSML allows referring to any orchestration description identified by an IRI.

WSML provides a language for describing choreographies, called the **WSML choreography language**. It is also possible to refer to any choreography that has an IRI. However, WSML does not prescribe how such a choreography should be interpreted.

A choreography description is a model for possible conversations—that is, sequences of message exchanges—between the service requester and the service provider. Patterns of message exchanges are governed by transition rules; given the current state, the transition rules determine the next step in the conversation. The messages themselves consist of ontology instance information, and the background knowledge contained in ontologies is taken into account when evaluating the transition rules.

A central notion in WSML choreographies is the **state** of a conversation. Technically, a state consists of instance data of some ontology specific to a certain point during the service execution. State transitions correspond to update, insertion, or deletion of instance data. Communication between the requester and provider is modeled by marking certain ontology concepts as **in** or **out** concepts; an incoming message (from the requester to the provider) results in the insertion of an instance of an **in** concept; inserting an instance of an **out** concept results in a message being sent from the provider to the requester.

A single conversation corresponds to a choreography run, which consists of a start state, a sequence of intermediate states, and an end state. State transitions are governed by the transition rules; firing of a rule corresponds to a state transition. Such a state transition may or may not correspond to a message exchange between the requester and provider.

Listing 8.4 shows an example of an interface description with a choreography. A choreography is defined by a **state signature** and a set of **rules**. The state signature defines the state ontology over which the rules are evaluated and updates performed. In addition, the state signature assigns **modes** to ontology concepts and relations. These modes determine the role that a particular concept/relation plays in the choreography, as well as the relationship between instances of such a concept/relation and message formats, defined using, for example, WSDL.

Following the state signature block are the **transition rules** which express conditions that are evaluated with respect to the state and the background ontologies. If the condition of a transition rule holds in this state, the rule is **fired**. If a transition rule fires, the enclosed **update rules** change the state by adding, deleting, or updating instances.

The WSML choreography language is very general, and can be used in a number of different ways. For example, there is no explicit control flow between the transition rules—that is, the order of rule firing is not determined by the order the rules

```

interface WeatherUSAInterface
  annotations
    dc#title hasValue "WeatherUSA Choreography"
    dc#description hasValue "An example of a choreography for retrieving weather forecast"
  endAnnotations

  choreography WeatherUSAInterfaceChoreography

    stateSignature WeatherUSAInterfaceSignature

    importsOntology _"http://example.org/ontology/Weather"

    in
      wea#WeatherQuery withGrounding
      _"http://example.org/webservices/USWeatherForecast/Weather#wsdl.interfaceMessageReference(
        WeatherServicePortType/GetWeatherForCityState/In)",

    out
      wea#WeatherForecast withGrounding
      _"http://example.org/webservices/USWeatherForecast/Weather#wsdl.interfaceMessageReference(
        WeatherServicePortType/GetWeatherForCityState/Out)",

    transitionRules WeatherUSAInterfaceTransitions

    forall ?search
      with
        (?search[
          hasCity hasValue ?city,
          hasState hasValue ?state
        ] memberOf wea#WeatherQuery
        and exists{?location} (
          ?location memberOf loc#Location and
          ?location[hasCity hasValue ?city, hasState hasValue ?state]
        )
      )
      do
        add(?forecast[
          hasLocation hasValue ?location
        ] memberOf wea#WeatherForecast
      )
        delete(?search memberOf wea#WeatherQuery)
    endForall
  
```

**Listing 8.4** An example interface declaration

are written, but rather by the conditions (current state) at the time of the execution. It is certainly possible to add control flow to the transition rules, for example, by defining a controlled concept that mimics an explicit state using an integer.

#### 8.2.4.3 Non-Functional Properties

Non-functional properties of a service are not concerned with aspects of the service directly related to functionality, but that are nonetheless of interest to the requester. Examples of non-functional properties are: provider of the service, cost of service invocation, availability, and security [26].

Non-functional properties should not be confused with annotations: whereas annotations are concerned with the **description** of the service, e.g., creator of the

```

nonFunctionalProperty
prop#accuracy hasValue prop#highAccuracy
annotations
dc:description hasValue "The accuracy of the data returned by the service is high."
endAnnotations

nonFunctionalProperty
prop#price hasValue ?price
annotations
dc:description hasValue "The price for service invocation is determined by the relation servicePrice,
which is defined in the service properties ontology."
endAnnotations
definedBy
exists ?x(?x memberOf prop#ServiceRequester and servicePrice(?x, ?price)).

```

**Listing 8.5** Example non-functional properties

description, natural language description, non-functional properties are concerned with the service itself.

As the name indicates, non-functional properties are **properties**, and are thus essentially name–value pairs. However, in contrast to annotations, it is possible to say more about non-functional properties using logical expressions. Consequently, there are three kinds of non-functional properties: (i) simple name–value pairs, (ii) conditional name–value pairs which should only be considered if the given logical expression is true (i.e., follows from the ontology), and (iii) open name–value pairs where a given logical expression determines the value(s) of the property [32]. Listing 8.5 shows examples of all three kinds of non-functional properties, all related to the service description in Listing 8.2.

### 8.2.5 Goals in WSML

Goal represents the requester's view over the desired functionality of a service. Thus, goals are described in a symmetric way with respect to Web service descriptions. Therefore, goal descriptions comprise the same modeling elements as Web service descriptions, namely a capability, one or more interfaces, and non-functional properties. Whereas services are described from the provider's point of view, goals are described from the requester's point of view.

In an ideal world, requesters and providers would use the same domain ontologies for the description of their goals and Web services, respectively. However, this cannot be assumed in an open environment. A requester might find it more convenient to use his own terminology rather than the terminology of the service providers; furthermore, different providers may use different terminologies to describe the same functionality. In general, it cannot be assumed that the requester is aware of the terminologies used by all the service providers with which it interacts.

In case there is a mismatch between the terminologies used in a goal and a Web service, mapping or mediation is required during discovery, selection, as well as usage of the service. To mediate between the requester and provider ontologies during

```

goal "http://example.org/service/WeatherInnsbruck"
importsOntology {
    _"http://example.org/ontology/Location",
    _"http://example.org/ontology/Weather"
}
capability
postcondition
annotations
    dc:description hasValue "I want to know the weather forecast for Innsbruck."
endAnnotations
definedBy
?forecast[hasLocation hasValue loc#Innsbruck] memberOf WeatherForecast.

```

**Listing 8.6** Example goal

discovery, it is necessary to be aware of the mappings between the ontologies (e.g., [27, 31]).

WSML does not prescribe how a discovery engine should specify or use ontology mappings. There are, however, a number of ways in which such mappings can be made explicit in WSML. A **wgMediator** specifies a link between a goal and a Web service. Such a mediator can import an ontology that contains mappings between the ontologies used in the goal and Web service descriptions. WSML does not prescribe how such a mediator should be used, or how the mappings in the imported ontology should be processed.

In case a requester or provider knows about ontologies that are used in potentially matching Web services or goals, the mappings can be imported in the goal or Web service directly using the **importsOntology** directive. Mappings that are imported in this way are added as mapping rules to the ontologies that are used for the goal or Web service specification [23]. Listing 8.6 shows an example of a goal represented in WSML.

### 8.2.5.1 Capability

On the level of a set-based capability description, there is not much difference between the requester's and provider's point of view—they both specify the capability of the requested or provided service. In a state-based capability description, however, the requester and provider will have different objectives in their descriptions of the preconditions, postconditions, assumptions, and effects.

**Preconditions** The service provider will want to make sure that all information required to execute the service is present when the service is executed. Therefore, he has an interest in describing the preconditions of the service in detail. The requester, on the other hand, may want to disclose some of his (personal) information, but most likely not all of it. Furthermore, if requesters were to include all their information in the preconditions of every goal, the task of writing the goal would become unmanageable, and even more so the processing of that goal. Therefore, we may assume that the requester might include some of his information in the precondition, but it will in no way be complete. This has implications for possible discovery

mechanisms: even if a Web service provides the exact requested functionality, it cannot be assumed in general that the preconditions of a goal will exactly match the preconditions of the service.

**Assumptions** With respect to assumptions, neither the requester nor the provider can provide complete descriptions. There are many things that must hold in the real world to ensure that the service can be executed successfully, e.g., a delivery address must exist, the company providing the service must not go bankrupt, and war must not break out. While a provider may want to model the assumption that the provided delivery address must exist in the real world in order to deliver a product, he will not be poised to model aspects related to bankruptcy and war. Considerations for the requester are similar: the requester can guarantee a number of things to hold, but an exhaustive list (e.g., including guarantees that his house does not have a leaky roof and his car will have enough gas to drive to the supermarket) would become too long for any practical use. In general, it may be expected that the provider will include some assumptions that are directly related to the provided service (e.g., for payment, the limit of the credit card must not have been reached) that might help a potential requester to determine whether the service is suitable and to explain possible failures in execution of the service, but the description will not be exhaustive.

From the above description, it may seem that the prospect of using state-based capabilities for automated matching is rather bleak when looking at our expectations concerning the description of preconditions and assumptions, especially from the side of the requester; if the formal descriptions are not detailed enough, or even missing, automated matching is not possible. Fortunately, the situation looks a bit better when considering postconditions and effects. There are two reasons for this: first, postconditions and effects are only concerned with the desired and actual outputs and real-world effects of the service; and secondly therefore, requesters have an interest in including detailed descriptions of the postconditions and effects in their goals.

**Postconditions and Effects** The requester's main interest during a service invocation (or even prior to the invocation) is what the service actually "does" for him. If he is interested in specific information (e.g., product availability) that should be an output of the service, he will describe the postconditions concerning this information. If he is interested in certain real-world effects (e.g., product delivery), he will describe the corresponding effects. The provider may find it more important to ensure that all preconditions and assumptions are met before executing the service, rather than describing the output and effects in detail; however, if the provider wants the service to be found, he will have to describe the postconditions and effects in sufficient detail.

### 8.2.5.2 Interfaces and Choreographies

Considerations for interfaces are similar to those for preconditions. There is typically a fixed way (or number of ways) in which a service can interact with a requester. The service provider has an interest in accurately describing this interface

```

ooMediator _"http://example.org/ooMediatorUSUnitsToMetricUnits"
  annotations
    dc#description hasValue "This ooMediator transform units expressed
      in the US Measure System to the Metric Measure System."
  endAnnotations
  source _"http://example.org/ontology/USUnits"
  target _"http://example.org/ontology/MetricUnits"

```

**Listing 8.7** Example mediator

so that potential requesters know how to invoke it. The requester, on the other hand, may have a large repository of information that could be sent; however, it is not practical to describe all this in a goal. We conjecture that there will be many situations in which a Web service description contains an accurate interface description, but a matching goal description does not. That said, if the requester is an automated agent, there may be a limited number of ways it can interact with services. In this case, the goal would contain descriptions of the ways the requester can interact with services. Note that it is, in the general case, unlikely that such interface descriptions and goals would easily match with interface descriptions in Web services, even if ontology mapping is applied, thus it might be necessary to use a **mediator** (see the next section).

### 8.2.6 *Mediators in WSML*

Mediators connect different goals, Web services and ontologies, and enable inter-operation 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 for the specification of a number of used mediators through the **usesMediator** keyword.
2. Each mediator has (depending on the type of mediator) one or more **sources** and one **target**. Both source and target are optional in order to allow for 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.

Listing 8.7 provides an example of ontology-to-ontology mediator that translates between two different metric units.

### 8.2.7 *Technologies for Using WSML*

Interested readers can experiment WSML with a number of technologies. In particular, the Web Service Modeling Toolkit is an IDE tool to support the design of

WSML descriptions, while the WSML2Reasoner framework is a reasoning framework that enables the user to register ontologies belonging to the different variants of WSML and query the registered knowledge base.

### 8.2.7.1 The Web Service Modeling Toolkit

The Web Service Modeling Toolkit (WSMT)<sup>2</sup> is an integrated development environment for Semantic Web Services that enables developers to develop Ontologies, Web Services, Goals and Mediators through the Web Service Modeling Ontology (WSMO) formalism. The WSMT is implemented as a collection of plug-ins for the Eclipse framework such that it can be integrated with other toolkits like the Java Development Toolkit JDT or the Web Tools Platform (WTP) so that a developer can develop his Java code, Web services and Semantic Web Services side by side in one application. The main aim of the WSMT is to support the developer through the full Software Development Cycle of his Semantic Web Service from requirements, through design, implementation, testing, and deployment. In this way, the process of developing Semantic Web Services can be achieved more cheaply, and remove many of the tedious activities that the developer must currently perform.

As shown in Fig. 8.2, WSMT supports advanced visualization of ontologies and enables the validation of ontologies over reasoners and their query.

### 8.2.7.2 WSML2Reasoner

WSML2Reasoner<sup>3</sup> is a highly modular framework that combines various validation, normalization and transformation algorithms that enable the translation of ontology descriptions in WSML to the appropriate syntax of several underlying reasoning engines. It first maps WSML to either generic Datalog or OWL (OWL1.0), and then uses a plug-in layer that translates to the specific internal representation of a single reasoning engine. Reasoning engines currently integrated and tested within the WSML2Reasoner framework are: IRIS (WSML-Core, WSML-Flight, WSML-Rule), KAON2 (WSML-Core, WSML-Flight, WSML-DL), MINS (WSML-Core, WSML-Flight, WSML-Rule) and PELLET (WSML-Core and WSML-DL). An on-line demo can be found at: <http://tools.sti-innsbruck.at/wsml2reasoner/demos>.

## 8.3 Extensions

The standard ontology languages utilized for service annotation vary with high complexity ranging from PTIME (RDFS and WSML-Core) to NEXPTIME (OWL and

---

<sup>2</sup> Available at: <http://wsmt.sourceforge.net>.

<sup>3</sup> Available at: <http://tools.sti-innsbruck.at/wsml2reasoner/>.

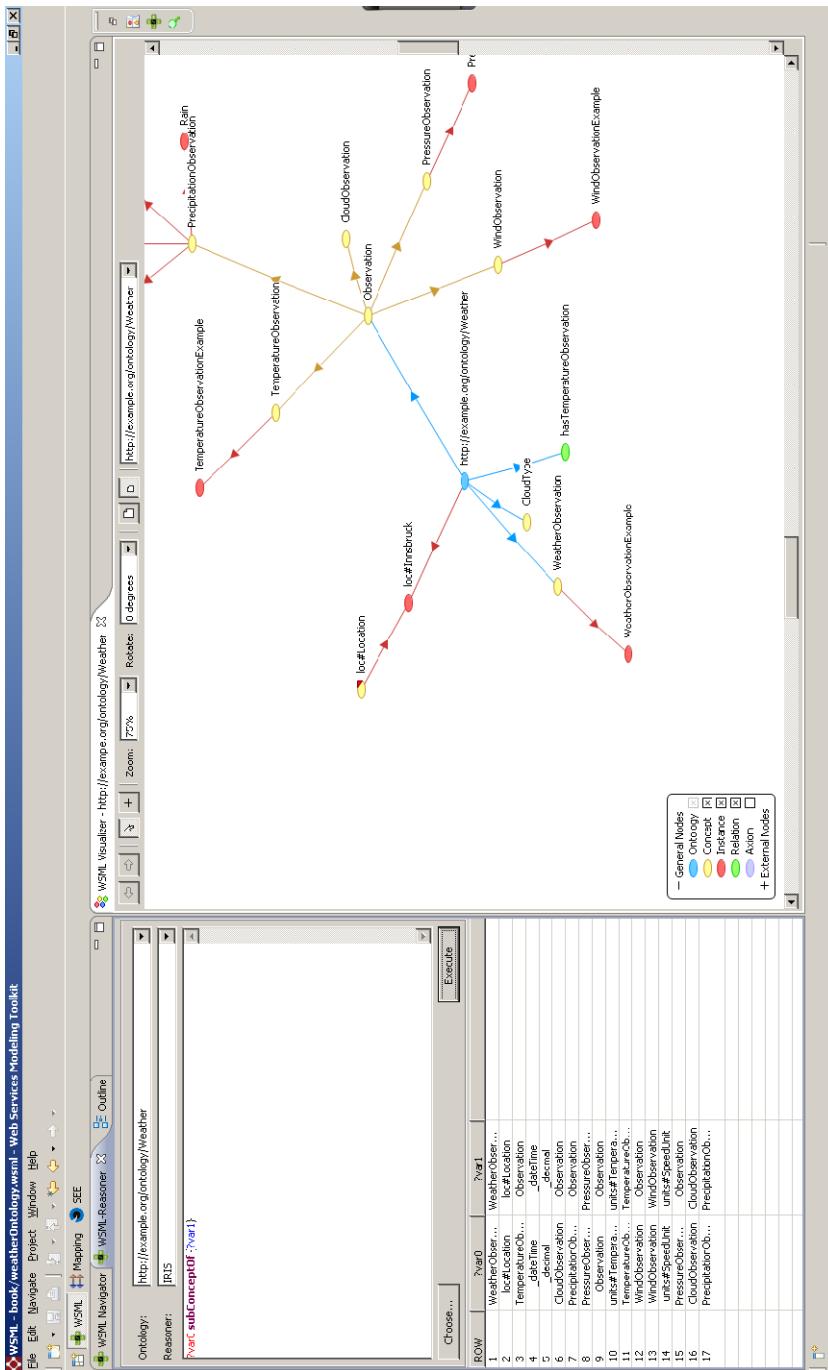


Figure 8.2 The Web Service Modeling Toolkit

WSML-DL). A paradigm shift in constructing ontology languages pushing language features no longer towards the boundaries of expressivity but eliminating language features to reach tractability can be observed in recent research on semantics. This trend leads to the further simplification of WSMO to obtain WSMO-Lite and MicroWSMO.

Current ongoing research envisions the following layering:

- WSMQ—variant that can be used to tag Web services. This variant is inspired by SKOS and contains concept declarations with subConceptOf as the only allowed relationship.
- WSMC v2.0—an evolution of WSMC that has minimal expressivity and thus hugely improved reasoning scalability, ideally with sub-polynomial complexity.
- WSDL v2.0—a new description logic variant of WSM inspired by recent progress of the OWL 2 family of languages. An OWL 2 profile (equivalent to ELP) is used as the basis of this new WSM language variant such that useful expressivity is obtained with minimal computational cost. The semantics of this new language variant can be captured in Datalog rules and therefore should have polynomial reasoning complexity.
- WSR v2.0—a new rule variant of WSM is inspired by recent progress in the W3C Rule Interchange Format (RIF) and is further modified such that its logical expression syntax properly subsumes that of the new WSMC v2.0 and WSDL, respectively [33].

## 8.4 Illustration by a Larger Example

In this section, we present a set of WSM description modeling the example scenario of the Virtual Travel Agency first introduced in Chap. 4.

In Chap. 7, we collected requirements on semantic descriptions for the scenario. In this section, starting from those requirements, we present the descriptions needed to support the scenario. In particular, we will cover a scenario illustrating and ontology modeling the travel domain, a Web service of an Austrian railway company, a service for a international flight company, and a goal to book a trip from Austria to the UK.

### 8.4.1 Travel Ontology

The definition of the ontology is based on the travel itinerary ontology from the DAML ontology library.<sup>4</sup> The ontology defines basic travel related concepts like Ticket and Vehicle, as illustrated by the following listing.

---

<sup>4</sup> Available at: <http://www.daml.org/2001/06/itinerary/itinerary-ont>.

```

wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {_ "http://www.gsmo.org/dip/travel/domainOntology#",
  dc _ "http://purl.org/dc/elements/1.1#",
  wsml _ "http://www.wsmo.org/wsml/wsml-syntax#"}

ontology TravelOntology

concept Ticket
  annotations
    dc#description hasValue "concept of a ticket"
  endAnnotations
  from ofType Region
  to ofType Region
  vehicle ofType Vehicle

concept Region

concept Country subConceptOf Region
  name ofType _string

concept City subConceptOf Region
  name ofType _string
  country ofType Country

concept EUCity subConceptOf City

concept GermanCity subConceptOf EUCity

concept AustrianCity subConceptOf EUCity

concept UKCity subConceptOf EUCity

concept USCity subConceptOf City

concept Vehicle
  seats ofType _integer

concept Airplane subConceptOf Vehicle

concept Train subConceptOf Vehicle

axiom GermanCityDef
  definedBy
    ?city memberOf GermanCity implies ?city[country hasValue Germany].

axiom AustrianCityDef
  definedBy
    ?city memberOf AustrianCity impliedBy ?city[name hasValue "Austria"] memberOf country.

axiom UKCityDef
  definedBy
    ?city memberOf UKCity implies ?city[country hasValue UK]. 

instance Innsbruck memberOf AustrianCity

instance Germany memberOf Country
  name hasValue "Germany"

instance UK memberOf Country
  name hasValue "United Kingdom"

instance Austria memberOf Country
  name hasValue "Austria"

```

### 8.4.2 Services

The first Web service provides the capability of booking train tickets to travel within Austria, as expressed by the logic expression fragments `?from memberOf dO#AustrianCity and ?to memberOf dO#AustrianCity`, as presented in the following listing.

```
wsmiVariant_ "http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {"http://www.gsmo.org/dip/travel/webServiceAustrianTrains#",
dO_ "http://www.gsmo.org/dip/travel/domainOntology#",
dc_ "http://purl.org/dc/elements/1.1#"}

webService_ "http://www.gsmo.org/dip/travel/webServiceAustrianTrains"
importsOntology_ "http://www.gsmo.org/dip/travel/domainOntology#TravelOntology"
capability webServiceAustrianTrainsCapability
postcondition
definedBy
?ticket [
    dO#from hasValue ?from,
    dO#to hasValue ?to,
    dO#vehicle hasValue ?vehicle
] memberOf dO#Ticket and
?from memberOf dO#AustrianCity and
?to memberOf dO#AustrianCity and
?vehicle memberOf dO#Train.
```

The second Web service provides reservations for flights connecting German and Austrian cities to other European cities.

```
wsmiVariant_ "http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {"http://www.gsmo.org/dip/travel/webServiceFlights#",
dO_ "http://www.gsmo.org/dip/travel/domainOntology#",
dc_ "http://purl.org/dc/elements/1.1#"}

webService_ "http://www.gsmo.org/dip/travel/webServiceFlights"
importsOntology_ "http://www.gsmo.org/dip/travel/domainOntology#TravelOntology"
capability webServiceFlightsCapability
postcondition
definedBy
?ticket [
    dO#from hasValue ?from,
    dO#to hasValue ?to,
    dO#vehicle hasValue ?vehicle
] memberOf dO#Ticket and
(?from memberOf dO#AustrianCity or ?from memberOf dO#GermanCity) and
?to memberOf dO#EuropeanCity and
?vehicle memberOf dO#Airplane.
```

### 8.4.3 Goal

The following listing presents a goal of buying a ticket from an Austrian city to a city in the UK. The goal can match only the second Web service, since it is the only option providing connections between Austria and the UK.

```

wsmlVariant _ "http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {_
    "http://www.gsmo.org/dip/travel/goal#",
    dO _ "http://www.gsmo.org/dip/travel/domainOntology#",
    dc _ "http://purl.org/dc/elements/1.1#"}

goal _ "http://www.gsmo.org/dip/travel/goal.wsml"
importsOntology _ "http://www.gsmo.org/dip/travel/domainOntology#TravelOntology"
capability goalCapability
postcondition
definedBy
?ticket [
    dO#from hasValue ?from,
    dO#to hasValue ?to
] memberOf dO#Ticket and
?from memberOf dO#AustrianCity and
?to memberOf dO#UKCity.

```

## 8.5 Summary

In this chapter, we have introduced various aspects of the Web Service Modeling Language. We started by introducing principles underlying WSML design. In particular, we discussed OWL limitations and showed how these limitations can be overcome by a proper combination of Description Logics and Logic Programming, by presenting the WSML descriptions of WSMO concepts.

After these initial considerations, we introduced in more detail WSML variants and their layering in order to achieve the maximum flexibility in term of expressive power of the language. WSML enables going from the maximum common intersection of Description Logics and Logic Programming to the minimum common conjunction of them. The discussion on WSML variants is followed by a detailed introduction to WSML syntax, covering all the WSML elements: ontologies, services, goals and mediators. We also provided a quick insight to tools that students can use to create and test WSML descriptions, and discussed extensions of WSML to support large scale reasoning and compatibility with OWL2 and RIF.

The chapter has concluded with the presentation of WSML descriptions for the scenario introduced in Chap. 4.

## 8.6 Exercises

**Exercise 1** According to the Blue Hotel service example discussed in Chap. 4, create an ontology for the description of the hotel domain and a service description for the Blue Hotel. In the service description, also include the definition of the choreography.

**Exercise 2** Extend the travel ontology presented in Sect. 8.4 by introducing the bus travels, timetables and routes. Define a service for the public transport system of Innsbruck capable of providing the departures from a given bus stop and the next bus stop on the route.

**Exercise 3** Define a goal for booking a double room for non-smokers, according to the ontology modeled in Exercise 3.

**Exercise 4** Consider the ontology in the following listing, representing knowledge about a famous American cartoon.

```
wsmVariant _"http://www.wsmo.org/wsmi/wsml-syntax/wsml-flight"
namespace { _"http://ontologies.sti2.at/"

,
wsml _"http://www.wsmo.org/wsmi/wsml-syntax#", dc _"http://purl.org/dc/elements/1.1/" }

ontology simpsons

concept gender

concept character
hasName ofType _string
hasGender ofType gender
hasSpouse ofType character
hasChild ofType character
hasParent ofType character
hasSibling ofType character
hasFriend ofType character
hasCatchPhrase ofType _string
inLoveWith ofType character
isCustomerOf ofType workplace
hasWorkingPlace ofType place
attends ofType school

instance male memberOf gender

instance female memberOf gender

instance homer_simpson memberOf character
annotations
    dc#title hasValue "Homer J Simpson"
endAnnotations
hasName hasValue "Homer J Simpson"
hasGender hasValue male
hasSpouse hasValue marge_simpson
hasParent hasValue abe_simpson
hasChild hasValue {bart_simpson, lisa_simpson, maggie_simpson }

instance marge_simpson memberOf character
annotations
    dc#title hasValue "Marge Simpson"
endAnnotations
hasName hasValue "Marge Simpson"
hasGender hasValue female
hasSpouse hasValue homer_simpson
hasChild hasValue {bart_simpson, lisa_simpson, maggie_simpson }
hasSibling hasValue {patty_bouvier, selma_bouvier }

instance lisa_simpson memberOf character
annotations
    dc#title hasValue "Lisa Simpson"
endAnnotations
hasName hasValue "Lisa Simpson"
hasGender hasValue female
hasParent hasValue {homer_simpson, marge_simpson }
hasSibling hasValue {bart_simpson, maggie_simpson }

instance bart_simpson memberOf character
annotations
```

```

dc#title hasValue "Bart Simpson"
endAnnotations
hasName hasValue "Bart Simpson"
hasGender hasValue male
hasParent hasValue {homer_simpson, marge_simpson }

```

Define

1. An axiom that states that all married people are in love with their spouse (Hint: similar to the axioms from the travel Ontology example).
2. A logical expression to find out who is in love with whom in the Simpsons world (Hint: logical expressions are modeled using axioms, for example, hasTemperatureObservationAxiom).
3. A axiom that states that if a character A is sibling with a character B then also the B is sibling with A.
4. A axiom that defines the mother of a character as parent of female gender.
5. A axiom that states that male and female characters are disjoint.

## References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
2. Biron, P.V., Malhotra, A.: XML schema part 2: Datatypes second edition. Recommendation 28 October 2004, W3C (2004). <http://www.w3.org/TR/xmlschema-2/>
3. de Bruijn, J., Polleres, A., Lara, R., Fensel, D.: 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, pp. 623–632. ACM, New York (2005)
4. de Bruijn, J., Eiter, T., Polleres, A., Tompits, H.: On representational issues about combinations of classical theories with nonmonotonic rules. In: Proceedings of the 1st International Conference on Knowledge Science, Engineering and Management (KSEM2006), pp. 1–22. Springer, Berlin (2006)
5. de Bruijn, J., Eiter, T., Polleres, A., Tompits, H.: Embedding non-ground logic programs into autoepistemic logic for knowledge-base combination. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI2007), January 6–12, Hyderabad, India, pp. 304–309. AAAI Press, Menlo Park (2007)
6. de Bruijn, J., Heymans, S.: A semantic framework for language layering in WSML. In: Proceedings of the First International Conference on Web Reasoning and Rule Systems (RR2007), June 7–8, Innsbruck, Austria, pp. 103–117. Springer, Berlin (2007)
7. de Bruijn, J., Heymans, S.: WSML ontology semantics. Working Draft d28.3, WSML Working Group (2007). <http://www.wsmo.org/TR/d28/d28.3/v0.2/>
8. de Bruijn, J., Pearce, D., Polleres, A., Valverde, A.: Quantified equilibrium logic and hybrid rules. In: Proceedings of the 1st International Conference on Web Reasoning and Rule Systems (RR2007), June 7–8, Innsbruck, Austria, pp. 58–72. Springer, Berlin (2007)
9. de Bruijn, J., Fensel, D., Kerrigan, M., Keller, U., Lausen, H., Scicluna, J.: Modeling Semantic Web Services: The Web Service Modeling Language. Springer, Berlin (2008)
10. Dean, M., Schreiber, G.: OWL web ontology language reference. Recommendation 10 February 2004, W3C (2004). <http://www.w3.org/TR/owl-ref/>
11. Duerst, M., Suignard, M.: Internationalized resource identifiers (IRIs). Proposed standard RFC 3987, Internet Engineering Task Force (2005)

12. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the Semantic Web. In: Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR2004). AAAI Press, Menlo Park (2004)
13. Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services—The Web Service Modeling Ontology. Springer, Berlin (2006)
14. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: Proc. Int. Conf. on the World Wide Web (WWW-2003), Budapest, Hungary (2003)
15. Group, W.W.: WSML language reference. Final Draft D16.1 v1.0, WSML (2008). <http://www.wsmo.org/TR/d16/d16.1/v1.0/>
16. Hidders, J., Paredaens, J.: Encyclopedia of Database Systems (2009)
17. Horrocks, I., Patel-Schneider, P.F.: A proposal for an OWL rules language. In: WWW '04: Proceedings of the 13th International Conference on World Wide Web, pp. 723–731. ACM, New York (2004). doi:[10.1145/988672.988771](https://doi.org/10.1145/988672.988771)
18. Jacobs, I.: Architecture of the World Wide Web, volume one. Recommendation 15 December 2004, W3C (2004). <http://www.w3.org/TR/webarch/>
19. Kifer, M., de Bruijn, J., Boley, H., Fensel, D.: 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. Lecture Notes in Computer Science, vol. 3791, pp. 17–29. Springer, Berlin (2005)
20. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM **42**(4), 741–843 (1995)
21. Levy, A.Y., Roussel, M.-C.: Combining horn rules and description logics in CARIN. Artificial Intelligence **104**, 165–209 (1998)
22. McGuinness, D.L., van Harmelen, F.: OWL web ontology language overview. Recommendation 10 February 2004, W3C (2004). Available from <http://www.w3.org/TR/owl-features/>
23. Mocan, A.: Ontology-based data mediation for semantic environments. PhD Thesis, National University of Ireland (2008)
24. Motik, B., Rosati, R.: A faithful integration of description logics with logic programming. In: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07), Hyderabad, India (2007)
25. Motik, B., Horrocks, I., Rosati, R., Sattler, U.: Can owl and logic programming live together happily ever after? In: Proc. of the 5th Int. Semantic Web Conf. (ISWC 2006), Athens, GA, USA (2006)
26. O'Sullivan, J., Edmond, D., ter Hofstede, A.H.: Formal description of non-functional service properties. Technical report, Queensland University of Technology, Brisbane (2005). <http://www.service-description.com/>
27. Polleres, A., Scharffe, F., Schindlauer, R.: SPARQL++ for mapping between RDF vocabularies. In: OTM 2007, Part I: Proceedings of the 6th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2007), Vilamoura, Algarve, Portugal, pp. 878–896. Springer, Berlin (2007)
28. Przymusinski, T.C.: On the declarative and procedural semantics of logic programs. Journal of Automated Reasoning **5**(2), 167–205 (1989)
29. Roman, D., Lausen, H., Keller, U.: Web service modeling ontology (WSMO). Final Draft D2v1.3, WSMO (2006). <http://www.wsmo.org/TR/d2/v1.3/>
30. Rosati, R.:  $\mathcal{DL}+\text{log}$ : tight integration of description logics and disjunctive datalog. In: KR2006 (2006)
31. Scharffe, F., de Bruijn, J.: A language to specify mappings between ontologies. In: Proceedings of the 1st International Conference on Signal-Image Technology and Internet-Based Systems (SITIS2005). Dicolor Press, Yandoué (2005)
32. Toma, I., Foxvog, D., Jaeger, M.C.: Modeling QoS characteristics in WSMO. In: Proceedings of the 1st Workshop on Middleware for Service Oriented Computing (MW4SOC 2006), Melbourne, Australia, pp. 42–47 (2006)

33. Toma, I., Bishop, B., Fischer, F.: Defining the features of the WSML-Rule v2.0 language. SOA4All Deliverable D3.1.4 (2009)
34. Weibel, S., Kunze, J., Lagoze, C., Wolf, M.: Dublin core metadata for resource discovery. RFC 2413, IETF (1998)

# Chapter 9

## The Web Service Execution Environment

**Abstract** Computer science is on the edge of an important new period of abstraction. A generation ago we learned to abstract from hardware and currently we learn to abstract from software in terms of Service Oriented Architectures (SOA). A Service Oriented Architecture is essentially a collection of services. However, we believe that these SOAs will not scale without significant mechanization of service discovery, service adaptation, service negotiation, service composition, service invocation, and service monitoring, as well as data-, protocol-, and process-mediation. We envisage the future of applied computer science in terms of service-oriented architectures which is empowered by adding semantics as a means of dealing with heterogeneity and mechanization of service usage. This approach is called Semantically Enabled Service-oriented Architectures, or SESA for short. In this chapter, we give an introduction to SESA and Web Service Execution Environment (WSMX) as its most prominent implementation. First, we are motivating the SESA approach, followed by an analysis of SESA vision, and governing principles. Special attention is paid to the notion of Execution Semantics, basic SESA services and WSMX. The elaboration is followed by a larger example demonstrating steps needed to achieve a WSMO goal.

### 9.1 Motivation

In this section, we provide an overview of the fundamental elements of the SESA architecture, which enable the execution of Semantic Web Services (SWS) and resolve the fundamental challenges related to the open SOA environment. We expect that in the near future a service-oriented world will consist of an extremely high number of services. Their computation will involve services searching for other services based on functional and non-functional requirements, and then resolving any interoperability conflicts from the services selected. However, services will not be able to interact automatically and existing SOA solutions will not scale without significant mechanization of the service provisioning process. Hence, machine processable semantics is essential for the next generation of service-oriented computing to reach its full potential. In this chapter, we define methods, algorithms and tools forming a skeleton of SESA, introducing automation to the service provisioning process including service discovery, negotiation, adaptation, composition, invocation,

and monitoring, as well as service interaction requiring data and process mediation. SOA outside of a tightly controlled environment cannot succeed until semantic issues are addressed and critical tasks within the service provisioning process are automated, leaving humans to focus on higher level aspects like problem-solving and business processes. This chapter describes how these building blocks are consolidated into a coherent software architecture which can be used as a blueprint for implementation.

### ***9.1.1 Service Orientation***

The design of enterprise information systems has gone through several changes in recent years. In order to respond to the requirements of enterprises for flexibility and dynamism, the traditional monolithic applications have become substituted by smaller composable units of functionality known as services. Information systems must then be re-tailored to fit this paradigm, with new applications developed as services, and legacy systems to be updated in order to expose service interfaces. The drive is towards a design of information systems based on Service Oriented Computing (SOC) paradigm, SOA implementation architecture and relevant Web service technologies.

#### ***9.1.1.1 Service Oriented Computing***

Service Oriented Computing (SOC) is a new computing paradigm that utilizes services as the fundamental elements for the development of rapid, low-cost and easily integrable enterprise applications [17, 20]. One of the main goals of SOC is to enable the development of networks of the integrated and collaborative applications, regardless of both the platform where applications or services run (e.g., the operating system) and programming languages used to develop them.

In this paradigm, services are autonomous, self-describing and platform-independent computational entities which provide uniform and ubiquitous access to information for a wide range of computing devices (such as desktops, PDAs, cellular phones) and software programs across different platforms. Any piece of code and any application component deployed in a system can be reused and transformed into a network-wide available service, while service providers and service consumers remain loosely coupled.

The main goal of SOC is to facilitate the integration of newly built and legacy applications which exist both within and across organizational boundaries. SOC must overcome and resolve conflicts due to different platforms, programming languages, security firewalls, etc. The basic idea behind this service-orientation is to allow applications which were developed independently (using different languages, technologies, or platforms) to be exposed as services and then interconnect them exploiting the Web infrastructure with respective standards such as HTTP, XML, SOAP, WSDL, and complex service orchestration standards like BPEL.

**Fig. 9.1** Web Services Programming Model



### 9.1.1.2 Service Oriented Architectures

The service oriented paradigm of computation can be abstractly implemented by the system architecture called Service Oriented Architecture (SOA) [1, 2]. The purpose of this architecture is to address the requirements of loosely coupled, standard-based, and protocol-independent distributed computing, mapping enterprise information systems isomorphically to the overall business process flow [19]. This attempt is considered to be the latest development in a long series of advancements in software engineering addressing the reuse of software components.

Historically, the first major step of this evolution has been the development of the concept of a *function*. Using functions, a program is decomposed into smaller subprograms, while writing code is focused on the notion of the application programming interface (API). An API, practically, represents the contract to which a software component must commit. The second major step was the development of the notion of an *object*. An object is a basic building block which contains both data and functions within a single encapsulated unit. With the object-oriented paradigm, the notions of classes, inheritance and polymorphism were introduced. The concept of a *service* becomes the next evolutionary step introduced with the advent of SOC and SOA.

The following Fig. 9.1 is the *Web Services Programming Model* which consists of three components: service consumers, service providers and service registrars. Ignoring the detailed techniques for the connection of the three components, this model also represents the SOA basic model. A service registrar (also known as service broker) acts as an intermediary between the provider and the consumer, so that they are able to find each other. A service provider simply publishes the service. A service consumer tries to find services using the registrar; if it finds the desired service it can set up a contract with the provider in order to consume such a service.

The fundamental logic view of services in SOA is based on the division of service description (called usually interface) and service implementation [19]. Service interface defines the identity of a service and its invocation logistics. Service implementation implements the functionality that the service is designated to provide.

Following this division, service providers and services consumers are loosely coupled. Furthermore, the services can be to a large extent reused and adapted according to certain requirements. Because service interfaces are platform independent and implementation is transparent to the service consumers, a client using any communication device on any computational platform, operating system and any programming language should be capable of using the service. The two facets of the service are distinct; they are designed and maintained as distinct items, though their existence is highly interrelated.

SOA provides a more flexible architecture for realizing distributed applications than its predecessor approaches (e.g., CORBA) by unifying business processes by modularizing large applications into services. Furthermore, enterprise-wide or even cross-enterprise applications can be realized by means of services development, integration, and adaptation. Any system that claims to be SOA compliant must fulfill a set of requirements that were identified and analyzed in [21]:

- **Loose Coupling** Interacting services are loosely coupled by nature. They run on different platforms, are implemented independently and have different owners. The model has to consider the loose coupling of services with respect to one another.
- **Implementation Neutrality** The interface should be the thing that matters, not its implementation. Services are defined independently of their implementation and, with respect to them, should behave neutrally.
- **Flexible Configuration** Services are invoked dynamically after the discovery process through the service requestor. That is, the binding of a service provider to the requester occurs at run time in the final phase.
- **Long Lifetime** Components/services should exist long enough to be discovered, relied upon, and engender trust in their behavior.
- **Granularity** The granularity of a service defines the complexity and number of functionalities defined by an individual service and is thus part of the service model. An appropriate balance between coarse and fine grained services depends on the way services are modeled. For too fine grained services, problems arise when there are frequent and rapid changes.
- **Teams** Computation in open systems should be conceptualized as business partners working as a team. Therefore, a team of cooperating autonomous components/services is a better modeling unit, as opposed to framing computations centrally.

Besides the basic SOA model shown in Fig. 9.1, there are a few extension works towards SOA, which include more detailed concepts. The Extended Service-Oriented Architecture (xSOA) accounts for SOA deficiencies in such areas as management, security, service choreography and orchestration, and service transaction management and coordination [16]. The xSOA is an attempt to streamline, group together and logically structure the functional requirements of complex applications that make use of the SOC computing paradigm. xSOA provides a layer server-based architecture that extends conventional SOA and provides a logical separation of concerns. Extensions, such as xSOA, are steps towards defining a more accurate and

detailed picture of SOA components. In the following subsections, after discussing how SOA can be implemented, we propose a set of semantic-based extensions to SOA that will enable automation support for service related tasks.

### 9.1.1.3 SOA Implementations

Web services become the preferred implementation technology for realizing the SOA promise of maximum service sharing, reuse, and interoperability. As illustrated in Fig. 9.1, the Web service programming model is a typical SOA implementation. Its central concept is the concept of a Web service that is defined by W3C as a software system identified by a URI whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols. Interactions between Web services typically occur as Simple Object Access Protocol (SOAP) calls carrying XML data content. Interface descriptions of the Web services are expressed using Web Services Definition Language (WSDL). The Universal Description, Discovery, and Integration (UDDI) standard defines a protocol for directory services that contain Web service descriptions. UDDI enables Web service clients to locate candidate services and discover their details. Service clients and service providers utilize these standards to perform SOA's basic operations. Service aggregators may use the Business Process Execution Language for Web Services (BPEL4WS) to create new Web services by defining corresponding compositions of the interfaces and internal processes of existing services.

Web services technology is a widespread accepted instantiation of SOC providing a platform on which it is possible to develop applications taking advantage of the already existing Internet infrastructure. This does not mean that the set of technologies we are presenting are the only ones which make possible the realization and implementation of SOC and SOA. Other, more conventional programming languages or middleware platforms may be adopted as well, such as, for instance, established middleware technologies like J2EE, CORBA and IBM's WebSphere MQ, which can now also participate in an SOA, using new features that work with WSDL.

The broad use of Enterprise Application Integration (EA) middleware support a variety of hub-and-spoke integration patterns [18]. EA comprises message acceptance, transformation, translation, routing, message delivery and business process management. All of these can be used to develop services and service orientated architectures. The Enterprise Service Bus (ESB) is an open, standards-based message bus designed to enable the implementation, deployment, and management of SOA based solutions with a focus on assembling, deploying, and managing distributed SOAs. The ESB provides the distributed processing, standards based integration, and enterprise-class backbone required by the extended enterprise [13].

### ***9.1.2 Execution Environment for Semantic Web Services***

With the underpinning computation approach SOC, SOA is one of the most promising software engineering trends for future distributed systems. Pushed by major industry players and supported by many standardization efforts, Web services are a prominent implementation of the service-oriented paradigm. They promise to foster reuse and to ease the implementation of loosely coupled distributed applications.

Although the idea of SOA targets the need for integration that is more adaptive to changes in business requirements, existing SOA solutions will prove difficult to scale without a proper degree of automation. While today's service technologies around WSDL, SOAP, UDDI, and BPEL certainly brought a new potential to SOA, they only provide a partial solution to interoperability, mainly by means of unified technological environments. Where content and process level interoperability is to be solved, ad-hoc solutions are often hard-wired in a manual configuration of services or workflows while at the same time they are hindered by dependence on XML-only descriptions. Although flexible and extensible, XML can only define the structure and syntax of data. Without machine-understandable semantics, services must be located and bound to service requesters at design-time which, in turn, limits possibilities for automation. In order to address these drawbacks, the extension of SOA with semantics offers a scalable integration, more adaptive to changes that might occur over a software systems lifetime. Semantics for SOA allow the definition of semantically rich and formal service models where semantics can be used to describe both services offered and capabilities required by potential consumers of those services. The data to be exchanged between business partners can also be semantically described in an unambiguous manner in terms of ontologies. By means of logical reasoning, semantic SOA thus promotes a total or partial automation of service discovery, mediation, composition and invocation. Semantic SOA is not, however, meant to replace existing integration technologies. The goal is to build a new layer on the top of existing service stack while at the same time adopt existing industry standards and technologies being used within existing enterprise infrastructures.

### ***9.1.3 Governing Principles***

We have identified a number of underlying principles which govern the design of the architecture, its middleware, as well as the modeling of business services. These principles reflect fundamental aspects for service-oriented and distributed environments which promote intelligent and seamless integration and provisioning of business services. These principles include:

- **Service Oriented Principle** represents a distinct approach for analysis, design, and implementation which further introduces particular principles that govern aspects of communication, architecture, and processing logic. This includes service

- reusability, loose coupling, abstraction, compositability, autonomy, and discoverability.
- **Semantic Principle** allows a rich and formal description of information and behavioral models enabling automation of certain tasks by means of logical reasoning. Combined with the service oriented principle, semantics allows the definition of scalable, semantically rich and formal service models and ontologies allowing the promotion of total or partial automation of tasks such as service discovery, contracting, negotiation, mediation, composition, invocation, etc.
  - **Problem Solving Principle** reflects Problem Solving Methods as one of the fundamental concepts of Artificial Intelligence. It underpins the ultimate goal of the architecture which lies in the so-called *goal-based discovery and invocation of services*. Users (service requester's) describe requests as goals semantically and independently from services, while architecture solves those goals by means of logical reasoning over goals' and services' descriptions. Ultimately, users do not need to be aware of processing logic but only care about the result and its desired quality.
  - **Distributed Principle** allows aggregation of the power of several computing entities to collaboratively run a task in a transparent and coherent way, so that from service requester's perspective they can appear as a single and centralized system. This principle allows executing a process across a number of components/services over the network which, in turn, can promote scalability and quality of the process.

## 9.2 Technical Solution

### 9.2.1 SESA Vision

The global view on the architecture, depicted in Fig. 9.2, comprises several layers, namely (i) Stakeholders forming several groups of users of the architecture, (ii) Problem Solving Layer building the environment for stakeholders to access the architecture, (iii) Service Requesters as client systems of the architecture, (iv) Middleware providing intelligence for the integration and interoperation of business services, and (v) Service Providers exposing the functionality of back-end systems as Business Services. In this section, we describe these layers and define service types within the architecture, as well as underlying concepts and technology we use for architecture implementation.

As part of the five layers infrastructure illustrated in Fig. 9.2, four types of services are provided:

- **The stakeholders' layer services** which consist of Ontologies, Applications (e.g., e-tourism, e-government) and Developer tools (GUI tools such as those for engineering ontology/Web Service descriptions; generic developer tools such as language APIs, parsers/serializers, converters, etc.).

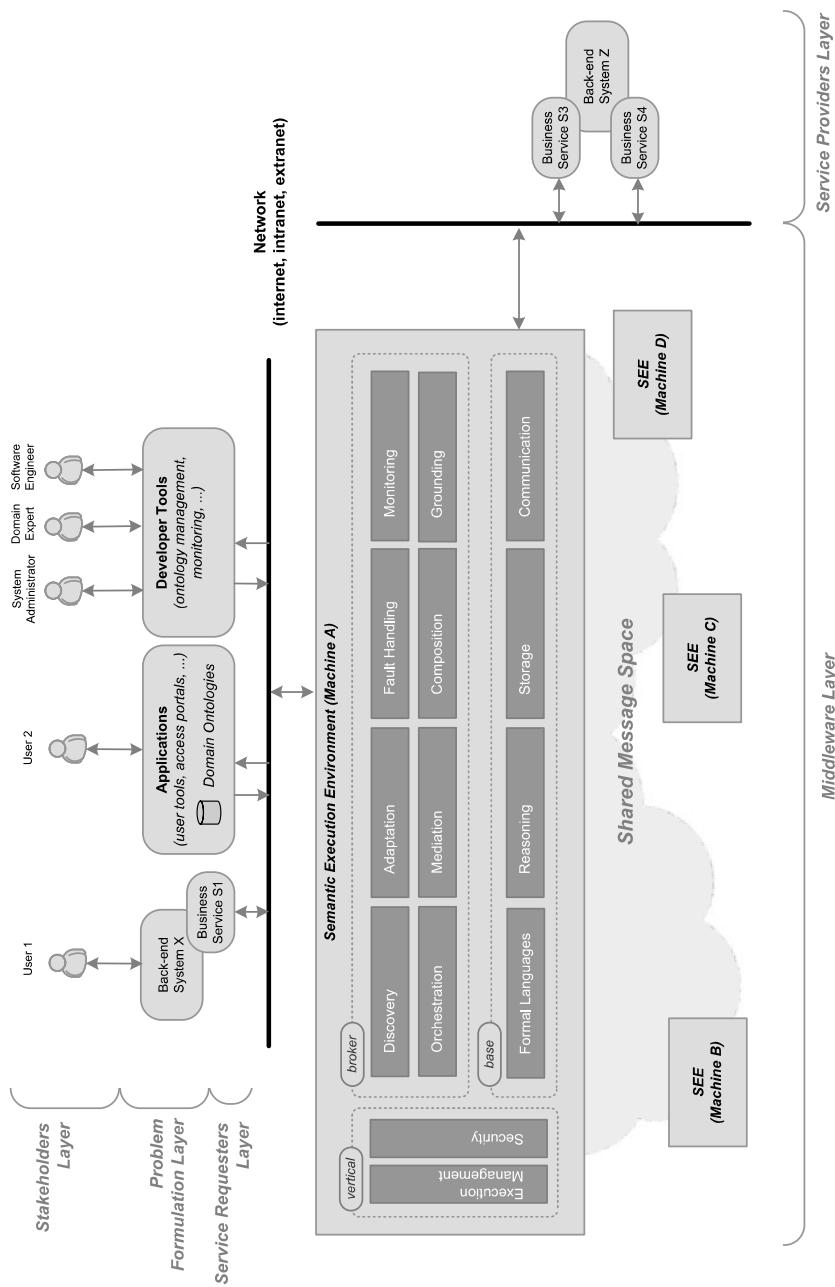


Fig. 9.2 Global view on SESA architecture

- **Broker services** which consist of Discovery, Adaptation (including selection and negotiation), Composition (Web Service composition techniques such as planning), Choreography, Mediation ((a) Ontology mediation—techniques for combining Ontologies and for overcoming differences between Ontologies; (b) Process mediation—overcoming differences in message ordering, etc.), Grounding, Fault Handling (Transactionality, Compensation, etc.), and Monitoring.
- **Base services** that are providing the exchange formalism used by the architecture, i.e., Formal languages (static ontology and behavioral, i.e., capability/choreography/orchestration languages, connection between higher-level descriptions, e.g., WSML), Reasoning (techniques for reasoning over formal descriptions; LP, DL, FOL, behavioral languages, etc.), and Storage and Communication.
- Finally, **vertical services** such as Execution management and Security (authentication/authorization, encryption, trust/certification).

### 9.2.1.1 Stakeholders' Layer

Stakeholders are the users of the functionality of the architecture. Two basic groups of stakeholders are identified: users and experts. For users the architecture provides end-user functionality through specialized applications. For example, users can perform electronic exchange of information to acquire or provide products or services, to place or receive orders and to perform financial transactions. In general, the goal is to allow users to interact with business processes online while simultaneously reducing their physical interactions with back-office operations. Experts perform development and administrative tasks in the architecture. These tasks support the entire SOA life cycle including service modeling, creation (assembling), deployment (publishing), and management. Different types of experts could be involved in this process ranging from domain experts (modeling, creation), system administrators (deployment, management) and software engineers.

### 9.2.1.2 Problem Solving Layer

The Problem Solving layer contains applications and tools which support stakeholders during the formulation of problems/requests and generates descriptions of such requests in the form of user goals. Through the Problem Solving layer, users are able to solve their problems, i.e., formulate a problem, interact with the architecture during processing and get their desired results. This layer contains back-end systems which directly connect the middleware within business processes, specialized applications built for specific purposes in a particular domain, which also provide specific domain ontologies, and developer tools providing functionality for development and administrative tasks within the architecture. Developer tools provide a specific functionality for experts, i.e., domain experts, system administrators and software engineers. The functionality of developer tools cover the whole SOA life cycle including service modeling, creation (assembling), deployment (publishing),

and management. Using tools such as these, a developer will be able to create and manage ontologies, Web Services, goals and mediators, create ontology-to-ontology mediation mappings and deploy these mappings to the middleware.

### 9.2.1.3 Service Requesters' Layer

Service requesters act as client systems in a client–server setting of the architecture. Their requests are represented as goals created through problem/request formulation as well as interfaces through which conversation with potential services is expressed.

### 9.2.1.4 Middleware Layer

Middleware is the core of the architecture, providing the intelligence for the integration and interoperation of Business Services. For the SESA purposes, we call this middleware Semantic Execution Environment. The architecture defines the necessary conceptual functionality that is imposed on the architecture through the underlying principles defined in Sect. 9.1. Each such functionality could be realized (totally or partially) by a number of the so-called middleware services (see Sect. 9.2.1.6). We further distinguish this functionality through the following layers: base layer, broker layer, and vertical layer.

**Vertical Layer** The Vertical layer defines the middleware framework functionality that is used across the Broker and Base Layers but which remains invisible to them. This technique is best understood through the so-called “Hollywood Principle” that basically means “Don’t call us, we’ll call you”. In this respect, framework functionality always consumes functionality of Broker and Base Layers, coordinating and managing overall execution processes in the middleware. For example, Discovery or Data Mediation is not aware of the overall coordination and distributed mechanism of the Execution Management.

- **Execution Management** defines control of various execution scenarios (called execution semantics) and handles distributed execution of middleware services.
- **Security** defines secure communication, i.e., authentication, authorization, confidentiality, data encryption, traceability or non-repudiation support applied within execution scenarios in the architecture.

**Broker Layer** The Broker layer defines the functionality which is directly required for a goal based invocation of Semantic Web Services. The Broker Layer includes:

- **Discovery** which defines tasks for identifying and locating of business services which can achieve a requester’s goal.
- **Choreography** which defines formal specifications of interactions and processes between the service providers and the client.

- **Monitoring** which defines monitoring of the execution of end point services; this monitoring may be used for gathering information on invoked services, e.g., QoS related or in identifying faults during execution.
- **Fault Handling** which defines handling of faults occurring within execution of end point Web Services.
- **Adaptation** which defines an adaptation within particular execution scenario according to user preferences (e.g., service selection, negotiation, contracting).
- **Mediation** which defines interoperability at the functional, data and process levels.
- **Composition** which defines a composition of services into an executable workflow (business process). It also includes orchestration which defines the execution of a composite process (business process) together with a conversation between a service requester and a service provider within that process.
- **Grounding** which defines a link between semantic level and a non-semantic level (e.g., WSDL) used for service invocation.

**Base Layer** The Base layer defines functionality that is not directly required in a goal based invocation of business services; however, they are required by the Broker Layer for successful operation. Base layer includes:

- **Formal Languages** which define syntactical operations (e.g., parsing) with semantic languages used for semantic description of services, goals and ontologies.
- **Reasoning** which defines reasoning functionality over semantic descriptions.
- **Storage and Communication** which define persistence mechanism for various elements (e.g., services, ontologies) as well as inbound and outbound communication of the middleware.

The SESA middleware can operate in a distributed manner when a number of middleware systems connected using a shared message space operate within a network of middleware systems and empowering this way a scalability of integration processes. SESA consists of several decoupled services. These services can be refined and extended. Each of these services can have its own structure without hindering the overall SESA architecture. Following the SOA design principles, the SESA architecture separates concerns of individual middleware services, thereby separating service descriptions and their interfaces from implementation. This adds flexibility and scalability for upgrading or replacing the implementation of middleware services which adhere to required interfaces.

#### 9.2.1.5 Service Providers' Layer

Service providers represent various back-end systems. Unlike back-end systems in the service requesters' layer which act as clients in the client–server setting of the architecture, the back-end systems in the service providers' layer act as servers which provide certain functionality for a certain purpose exposed as a business service to the architecture. Depending on the particular architecture deployment and integration scenario, the back-end systems could originate from one organization (one

service provider) or multiple organizations (more service providers) interconnected over the network (Internet, intranet or extranet). The architecture thus can serve various requirements for Business-to-Business (B2B) integration, Enterprise Application Integration (EAI) or Application-to-Application (A2A) integration. In all cases, functionality of the back-end systems is exposed as semantically described business services.

### 9.2.1.6 Services in SESA

Most of the SESA functionality is provided as services, while a very small part of the functionality required for the overall management is not. While the Middleware and Service requesters' layers build SESA architecture in terms of services (with some exceptions), the Problem Solving layer adds the set of tools and entities which make SESA a fully-fledged Semantic Service-Oriented Architecture.

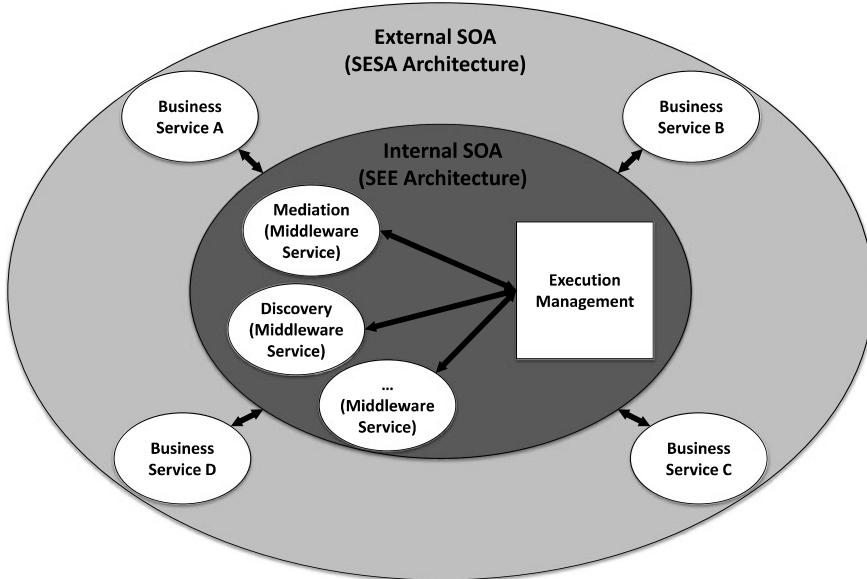
The core aspect of SOA is the service. In this respect, we distinguish between two types of services in SESA, namely middleware services and business services.

1. **Middleware Services** are necessary to enable some particular functionality of the architecture—they are the main facilitators for integration, search, and mediation of business services.
2. **Business Services** are exposed by service providers whose back-end systems are external to SESA. Business services are subject of integration and interoperation within the architecture (facilitated by the middleware services) and usually provide a certain value for architecture stakeholders. Such services are semantically described in the SESA architecture, conforming to a specific semantic service model.

SESA defines **the scope of particular middleware services** in terms of the functionality they should provide. In addition, it defines a **semantic service model for the business services** on which it operates. Business services are modeled in accordance to the requirements of the application scenario in which they are used. For this purpose, domain-specific ontologies can be designed as part of SESA application design or evaluation. With respect to middleware services and business services distinction, the SESA middleware is designed as the Service Oriented Architecture on its own. However, it serves as a facilitator for the integration of semantic business services and as such is not currently considered as semantically-enabled but rather as semantics-enabling. Illustrating the above, Fig. 9.3 depicts middleware and business services within the scope of SESA architecture.

### 9.2.1.7 Underlying Concepts and Technology

The SESA architecture builds on the underlying concepts and technologies and further extends specifications around conceptual model defined by Web Service Modeling Ontology (WSMO), Web Service Modeling Language (WSML), and the reference middleware implementation called WSMX. WSMO (see Chap. 7) provides a



**Fig. 9.3** Middleware services and business services

conceptual model describing all relevant aspects of Web Services in order to facilitate the total or partial automation of service discovery, composition and invocation. The description of WSMO elements is represented using the WSM family of ontology languages (see Chap. 8) which consists of a number of variants based on different logical formalisms and different levels of logical expressiveness. WSMO and WSM thus provide grounds for semantic modeling of services as well as for a Semantic Web services based middleware system which specifically enacts Semantic Service Oriented Architectures. Reference implementation of this middleware system is called WSMX. WSMO is the model being developed for modeling of business services. WSMX is the reference implementation of the SESA and provides various middleware services which facilitate integration of semantic business services. In addition, Web Service Modeling Toolkit (WSMT) provides an end-user IDE for modeling of business services and run-time management of the middleware environment.

### 9.2.2 SESA Middleware

This section provides a technical description for the middleware layer of the SESA architecture, introduced in the last section, following the view-centric guidelines of the IEEE Recommended Practice for Architectural Descriptions of Software-

intensive systems (IEEE 1471<sup>1</sup>). We look in more detail at the functionality required by each of the middleware services and suggest technology-neutral interfaces for them. This section also includes a description of the technology used to design and implement an open source prototype for the SESA middleware. The last section provided a high-level view of what is meant by SESA. This section extends the technical description by providing a view on the processes supported by SESA and how they are defined in two additional views: (i) Services View—functional descriptions and interfaces of the middleware and base services and (ii) Technology View—a look at the technology used in the design and implementation of the WSMX prototype for SESA.

### 9.2.2.1 Services Viewpoint

Services are the basic building blocks of the SESA architecture. In other words, SESA uses a Service-Oriented Architecture to provide an intelligent middleware for service-oriented applications. This means that the middleware services provided by SESA can themselves be used by SESA to allow for exchanging data models, process strategies, implementations or policy.

In the following subsections, we describe the three middleware service layers with one exception. Security is a major topic in its own right, and we make the assumption that SESA adopts the security mechanisms of the underlying technologies on which it builds (Web service and message-based systems). We are primarily interested in the broker and base services as they provide the novel aspects of the middleware.

### 9.2.2.2 Broker Services

**Service Discovery** The discovery service is concerned with finding Web service descriptions that match the goal specified by the service requester. A service requester provides the WSMO description of a goal to be achieved (described in terms of the desired capability with preconditions, assumptions, effects, and postconditions). This is matched against WSMO descriptions of Web services available to the discovery service as the level of description between goals and Web Services will naturally vary and because it is impractical for service descriptions to include all possible information on the functionality they provide.

Various discovery approaches have been developed and implemented as part of the WSMX prototype for SESA. The theoretical foundation for service discovery developed in [8, 14] clarifies a number of conceptual issues related to service discovery. It proposes a distinction between service discovery and Web service discovery and identifies three major steps in service discovery, following the idea of

---

<sup>1</sup><http://www.iso-architecture.org/ieee-1471/>.

**Table 9.1** Discovery service interface

Aspect	Detailed information
Description	Provides access to the Semantic Web Service discovery mechanism of SESA based on the conceptual model provided by WSMO
Provided To	The Core service which has the responsibility of managing the life cycle of instances of Execution Semantics, and it may therefore need to invoke the Discovery Service. Processes on SESA are executed as instances of Execution Semantics. Additionally, the Orchestration Engine service may also need to make invocations to the Discovery service (see Sect. Sect. 9.2.2)
Inputs	A WSMO Goal and, optionally, input data specified as instances of concepts from one or more WSMO ontologies
Outputs	A list of unordered WSMO Web service descriptions

heuristic classification that has been identified in [3]. In this approach, a problem is first lifted to a more abstract level, then matching of abstract entities (requests and services) is performed, and finally the candidate set is refined with respect to concrete data.

The interface provided by the discovery service is described in Table 9.1.

**Service Ranking** The ranking service provides a means to identify the best services given a set of services functional compatible with the user's request. It produces an ordered list of services according to user's preferences and non-functional properties criteria like price, availability, etc. Once a ranked list of Web services that can fulfill the user's concrete goal is prepared, SESA or the user must then choose which of the services to invoke.

Our approach for ranking and selection [22] is a multi-criteria ranking approach based on evaluation of non-functional properties. Non-functional properties specified in goal and service descriptions are formalized in a semantic language (i.e., WSML) by means of logical rules using terms from NFP ontologies. The logical rules used to model NFPs of services are evaluated during the ranking process by a reasoning engine. Additional data is required during this process: (i) in which NFPs the user is interested, (ii) the importance of each of these NFPs, (iii) how the list of services should be ordered (i.e., ascending or descending), and (iv) concrete instance data extracted from the goal description. The non-functional properties values obtained by evaluating the logical rules are sorted and the order list of services is built. The corresponding ranking interface is described in Table 9.2.

**Data Mediation** Every time data is exchanged as messages between services, the owner of each service must be confident that they will be able to correctly interpret the data types used in the messages and that all parties engaged in the message exchange share a common understanding of the meaning of those data types.

In the SESA architecture, the Data Mediation service has the role of reconciling data heterogeneity problems that might appear during discovery, composition,

**Table 9.2** Ranking service interface

Aspect	Detailed information
Description	Provides access to the ranking mechanism that takes into account non-functional properties descriptions
Provided To	Ranking may be called by the Core service, after Discovery has been invoked, as part of the execution of an instance of Execution Semantics
Inputs	A WSMO goal including non-functional properties, preferences over these properties and input data
Outputs	A list of WSMO Web service descriptions, ordered based on the values of the non-functional properties specified by the requester in the goal description

**Table 9.3** Data mediation interface

Aspect	Detailed information
Description	Applies mapping rules between ontologies on instances of the source ontology to create instances of the target ontology
Provided To	Data mediation may be required during service discovery, orchestration and as an essential part of process mediation
Inputs	IRIs identifying the source and target ontologies, and a set of instances of concepts from the source ontology
Outputs	A set of instances of concepts from the target ontology corresponding to the input source ontology instances. The relationship between the input and output instances is defined by mapping rules

selection or invocation of Web services. This service uses the inter-ontology mappings described in a declarative mapping language, such as that defined in [15]. This facilitates scalability, as a single logical service can provide all data mediation requirements, independent by the number of ontologies that must be mediated. Naturally, at run-time this service depends on the availability of the mappings. The creation of such mappings requires the involvement of domain experts supported by the availability of mapping tools such as the one provided in WSMT.

Two aspects of these mappings need to be taken into account in the design of the Data Mediation service. The Data Mediation service needs to have access to a persistent repository in which the mappings are stored. In addition, as the mappings are not tied to a formal language, they require grounding to such a language before they can be used. Once grounded, the mappings can be used by a reasoner to generate instances of a target ontology based on instances of the source ontology. This implies a dependency from the Data Mediation service to the Logical Reasoner service. There is a single interface for the data mediation service and this is shown in Table 9.3.

**Choreography Engine** In WSMO, choreography interface descriptions define both the messages that an entity expects to send and receive, and the control flow between those messages. Choreography interfaces belonging to Web services are related to how the overall capability of the service can be achieved while those belonging to goals are related to how the supplier of the goal wishes to interact with a service in order to invoke its capability. For example, a Web service selling IBM notebook computers may use the RosettaNet PIP3A4<sup>2</sup> protocol to specify that it expects to send and receive the following messages in this order:

- Receive a purchase order (PO) message.
- Send a purchase order acknowledgment message (POA).
- Send a purchase order conformation message (POC).
- Receive a purchase order conformation acknowledgment message (POCA).

This message exchange can be considered as a definition of a business process to achieve a given capability. It does not care whether a service with that capability is designed to implement all its functionality by itself, or if it uses other services to do so. What the choreography interface is explicitly stating is that the only way to obtain the service functionality is through the exchange of the specified messages in the specified order. WSMO choreography interfaces are defined using ontologized abstract state machines. In essence, each time a goal and Web service are determined to match each other, an instance of each of their choreographies must be created and the flow of messages through those choreography interfaces handled. The Choreography Engine is responsible for this task.

As abstract state machines, choreography interfaces consist of a universe (the kinds of data available to the choreography) and a set of rules (conditional, update, block-parallel) that operate on instances of data from the universe. The Choreography Engine service for SESA has to be able to handle these descriptions where the universe is represented by the State Signature, in terms of WSMO ontologies, and the rules are specified using logical expressions written in terms of the WSML variant being used.

The engine works as follows. First, choreography interface instances for the goal and Web service are created. Each has an associated knowledge base to which instances of ontological concepts can be added. After each change to either knowledge base, the conditions in the headers of the respective sets of rules are examined to see if any of the rules *fire*. The firing of a rule may result in data being either sent to or from the goal or Web service owning that choreography interface. The interface for the choreography engine is described in Table 9.4.

**Orchestration Engine** In WSMO, each Web service may achieve its functionality through a composition of other Web services. This is described as the *orchestration* of the Web service and resembles a process definition. Each step in the orchestration can be represented as a WSMO goal or Web service description. Describing a process in terms of goals to be achieved at each step greatly adds flexibility as the

---

<sup>2</sup><http://www.rosettanet.org/>.

**Table 9.4** Choreography engine interface

Aspect	Detailed information
Description	Allows registration and state-update operations. Registration involves telling the engine to create an instance of a choreography interface (and associated knowledge base) in its internal memory. This may be a new or pre-existing choreography interface. The latter is the case for long running interactions whose state should be stored in persistent storage and re-loaded when new messages for that choreography instance arrive. Updating the state means providing the instance of the choreography interface with data needed by the control flow it specifies. Rules are run over the provided data and may result in the invocation of a service
Provided To	The choreography engine can be invoked directly after discovery if there is no data or process mediation required. Otherwise, it is expected that the Process Mediation service creates the messages that this service receives
Inputs	A set of ontological instances corresponding to a subset of the concepts defined in the state signature of the choreography
Outputs	No value is returned

binding of goals to services can be deferred to execution time and handled by the execution environment.

Although it may seem counterintuitive to provide internal details of how a service is composed in the service description itself, this can prove to be very useful in environments where dynamic service composition is a requirement or where the software process is stable, but the providers of steps in the process may change. For example, a Web portal for a telecom broker may allow a customer to create a request for a product bundle including broadband, landline and provisioning for a Web site. The broker has access to multiple service providers for each product and wants to defer the choice of the best set of providers for each bundle request until run-time. The customer request is written as a goal and matched against a Web service with an orchestration having three steps (one for each product being ordered) each represented by a WSMO goal. Using goals means that the broker retains the flexibility of allowing each goal to be bound to the most suitable service at run-time when the service is needed. This is in contrast to hard-wiring a business process to specific services when the process is designed.

As with any other form of integration, each Goal in a composition described by an orchestration may have its own information model. This mirrors a common situation in workflows where the data flow aspect can only be defined across heterogeneous data models through the use of data transformations. These data transformations often operate on the structure rather than the semantics of the data. We have already described how the data mediation service overcomes data heterogeneity through mappings between ontological concepts. This more generic and flexible approach can be applied to data flow between steps in a service orchestration. Data mediation in this context can be explicitly or implicitly declared. Implicit means that when the orchestration description is created, specific data mediation steps are introduced to the orchestration description. Explicit means that the engine executing the orchestration description has sufficient information available at run-time to

**Table 9.5** Orchestration engine interface

Aspect	Detailed information
Description	An instance of an orchestration must be registered to the engine before it can be used. The registration operation creates an orchestration instance and its associated knowledge base. The ontologies used by the orchestration may be imported to the knowledge base at this point or later, e.g., when the first message for the orchestration arrives. A second operation of the interface called updateState is required to allow new data to be made available to the orchestration instance so that its state can be updated
Provided To	The orchestration engine is accessed by the Core of SESA which executes the abstract descriptions of the various supported execution semantics We describe the Core in Sect. 9.2.2.3
Inputs	The initialization of the orchestration requires the identifier (IRI) of the Web service description containing the orchestration
Outputs	The registration returns an identifier to the instance of the orchestration. The update state operation does not return any value

determine if (i) mediation is required and (ii) if a mapping between the source and target ontologies is available to be executed.

Assuming that suitable mappings exist between the two ontologies, the orchestration engine finds and applies them on-the-fly at run-time. This is because an orchestration is essentially a composition of choreographies where the output of the choreography of a goal (or service) at one step provides input for the choreography of another step. Where a data mismatch occurs, the orchestration engine can examine the corresponding choreographies to determine the ontologies used and the associated mappings (see Sect. 9.2.2.2) and apply these mappings to the instance of the concept to be transformed.

In this way, it may prove more effective to carry out this work when the orchestration is being composed. Explicit calls to the Data Mediation service can be incorporated into the orchestration description as required. Table 9.5 describes the interface of the Orchestration Engine.

### 9.2.2.3 Base Services

The base services represent functional services fundamental to the infrastructure of the SESA system. They are described in the following sections in terms of their functionality and interfaces.

**Core** The SESA architecture essentially supports distribution as each part of its functionality is provided as a service that can be designed and implemented in a distributed system. A coordinating service (the Core) takes the responsibility for providing a messaging infrastructure and for managing administrative tasks such as the loading of services and fault monitoring. Service loading can either occur through a defined list when the Core is started (bootstrapping) or on-the-fly (hot-deployment).

**Table 9.6** Core: ManageSystem interface

Aspect	Detailed information
Description	In this design, the Core provides the microkernel for the SESA system. It must be explicitly started and stopped, it must be possible to add and remove services while the system is running, and it should be possible to extract information on the health of the system as a whole
Provided To	The operations provided by this interface are aimed at system administrators and would most likely be best presented through a graphical user interface dashboard type of application
Inputs	Starting takes no input. Stopping takes an identifier of the system. Monitoring requires the system identifier and identifiers of one or more monitoring types on which information should be gathered
Outputs	Starting returns an identifier for the started system. Stopping and service management have no explicit functional return information aside from possible confirmation messages. Monitoring returns information regarding the health and throughput of services as well as information about the system as a whole

It is important to note that the only functional restriction on the location of services with respect to the Core is that they can be identified using Web identifiers and reachable through the chosen messaging implementation. For example, a shared tuple space would mean that services could be located on any physical machine or network that has access to that shared space (see Chap. 10).

The messaging infrastructure must be transparent to the other services in the architecture, thus enabling strong decoupling. To achieve this transparency, the loading and hot-deployment of the services must take care of adding whatever deployment code is necessary to the services. One possible technique for this is the use of deployment configuration files and wrappers. The Core is responsible for handling three main functional requirements:

- A framework for management including monitoring, and starting and stopping the system.
- Facilitating communication between services. Messages must be accepted and routed to a suitable target service, enabling the communication and coordination of services.
- Support for the life cycle of Execution Semantics. Execution Semantics specify the process-types that are available to run on the SESA architecture. They enable the separation between the definition of what the architecture does from how it achieves. Multiple definitions of Execution Semantics are supported and multiple instances of each Execution Semantics may run concurrently.

There are three interfaces for the Core. These are ManageSystem, Messaging and ExecutionSemantics, shown in Tables 9.6, 9.7, and 9.8, respectively.

**External Communications Manager** While the Core provides the infrastructure for messaging between the various services that make up the SESA middleware, there is also a need to interact with services and entities that are external to SESA.

**Table 9.7** Core: Messaging interface

Aspect	Detailed information
Description	The Core provides the functionality of a message bus accepting messages represented as events and routing each of them to a target service that has subscribed to that event. The routing takes place in the context of a particular instance of an Execution Semantics
Provided To	All services in the architecture participate in the messaging mechanism but this should be transparent to them. The transparency should be made possible by the addition of whatever additional code is necessary at the time the service is deployed to the SESA system, using a service deployment descriptor (typically, an XML file). It is this additional code that interacts with the messaging system, isolating it from the service. The service should only be invoked by its defined interface
Inputs	Subscription requires the specification of an event type for the message and an identifier for the service instance. Sending a message must specify the message type and the instance of the Execution Semantics to which it belongs. Receiving a message means that the Core must be able to <i>push</i> a message to services that have subscribed to that particular message type. Another possibility that does not require subscription is to use a <i>pull</i> method where services check the Core at regular intervals to see if any message of a suitable type is available. Either way, the target service must receive the message and the instance of the Execution Semantics specifying the context in which it was sent
Outputs	In terms of the logical functionality, there is no explicit output

**Table 9.8** Core: Execution Semantics interface

Aspect	Detailed information
Description	Execution Semantics is the name given to the process definitions that the SESA architecture can support. It separates the description of the process from its implementation
Provided To	SESA provides an external interface that allows for a service requester to request that a Goal be achieved or that a specific service be invoked. This interface is attached to the Communication Manager service. Each such user request results in the creation of an instance of a suitable Execution Semantics to deal with the request. Once an instance of an Execution Semantics exists, it is driven by the receipt of events representing messages from specific services. For example, in the Execution Semantics description for the AchieveGoal process, once the Execution Semantics receives an event indicating that the Discovery Service has completed successfully, it will raise an event for Process Mediation to take place
Inputs	Creation of an instance of Execution Semantics requires the identifier of the corresponding type. Updating an instance of an Execution Semantics requires its identifier and an event indicating the outcome of some services functionality
Outputs	Starting an execution semantics returns an identifier. Receiving an event requires the event identifier and the identifier for the instance of the Execution Semantics (assuming that these identifier point to the complete data in persistent storage)

This is the role of the External Communications Manager service (ECM). The service should be open to supporting as many transport and messaging protocols as possible and this support should be transparent to SESA.

**Table 9.9** ECM: Invoker interface

Aspect	Detailed information
Description	The ECM Invoker handles all invocations from the SESA architecture to external services. It is responsible for carrying out any necessary transformations and if security or encryption is required, it should coordinate any associated tasks
Provided To	The Choreography Engine service (see Sect. 9.2.2.2) is responsible for raising events that lead to the Invoker interface being called
Inputs	The identifier of the service being invoked, the list of WSML instances to be sent and the identifier of the grounding that identifies the exact operation and endpoint for the invocation
Outputs	The invocation of a service is treated as asynchronous from the perspective of the Choreography Engine raising the invocation event. In actual fact, the <i>wrapper</i> attached to the ECM at deployment times shields the Choreography Engine from needing to know about this. If the Web service invocation is synchronous, then the wrapper should use an individual thread to wait until the Web service response is returned and then raise a suitable event to be passed to the messaging mechanism of the Core. Where the Web service invocation is synchronous, there is no return data immediately associated with the invocation. Rather the service must make a subsequent callback to an endpoint on the SESA architecture if it has data to return

The link between the semantic description of a service and the concrete specification of the service's endpoints and operations (e.g., provided by WSDL) is defined by the grounding. When a service is about to be invoked, the ECM service must be able to manage any transformations necessary in terms of data representation and/or communication protocols. For example, typically Web Services expect to receive SOAP messages over HTTP with the SOAP payload represented in XML conforming to a specified XML Schema definition. As the internal representation of data within SESA is in terms of WSMO, the data must be *lowered* from a rich semantic language to the structure syntax of XML.

Communication must also be supported in the opposite direction, i.e., from a service requester to SESA. Continuing the last example of a Web service using SOAP over HTTP with XML content, messages must be *lifted* to WSML before they can be further processed by the services making up SESA. Although, currently very popular, the combination of SOAP, HTTP and XML are not guaranteed to be the only message, transport and data specifications used by external services. Many other valid combinations are possible and, where practical, should be supported by the ECM. There are two interfaces defined for the ECM service: Invoker described in Table 9.9 and Receiver in Table 9.10.

**Parser** The information model for the content of all messages passed between the SESA services (e.g., Discovery, Selection, Mediation, etc.) is defined by the WSMO metamodel. The language used to represent the content and define the semantics of these messages is WSML. However, WSML in its native state is not easily consumable by the implementation of the individual services. It is useful to be able to parse WSML into a form that is more suitable for application programming. Such a form

**Table 9.10** ECM: Receiver interface

Aspect	Detailed information
Description	The ECM Receiver handles all incoming messages from external entities. There are two categories of messages. The first are from SESA users who wish to run one of the processes defined by the definitions of Execution Semantics provided for SESA. These include service discovery, Goal achievement and specific Web Service invocation. The second category is for messages that are sent to SESA from a service as part of an already on-going conversation, e.g., the asynchronous response to a Web service operation mentioned in the description of the Invoker interface
Provided To	External entities using SESA for particular tasks
Inputs	When achieving a Goal, the Goal description or an IRI that resolves to the Goal description along with WSML instance data to act as input must be provided. The inputs for executing a Web service are almost the same, with Web service description required in place of the Goal description. For discovery, the Goal identifier is needed along with optional ontological instance to restrict the set of valid discovered services
Outputs	Achieving a Goal may result in output WSML data being returned to the external requester. Likewise, the execution of a specific service. For discovery, a possibly empty set of Web service descriptions is returned

is provided by WSMO4J [23], an API and reference implementation for WSMO using Java.

WSMO4J is a convenient programming abstraction for WSMO semantic descriptions. However, it is not a prerequisite. Likewise, the parser functionality is only required if the SESA design cannot natively work with WSML at the interfaces of each component. For descriptive purposes and convenience, we make the assumption that WSMO4J is used in SESA for the remainder of this chapter.

When and where parsing takes place is a design and implementation issue. In designs where the services are implemented in Java and hosted on a common Java virtual machine (JVM), or on multiple JVMs that can exchange objects, it may make sense to parse WSML messages as soon as they arrive at the Receiver interface and then use the internal WSMO4J model as the internal SESA representation. Where services are designed as standalone Web Services, it may makes sense to exchange messages between SESA services in WSML and allow each service to carry out WSML parsing as they require. Table 9.11 describes the Parser interface.

**Reasoner** To increase the level of automation for processes performed in distributed systems following the Service-Oriented Architecture principles, SESA proposes a solution based on semantic descriptions of the entities involved in the use of Web services as a technology for application and system integration. Such semantic descriptions are processed and understood by a reasoning engine that is able to answer questions. The reasoning engine or services provides a basic functionality that is required other SESA services. Discovery, Data Mediation and Process Me-

**Table 9.11** Parser interface

Aspect	Detailed information
Description	The Parser transforms WSML descriptions into the Java object model provided by WSMO4J
Provided To	Accessible by any of the SESA services that wish to use WSMO4J as their internal object model
Inputs	A WSML document
Outputs	A set of WSMO4J objects

**Table 9.12** Reasoner interface

Aspect	Detailed information
Description	The reasoner enables queries that may require logical inferencing to be made over a knowledge base defined by ontologies using a defined formal logic model, e.g., Description Logics (DL)
Provided To	Accessible by any of the SESA services but is required by Discovery, Data and Process Mediation. It is also required by the algorithms defined in this chapter for the Choreography and Orchestration engines
Inputs	One or more ontologies must be available to the reasoning engine to establish its knowledge base. Queries to the engine take the form of logical expressions
Outputs	A possibly empty set of identifiers that provide an answer to each query put to the Reasoner

diation services rely on the logical reasoning. For service discovery, the reasoner must determine if the capability of a Goal can be matched by the capability of one or more Web services, also expressed as logical expressions. This requires a reasoning engine that answers queries on the basis of the underlying formal logic model used by the semantic language. It is desirable that the Reasoning Engine interface be provided either through Java, supporting WSMO4J, or as a Web service that accepts and returns WSML expressions. The interface for the Reasoner is described in Table 9.12.

**Resource Manager** The Resource Manager Service provides access to persistent storage for the SESA architecture. Persistence at the system level is required for all WSMO entities used during the operation of the WSMX system as well as the non-WSMO entities, e.g., those representing messages and their states used during the execution of processes provided by the architecture (through execution semantics). There are two interfaces defined for the Resource Manager: WSMO Repository in Table 9.13 and Non-WSMO Repository in Table 9.14.

**Table 9.13** WSMO repository interface

Aspect	Detailed information
Description	This provides retrieval, add, update, remove (and possibly archive) operations on a repository of WSMO descriptions for Web services, Goals, Ontologies and Mediators
Provided To	Accessible by any of the SESA services
Inputs	When adding or updating a WSMO element, a valid WSML description or corresponding set of WSMO4J Java objects is required. When removing, the identifier(s) for the element(s) to be removed must be provided. Individual WSMO elements can be retrieved by specifying their unique identifier while it should be possible to retrieve all WSMO descriptions of a particular type, i.e., Web service or Goal or Mediator or Ontology
Outputs	When adding, updating, or removing there is no specific output beyond possible confirmation messages. When retrieving a WSML document containing all the requested elements or a corresponding set of WSMO4J objects should be returned

**Table 9.14** Non-WSMO repository interface

Aspect	Detailed information
Description	There are many aspects of the operation of the SESA architecture that require persistent storage. One example is that instances the Choreography Engine service should remain stateless, storing and retrieving instances of choreographies as required based on unique identifiers for those instances. Another example is that the architecture should keep track of the life cycle of events, representing messages between services in the context of an instance of an Execution Semantics. This is important for monitoring the health of the system and for maintaining historical information for performance monitoring over time
Provided To	Accessible by any of the SESA services. Ideally, this service takes advantage of an object-relational mapping framework like Hibernate that abstracts the details of the persistence from the application developer
Inputs	Depends on the SESA service
Outputs	Also depends on the SESA service

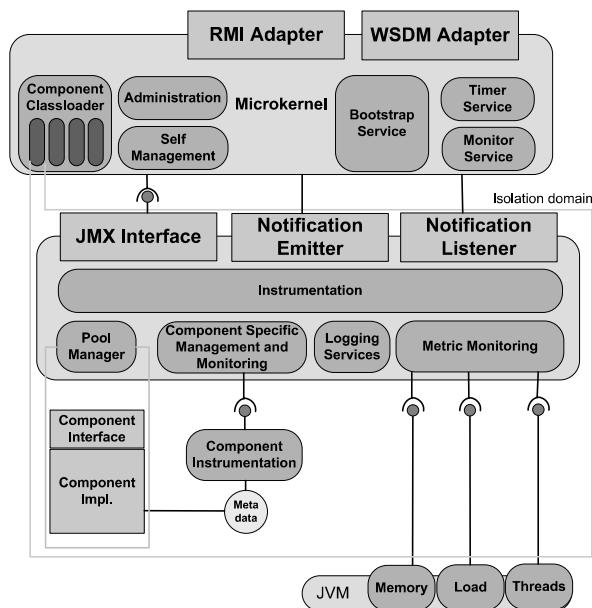
#### 9.2.2.4 Technology Viewpoint

In this section, we provide some details on the technology used in the design for the Core component of the Web Services Execution Environment (WSMX),<sup>3</sup> an open-source prototype of the SESA architecture. The purpose is to provide some insight into the challenges of creating a software design that realizes the SESA architecture.

**Core Design** The Core component coordinates the other components of the architecture. It manages the interaction between the other components through the exchange of messages containing instances of WSMO concepts expressed in WSML,

<sup>3</sup><http://sourceforge.net/projects/wsmx>.

**Fig. 9.4** Component management in the architecture core



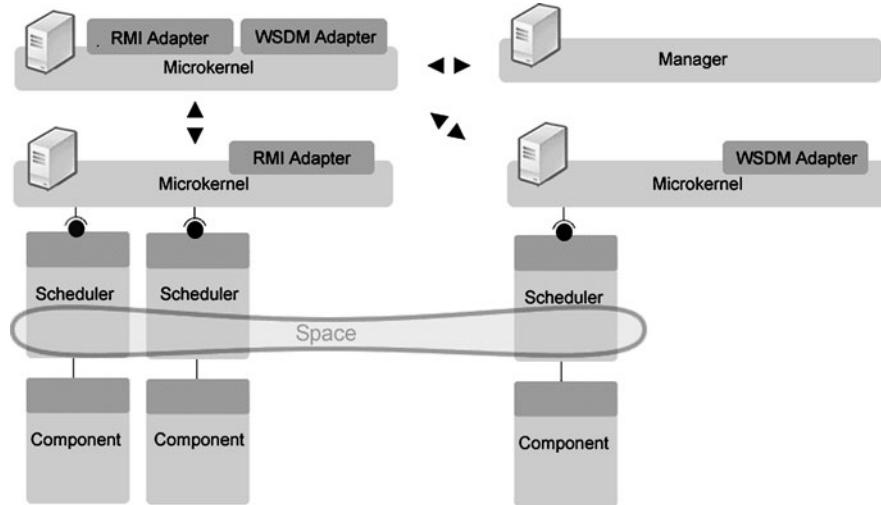
and provides the microkernel and messaging infrastructure for the architecture. This component is responsible for handling three main functional requirements:

- A framework for the management and monitoring the start and stop of the system as well as its health.
  - The Core facilitating the communication between components; it accepts messages and routes them to a suitable target component, enabling communication and coordination of components.
  - Supporting the life cycle of Execution Semantics (declarative definitions of the allowed operational behavior of the architecture). Multiple definitions of execution semantics are supported and multiple instances of each execution semantics may run concurrently.

The WSMX prototype of the Core component is realized as a microkernel utilizing Java Management Extensions (JMX) and is described in detail in [12].

**Management** Managing system components is a critical issue for the healthy functioning of any system. The very process of making management explicit, captures an invariant that helps to leverage the support for dynamic change of the rest of the system. Figure 9.4 presents an overview of the infrastructure provided by the Core to its components, which allows the management and monitoring of the system.

The Core offers several dedicated services, the most essential of which is perhaps the *bootstrap service* responsible for loading and configuring the application components. The Core also employs *self-management* techniques such as scheduled operations, and allows *administration* through a representation independent



**Fig. 9.5** Communication and coordination in the architecture core

management and monitoring interface that allows for a number of different management consoles serving different purposes. Text-Terminal-, Web-Browser- and Eclipse-Framework-based consoles have been implemented.

As is common with middleware and application servers, the Core hosts a number of subsystems that provide services to components and enable inter-component communication. *Pool management* takes care of handling several component instances and subsystems. The Core is in the unique position of offering services to the individual components, such as logging services, transport services and life cycle services.

One of the principles underlying the architecture design is support for distribution. The design of the Core allows for transparent distribution of middleware services in two ways. Firstly, the use of a shared virtual message space allows services to be physically located on different servers. Secondly, it is possible for multiple WSMX implementations to be configured as a federation, where if a middleware service is not available at one site, the Core at that location can send a message to another WSMX requesting the middleware service at that remote site.

**Communication and Coordination** The architecture avoids hard-wired bindings between components—communication is based on events. Whenever a middleware service is required, an event representing a request for that service is created and published. Any services subscribed to the event type can fetch and process the event. The event-based approach naturally allows asynchronous communication. As illustrated in Fig. 9.5 events exchange is conducted using a shared Tuple Space (originally demonstrated in Linda [9]; see Chap. 10 for more details) which provides persistent shared space enabling interaction between parties without direct event exchange between them.

The Tuple Space enables communication between distributed units running on remote machines or on the local machine. It should be emphasized that the functional components themselves (discovery, mediation, etc.) are completely unaware of this distribution. That is, an additional layer provides them with a mechanism of communication with other components that shields them from the actual mechanism of transport which can be local or remote.

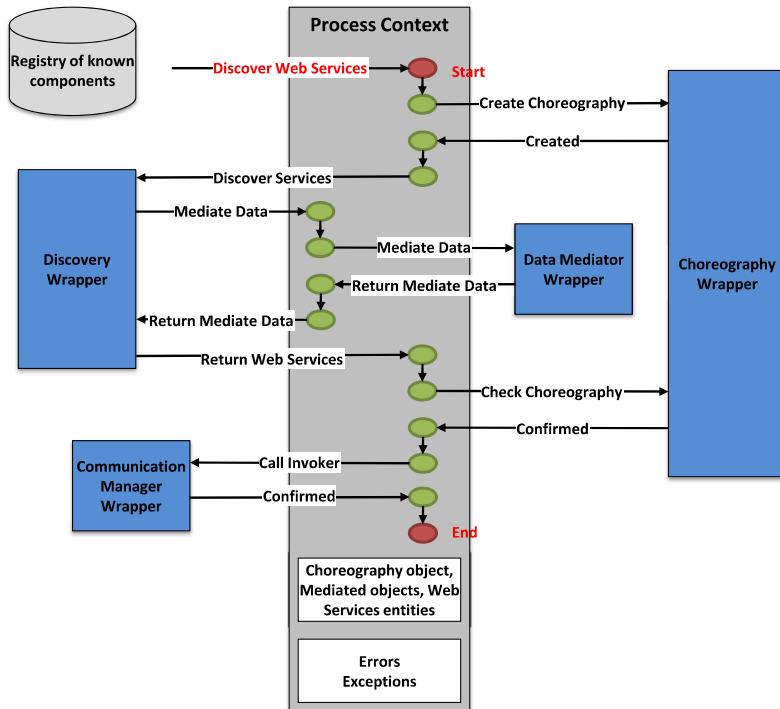
Through the infrastructure provided by the Core, component implementations are separated from communication issues. This infrastructure is made available to each component implementation during instantiation of the component carried out by the Core during a *bootstrap* process. This is the process that occurs when a component is identified and loaded by the WSMX prototype. Through the use of JMX and reflection technology, this can occur both at start-up as well as after the system is up and running. In particular, the communication infrastructure carries the responsibility to interact with the transport layer (a Tuple Space instance).

To enable one middleware service request functionality of another, a proxy mechanism is utilized. The calling service asks the WSMX Core for a proxy to the other middleware service it wishes to invoke. Once provided, the proxy can be used as if it were a direct interface to the required service. In fact, the proxy is implemented to publish an event with a type targeted at the required service and returns any results to the calling service.

Each middleware service uses configuration metadata which allows the Core to select the service and generate wrappers for it as it is deployed. Figure 9.6 shows a process represented by an Execution Semantic including the middleware services and associated wrappers. The wrappers are generated and managed by the Core to separate middleware services from the transport layer for events. One wrapper raises an event with some message content and another wrapper can at some point in time consume this event and react to it.

*Wrappers* are generic software units that separate the implementation of WSMX components from the communication mechanism. They provide methods to components, enabling communication with other components. *Wrappers* are automatically generated and attached to each component during instantiation. This is carried out by the WSMX Kernel. There are two major parts:

- **Reviver** Its responsibility is to interact with the transport layer (i.e., the Tuple Space). Through the transport layer, the *Reviver* subscribes to a proper event-type template. Similarly, the *Reviver* publishes result events in the Tuple Space. This level of abstraction reduces the changes required to code if the transport layer changes.
- **Proxy** A *Proxy* is utilized for one component to be able to request another's functionality. The calling component specifies the component name, the method to be invoked and the required parameters. The proxy creates the proper event for this data and publishes it in the Tuple Space. A unique identifier is assigned to the outgoing event and is preserved in the response event. Proxies subscribe to the response event by specifying a unique identifier in the event template. This guarantees that only the *Proxy* that published this event will receive the result event.



**Fig. 9.6** Process, its wrappers and context

### 9.2.3 SESA Execution Semantics

The advent of SOA makes it more practical to separate the definition of how a system operates from the implementation of the system itself. We define execution semantics as formal descriptions of the operational behavior of the system (analogous to business process descriptions in BPEL, but at the system level). By separating the description of the system behavior from the system implementation, we can achieve greater flexibility of how the implementations of the SESA architecture can be used and avoid rebuilding the system in the event of a new or changed requirement of how the system should operate. In terms of the behavior of the architecture, we introduced two types of services: middleware services and business services. Middleware services provide the functional services required by SESA to operate. Business services are exposed by information systems external to the SESA and the role of the SESA platform is to coordinate interoperation between them. The execution semantics are defined as middleware services describing in a formal and unambiguous way how the system built of loosely-coupled middleware services operates.

This chapter describes the execution semantics which should be supported by SESA. We present three fundamental scenarios (using UML activity diagrams). We envision new, dynamically created execution semantics reflecting changing busi-

ness needs where middleware services can be composed in the flexible way according to the principles of SOA. We provide an example which shows how execution semantics operating on middleware services can be used in practice. We also look at the execution semantics from WSMX technical perspective.

### 9.2.3.1 Motivation

The specification of system behavior can be viewed as the data and control flow between system components (services in SOA), where the actual actions take place. System designers tend to create architectures for specific needs that often result in rigid system behavior difficult to adapt to changing requirements. In SESA, middleware services that make up the system building blocks can be used in various scenarios some of which may not be known at design time. For instance, one execution semantics may specify Goal-based discovery of Semantic Web Services, while another may define Semantic Web service invocation. Both execution semantics are defined explicitly using the same middleware services, but with different data and controls flows. Middleware services cooperate with each other at the interface level but they do not refer to each others implementation directly.

A software design process should result in a design that is both an adequate response to the user's requirements and an unambiguous model for the developers who will build actual software. A design therefore serves two purposes: to guide the builder in the work of building the system and to certify that what will be built will satisfy the user's requirements. We define execution semantics, or operational semantics, as the formal definition of system behavior in terms of computational steps. As such, it describes in a formal, unambiguous language how the system operates. Since, in a concurrent and distributed environment, the meaning of the system to the outside world consists of its interactive behavior, this formal definition is called execution semantics.

The following lists major advantages of specifying system behavior by execution semantics over informal methods are the following:

- **Foundations for model testing** It is highly desirable to perform simulation of the model before the actual system is created and enacted. It allows one to detect anomalies like deadlock, livelock or tasks that are never reached. However, as pointed out by Dijkstra in [6], model simulation allows pointing out the presence of errors, but not the lack of them. Nevertheless, it is paramount to detect at least some of system malfunctions during design time rather than during the run-time. Therefore, semantics of utilized notations have to be perfectly sound in order to create tools enabling simulation of created models. Only formal, mathematical foundations can meet these requirements.
- **Executable representation** Similarly, as in the case of model testing, using formal methods provides a sound foundation for building an engine able to execute created models. Such an engine would not necessarily need to be able to detect livelock and other model faults. This fact distinguishes this bullet from the previous one.

- **Improved model understanding among humans** Soundness of the utilized method improves the comprehensiveness of the model.

Several methods exist to model software behavior. Some of them model system behavior in a general way (like UML diagrams), others impose more formal requirements on models (for instance, Petri net-based methods, process algebra, modal and temporal logics, and type theory). These methods have different characteristics: some are more expressive than others; some are more suited for a certain problem domain than others. Some methods provide graphical notation like UML or Petri nets, some are based on syntactic terms like process calculi and logics; some methods have tool support for modeling, verification, simulation or automatic code generation.

We impose two major requirements on the methodology utilized for modeling fundamental SESA behavior. Firstly, it has to use a understandable and straightforward graphical notation. Secondly, it has to be unambiguous. These two requirements are met by UML Activity Diagrams which are familiar to the engineering community and whose execution semantics can be disambiguated, for instance, in the semantics specified by Eshuis [7].

### 9.2.3.2 Description Formalism

UML 2.0 (Unified Modeling Language) is a widely accepted and applied graphical notation in software modeling. It consists of a set of diagrams for system design, capturing two major aspects, namely static and dynamic system properties. Static aspects specify system structure, its entities (objects or components) and dependencies between them. These structural and relational aspects are modeled by diagrams like Class Diagrams, Component Diagrams and Deployment Diagrams. Dynamic aspects of the system are constituted by control and data flow within the entities, specified as Sequence Diagrams, State Machine Diagrams and Activity Diagrams. Originally, UML was created for modeling aspects of object-oriented programming (OOP). However, it has to be emphasized that UML is not only restricted to the usage in OOP area, but is also applied in other fields such as Business Process Modeling.

The primary goal of UML diagrams is to enable common comprehension of structure and behavior of modeled software among involved parties (e.g., designers, developers, other stakeholders, etc.). To the detriment of UML notation, this information is conveyed in an informal way that may lead to ambiguities in certain cases as pointed out in [11]. Since we want to model the behavior of SESA in a widely used and understood manner, UML is the natural choice.

UML Activity Diagrams depict a coordinated set of activities that capture the behavior of a modeled system. They are used to specify control and data flow between the entities providing language constructs that enable it to model elaborate cases like parallel execution of entities or flow synchronization. For the SESA architecture, they are used to identify the middleware services used in each specific execution semantics along with the control and data flow between them.

In each of the activity diagrams provided in this document, the actions that provide steps in the execution semantics are identified inside a red dashed-line box. The middleware services for Discovery, Mediation, Selection, etc. are drawn as UML activities with interfaces that accept inputs and provide output as data objects. These interfaces are consistent with the defined SESA middleware services. In each of the provided activity diagrams, rounded rectangles with small square input pins denote actions. The pins represent input and output data ports. Rounded rectangles with large rectangular boxes at their boundaries (e.g., Communication Manager in Fig. 9.7) indicate activities that correspond to SESA middleware services. The rectangular boxes at the boundaries represent parameters that the activity accepts as input or provides as output. Data flow directional arrows can be identified as those beginning or ending at a data pin or a parameter box. The actions inside the red boundary are the responsibility of the execution semantics. Control flow directional arrows have no associated data and go from action to action inside the execution semantics. Finally, the vertical thick black bars are data-flow branches. Data flowing into a branch is made available at all outgoing flows. Where multiple input pins are available on an action, the action does not commence until data has arrived at each of those pins. Data becomes available on all outgoing pins of an action, as soon as that action is completed.

Due to the wide proliferation of UML notation several efforts were made to make concrete its semantics, especially regarding the dynamic aspects of UML notation. Semantics for UML Activity Diagrams, in the context of workflow modeling, was specified in [7]. UML Activity Diagrams fulfill our requirements; therefore, they are used as the graphical notation in this document, consistently with the semantics given by Eshuis, to specify operational behavior of SESA.

### 9.2.3.3 Mandatory Execution Semantics

In the following subsections, we define three fundamental execution semantics identified for the SESA and describe the details of each. The SESA defines a set of middleware services that are required to achieve the functionality of a Semantic Web Services execution environment. Interfaces are defined for each middleware service, defined in terms of the conceptual model provided by WSMO. The combination of all interfaces provides the basis for the SESA application programming interface (SESA API). Section 9.2.3.3 describes the scenario where Goal based service discovery without service invocation is desired. Section 9.9 is an extension of that described in Sect. 9.2.3.3—both service discovery and service invocation are required. The client’s Goal and all data instances the client wishes to send to the potential provided service are specified together as input. Section 9.2.3.3 describes the situation in which clients already know the service they wish to invoke and have the required data for this invocation available.

In each of the subsequent sections, execution semantics are linked to defined entry points to the SESA architecture. Entry points can be considered as input ports, to which messages can be sent, for initiating specific execution semantics. The following notation is used (similar to the convention used by the JavaDoc documentation

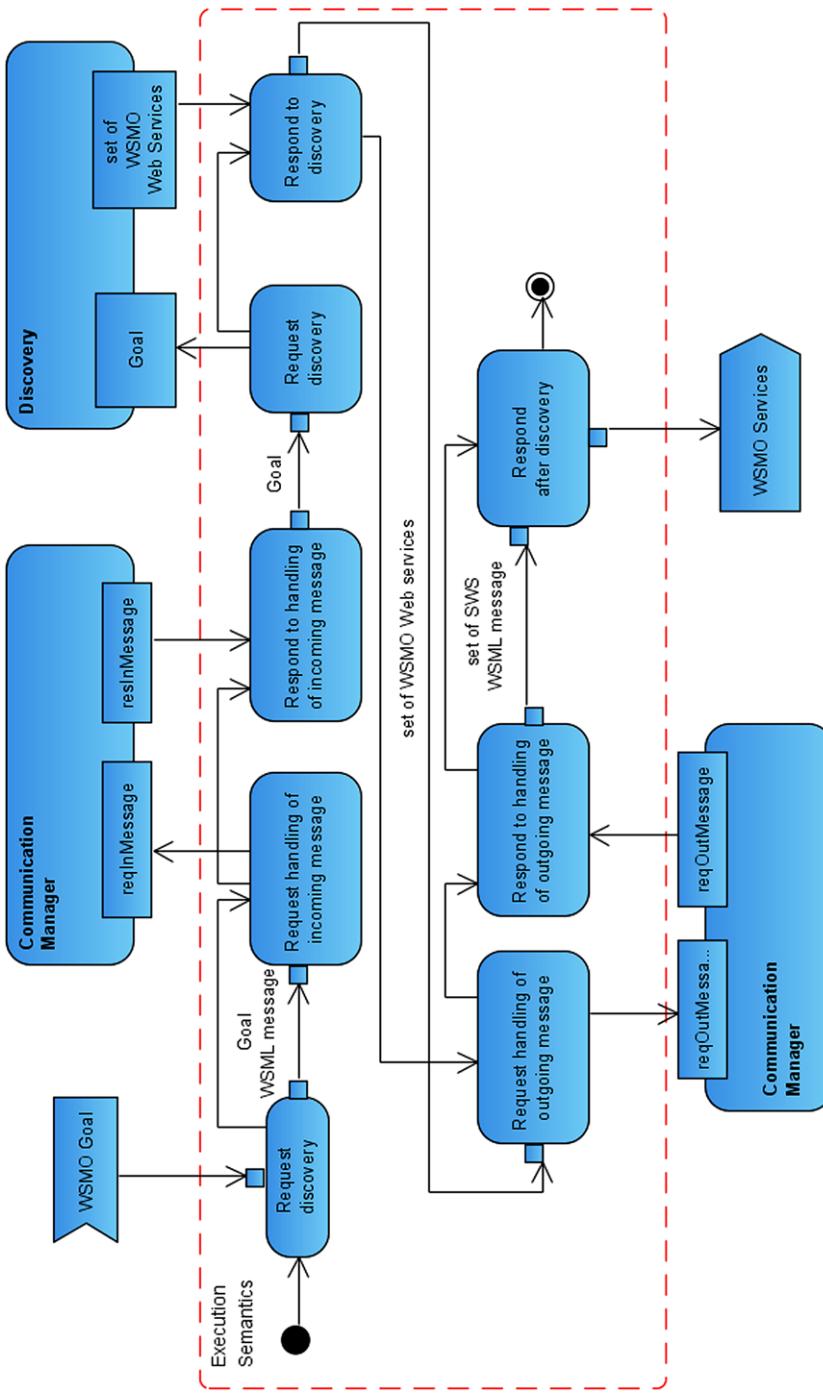


Fig. 9.7 Goal based service discovery

system), where execSemName stands for the name of the entry point to the system for the execution semantics, input data types are the types of the input parameters required to start the execution semantics, and return data type is the data type returned if the execution semantics completes successfully.

#### **execSemName (Input data types): Return data type**

We do not show Parser component in the following sections since we treat it as a transformation component whose only role is to perform the transformation and validation from the textual form of WSM to object-oriented representation which can be more easily utilized by other SESA components.

**Goal-based Web Service Discovery** The following entry-point initiates this system behavior:

#### **getWebServices(WSMLDocument): Web services**

A service requestor wishing to discover a list of Semantic Web services fulfilling its requirements provides its requirements as a Goal description using WSM. A set of WSM Web service descriptions whose capability matches the Goal is returned.

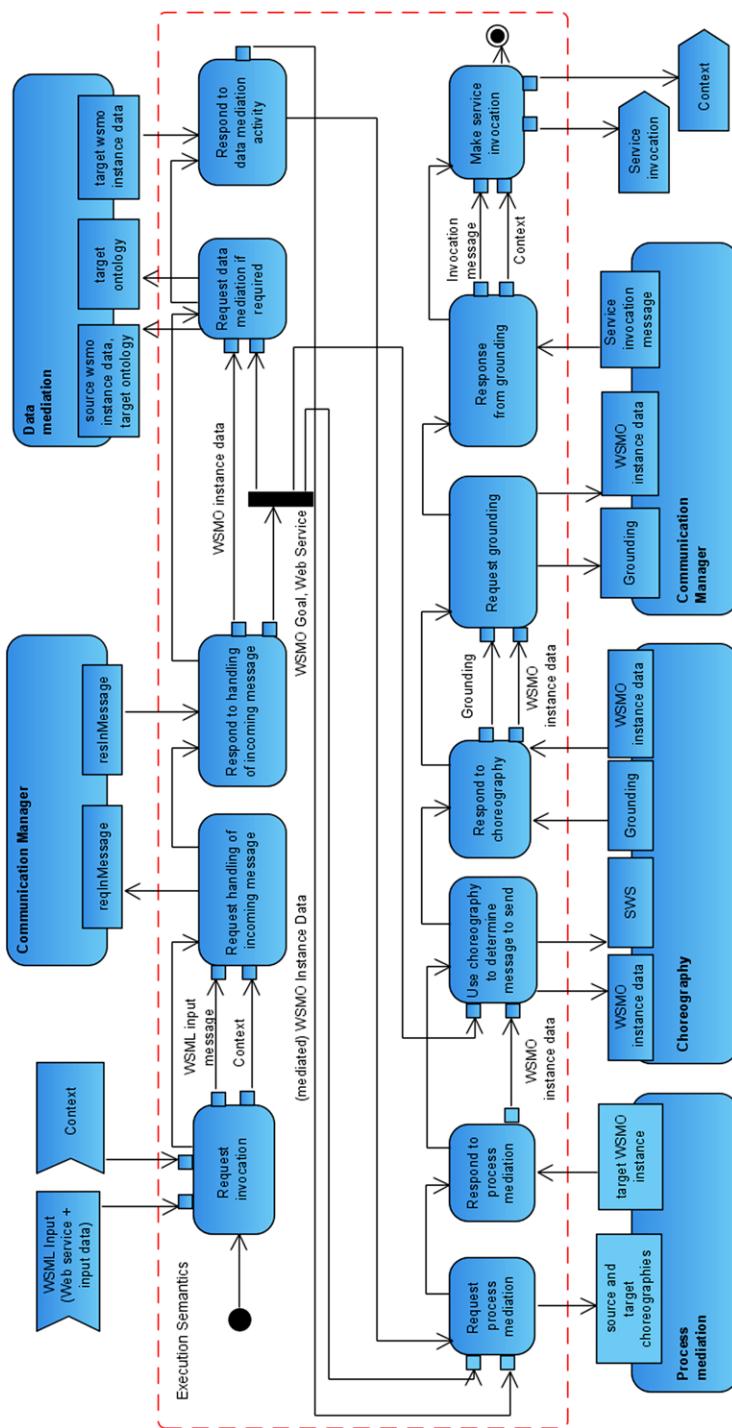
Figure 9.7 shows the activity diagram. A message containing a WSM Goal is received and passed to the Communication Manager which takes care of whatever message persistence that may be required. The Goal is sent to the Discovery activity which looks for WSMO Web service descriptions with capabilities that match that of the Goal. The discovery activity may need Data Mediation as part of its internal design but this internal aspect of that activity is not modeled as part of this execution semantics. Discovery continues until a set of matching Web services is compiled. Assuming this set is non-empty, the selection activity determines which of the discovered services to return to the service requester. The WSMO description of the selected service must be transformed to WSM and packaged into a message that can be sent back to the requester. The Communication Manager takes care of this activity ending the execution semantics for Goal based discovery.

**Web Service Invocation** The following entry-point initiates this system behavior:

#### **invokeWebService(WSMLDocument, Context): Context**

where the service requester already knows which Semantic Web Service they wish to invoke, the execution semantics illustrated in Fig. 9.8 is used. The first input signal provides the WSM content while the second provides a conversation context. If this is the first message being sent to the Web service, there will be no existing conversation and this context signal will be empty. As in the other two scenarios, the WSM input message is unpacked by the Communication Manager activity. In this case, however, the WSM input contains both the target Web service description and the input instance data.

After the WSM has been received, the WSMO Web service description and the instance data are passed to the Data Mediation activity to handle any data mapping that may be required (may require a WSM reasoner to be invoked internally).



**Fig. 9.8** Web service invocation

After that, Process Mediation has all input data at its input pins and can commence. We do not describe the internal design of the Process Mediation activity here but we do note that it is responsible for matching the message exchange patterns required by the Goal and offered by the Semantic Web Service description. This matching includes both the message patterns and the data types used by the respective messages. Process mediation determines how the input data instances need to be organized so that they can be understood by the Web service choreography description. The output of Process Mediation is WSMO data instances organized in this manner.

Once output from Process Mediation is available, the Choreography activity has all required input and (most likely using a reasoning engine as part of its internal design) determines which messages need to be sent out based on the choreography description (an abstract state machine) of the Web service). This WSMO data must be transformed into a format defined by the Web service grounding information (taken from the WSDL service binding) and sent to the Web service endpoint. This activity is carried out by the Communication Manager.

**Goal-based Service Execution** The following entry-point initiates this system behavior:

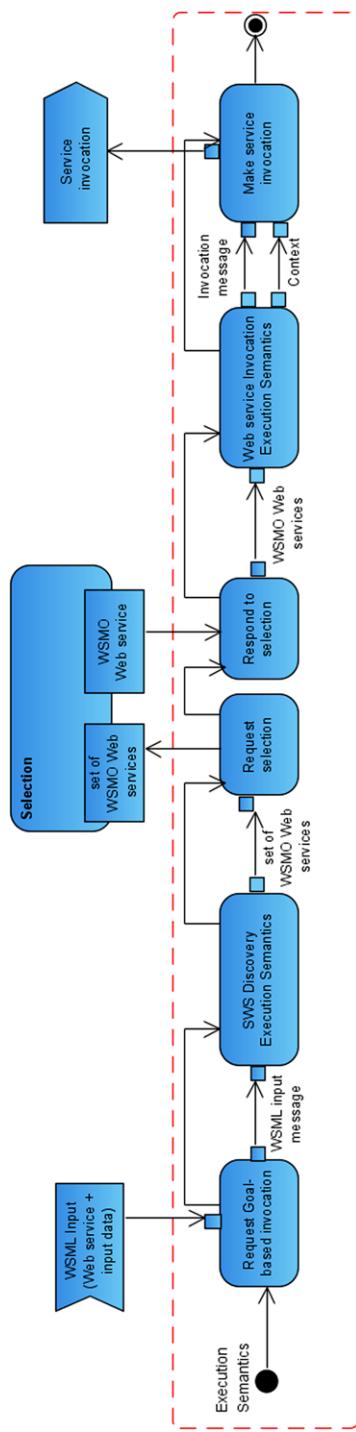
#### achieveGoal(WSMLDocument): Context

A service requestor wishing to use SESA for all aspects of Goal based service invocation (discovery, mediation, invocation) provides both the Goal and input data descriptions in a single WSML document. A unique identifier for the conversation initiated by the receipt of the Goal is returned by the execution semantics to the service requester. In parallel, the Goal and the WSML instance input data is processed by the execution semantics to discover and invoke a service whose capability matches that described by the Goal description.

This scenario illustrated in the UML activity diagram of Fig. 9.9 is based on the assumption that the service requestor is able to provide, up-front, all input data required by the discovered Web service.

The execution semantic is initiated by the receipt of the Goal and input data messages. As in Fig. 9.7, the Communication Manager activity is used to unpack the WSML content of the messages. Then the Discovery Execution Semantics commences as described in Sect. 9.2.3.3, after which the Selection takes place complete, the resulting Semantic Web Service is sent to the Web service Invocation Execution Semantics as presented in Sect. 9.2.3.3.

Note that this execution semantic, without the discovery and selection activities, could be used to handle messages returned from the Web service to be sent back to the service requester, assuming that the service requester provided an invocable endpoint at which it could receive such messages.



**Fig. 9.9** Goal based service execution

### 9.3 Illustration by a Larger Example

In this section, we present a case scenario which tends to solve B2B integration issues. Enterprise integration is an essential requirement for today's successful business. With the aim of overcoming heterogeneity, various technologies and standards for the definition of languages, vocabularies and integration patterns are being developed. For example, RosettaNet defines standardized Partner Interface Processes (PIPs) which include standard inter-company choreographies (e.g., PIP3A4 Request Purchase Order (PO)) and the structure and semantics of business messages. Although such standards certainly enable B2B integration, they still suffer from several drawbacks. All partners must agree on using the same standard, and often the rigid configuration of standards makes them difficult to adapt to local business needs.

The SESA architecture when applied to the B2B integration can help in resolving interoperability problems between business partners and in maintaining the flexible and more reliable integration. In order to demonstrate the value of the SESA in the B2B integration domain, we implemented a scenario from the SWS Challenge.<sup>4</sup> The SWS Challenge aims at establishing a common understanding, evaluation scheme, and testbed to compare and classify various approaches to services integration in terms of their abilities as well as shortcomings in real-world settings.

As Fig. 9.10 depicts, the scenario introduces various service providers (such as Racer and Mueller) offering various purchasing and shipment options for various products through the e-marketplace called Moon. On the other hand, there is a service requester called Blue who intends to buy and ship a certain product for the best possible price. The technology that the Moon operates on is the middleware system enabling the SESA architecture. The following are the basic characteristics of the scenario.

- **Back-end systems** Service requesters and providers use various back-end systems for handling interactions in their environment. For example, Mueller uses Customer Relationship Management system (CRM) and Order Management System (OMS) while Blue uses a system based on the RosettaNet<sup>5</sup> standard. RosettaNet is the B2B specification defining standard components called Partner Interface Processes (PIPs) which include standard intercompany choreographies (e.g., PIP3A4 Purchase Order), and structure and semantics for business messages. Service requesters and service providers maintain the integration with their back-end systems through Web services (adapters for their back-end systems) while at the same time they are responsible for their integration with the middleware through semantic descriptions of services and/or through interfaces with the middleware.
- **Modeling and publishing of semantic descriptions** Engineers on the requester's and provider's side model services and requests using the WSMO model and publish them in the Moon middleware repositories. Engineers of the Moon define mappings between different ontologies in repositories and store them in the middleware.

---

<sup>4</sup><http://www.sws-challenge.org>.

<sup>5</sup>RosettaNet (<http://www.rosettanet.org>).

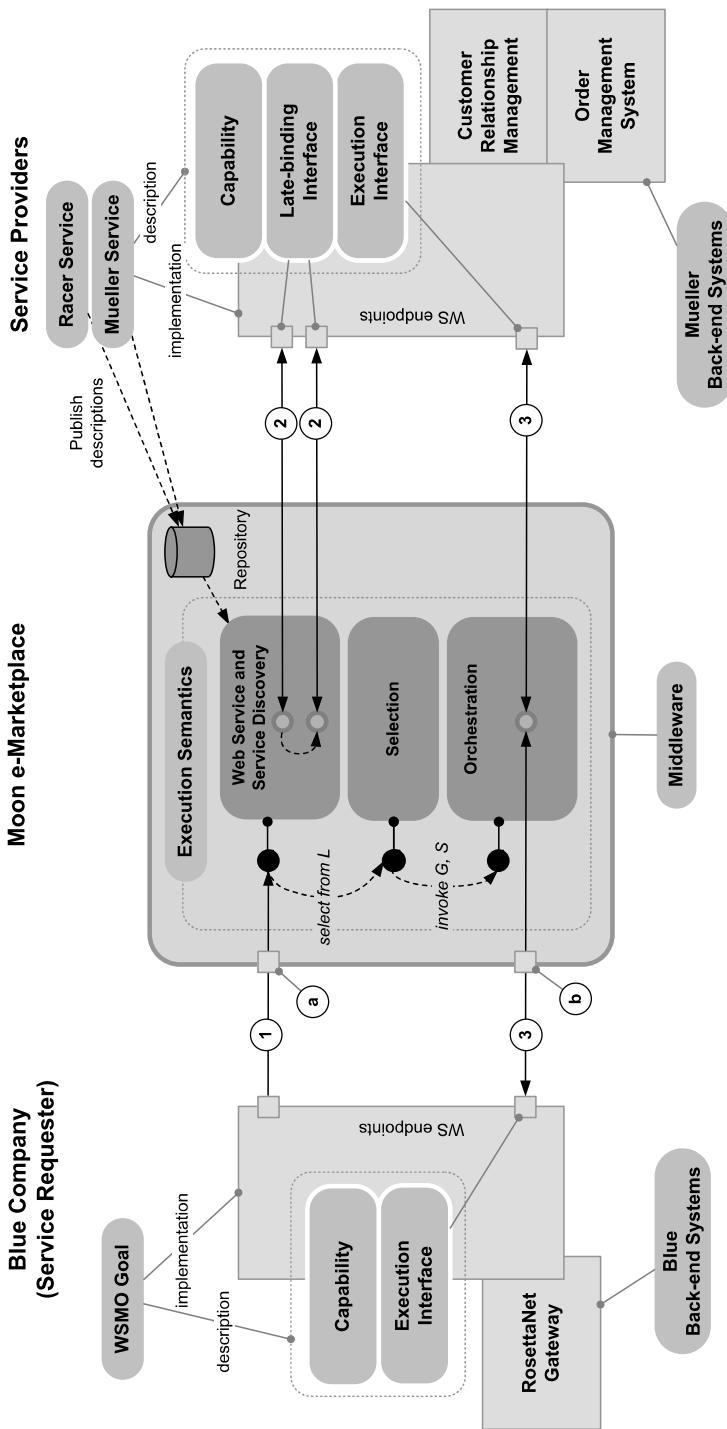


Fig. 9.10 Running example

- **Interoperability issues** Engineers on the requester's and provider's side model services independently, meaning that they use different ontologies as well as different descriptions of choreographies. For example, the Blue models the request and response messages according to the RosettaNet PIP3A4 Purchase Order (PO) specification while Mueller models the messages using proprietary information and choreography specifications of the CRM/OMS systems. For a request message sent by Blue, the Mueller must perform several interactions with the backend systems such as identify a customer in the CRM, open the order in the OMS, add all items to be ordered to the OMS and close the order in the OMS.

The scenario runs as follows: All business partners first model their business services. After that, Blue sends the purchase order request captured in WSMO goal to the middleware system which on reception of the goal executes the **achieveGoal** execution semantics including: (i) **discovery**, (ii) **selection**, and (iii) **orchestration**. During discovery, the matching is performed for the goal and potential services at abstract level as well as instance levels (abstract-level discovery allows narrowing down the number of possible Web services matching a given Goal while instance-level discovery carries out detailed matchmaking considering instance data of the service and the goal). During selection, the best service is selected (in our case Mueller service) based on preferences provided by Blue as part of the WSMO goal description. Finally, during orchestration, the execution and conversation of Blue and Mueller services is performed by processing of choreographies' descriptions from the Blue's goal and the Mueller's service.

### **9.3.1 Modeling of Business Services**

In this section, we show how Blue and Mueller systems can be modeled using WSMO and WSML formalisms. The modeling phase involves following steps: (i) **Web Service Creation** when underlying services as Web services with WSDL descriptions are created, and (ii) **Semantic Web Service and Goals Creation** when semantic service and goal descriptions are created using WSMO. For our scenario, we create two services, namely PIP3A4 and CRM/OMS service (we model both systems as one business service).

- **Web Services Creation** This step involves the creation of Web services as adapters to existing systems, i.e., WSDL descriptions for these adapters including XML schema for messages, as well as definitions of interfaces, their operations, binding and endpoint. In our scenario, we use two adapters: (i) PIP3A4 adapter and (ii) CRM/OMS adapter. These adapters allow Mueller's CRM and OMS systems to connect to the middleware, and perform lifting and lowering functionality for XML schema and ontologies according to the grounding definitions of WSMO services.
- **Semantic Web Services and Goals Creation** In order to create Semantic Web Services and Goals, the ontologies must be created (or reused) together with non-functional, functional and interface description of services. In addition, a grounding must be defined from the semantic (WSMO) descriptions to the syntactic

(WSDL) descriptions. Semantic Web Services and Goals are described according to WSMO Service and WSMO Goal definitions respectively. We create a WSMO Goal as PIP3A4 service and WSMO Service as CRM/OMS service. Please note that WSMO Goal and WSMO Service have the same structural definition, but differ in what they represent. The difference is in the use of defined capability and interface—WSMO Goal describes a capability and an interface required by a service requester whereas WSMO service describes a capability and an interface provided by a service provider. In our scenario, this task is performed by domain experts (ontology engineers) using WSMT. In this section, we further elaborate on this step.

### 9.3.1.1 Creation of Ontologies and Grounding

One possible approach towards creation of ontologies would be to define and maintain one local domain ontology for Moon’s B2B integration. This approach would further allow handling message level interoperability through the domain ontology when lifting and lowering operations would be defined from underlying message schema to the domain ontology. Another option is the definition of independent ontologies by each partner and its systems. In our case, these are different ontologies for RosettaNet and ontologies for CRM/OMS systems. The message level interoperability is then reached through mappings between used ontologies which are defined during design-time and executed during runtime. Although both approaches have their advantages and limitations, we will use the latter approach in our scenario. The main reason is to demonstrate mediators’ aspects to integration of services which are available as independent and heterogeneous services.

We assume that all ontologies are not available up-front and they need to be created by an ontology engineer. The engineer takes as a basis the existing standards and systems, namely RosettaNet PIP3A4 and **CRM/OMS** schema, and creates *PIP3A4* and *CRM/OMS* ontologies, respectively. When creating ontologies, the engineer describes the information semantically, i.e., with richer expressivity as opposed to the expressivity of underlying XML schema. In addition, the engineer captures the logic of getting from XML schema level to semantics introduced by ontologies by lifting and lowering rules executed on non-semantic XML schema and ontologies, respectively. These rules are part of grounding definition between WSMO and WSDL descriptions and physically reside within adapters. In Listing 9.1, an example of the lifting rules and the resulting WSML instance is shown for an extract of a RosettaNet PIP3A4 message.

### 9.3.1.2 Creation of Functional and Non-Functional Descriptions

WSMO functional description (modeled as WSMO service capability) contains the formal specification of functionality that the service can provide, which is definition of conditions on service “inputs” and “outputs” which must hold before and

```

/* Lifting rules from XML message to WSML */

...
instance PurchaseOrderUID memberOf por#purchaseOrder
    por#globalPurchaseOrderTypeCode hasValue "<xsl:value-of select='dict:
        GlobalPurchaseOrderTypeCode' />"
    por#isDropShip hasValue
        IsDropShipPo<xsl:for-each select='po:ProductLineItem'>
            por#productLineItem hasValue ProductLineItem<xsl:value-of select='position()' />
        </xsl:for-each>
        <xsl:for-each select='core:requestedEvent'>
            por#requestedEvent hasValue RequestedEventPo
        </xsl:for-each>
        <xsl:for-each select='core:shipTo'>
            por#shipTo hasValue ShipToPo
        </xsl:for-each>
        <xsl:for-each select='core:totalAmount'>
            por#totalAmount hasValue TotalAmountPo
        </xsl:for-each>
    ...
    ...
/* message in WSML after transformation */

...
instance PurchaseOrderUID memberOf por#purchaseOrder
    por#globalPurchaseOrderTypeCode hasValue "Packaged product"
    por#isDropShip hasValue IsDropShipPo
    por#productLineItem hasValue ProductLineItem1
    por#productLineItem hasValue ProductLineItem2
    por#requestedEvent hasValue RequestedEventPo
    por#shipTo hasValue ShipToPo
    por#totalAmount hasValue TotalAmountPo
...

```

**Listing 9.1** Lifting from XML to WSML

after the service execution, respectively. Functional description for our back-end systems contains conditions that input purchase order data must be of specific type and contain various information such as customer ID, items to be ordered, etc. (this information is modeled as preconditions of the service). In addition, the service defines its output as purchase order confirmation as well as the fact that the order has been dispatched. Functional description of service is used for discovery purposes in order to find a service which satisfies the user's request. Non-functional properties contain descriptive information about a service, such as author, version or information about Service Level Agreements (SLA), Quality of Services (QoS), etc. In our example, we use the non-functional properties to describe user preference for service selection. In our case, the Blue company wants to buy and get shipped a product for the cheapest possible price which is encoded in the WSMO goal description.

### 9.3.1.3 Creation of Interfaces and Grounding

Interfaces describe service behavior, modeled in WSMO as (i) **choreography** describing how service functionality can be consumed by service requester and (ii) **orchestration** describing how the same functionality is aggregated out of other services (in our example, we only model choreography interfaces as we currently do

```

/* late-binding choreography for service discovery stage */
choreography dbl#DiscoveryLateBindingChoreography
stateSignature
in mu#purchaseQuoteReq withGrounding { ... }
out mu#PurchaseQuoteResp withGrounding { ... }

forall {?purchaseQuoteReq} with (
    ?purchaseRequest memberOf mu#PurchaseQuoteReq
) do
    add( # memberOf mu#PurchaseQuoteResp)
endForall
...
...

/* execution choreography for service execution stage */
choreography exc#ExecutionChoreography
stateSignature
in mu#SearchCustomerRequest withGrounding { ... }
out mu#SearchCustomerResponse withGrounding { ... }

transitionRules MoonChoreographyRules
forall {request} with (
    ?request memberOf mu#SearchCustomerRequest
) do
    add(_# memberOf mu#SearchCustomerResponse)
endForall
...
...

```

**Listing 9.2** CRM/OMS choreography

not use WSMO service orchestration). The interfaces in WSMO are described using Abstract State Machines (ASM) defining rules modeling interactions performed by the service including grounding definition for invocation of underlying WSDL operations. In our architecture and with respect to types of interactions between service requester/provider and the middleware, we distinguish two types of choreography definitions, namely **late-binding choreography** and **execution choreography**. Listing 9.2 shows a fragment depicting these two choreographies for the **CRM/OMS** service. The first choreography, marked as *DiscoveryLateBindingChoreography*, defines the rule how to get the quote for the desired product from purchase order request (here, the concept *PurchaseQuoteReq* must be mapped to corresponding information conveyed by the purchase order request sent by the Blue). This rule is processed during the service discovery and the quote information obtained is used to determine whether a concrete service satisfies the request (e.g., if the requested product is available, which is determined through quote response). The second choreography, marked as *ExecutionChoreography*, defines how to get information about customer from the CRM system. Decision of which choreography should be used at which stage of execution (i.e., service discovery or conversation) is determined by the choreography namespace (in the listing this namespace is identified using prefixes *dbl#* for discovery late-binding and *exc#* for execution, respectively). In general, choreographies are described from the service point of view. For example, the rule on line 21 says that in order to send *SearchCustomerResponse* message, the *SearchCustomerRequest* message must be available. By executing the action of the rule (*add(SearchCustomerResponse)*), the underlying operation with

```

axiom mapping001 definedBy
mediated(X, o2#searchString) memberOf o2#searchString :-
X memberOf o1#customerId.
```

**Listing 9.3** Mapping rules in WSML

corresponding message is invoked according to the grounding definition of the message which, in turn, results in receiving instance data from the Web service.

### 9.3.1.4 Creation of Ontology Mappings

Mappings between used ontologies must be defined and stored in the middleware repositories before execution. In Listing 9.3, the mapping of the *searchString* concept of the *CRM/OMS* ontology to concepts *cusomterId* of the *PIP3A4* ontology is shown. The construct *mediated(X, C)* represents the identifier of the newly created target instance, where *X* is the source instance that is transformed, and *C* is the target concept we map to. Such a format of mapping rules is generated from the ontology mapping process by the WSMT ontology mapping tool.

### 9.3.2 Execution of Services

In this section, we describe the execution phase run in the middleware for our example. This phase is depicted in Fig. 9.11 and implements the **AchieveGoal** execution semantics. We further divide this phase into the following stages to which we refer using numbers before each paragraph: (i) **Sending Request**, (ii) **Late-Binding–Discovery, Selection and Conversation Set-up**, (iii) **Execution–Conversation with Service Provider**, (iv) **Execution–Conversation with Service Requester**.

#### 9.3.2.1 (i) Sending Request

A RosettaNet PIP3A4 PO message is sent from the Blue Company to the entry point of the PIP3A4 adapter. In the PIP3A4 adapter, the PO XML message is lifted to WSML according to the PIP3A4 ontology and the rules for lifting using XSLT. Finally, a WSMO Goal is created from the PO message, including the definition of the desired capability and choreography as well as instance data (this goal is created as an abstract goal which contains separately defined data). The capability of the requester (Blue company) is used during the discovery process whereas the Goal choreography describes how the requester wishes to interact with the environment. After the Goal is created, it is sent together with the data to middleware by invoking the **AchieveGoal** execution entrypoint.

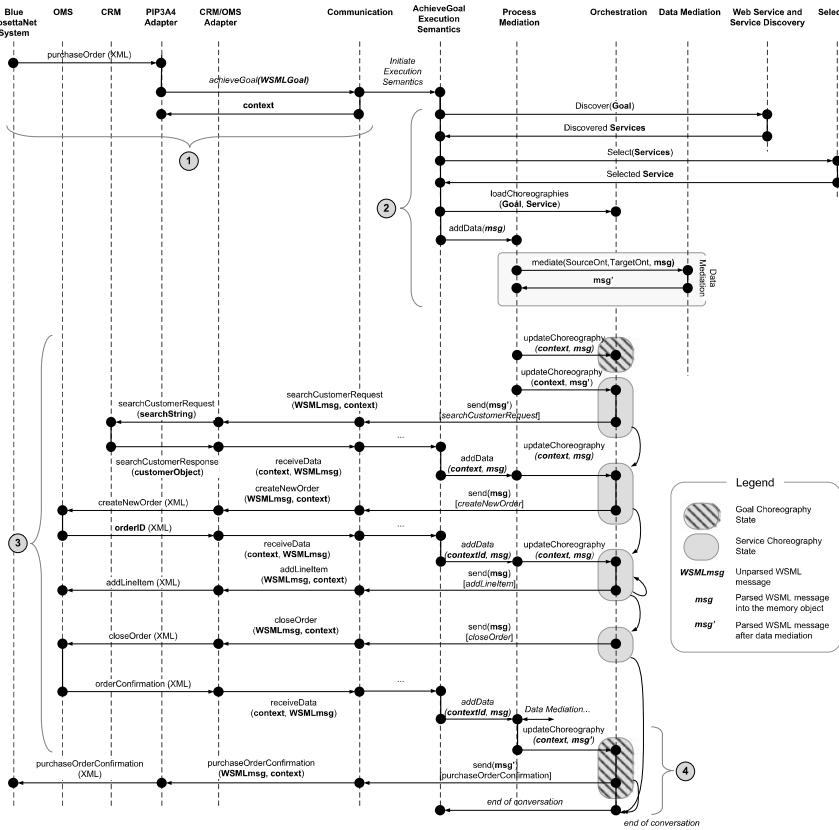


Fig. 9.11 Sequence diagram

### 9.3.2.2 (ii) Late-Binding–Discovery, Selection, and Conversation Set-up

The **AchieveGoal** execution entrypoint is implemented by the Communication service which facilitates inbound and outbound communication with the middleware. The AchieveGoal execution entrypoint starts the AchieveGoal execution semantics which performs the following steps:

- The parser parses the message into the internal memory representation.
- The discovery finds appropriate services by processing their abstract descriptions, that is, by matching capability descriptions of the goal with each service's capability descriptions from the repository.
- When a match is found at the abstract level, the match is further performed at the instance level. For this purpose, discovery fetches some additional data from service requester through processing of discovery late-binding interface of each candidate service (if interface exists). Listing 9.2 shows such an interface for Mueller's service. Through this interface, discovery obtains some additional data which is then used for fine-grained querying of the knowledge base containing all

instance data of the goal and the candidate service. During this process, interactions with data mediation as well as communication services are also performed (for brevity, these are not shown in Fig. 9.11). Upon result, the list of services all matching at the abstract and instance levels are returned.

- The next step is to perform the selection of the best service which satisfies service requester's preference. For this purpose, the preference on price is defined as part of the goal non-functional properties definition which is used for ranking of candidate services in the list of discovered services. In our implementation, the list is sorted by the reference property. The selection selects the first service from the list, in our case, it is the Mueller's service.
- The final step of this stage is to set-up the conversation between the service requester and the selected service. For this purpose, loading of both the requester's (Goal) and the provider's (selected service) choreography in the Orchestration service is performed (these choreographies are part of the goal and service descriptions). Both choreographies are then set to a state where they wait for incoming messages that could fire a transition rule. This completes the conversation set-up.
- In the next step, the Blue can send the data according to the Blue's goal choreography definitions. However, since our implementation is a bit simplified, all data from the Blue is already in the goal description. The execution semantics extracts all the data from the goal description and passes them to the process mediation to decide which data to add to requester's or provider's choreographies (this decision is based on the analysis of both choreographies and concepts used by these choreographies). Process Mediation service first updates the memory of the requester's choreography with the information that the PO request has been sent. The Process Mediation then evaluates how data should be added to the memory of the provider's choreography—data must be first mediated to the ontology used by the provider. For this purpose, the source ontology of the requester, target ontology of the provider and the instance data are passed to the Data Mediation service. Data mediation is performed by execution of mapping rules between both ontologies (these mapping rules are stored within middleware repositories and have been created during the Business Service Modeling phase).

### 9.3.2.3 (iii) Execution–Conversation with Service Provider

Once the requester's and provider's choreographies have been updated, the Orchestration service processes each to evaluate if any transition rules can be fired. The requester's choreography remains in the waiting state as no rule can be evaluated at this stage. For the provider's choreography, the Orchestration service finds the rule shown in Listing 9.2 (lines 8–12). Here, the Orchestration service matches the data in the memory with the antecedent of the rule and performs the action of the rule's consequent. The rule says that the message **SearchCustomerRequest** with data **searchString** should be sent to the service provider (this data has been previously added to the processing memory after the mediation—here, **searchString** corresponds to the **customerId** from the requester's ontology). The Orchestration service

then waits for the **SearchCustomerResponse** message to be sent as a response from the provider. Sending the message to the service provider is initiated by Orchestration service passing the message to the Communication service which, according to the grounding defined in the choreography, passes the message to the **searchCustomer** entrypoint of the CRM/OMS Adapter. In the adapter, lowering of the WSML message to XML is performed using the lowering rules for the CRM/OMS ontology and the CRM XML Schema. After that, the actual service of the CRM system behind the adapter is invoked, passing the parameter of the **searchString**. The CRM system returns back to the CRM/OMS Adapter a resulting **customerObject** captured in XML. The XML data is lifted to the CRM/OMS ontology, passed to middleware, evaluated by the Process Mediaton service and added to the provider's choreography memory. Once the memory of the provider's choreography is updated, the next rule is evaluated resulting in sending a **createNewOrder** message to the Mueller's service (OMS system). This process is analogous to the one described before. As a result, the **orderId** sent out from the OMS system is again added to the memory of the provider's choreography. After the order is created (opened) in the OMS system, the individual items to be ordered need to be added to that order. These items were previously sent in one message as part of order request from Blue's RosettaNet system (i.e., a collection of **ProductLineItem**) which must be now sent to the OMS system individually. As part of the data mediation in step (ii), the collection of items from the RosettaNet order request were split into individual items whose format is described by the provider's ontology. At that stage, the Process Mediation service also added these items into the provider's choreography. The next rule to be evaluated now is the rule of sending the **addLineItem** message with data of one **lineItem** from the processing memory. Since there is more then one line item in the memory, this rule will be evaluated several times until all line items from the ontology have been sent to the OMS system. When all line items have been sent, the next rule is to close the order in the OMS system. The **closeOrder** message is sent out from the middleware to the OMS system and since no additional rules from the provider's choreography can be evaluated, the choreography comes to the end of conversation state.

#### 9.3.2.4 (iv) Execution–Conversation with Service Requester

When the order in OMS system is closed, the OMS system replies with **orderConfirmation**. After lifting and parsing of the message, the Process Mediation service first calls for the mediation of the data to the requester's ontology and then adds the data to the memory of the requester's choreography. The next rule of the requester's choreography can be then evaluated which says that the **purchaseOrderConfirmation** message needs to be sent to the RosettaNet system. After the message is sent, no additional rules can be evaluated from the requester's choreography, thus the choreography comes to the end of conversation state. Since both requester's and provider's choreography are in the state of end of conversation, the Orchestration service closes the conversation.

## 9.4 Possible Extensions

The reference implementation of the SESA middleware is constantly evolving in order to incorporate additional functionality. Additionally, complete synchronous behavior of the presented Execution Semantics has proven to be a limiting factor in situations when a service consumer is continuously interested in services capable to fulfilling its needs. The solution can be found in a goal subscription mechanism based on asynchronous communication model.

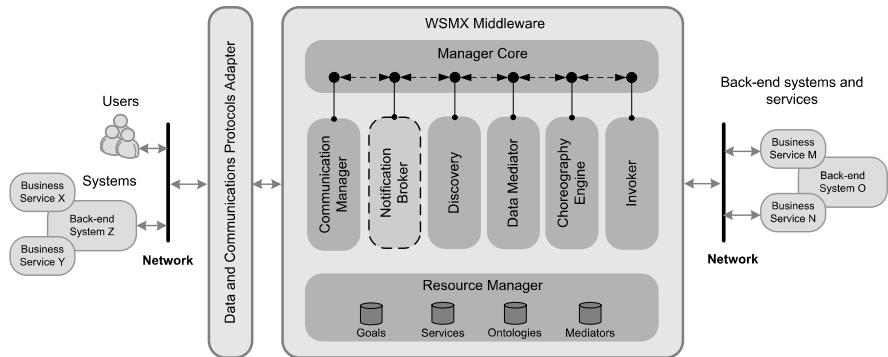
### 9.4.1 Goal Subscription

Large scale applications are demanding for more and more support of loosely coupled distributed interaction mechanisms. The Publish/Subscribe paradigm adapts well to the need, thus gaining momentum in the Web service community (e.g., WS-Events, WS-Notification). The main goal of such a mechanism is decoupling in time, space and synchronization publishers and subscribers. One current limitation of existing standards and supporting middleware is the need for the subscriber to know the topic offered by the publisher and to be able to process natively the published messages. The architecture of the semantically enabled middleware for Publish/Subscribe applications is based on the Semantically Enabled Service Oriented Architectures (SESA), enhanced with message brokering capability following the paradigm.

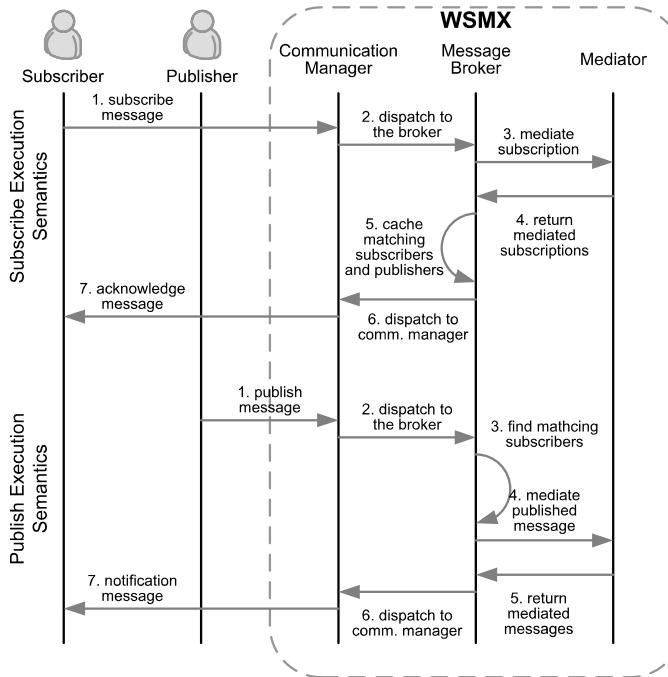
The existing WSMX behavioral models are based on the synchronous messaging paradigm. In order to support for asynchronous message processing, a new component called Notification Broker is introduced into the WSMX stack (as depicted in Fig. 9.12). The functionality of the component targets overcoming some shortcuts of the WS-Notification specification [10]. In particular, WS-Notification requires knowledge of the hierarchy topic prior to binding to the particular one. Furthermore, WS-Notification defines three types of filters (Topic filter, Message filter, Producer state filter) which base their functionality on XPath [4] expression processing over rather constrained data space.

The broker's operation relies upon the processing of data represented in an ontological format. In this way, both subscribers and producers of the messages can employ powerful knowledge description formalism (i.e., Web Service Modeling Language [5]) in order to precisely describe their needs. Topic hierarchies of arbitrary complexity can be described in terms of ontologies. Furthermore, filtering rules can be represented in the form of axioms which could govern specific constraints over the WSMX knowledge base. Synergy of the data mediator and the notification broker, operating on the semantic level, adds additional level of indirection between individual data formats of parties interested in communication over the messaging infrastructure.

The introduction of the Notification Broker enables also WSMX itself to be a subscriber and to trigger a proper execution semantics according to incoming messages, making it not only a Semantically enabled SoA, but also a Semantically enabled EDA (Event Driven Architecture).



**Fig. 9.12** The extension of the WSMX with message broker component



**Fig. 9.13** The execution semantics of the Publish (*lower part*) and Subscribe (*top part*) entry points

Figure 9.13 presents the execution semantics of the Publish and Subscribe entry points. In the case of the **Subscribe** execution semantics, when a subscribe message is received by the Communication Manager (i), the message is then forwarded to the Message Broker (ii); at this point the Message Broker registers the subscription and, to support the caching mechanism, forwards the subscription message to the

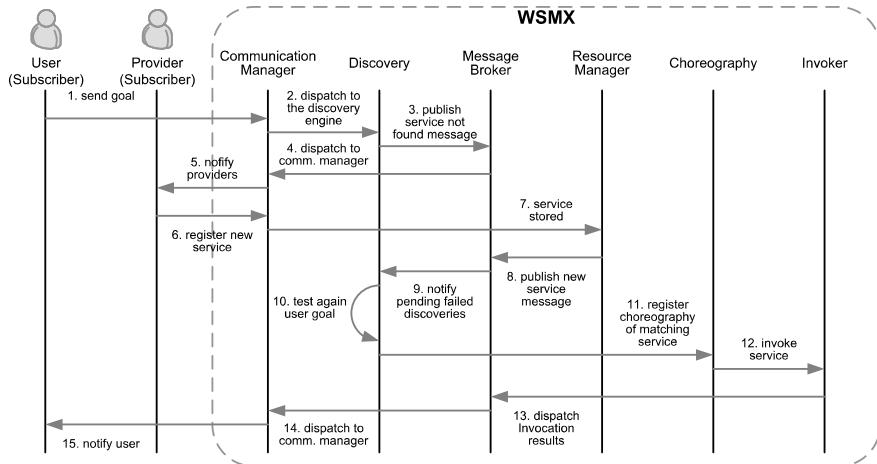


Fig. 9.14 The adapted achieveGoal execution semantics

Data Mediator (iii); the Data Mediator mediates the subscription to transform the subscribed topic concept and the filtering query contained in the subscription message according to the ontologies registered in the middleware by the Publishers. The mediated subscriptions are forwarded to the Message Broker (iv) that caches for optimization the mediated filtering queries and topic concepts. Finally, the control is returned to the Communication Manager (vi) that acknowledges the Subscriber (vii).

With regards to the **Publish** execution semantics, when a publish message is received (i), the message is then forwarded to the Message Broker (ii); at this point, the Message Broker uses the cache to find out possible matching subscriptions and applies registered mediated queries to complete the matching of the message against subscriptions (iii); the message is sent to the Data Mediator (iv) that mediates the published message to the ontologies of the matching subscriptions. The mediated messages are returned to the Message Broker (v) that returns the control to the Communication Manager (vi) that notifies the matched Subscribers (vii).

The **achieveGoal** entry point allows finding a service that matches user's goals and actually invoking it. WSMX may play both the role of a subscriber and a publisher, by introducing some small changes to the achieveGoal execution semantics (see Fig. 9.14). When a user invokes the achieveGoal entry point by sending a goal (i), the Discovery engine picks up the goal and tries to match it against registered services (ii). At this point, in the original execution semantics, if a matching service is not found, the user is informed that no matching service was found. In the modified version discussed here, a "service not found" message is notified to subscribed service providers, containing also the specification of the desired service (iii–v). When a Provider creates a new service and registers it to the middleware (vi), the service is stored by the Resource Manager (vii) that issues a "new service" message to the Message Broker (viii). At this point, the Message Broker notifies the pending Discovery processes (if, according to the user policy, they are not expired)

and restores them (ix); at this point, the original goal is tested against new registered services (x) and if it matches any of them, one is selected and invoked as in the original execution semantics (xi–xii). Finally, the results of the invocation are dispatched to the Message Broker (xiii) that triggers the Communication Manager (xiv) to send notification of the requested service execution to the User (xv).

## 9.5 Summary

This chapter presented basic notions comprising Semantically enabled Service Oriented Architecture (SESA). At the beginning, we introduced basic motivation for having SESA, by laying down its background and governing principles. Furthermore, a notion of Execution Environment for Semantic Web Services was introduced together with accompanying challenges.

The chapter continued with a discussion related to SESA technical solution where SESA was represented as a multi-layered system composed of stakeholders' layer, broker layer, base layer, and vertical services. Special attention was devoted to the description of specific broker and base services which comprise SESA middleware. We introduced the notion of Execution Semantics together with main three implementations. This discussion was followed by an example which tended to give a solution to the B2B integration problem. The chapter concluded with a discussion of possible SESA extensions, namely goal subscription.

## 9.6 Exercises

**Exercise 1** Given the following scenario description, try to formulate a set of WSMO ontologies and service descriptions.

With an invocation of one of the Web services, you can order a package shipment by specifying sender's address, receiver's address, package information, and a collection interval during which the shipper will come to your premises to collect the package. A shipper either responds with the estimated pickup (respecting the given time constraints) or with a fault message indicating that a pick up is not possible in the requested time interval. We have four distinct shippers with the following constraints:

### Racer

- Rates (flat fee/each lb): Europe (41/6.75), Asia (47.5/7.15), North America (26.25,4.15), Rates for South America as for North America, Rates for Oceania as for Asia; Furthermore, for each collection order, 12.50 is added!
- The list of countries Racer ships to is included in a WSDL file.
- Only packages weighing 70 lbs or less are shipped.
- Constraints on Collection:

- There should be at least an interval of 120 minutes for collection.
- Latest Collection time is 8 pm.
- Delivery Time
  - Ships in 2/3 (domestic/international) business days if collected by 6 pm.

### **Runner**

- Rates (flat fee/each lb): Europe (50/5.75), Asia (60/8.5), North America (15/0.5), South America (65.75/12), Africa (96.75/13.5), Oceania has the same rates as Asia.
- The exact list of countries is included in a WSDL file.
- When ordering a shipment using the Web services, the shipment of one package can be ordered per invocation.
- If the package weight exceeds 70 lbs, weight, length and height are required (the order has to be done via phone or fax).
- Constraints on Collection:
  - Collection can be ordered maximum 5 working days in advance.
  - Minimum Advance notice for collection is 1 hour.
  - Collection is possible between 1 am–12 pm (keep in mind that 12 pm is noon!).
  - There must be at least an interval of 30 minutes for collection.
- Delivery Time:
  - Ships in 2 business days if collected by 10 am.
  - Ships in 3 business days otherwise.

### **Walker**

- Rates (flat fee/each lb): Europe (41/5.5), Asia (65/10), North America (34.5/3), South America (59/12.3), Africa (85.03/13), Rates for Oceania as for Asia.
- Only packages weighing 50 lbs or less are shipped.
- The exact list of countries is included in a WSDL file.
- Constraints on Collection:
  - Shipment can be ordered maximum 2 business days in advance (the end of the pickup interval must be at most two business days in advance at the time of ordering).
  - Pickup time must be between 6 am and 11 pm.
  - There must be at least an interval of 30 minutes for collection.
- Delivery Time
  - Ships in 2 business days if collected by 5 pm.

### **Weasel**

- Rates (flat fee/each lb): United States (10/1.5).
- Delivery in the United States only.
- Constraints on Collection:
  - The pick up interval must be at least 5 hours.
  - The maximum pick up interval is 4 days.
  - Collection can be ordered until 8 pm.

- Delivery Time
  - 1 day if collected before 2 pm.

Next, try to express WSMO goals following given constraints:

### Goal A1

- Destination: Tunis
- Package dimensions: (l/w/h) 7/6/4 (inch)
- Package weight: 1 lb

### Goal A2

- Destination: Luxembourg
- Package dimensions: (l/w/h) 7/7/5 (inch)
- Package weight: 1.5 lb

### Goal B1

- Destination: Tunis
- Package dimensions: (l/w/h) 40/10/10 (inch)
- Package weight: 30 lb

### Goal B2

- Destination: Bristol
- Package dimensions: (l/w/h) 40/24/10 (inch)
- Package weight: 30 lb

Discuss which Web services are fulfilling each of the aforementioned goals.

## References

1. Booth, D., Hugo Haas, W., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Service Architecture. W3C, Working Notes (2003/2004). <http://www.w3.org/TR/ws-arch/>
2. Burbeck, S.: The tao of e-business services (2000). <http://www-128.ibm.com/developerworks/library/ws-tao/>
3. Clancey, W.J.: Heuristic classification. Artificial Intelligence **27**(3), 289–350 (1985). doi:[10.1109/MC.2004.172](https://doi.org/10.1109/MC.2004.172)
4. Clark, J., DeRose, S. (eds.): XML Path Language (xpath) Version 1.0. W3C Recommendation (1999). <http://www.w3.org/TR/xpath>
5. de Bruijn, J., Lausen, H., Polleres, A., Fensel, D.: The web service modeling language wsml: an overview. In: Sure, Y., Domingue, J. (eds.) ESWC. Lecture Notes in Computer Science, vol. 4011, pp. 590–604. Springer, Berlin (2006)
6. Dijkstra, E.W.: Notes on structured programming. Tech. rep. (1972)
7. Eshuis, H.: Semantics and verification of uml activity diagrams for workflow modelling. PhD thesis, University of Twente, Twente, Netherlands (2002)

8. Fensel, D., Keller, U., Lausen, H., Polleres, A., Toma, I.: Www or what is wrong with web service discovery. Tech. rep. (2005). Available at [http://www.w3.org/2005/04/FSWS/Submissions/50/WWW\\_or\\_What\\_is\\_Wrong\\_with\\_Web\\_service\\_Discovery.pdf](http://www.w3.org/2005/04/FSWS/Submissions/50/WWW_or_What_is_Wrong_with_Web_service_Discovery.pdf)
9. Gelernter, D., Carriero, N., Chang, S.: Parallel programming in Linda. In: Proceedings of the International Conference on Parallel Processing (1985)
10. Graham, S., Niblett, P., Chappell, D., Lewis, A., Nagaratnam, N., Parikh, J., Patil, S., Samdarshi, S., Tuecke, S., Vambenepe, W., Weihl, B.: Web services notification. Technical report, IBM, Akamai Technologies, Computer Associates International, SAP AG, Fujitsu Laboratories of Europe, Globus, Hewlett-Packard, Sonic Software, TIBCO Software (2004)
11. Harel, D., Rumpe, B.: Meaningful modeling: What's the semantics of "semantics"? Computer **37**(10), 64–72 (2004). doi:[10.1109/MC.2004.172](https://doi.org/10.1109/MC.2004.172)
12. Haselwanter, T.: WSMX core—A JMX microkernel. Bachelor Thesis (2005)
13. Keen, M., Acharya, A., Bishop, S., Hopkins, A., Nott, S.M.C., Robinson, R., Adams, J., Verschueren, P.: Patterns: Implementing an SOA using an enterprise service bus. IBM Red Book (2004)
14. Keller, U., Lara, R., Polleres, A., Toma, I., Kiffer, M., Fensel, D.: Web service modeling ontology—discovery. Working draft, Digital Enterprise Research Institute (DERI) (2004). Available from <http://www.wsmo.org/2004/d5.1/v0.1>
15. Mocan, A., Cimpian, E.: An ontology-based data mediation framework for semantic environments. International Journal on Semantic Web and Information Systems **3**(2), 66–95 (2007)
16. Papazoglou, M.P.: Extending the service-oriented architecture. Business Integration Journal (2002)
17. Papazoglou, M.P.: Service-oriented computing: concepts, characteristics and directions. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering. WISE 2003 (2003)
18. Papazoglou, M.P., Ribbers, P.: e-Business: Organizational and Technical Foundations. Addison Wesley, Reading (2006)
19. Papazoglou, M.P., van den Heuvel, W.-J.: Service oriented architectures: Approaches, technologies and research issues. VLDB Journal (2007)
20. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing research roadmap (2006)
21. Singh, M.P., Huhns, M.N.: Service-oriented Computing: Semantics, Processes, Agents. Wiley, New York (2005)
22. Toma, I., Roman, D., Fensel, D., Sapkota, B., Gomez, J.M.: A multi-criteria service ranking approach based on non-functional properties rules evaluation. In: ICSOC '07: Proceedings of the 5th International Conference on Service-Oriented Computing, pp. 435–441. Springer, Berlin (2007)
23. Wsmo4j—an api and a reference implementation for building semantic web services applications compliant with wsmo. Available at <http://wsmo4j.sourceforge.net>

**Part III**

**Complementary Approaches for Web  
Service Modeling Ontology**



# Chapter 10

## Triple Space Computing for Semantic Web Services

**Abstract** Semantic Web Services promise seamless interoperability of data and applications on a semantic level, thus turning the Web from a world-wide information repository for human consumption only to a device of distributed computation. To this end, appropriate semantic descriptions of Web services and intelligent mechanisms working on this need a solid foundation in terms of the underlying semantically-enabled communication technologies. Recent advances in middleware technologies propose semantics-aware tuplespaces as an instrument for coping with the arising requirements of scalability, heterogeneity and dynamism. An additional argument is that Semantic Web Services have inherited the communication model of XML-based Web services, which is based on message exchange, thus being a priori incompatible with the REST architectural model of the Web. Analogously to the conventional Web, truly Web-compliant service communication should be based on persistent publication instead of message passing. In this chapter, we present “Triple Space computing”, a Triple Space-based coordination middleware for Semantic Web Services. We take a closer look at the respective data, infrastructure and coordination models that enable the management of formal knowledge and that support the interaction patterns characteristic for the Semantic Web and Semantic Web Services. Applying Triple Space computing to Semantic Execution Environments and Service-Oriented Infrastructures is the second main topic of this chapter.

### 10.1 Motivation

Most of the available IT applications—and in particular Web services—depend largely on synchronous communication links between information producers and consumers. Instead of following the “persistently publish and read” paradigm of the Web, traditional XML-based Web services establish tightly coupled communication cycles and send directed messages back and forth. Indeed, aside from the name, these Web services do not have much in common with the Web. This is similar to the situation in the pre-Web age where one had to send an e-mail message to a scientific colleague, asking for a specific paper or piece of information. In the current Web that offers a communication infrastructure for humans, this pattern of communication has widely been replaced by persistent publication and asynchronous retrieval. We no longer request the biggest share of information in lengthy, synchronous communication cycles with the originators, but fetch it from persistent

sources, for instance, the scientist's personal Web page. We argue that this has significantly contributed to the success and scalability of the Web since it freed the sender from the need to handle individual requests, and it made resources identifiable and thus referable. Imagine one having to handle individual requests from friends and colleagues who visit a Web site. The human certainly becomes the main bottleneck for the dissemination of new ideas. The shift from information dissemination based on message exchanges to publication has not only made the Web scale tremendously, but it has also accelerated the dissemination process. When comparing Web services with this important principle of the Web, it becomes obvious that Web services are not following the core idea of the Web. It is true that Web services use the Internet as the transport medium, however, that is basically all they have in common with the Web.<sup>1</sup>

The negative side-effect of distributed applications that communicate via message exchange is that they require a strong coupling in terms of reference and time. This means that the sender and receiver of data must (i) maintain a connection at the very same time, (ii) agree on the data format, and (iii) know each other and share a common representation. Therefore, the communication has to be directed to a particular service endpoint and is synchronous as long as neither party implements asynchronous communication (or the two jointly agree on the specific way this mechanism is implemented).

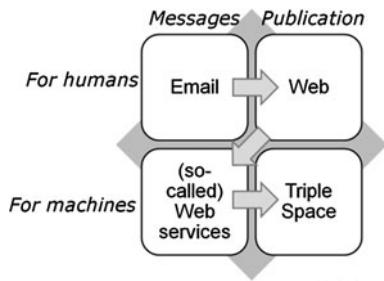
The above statement is in alignment with the worries expressed by the REST community [13]. Their two major criticisms on Web services were about the improper usage of URIs and the violation of the stateless architecture of the Web. It is one of the basic design principles of the Web and REST architecture to not provide stateful protocols and resources [14, 27]. In practice this means that when sending and receiving, for example, SOAP messages, the content of the information is hidden in the body and not addressed as an explicit Web resource with its own global and persistent URI. Therefore, most Web functionality dealing with caching or security checks is disabled, since using it would require parsing and understanding all possible XML schemas that can be used to write a SOAP message. Thus, when a stateful conversation is required, this should be explicitly modeled by different URIs. Moreover, 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 and the scalability of the resulting service-based system would be significantly increased.

These arguments motivated the development of a new communication paradigm at the intersection of tuplespace computing and semantics. Tuplespaces allow distributed sharing of information without requiring synchronous connections. The Semantic Web extends the Web with machine processable semantic data, allowing data exchange in heterogeneous application fields. Combining the two provides a

---

<sup>1</sup>In this line of reasoning, we obviously refer to XML-based Web services and similar communication protocols only. The recently emerging REST services pay tribute to many of the expressed criticisms. Still, the principles and added value of Triple Space-based communication and coordination, as it will be presented later in this chapter, also apply to REST services.

**Fig. 10.1** Triple Space conceptual architecture



communication paradigm with persistent and asynchronous dissemination via the sharing of machine understandable data that we term “Triple Space computing”: RDF triples provide a natural link from the Semantic Web and tuplespaces to Triple Spaces [11]. In the next section, we explore how Web services can be fully leveraged by the aforementioned “persistently publish and read” Web principle and propose a means to overcome the major flaws from which traditional IT service systems suffer. Notice, however, that the (Semantic) Web does not become obsolete due to the Triple Space paradigm, nor will the message-based systems exchange completely disappear as a communication paradigm for accessing computer functionality remotely. Triple Spaces will supplement Web services, but will not replace current technological approaches. Just as WWW technology has advanced message-oriented communication means (e.g., phone or e-mail) for humans (Fig. 10.1), Triple Space computing provides a complementary approach for machine-to-machine interaction and leverages the true added value of service-oriented infrastructures.

## 10.2 Technical Solution

In order to understand the concepts of Triple Space computing, we start with an introduction to tuplespace computing as a paradigm for exchanging data between processes. Adding the concept of semantics of information naturally leads to semantic tuplespaces and thus Triple Spaces.

### 10.2.1 Tuplespace Computing

Tuplespaces were first introduced in the context of parallel programming languages such as Linda to implement coordination between processes and were developed as a means to inject the capability of concurrent programming into sequential programming languages [16]. The principle is based on a simple communication interface and a shared data space (*tuplespace*) which contains the data in the form of (*tuples*). A tuple is an ordered list of typed fields, each of which either contains a value or is undefined. The coordination primitives are a small, yet elegant set of operations that permit agents to emit a tuple into the space (operation *out*) or associatively retrieve

tuples either removing them (*in*) or not (*rd*). Retrieval is governed by matching rules. Tuples are matched against a template which is a tuple that contains both literals and typed variables. A match occurs if the template and the tuple have the same length and field types and if the value of literal fields are identical. The tuple (“N70241”, EUR, 22.14) will match the template (“N70241”, ?currency, ?amount) and bind the variables currency and amount to the values EUR and 22.14, respectively. The operations are blocking and thus provide an inherited coordination mechanism.<sup>2</sup> In summary, a tuplespace

- supports **asynchrony** and **concurrency**, i.e., the producers and consumers of a tuple do not need to know one another’s address nor exist concurrently (this is also referred to as reference and time decoupling, respectively)
- permits **associative addressing**, i.e., data is retrieved on the basis of the type of data requested, rather than on which specific data is referenced, and
- **separates** the coordination **model** from characteristics of the host **implementation** environment.

This decoupling has obvious design advantages for defining reusable, distributed, heterogeneous, and agile applications. A communication paradigm based on tuplespace computing also resembles the core Web paradigm: information is persistently written on a globally accessible space where it is shared and can easily be accessed without a cascade of message exchange operations. It is thus quite obvious that the Web and tuplespaces share many underlying design principles. However, current tuplespace models lack support for namespaces, semantics, and structure in describing the information content of the tuples. In fact, tuplespaces provide a flat and simple data model that does not support any nesting or linking of data. Hence, tuples with the same number of fields and field order but different semantics cannot be distinguished. Triple Space computing thus had to extend tuplespaces to enhance the common field structure and field comparison matching with nested and inter-linked tuples (triples) and semantics-aware matching rules. Parts of these extensions are directly addressed by existing Web technology for resource identification in the Web:

1. URIs provide a reference mechanism that allows information to be distinguished on a world-wide scale.
2. Namespaces provide a separation mechanism that allows different applications to use the same vocabulary without blurring their communication.

In order to comply to the requirements of large scale Semantic Web applications, there is a further need to extend the tuplespace models in several directions. We distinguish between three categories of extensions to the original approach: (i) new types of tuples, (ii) new types of tuplespaces, and (iii) new types of operations.

The first extension looks at the necessary changes to tuples and tuple fields. The Semantic Web is mainly about the meaning of Web resources and their properties.

---

<sup>2</sup>The original Linda language included a further operation, *eval*, that is not exploited in Triple Space computing and thus not further discussed in this chapter.

The first problem with traditional tuplespace implementations is that tuples with the same number of fields and the same field typing cannot be distinguished, which does not comply with the Semantic Web principle which foresees that all information is encoded in triples of URIs. Moreover, as stated previously, the different semantic tuples in a space are not independent as, for example, in the Linda language, but highly correlated and dependent upon each other (semantic statements are connected in the so-called RDF graphs). Hence, the Semantic Web requires a reconsideration of the tuple model and the way tuples are matched in accordance with these concepts. The matching algorithms have to consider the meaning of the semantic tuples in order to provide the required degree of knowledge interpretation, and allow the retrieval of tuples taking into account the relationship to other tuples in the same space. This relates especially to the issues of resource identification previously discussed.

The second category of extensions aims at overcoming the technical problems of large scale distributed systems (e.g., heterogeneity, scalability, fault-tolerance, multiuser access) by proposing distribution strategies for multiple spaces or hierachic spaces and augmented naming approaches. Such approaches were already considered in various non-semantic tuplespace implementations applied to solve communication and coordination issues in areas such as open distributed systems, workflow execution, or XML middleware [2, 5, 6, 23, 26].

The last type of extensions reconsiders the tuplespace operations for publishing and retrieving tuples. Rather than handling individual tuples, in the context of the Semantic Web, we have to be able to work at the level of resources, semantic statements and graphs.

### 10.2.2 Triple Space Computing

Tuplespaces, as we discussed above, can help to overcome heterogeneity in communication and coordination, however, they do not provide an answer to data heterogeneity. That remains a task for the Semantic Web, as it provides standards to represent machine-processable semantics of data. In RDF [17], semantic data is described by interlinked triples of the form <subject, predicate, object>. The subject is the resource about which the statement is made. The predicate represents a specific property and the object is either a resource or a literal defining the property value of the statement. Resources are identified by URIs, while literals are, for instance, strings or numbers. RDF Schema [1] additionally defines classes, properties, domain and range inferences, and hierarchies of classes and properties on top of RDF. Therefore, a richer data model other than interlinked triples could also be used to model and retrieve information. This is even further extended by OWL [20], a data modeling language based on description logic. The Semantic Web aims at making problems in protocol and process heterogeneity transparent by its uniform and simple means for accessing and retrieving information. Space-based computing moreover allows complex message exchanges to be replaced by simple read and write operations in a globally accessible space. Therefore, Triple Spaces deliver a

global platform for application and service integration on the Internet just as tuplespace computing became a means for the local integration of parallel processes. In other words, Triple Spaces enable machines to use an equally powerful communication medium as the WWW provides for humans.

To conclude this section, we highlight again the advantages for providers and consumers of semantic data that arise from the introduction of Triple Space computing. The providers of data can publish it at any point in time (time autonomy), independent of its internal storage (location autonomy), independent of the knowledge about potential readers (reference autonomy), and independent of its internal data schema (schema autonomy):

- **Time autonomy** There is a only minimal time dependency between the data provider and reader, in the sense that a triple must first be written before it can be retrieved.
- **Location autonomy** The Triple Space as a storage location is independent of the storage space of the providers or readers of data. Complete independence is achieved by ensuring that triples are passed to and from the Triple Space by value and in the format required by the Triple Space.
- **Reference autonomy** Provider and reader of data might know about each other, but ex ante knowledge for purposes of communication through the Triple Space is not required. In the simplest case, the reading and writing of data is anonymous.
- **Data schema autonomy** Triple Space computing provides its own schema (based on triples according to RDF) and the data written and retrieved from a Triple Space will follow that data model. This makes the provider and reader independent of their internal data schemas.

There are two further positive side-effects. A Triple Space provides a trustworthy third-party infrastructure for data communication. Its involvement can enable secure data exchange and business processes negotiation and communication. Lastly, the Triple Space ensures persistent data storage. This guarantees that data is also available at a later point in time and that the data eventually can be read by all partners involved in a data exchange. This enables the sharing of information for collaborative activities.

### **10.2.3 Triple Space Conceptual Models**

In the following, we present the conceptual specification of a Triple Space system including the space and data models, and the extended interaction primitives.

#### **10.2.3.1 Data Models**

In Sect. 10.2.2, we have already presented some of the similarities between tuplespaces and Triple Spaces in terms of their conceptual models. The data model

of Triple Space alters the view on tuples towards established Semantic Web standards. Tuples receive the dimensions and semantics of RDF (<subject, predicate, object>) and can be interlinked by use of shared URIs to form RDF graphs. Following the RDF abstract syntax, each tuple field contains thus an URI (or, in the case of the object, also a literal). In this way, tuples sharing the same subject, predicate and object can still be addressed separately, which is consistent with the original tuplespace model, while the sharing of resources (URIs) brings the RDF graph concept to Triple Space computing. By associating graphs with globally unique identifiers, and thus by making graphs second-level resources in a space, sets of triples become addressable and retrievable. This principle of making graphs manageable resources has already proven useful in various other scenarios [3]. Graphs are then referred to as named graphs, and are used, for example, in [4] to attach trust and security information to RDF data. Analogously, the Triple Space offers the possibility to process entire semantic artifacts as one resource, e.g., Web service descriptions or business orders. For such objects, it is advisable to consider them as one addressable resource, rather than manipulating each triple individually, as this would normally be the case when triples are the minimal data unit of a Triple Space. The graph identifier is moreover used as a reference subject to add contextual data to the published RDF data. Contextual information such as access log information does not differ from one triple to another within a set that is written or accessed at the same time (e.g., the time of publication or the owner/publisher). This metadata can be used to refine retrieval operations and to compute provenance and trust via (a) the sources that delivered an answer or an indicator of how the answer was derived, and (b) the recency of a published data item by means of the timestamps. In the context of dataspaces—a novel view on distributed databases, where individual stores are no longer integrated but rather coordinated in loosely-coupled manners—[15] mentions metadata queries as an important element for the computation of the degree of uncertainty of an answer. This is highly relevant in a world of virtualized information providers, as they are enabled via Triple Spaces, because the origin of content can a priori no longer be determined. The assumed quality of data is no longer bound to the provider, but must be determined from a data item's own context.

### 10.2.3.2 Space Model

There are two different possibilities to structure Triple Spaces: either by means of a single space approach where the space abstracts from a single central server, as in the original Linda, or a multiple spaces structure that partitions the global interaction space. According to the partitioning strategy, it is further possible to differentiate between flat multiple spaces and hierarchical or nested spaces with the intermediate tree-like structure. Table 10.1 summarizes this discussion. The single space approach is not adequate for an infrastructure that aims at a global Triple Space platform for the Semantic Web and Web services, as it results in a bottleneck and single point of failure. This may become unacceptable in terms of performance and scalability. On the other hand, the nested and interlinked approach, in particular when

**Table 10.1** Possible space structures

	Pros	Cons
Single	Simplified implementation, simple access through single access point	Single point of failure, bad scalability
Flat	Distribution of data possible, better scalability	No structuring, no dependencies and relationships of spaces
Nested	High flexibility in structuring spaces and data	Difficulties in updating, high number of dependencies
Tree	Allows layering and limited nesting of spaces	Limited interlinking and dependencies

allowing overlapping spaces, necessitates complex data management mechanisms. Consistency checking and updates over multiple spaces imply a high processing overhead. The intermediate approaches of multiple flat spaces, or tree-like spaces, are more adequate for the goal of Triple Spaces. The flat management provides a simple means of organization, while the advantage of a tree-like space organization is the possibility to partially merge a number of spaces within a parent space. The former approach has, for example, been chosen by the TSC project [12], while the later space structuring approach was applied by the TripCom project and has shown to be effective for many of the targeted application scenarios [8, 19, 25].

In the continuation of this chapter, we rely on a tree-based space model in order to cope with the increasing complexity of open decentralized systems in terms of heterogeneity of applications and data, and the diversity and frequency of access. The space model thus defines the global Triple Space to consist of a forest of space trees in a meronymic structure. The overall space is composed of a set of disjoint trees, in which every Triple Space may contain multiple subspaces, while it can be contained in at most one parent. Triples are then associated with a single space, i.e., they are published to and maintained in only one space. Each space is identified by its own URL, independent of its tree affiliation. This identifier is important for accessing the space, as we will see when discussing the interaction models.

Although the Triple Space provides only one Triple Space infrastructure (one virtually singleton access point) to ensure scalability at the level of the applications that are built on top of the space, the division into smaller spaces is essential to guarantee scalability at the level of the platform as a whole. The number of servers that manage a space, the number of triples published to a space, and the number of users accessing a space influence the performance of the infrastructure in terms of responsiveness and communication overhead. Therefore, it is important to install means to control these figures at the level of individual spaces. The goal of installing spaces is to have dedicated means for information management, communication and coordination in the space layer. In other words, whenever there is a new interaction context or a new group of clients that intend to collaborate through the Triple Space platform, a new virtual space is installed that links together the interacting data consumers and providers. There might be a space for sport news providers and readers, and, independently, one for the processing of shoe orders. Any of these

**Table 10.2** Coordination primitives for publication and retrieval

1	write (URI space, Set(Triple) data):URI
2	read (URI space, Template t, [long timeout]):Set(Triple)
3	read (URI space, URI graph, [long timeout]):Set(Triple)

dedicated spaces can be hosted on a single machine, or shared amongst multiple co-authorizing machines. This exemplifies the importance of P2P-driven technology in Triple Space computing, as an arbitrary and dynamic number of machines should be able to share a space without central control (cf. Sect. 10.2.4 on the Triple Space architecture). This approach to space structuring guarantees at least local scalability by naturally grouping related data and clients in dedicated spaces, and by inherently limiting the scope of the individual interaction instances [18].

### 10.2.3.3 Interaction Model

The interaction model of the Triple Space provides operations for publishing and retrieving semantic data in the form of RDF statements and RDF graphs (cf. Table 10.2). Note that there are no removal operations defined. In the context of distributed installations, the removal of data, or worse of knowledge, that includes the deletion of inferred statements leads to complex data management issues and inevitably to decreased scalability. Omitting deletion seems to contradict the traditional CRUD functionalities (create, read, update, delete) of databases, however, makes Triple Space operations manageable under the eminent scalability requirements. Nonetheless, it is indeed possible to model removal by means of the write and read operations only. Instead of removing triples from a space, statements can be marked as being invalid, e.g., by writing invalidating statements.<sup>3</sup> The antipodal coordination primitives for publication and retrieval are thus seen to be sufficiently expressive.

The first two operations in Table 10.2 represent the Triple Space versions of the Linda ‘out’ and ‘rd’ primitives. The write operation asynchronously publishes a set of triples to a given space, whereas the read operation retrieves triples matching the given template from the indicated space. Templates are either triple or graph patterns (see Table 10.3), or full-fledged SPARQL CONSTRUCT queries [24]. Graph patterns are the fundamental element of any RDF query language, and are supported by most query engines. However, they have limited expressiveness compared to SPARQL, e.g., no optional patterns, no alternative matches via UNION statements and no filter expressions.

If the template is an SPARQL query, the indicated space can also be seen as a simulation of an SPARQL-endpoint with the space URL as the endpoint locator. Note that the retrieval operation does not give any guarantees with respect to the

<sup>3</sup>In the open world of the Web, this is an even stronger statement than ignorance. Even more, as deletion on the Web cannot be guaranteed due to the caching and archiving of Web resources.

**Table 10.3** Examples of semantic templates

Template	Description
?s a doap:Project; foaf:member ?o.	Matches all triples where the subject is of type doap:Project and where the same subject has triples indicating the members
?s ?p ?o.	Matches all triples where the object is of type foaf:Person
?o a foaf:Person.	
?s foaf:name ?a; foaf:mbox ?b.	Matches the triples that contain subjects for which the name and a mailbox (foaf:mbox) are indicated

**Table 10.4** Coordination primitives for publish–subscribe

4	advertise(URI space, Template t):URI
5	unadvertise(URI space, URI adv):void
6	subscribe(URI space, Template t, NotificationListener l):URI
7	unsubscribe(URI space, URI sub):void

completeness (i.e., the recall) of the set of triples returned. Analogously to traditional Linda spaces, querying only ensures that a triple that is published to a space will eventually be retrieved. Retrieval could be invoked with an infinite timeout, but even under these circumstances, the Triple Space cannot guarantee full recall. In the distributed scenarios that underlie the Triple Space platform, the state of a space can change under the processing of a request, as further data might get published, and locks would then be required to prevent this. Locking an open and distributed system is, however, neither feasible nor even desirable. For identical reasons, there is no guarantee of the consistency in terms of simultaneous accesses.

In order to offer Web-style access to resources, the template-based retrieval operation is complemented with a URI-based primitive (Table 10.2(3)) that allows the extraction of all the triples that are contained in the identified graph. Note that this operation only returns one graph of all available ones. This alternative read operation emphasizes the convergence of space and Web technology: the data (triples) is shared in a space, but retrieved as resources by their unique name, and not only by Linda-like associative template matching.

Recall that one of the fundamental objectives of Triple Spaces in terms of communication is the decoupling of producers and consumers of semantic artifacts. While the read and write operations via spaces provide a first step in this direction, they are still limited in terms of the decoupling of sequential operations of individual consumers or producers. The publish–subscribe paradigm adds such an additional dimension to the interaction patterns of the Triple Space. In systems based on publish–subscribe, subscribers register their interest in an event, or a pattern of events, and are subsequently asynchronously notified of events generated by publishers [10].

**Table 10.5** Management operations

8	createSpace(URI space):void
9	joinSpace(URI space):void
10	leaveSpace(URI space):void

The operation (6) in Table 10.4 allows subscribers to express their interest in a particular template. Upon the publication of matching triples, the subscribers are notified. Publish-subscribe operations allow for long-term coordination of services and applications. The operation (4) is put in place for publishers to advertise triple patterns that are likely to match the data they intend to publish. While advertisements are not necessary to realize the publish–subscribe functionality, they provide a convenient means to optimize the match making between publications and subscriptions. The methods ‘unadvertise’ and ‘unsubscribe’ are the inverse operations to (4) and (6).

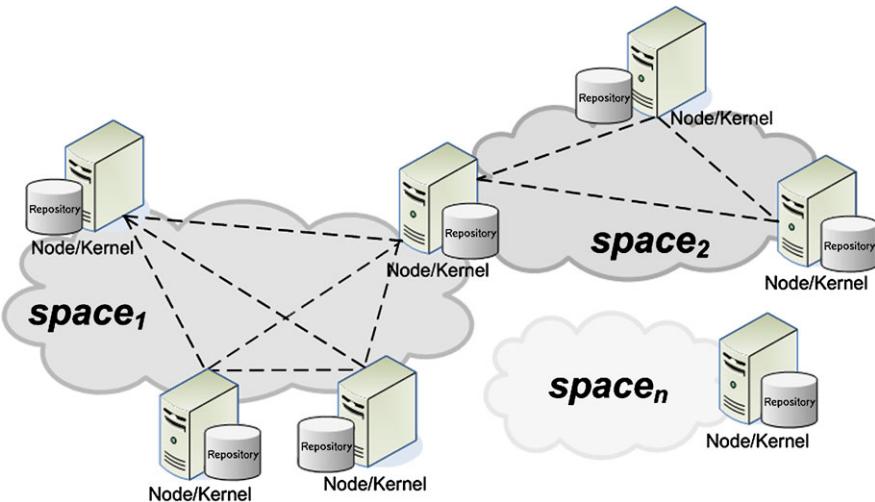
Besides the seven operations to publish and access semantic data in a space, the Triple Space offers three management operations (Table 10.5). Administrators have the possibility to create new spaces by means of the ‘createSpace’ operation.<sup>4</sup> As an effect of the creation of a space, the internal management structures are set up to allow other client and machines to discover, access and join a space. By joining, we refer to the process of becoming a co-administrator of a given space. While the creator of a space is per default the responsible host of a space, any other machine can explicitly request to share a space by managing and storing some of the published data (joinSpace as listed in Table 10.5(9)). The ‘leaveSpace’ operation inverts the effects, and a machine retires from sharing a space by moving all the locally maintained triples to another co-administrator and releasing all bindings to the space. If the machine that leaves happens to be the creator of the space, it needs to pass on the role of primary host to another co-administrator of the space.

#### 10.2.4 Triple Space Architecture

The Triple Space architecture fundamentally relies on distributed kernels. A so-called Triple Space kernel is a single physical implementation consisting of all necessary components that are needed to realize the functionality offered by the Triple Space interaction model. Depending on the level of functionality provided, kernels might have different sets of components. Any implementation, however, requires at least the components for the implementation of the interaction model, for the management of the distributed space infrastructure, and for the integration of various persistency frameworks, such as RDF repositories. Other possible components take

---

<sup>4</sup>The distinction between users and administrators is purely conceptual in the Triple Space as any entity accessing the Triple Space infrastructure is only categorized in terms of the operations invoked, and not in terms of roles or rights.



**Fig. 10.2** Triple Space conceptual architecture

care of data mediation, access control, or query optimization. The sum of these components, i.e., a kernel instance, runs on a machine that is deployed on the physical infrastructure underlying the Triple Space platform. Every kernel is addressable using a specific physical network address like a single physical machine or a single physical cluster of machines.

The goal of Triple Space is to provide a scalable and distributed infrastructure. Although scalability is a general property of a distributed system, its requirement depends largely on application-specific needs in terms of functional properties such as the type of operations, and non-functional properties such as availability, completeness, consistency or responsiveness [18]. Depending on the non-functional requirements, a space might be hosted on only one machine or distributed across any number of co-hosting peers (Fig. 10.2). While a space on a single machine realizes a Web server-like access point to RDF data, co-hosting kernels are organized as equally positioned peers in a distributed system. Distributed spaces inherit the advantages of the underlying network infrastructure: (i) there is no kernel that becomes a bottleneck and the single point of failure, (ii) having the measures in place to distribute a space allows for scaling up the system by adding more kernels with more processing power and memory (at the price of kernel coordination), and (iii) provides flexibility with respect to scalability-driven trade-offs, i.e., means to adjust the availability, completeness or latency guarantees by increasing or decreasing the degree of distribution. The Triple Space architecture hence allows for adding new hardware and software resources without hindering the existing configuration. The overall perceived performance can be kept at the desired level, while system dynamism, openness, and increasing data and workload can be handled at the kernel and space level, respectively.

### 10.2.5 Triple Space and Semantic Web Services

The motivation for Triple Space computing was from the very beginning the enhancement of the communication and storage layer of Web services in general, and Semantic Web Services in particular. Using Triple Space computing as paradigm for the core infrastructure services of Semantic Execution Environments (SEE) enables greater modularization, flexibility and decoupling, and as a result thereof easier distribution and deployment, and ultimately more scalability in terms of computing on the Web. In this section, we explain how Triple Space computing improves Semantic Web Service infrastructures at different levels. The improvements are illustrated by means of WSMX, the Web Service Execution Environment.

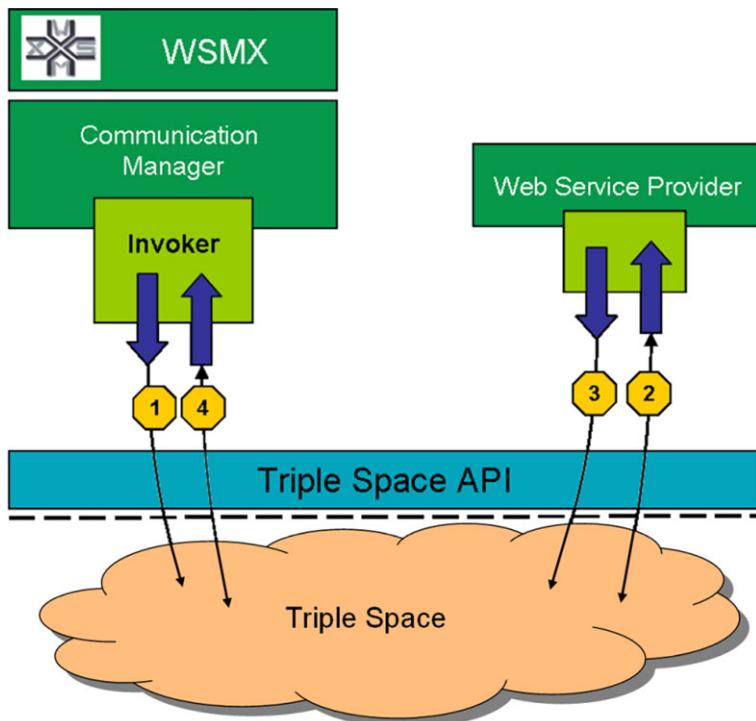
There are three main integration points for Triple Space and WSMX: external communication grounding, component coordination and resource management. The technological foundations of the integrations are subject to the next sections. In Sect. 10.3, we provide moreover a larger and integrated example of how the combination of WSMX and Triple Space adds a novel dimension to the Semantic Web Service communication paradigm.

#### 10.2.5.1 External Communication Grounding

Communication with service requestors or Semantic Web Services is an important aspect of WSMX that can be significantly improved by decoupling the different communication partners in time and reference. Currently, driven by the message-oriented communication paradigm underlying XML-based services, users communicate with WSMX and WSMX communicates with Web services via synchronous remote procedure calls. In order to counteract this habit and to enable the use of Triple Space, the external communication manager of WSMX has to be altered, i.e., the logics for sending and receiving messages has to be grounded to the interaction models of Triple Space. This grounding happens at several distinct access points:

1. The invoker component of WSMX needs to support Triple Space-enabled WSDL and SOAP bindings.
2. The entry point interfaces have to be linked to the space infrastructure in order to provide the glue between existing Web service standards and Triple Space computing.
3. The communication manager has to accept WSMO goals in RDF via the Triple Space in order to enable fully decoupled invocation.

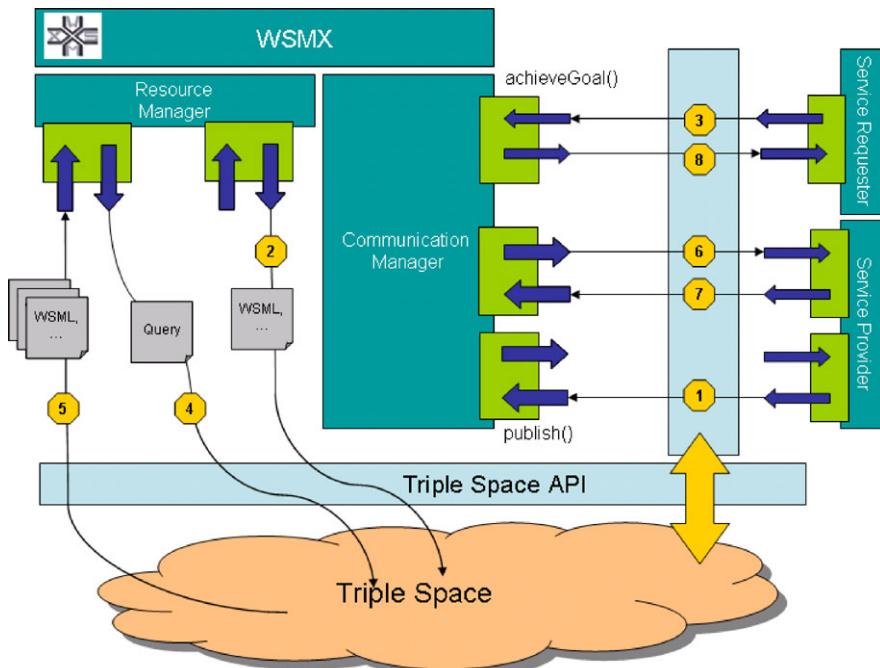
Without the addition of Triple Space as communication means, clients are tightly coupled to the execution on WSMX. Amongst other things, the use of Triple Space thus significantly decreases the waiting time from clients, and makes the use of WSMX a far more flexible approach to Semantic Web Services.



**Fig. 10.3** Triple Space-based Web service invocation

**Grounding of Web Service Invoker** The WSMX Web service invoker is part of the communication manager component and is responsible for mapping an invocation of a WSMX-external Web service provider to the transport protocol supported by the Web service. In combination with the WSMX grounding component, which implements a bi-directional mapping between the semantic data models of WSMX (WSMO-compliant objects) and the message and data format supported by external Web service endpoints (e.g., SOAP messages), it delivers the necessary facilities for the invocation of Web service endpoints.

In order to ground the Web service invoker in the Triple Space, an extended component implementation maps the operations of the invoker interface to the operations provided by the Triple Space interaction model. The mappings are offered by the grounding component in addition to the transformation facilities for WSMO-based semantic communication objects. Those are used within WSMX to transform (i) messages that match the Triple Space data model (e.g., SOAP/RDF representations) and (ii) message payloads that can be processed by the (Web) service provider. Figure 10.3 depicts the interaction scenario of asynchronous Triple Space-based Web service invocation by WSMX: (i) WSMO to RDF grounding and publication into the Triple Space, (ii) SOAP/RDF to SOAP/XML lifting, (iii) SOAP/XML to RDF transformation and publication, and (iv) SOAP/RDF lifting back to the WSMX-internal WSMO data models.



**Fig. 10.4** Client communication with WSMX via Triple Space

**Grounding of Client End-Points** The second type of grounding that is relevant in the context of WSMX communication concerns either service providers or service requesters that trigger read and write operations on the Triple Space, respectively. A typical example is the communication between clients that contact WSMX in order to register a service they offer or to find relevant services. In the first case, users invoke a ‘publish’ operation on the WSMX communication manager (Fig. 10.4—Step 1). For the latter case, to find a service, WSMX offers the ‘achieveGoal’ operation (Fig. 10.4—Step 3).

To realize the two scenarios by means of the integrated Triple Space platform, the interfaces of the communication manager are mapped to the interaction model of the Triple Space, and the WSMX end-points have to be registered with the Triple Space platform along with the appropriate graph patterns that have to be matched as initiators to the execution of the component, i.e., the end-point registers for particular data constructs in order to be notified about incoming requests. With such mappings in place, the resolving of a goal or the publication of a service can be mediated through the space infrastructure and thereby WSMX avoids the requirement for synchronicity and reduces the communication overhead. Additionally, this approach offers the benefit of effectively hiding the complexity of service interactions, e.g., combining discovery and execution of one or more services to achieve a certain goal, from the service requester.

Getting back to Fig. 10.4, Step 1 was already mentioned above. It represents the registration of a new service description. In Step 2, the WSMX component responsi-

ble for publications of services is triggered, and the content of the registration is read from the space and stored by the resource manager in the appropriate Triple Space (details on the realization of a triple space-enabled resource manager are given later in this section). A user request for a service via goal description (Step 3) triggers the resource manager and a query is sent to the Triple Space in order to retrieve the descriptions of services capable of handling the service requesters requirements (Step 4). In Step 5, matching service descriptions are returned to the resource manager, and WSMX starts to interact with service providers on behalf of the requester. By means of the grounding mechanisms for the Web service invoker, such interactions (Steps 6 and 7) can also be executed over a triple space. At last, in Step 8, WSMX returns the result of the ‘achieveGoal’ invocation to service requester, possibly again via Triple Space.

### 10.2.5.2 Component Management in WSMX

The different components of WSMX are centrally managed by a core component that is also in charge of enforcing the overall execution semantics [21]. In this way, the various components provide the business logic, while the management logic is implied by the WSMX core component. Every component is per design equipped with a wrapper to handle its communication tasks. In order to coordinate the components over Triple Space, the wrappers need be interfaced with Triple Space. The manager then communicates with the different components by publishing RDF data sets to the integration space, respectively by having components subscribed to particular triple patterns, i.e., components subscribe to pieces of data that are tailored to trigger activities in the respective components. Moreover, the component management over Triple Space is designed in ways that make sure that the WSMX manager can distinguish between the data flow related to business logics (execution of components based on the requirements of a concrete operational semantic) and the data flow related with the management logic (e.g., monitoring the components, load-balancing, instantiation of threads).

The main adaptation with respect to non-Triple Space-based realizations at the level of the communication wrappers is the addition of a Triple Space proxy (Fig. 10.5). The proxies bind the wrappers to a Triple Space kernel instance by incorporating the functionalities of a space client on behalf of the component. In this way, also adding a totally different communication paradigm between the manager and the different components, Triple Space does not change any of the execution semantics or WSMX internal component logic, nor even the interfaces of the components. Note, moreover, that it is still the manager that determines which components are used and have to be invoked. The addition of a Triple Space proxy solely enables alternative communication means and defines how to invoke a particular components by use of Triple Space primitives.

**Grounding of Component Manager** The implementation of the communication wrappers without Triple Space shows nicely what advantages the new bindings provide. With space-based coordination, every wrapper would have to host a service

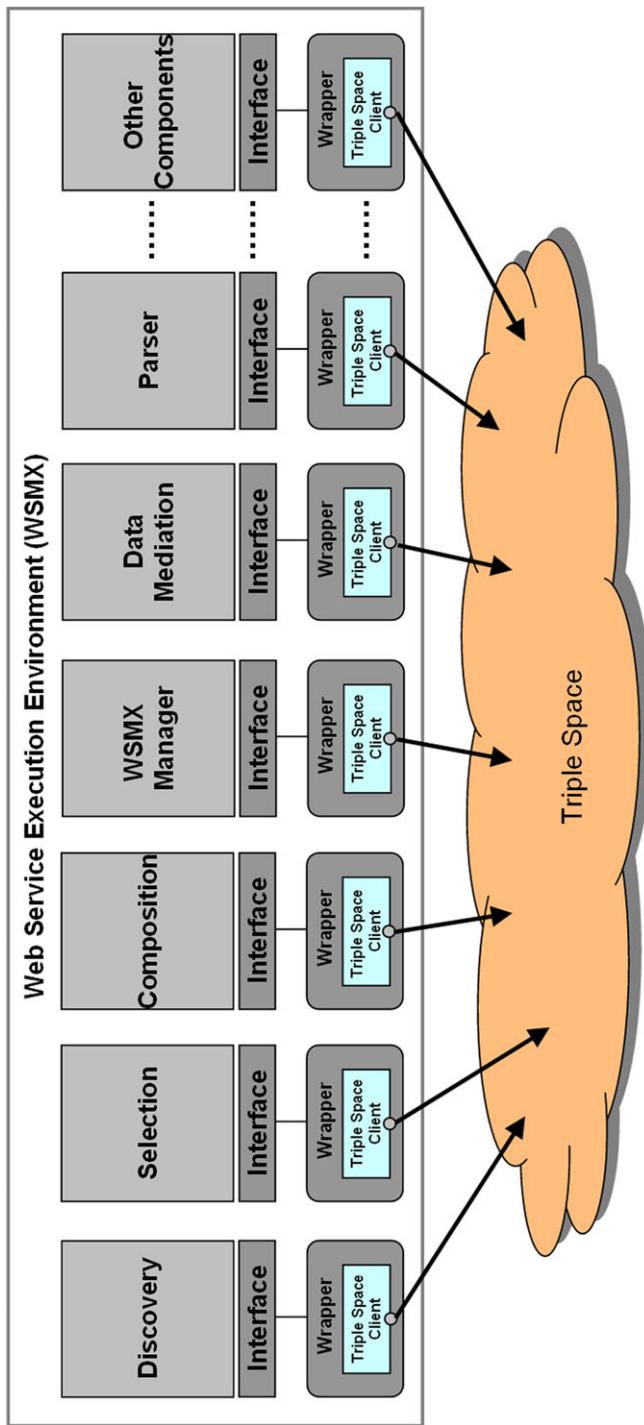
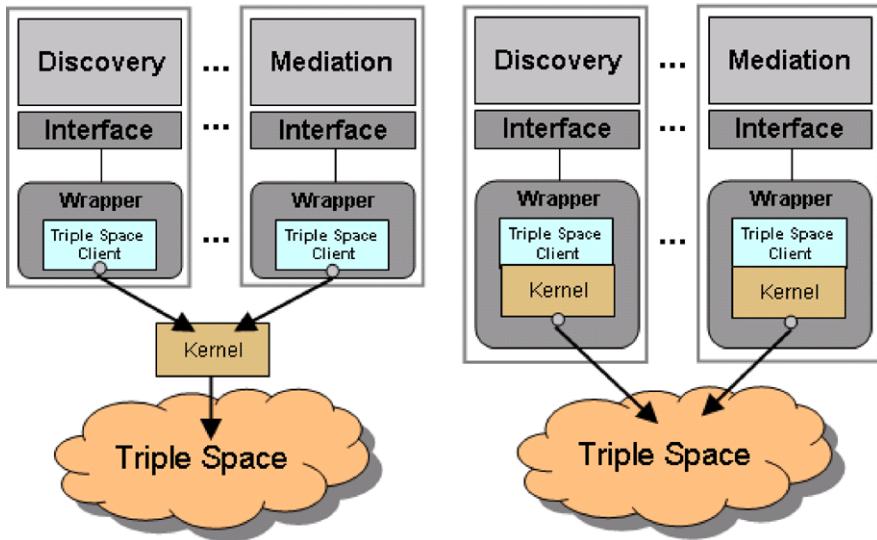


Fig. 10.5 WSMX component coordination over Triple Space



**Fig. 10.6** Alternative bindings: Triple Space “server” (left) or heavyweight proxies (right)

proxy for all other components. Adding new components or new functionality would in consequence require an update to all wrappers whose components might need the new value. Thanks to Triple Space, the proxies only implement the interface of the space infrastructure and the communication between components is guided by the publication of triples and the subscription to templates. The data that is exchanged between components are WSMO objects that are formalized by means of the Web Service Modeling Language (WSML, [7]) for which pre-defined serializations to RDF exist. Goal or service descriptions can thus be stored in the Triple Space as identifiable RDF graphs, i.e., isolatable sets of triples. The task of the WSMX manager becomes much simpler and is reduced to (i) the dispatching of requests to the appropriate components by writing destined triple sets, and (ii) the monitoring of the execution.

In terms of implementation, there are two distinguishable approaches, which both have their advantages and disadvantages. Firstly, each proxy embeds its own Triple Space kernel which transforms the proxies into rather heavyweight constructs. Secondly, the whole of WSMX offers one Triple Space kernel (server) to which the various proxies and the core connect remotely. This keeps the proxy implementation simpler as they only have to implement a simple image of a kernel implementation. A disadvantage of this approach is the fact that the kernel could become a bottleneck, as the chosen Triple Space adheres a single space model (Fig. 10.6); compare as well with the discussion of space models in Sect. 10.2.3.2.

**Table 10.6** Resource management operation on Triple Space

Resource Manager API	Triple Space API
void saveGoal(Goal goal)	URI write(URI space, Set<Triple> ts)
Set<Identifier> getGoalIdentifiers()	Set<Triple> read(URI space, Template t, long time)
Set<Identifier> getGoalIdentifiers(Set<Object> searchTerms, boolean conjunction)	Set<Triple> read(URI space, Template t, long time)
Set<Identifier> getGoalIdentifiers(Namespace namespace)	Set<Triple> read(URI space, Template t, long time)
boolean containsGoal(Identifier id)	Set<Triple> read(URI space, URI id, long time)
Goal loadGoal(Identifier id)	Set<Triple> read(URI space, URI id, long time)
Goal loadAllGoals()	Set<Triple> read(URI space, Template t, long time)

### 10.2.5.3 Resource Management in WSMX

Besides delivering the publish and read functionality exploited above for the various types of communication grounding and the component coordination, Triple Space particularly also emphasizes on the persistency and sharing of data. As it was stated in the section about the Triple Space architecture (Sect. 10.2.4), a persistency framework in form of RDF repositories is a core element of a Triple Space kernel. The persistency is exploited for the realization of a Triple Space-based resource manager for WSMX.

Per design, WSMX has six different repositories. Four of them to store WSMO ontologies, goals, mediators and services descriptions, and another two to store events and intermediate instance data. In order to rely on unified and simple resource management procedures, WSMX makes use of the Triple Space to store WSML data in the form of RDF graphs. To some degree, this is very similar to the approaches described above for the grounding of invocations and the coordination of components. This time, however, the aim is to improve the durability of information objects rather than the exchange of information. Again, a first requirement is the transformation of WSMO top level entities from WSML into their RDF representation. A second requirement is the mapping of given WSMX resource management operations to the Triple Space operations. To exemplify the correlation between resource management and data management with Triple Space, Table 10.6 outlines the mappings of the goal-driven operations of the Resource Manager API onto the Triple Space API, as it was introduced in Sect. 10.2.3.3.

With this outline we conclude the section on Triple Space-enhances Semantic Execution Environments and turn to a related application area. While WSMX offers the core infrastructure services and a number of broker services for the access and management of Semantic Web Services, the so-termed Global Service Delivery Platforms yield the main building blocks for Service-Oriented Architectures at the

Internet scale. In the next section, we take a look at how Triple Space computing is an important enabler of such much larger service economies.

### 10.2.6 Triple Space and Semantic SOA

The integration of WSMX and Triple Space computing has shown, by means of some simple examples, the added value that spaces bring to Semantic Web Services. While WSMX is an early adopter of Triple Space technology, the future in service computing is concerned with much more open and distributed environments. Semantic Service-Oriented Architectures are the natural evolution of Semantic Execution Environments by bringing the results of WSMX to the Web in order to support the realization of the Service Web. The Service Web is one of the three main building blocks of the Future Internet, and comprises the world of distributed computing on the Web via composable and dynamically adjustable Web services. In such a world where massive numbers of parties expose and consume services by realizing a coherent and domain independent platform, scalability, flexibility and automation become even more crucial. Triple Space technology, with the aforementioned traits in terms of communication and coordination of distributed services, is a core ingredient to the deployment of SOA in the large. In this section, we thus go a step beyond WSMX over Triple Spaces and depict how a global service delivery platform should look when blended with spaces and semantics.

The overall architecture of a global service-oriented infrastructure for the Service Web can be structured into four main parts:

**User front-end** A rich Web platform that provides users with a unified view covering the whole life-cycle of services, including design-time provisioning, run-time consumption and analysis.

**Service cloud** An infrastructural backbone on which various services are deployed and over which distributed services communicate by exchanging or sharing data. This is the part that is enhanced with Triple Space technology to move from traditional service bus solutions to the Service Cloud.

**Platform services** A set of services that provide the basic functionalities of the service delivery platform, such as Service Ranking and Selection, Service Discovery, Service Adaptation, Service Composition, Service Execution, and Reasoning Services that are needed to support the automation of the other activities.

**Web services** The forth and last part comprises the business services (atomic or composed) that are made available by end users.

Figure 10.7 depicts a high-level architecture of a Semantic SOA infrastructure that offers an instance of a global service delivery platform. This infrastructure is an outcome of a European research project called SOA4All whose aim is to transform the Web into a domain where billions of services are coordinated in seamless and transparent fashion, by integrating the service world of large enterprises, SMEs, and end-users. Moreover, SOA4All seeks to bring the Service Web to the average user by exploiting Web and Web2.0 technology, contextual information and not at least semantics.

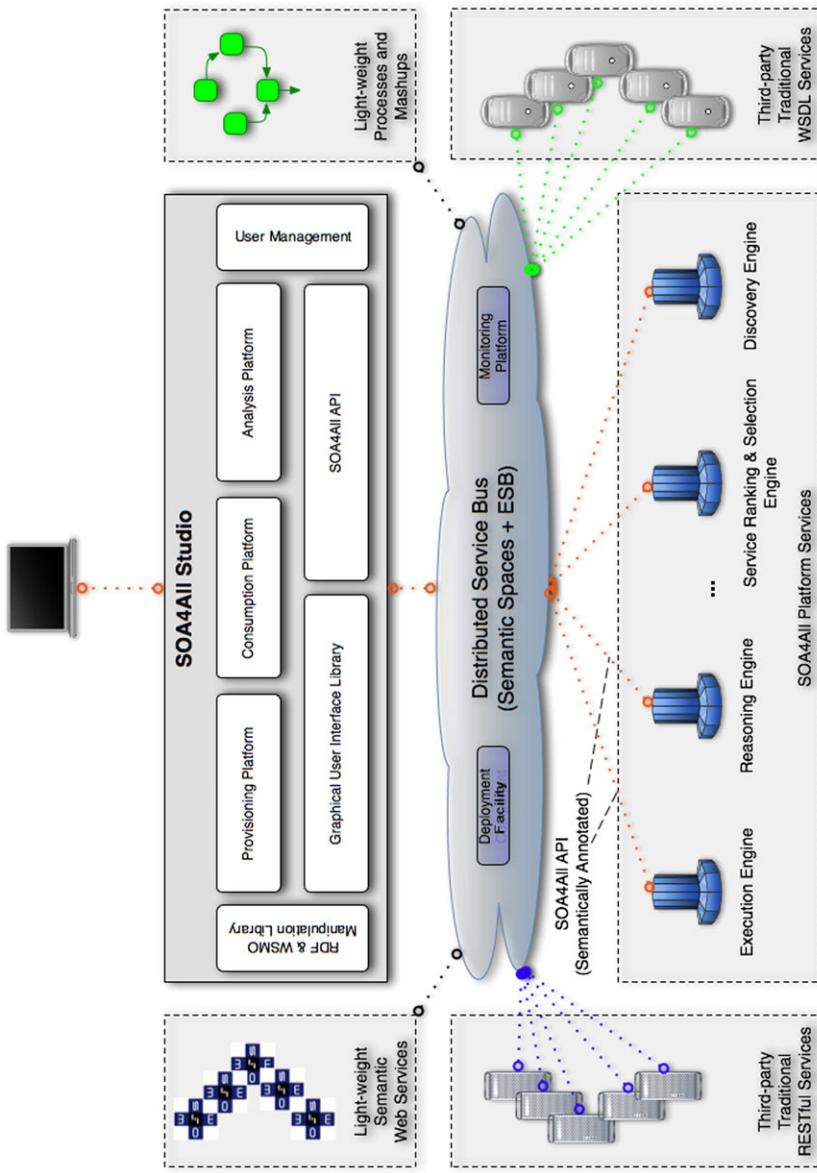


Fig. 10.7 SOA with Triple Spaces in the service cloud

**Service Cloud** In the very center of Fig. 10.7, there is the Service Cloud (in SOA4All called Distributed Service Bus) that offers the core infrastructure needs of the global service delivery platform. The Service Cloud takes on the role of the enterprise service bus of traditional SOA systems by abstracting from service particularities and implementation details. In order to enable a Semantic SOA at the Web scale, it is necessary to extend traditional service bus solutions with techniques to ease deployment, to cope with openness, dynamics and scalability in terms of services, service providers and consumers, and certainly the resulting data and work load on the system. In particular, when searching for novel solutions to the increasing load, Triple Space-based solutions bring along obvious added value. Triple Spaces can be used as a shared memory infrastructure to build repositories or to provide monitoring data about users, services and executions, and as a communication infrastructure to enhance the traditionally message-oriented bus towards a publication infrastructure for anonymous and asynchronous service communication with a notion of event-driven architecture; something currently not available from openly-accessible enterprise service bus technology. More details describing how traditional service bus solutions and Triple Space computing complement each other are given in Sect. 10.2.6.1 below.

**User Front-End and Platform Services** On top of the service cloud, there is the user front-end, in Fig. 10.7 termed SOA4All Studio, and at the bottom of the figure, one can find the platform services. The front-end is a fully Web-based user platform that enables the creation, provisioning, consumption and analysis of Web services that are published to global service delivery platform; this includes the use of platform services. In the particular case of SOA4All, different types of users are supported at different times of interaction. SOA4All knows two major groups of users: (i) a large group of service consumers that use SOA4All for the consumption of functionality that is provided by either platform services or by public Web services, and (ii) a significantly smaller group of (administrative) users that exploits the capabilities of the platform services (via the SOA4All Studio) to annotate, select and compose Web services. As stated above, the platform services deliver service discovery, ranking and selection, composition and invocation functionality, respectively, and are providers of the minimal functionality that is needed to establish a service delivery platform. In that sense, they are very much comparable to WSMX components in the examples of the previous section. As a whole, the ensemble of service cloud, user front-end, and platform services exposes an instance of a fully Web-based and Web-enabled service delivery platform: service integration and coordination at the level of the cloud, Web-style service access via the user front-end, and advanced state-of-the-art service processing, management and maintenance via platform services.

**(Semantic) Web Services** The figure also shows the semantic service descriptions and processes (composed services) that are created and processed by means of the Semantic SOA infrastructure. First, Web services that are exposed either as traditional RESTful services or as traditional WSDL-based services (lower parts of the

figure) are available. Those are invocable third-party business services that are enhanced in terms of automation, composition and invocation. Second, the top-left of the figure depicts the semantic annotations of the business services, the so-called Semantic Web Services. The semantic descriptions are published to the service cloud, and used for reasoning with service capabilities (functionality), interfaces and non-functional properties, as well as context data to deliver advanced and semi-automated discovery, ranking and composition via platform services. Third, in the top-right corner, light-weight processes and mash-ups are shown that are the basis for the definition and execution of service compositions. Both, mash-ups and semantic descriptions of service compositions are also published to the service cloud, and become a public good for automated large-scale service computing.

#### 10.2.6.1 Triple Space-Enhanced Service Delivery Platform

As explained in the previous section, the Service Cloud is the heart of the global service delivery platform. Various challenges that come along with the use of service infrastructures on the Web, such as heterogeneity, dynamics, openness and certainly scalability, cannot be addressed with traditional service bus approaches. For this reason, the service cloud that we present hereafter combines established enterprise service bus technology with Triple Spaces. The enhancement with Triple Spaces offers novel means for event-driven Web-style communication, coordination and data sharing.

The primary goal of the Service Cloud is to ensure that the Semantic SOA scales to the dimensions of the Web, by enabling appropriate distribution techniques that evolve the service buses towards a fully distributed middleware for service coordination and communication without altering standard interaction patterns used in XML-based SOA. While state-of-the-art service technology still mostly relies on client–server communication, as it was discussed in the beginning of this chapter (cf. Sect. 10.1), the service cloud can offer more powerful communication patterns thanks to Triple Space computing, e.g., publish–subscribe, event-driven, semantics-based communication means that allow for further decoupling of the communicating entities in terms of time, processing flow and data schema. In particular, the integrated support for semantics empowers direct links to reasoning or mediation techniques. By adding Triple Spaces, the Service Cloud moreover realizes an integrated infrastructure that grants access to

- Asynchronous and/or event-driven communication.
- Distributed semantic repositories for service, goal and process descriptions.
- A storage infrastructure for sharing monitoring and user data.
- And a computation platform for various types of collaborative activities.

It is important to note that all of the above are delivered without requiring any of the platform services and public Web services to take care of additional access points, i.e., the Service Cloud provides an all-in-one solution for service computing on the Web. This is an important principle and enabler for a SOA that scales beyond the boundaries of corporate networks. The Service Cloud allows any type of

communication efficiently and transparently over the Web by means of sharing or exchanging any type of data in between any type and number of distributed actors.

In summary, the Service Cloud exposes the traditional enterprise service bus functionality extended with scalable Web-style publishing and reading, integrated support for semantics and event-based communication mechanisms as basis for shared data management and collaborative activities. Thanks to the Web principles added by the integrated Triple Space infrastructure, the Service Cloud is offering the core infrastructure services that are necessary for the realization of the Service Web.

#### 10.2.6.2 Service Federations in the Space

In the section above, we argued that a Service Cloud that is blended with Triple Spaces is an enabler for various types of collaborative activities. To conclude this section, we shortly present the principle of service federations. Service federations are grounded in the idea of autonomous and self-configurable service compositions in the space. Similar to agents on a blackboard [9], or the processes in Linda, service are coordinated in the space, and as such create loosely-coupled service compositions without the need for choreographies or man-made orchestrations. This opens up new frontiers for the automatic creation of complex, but self-organized workflows. To a given degree, the control flow is implicitly implied by the data flow, which, in turn, is not centrally managed, but rather implied by the publish and read operations on the space that are triggered by individual services.

A related effort, on a much smaller scale than the Service Web, is realized by a Triple Space-based platform called Sedvice [22]. The core idea of the Sedvice is to establish a Semantic Web vision in form of shared, but localized and potentially personalized spaces. This view is based on the argument that the Semantic Web is characterized by dynamicity and monotonicity of information, and the fact that the meaning of knowledge is depending upon the locality, the use and the user. A space in Sedvice is a virtual closed world in the scope of which information is perceived according to a common context. The primary goal of Sedvice is to enable control-flow free interaction mechanisms for distributed agents running on mobile and personal devices. Through Sedvice, it becomes much easier to create innovative and previously unforeseen mobile applications, as various distributed functionalities can be coordinated on the fly.

### 10.3 Illustration by a Larger Example

In this section, we showcase the various uses of the Triple Space in the context of integration with the Semantic Web Service Execution Environment WSMX. The technical foundations for the example were given in Sect. 10.2.5. The example is based on a generic “Virtual Travel Agency” scenario. The travel agency serves a broker between passengers and different airlines and train services. Passengers can

choose a carrier and specify the departure location and destination of their trip. This information is taken as input for the travel agency service to discover the online booking service of the chosen carrier and to book the trip on behalf of the user. In the given example, the virtual travel agency integrates the booking services of two airlines and one railway company, namely United Airlines, Austrian Airways, and Austrian National Railways.

The showcase scenario is realized on top of the integrated WSMX/Triple Space platform and comprises four steps:

1. WSMX accesses ontologies and service descriptions through the Resource Manager. The Resource Manager retrieves these data from the Triple Space.
2. The user invokes WSMX and specifies the source, destination and carrier for the trip. The request is sent to WSMX via the external communication grounding.
3. In order for the client request to be processed, there is a need for multiple WSMX components to interact. The coordination of these interactions is performed via the Triple Space.
4. After the discovery of a suitable booking service, WSMX interacts with the service and returns the result to the client.

**Accessing Data** Upon start-up of a WSMX instance, the Resource Manager retrieves the known ontologies and service descriptions from memory. The Triple Space-enabled Resource Manager reads these resources non-destructively from its storage space. While the data in the space is available in RDF only, the WSMX runtime operates upon Java objects that represent the WSML language constructs. The Resource Manager must therefore not only query all the known resources from the space, but also transform them from RDF into the wsmo4j object model. wsmo4j is an API and a reference implementation for building Semantic Web Services applications based on the Web Service Modeling Ontology (WSMO).<sup>5</sup> The two listings (Listings 10.1 and 10.2) depict the semantic description of the OEBB service in RDF and WSML, respectively.

**Invoking WSMX** Based on user input about travel start and destination, the virtual travel agency application creates a goal. As we know, a goal in the context of Semantic Web Services is the formal specification of the objective (tasks or activities) that a user likes to have performed and for which fulfillment is sought. In other words, the goal is an implicit description of the service or services that have to be executed in order to satisfy the desired request. In order to discover the required service(s), the virtual travel agency sends the goal to WSMX by invoking the achieveGoal() operation on the client entry point to WSMX. For this second step of the example, WSMX serves as a magical black box that matches the goal description to one of the three reservation and booking services, invokes the best match, and returns the execution result to the client application, i.e., the virtual travel agency.

Listing 10.3 depicts an excerpt of a goal description that in a very simplified manner states that the request (input data) must be of type OEBBTrip, and the response

---

<sup>5</sup>wsmo4j: <http://wsmo4j.sourceforge.net/>.

**Listing 10.1** OEBB Web service in WSML

```

webService oebb
capability OEBB
postcondition OEBBPostCondition
definedBy
    ?request memberOf trip#OEBBTrip.

interface OEBBInterface
choreography OEBBChoreography
stateSignature OEBBStatesignature
importsOntology {trip#tripOntology}

in trip#OEBBTrip withGrounding {
    _ "http://tripcom.org/services/
    oebb.wsdl#twsdl.
    interfaceMessageReference(
        OEBBPortType/bookTrip/
        bookTripRequest")
    out trip#Reservation

transitionRules OEBBTransitionRules
forall {?request} with
    (?request memberOf trip#OEBBTrip) do
        add(_#1 memberOf trip#Reservation)
    endForall
}

```

**Listing 10.2** OEBB Web service in RDF

```

<rdf:RDF xmlns="http://example.org/">
<wsmo4rdf_wsmi:webService
    rdf:about="http://example.org/oebb">
<wsmo4rdf_wsmi:hasCapability>
<wsmo4rdf_wsmi:hasPostcondition>
    <rdf:Description rdf:about=
        "http://example.org/OEBBPostCondition">
        <rdfs:isDefinedBy>?request memberOf
            _ "http://trip.example.org/OEBBTrip".
        </rdfs:isDefinedBy>
    </rdf:Description>
</wsmo4rdf_wsmi:hasPostcondition>
</wsmo4rdf_wsmi:Capability>
<wsmo4rdf_wsmi:Interface rdf:about=
    "http://example.org/OEBBInterface">
    <wsmo4rdf_wsmi:Choreography>
        <rdf:Description rdf:about=
            "http://example.org/OEBBChoreography">
            ...
        </rdf:Description>
    </wsmo4rdf_wsmi:Choreography>
    ...
</wsmo4rdf_wsmi:Choreography>

```

should be of type Reservation. The instance shown in the same listing is the instantiation of the ?request variable in the goal description, and thus specifies the exact trip details that are sought. This goal description is transformed to RDF, analogously as in the ‘Accessing data’ step, and then published in a space from which the WSMX entry point process reads requests. The ‘achieveGoal’ operation is thus mapped to a series of Triple Space primitives, as it was shown in Sect. 10.2.5.3 for the Resource Manager operations. Once the goal description is discovered in the Triple Space by some idle entry point process, WSMX takes care of processing the user request.

**Listing 10.3** Excerpt of the goal description

```

namespace { trip _: "http://trip.example.org/" }

goal tripGoalOEBB
importsOntology { trip#tripOntology }
capability bookTripOEBBCapability
precondition bookTripOEBBPreCondition
definedBy ?request memberOf trip#OEBBTrip.
postcondition bookTripOEBBPostCondition
definedBy
?response memberOf q0#Reservation.

ontology TripOEBB
importsOntology { trip#tripOntology }
instance TripViennaInnsbruck
memberOf trip#OEBBTrip
trip#from hasValue "Vienna"
trip#to hasValue "Innsbruck"

```

**Listing 10.4** Excerpt of a WSMX response

```

namespace { trip _: "http://trip.example.org/" }

ontology _: "http://www.wsmo.org/ontologies/
resp366428579644871840"

instance Reservation341a696a49d34739
memberOf trip#Reservation
trip#from hasValue "Vienna"
trip#to hasValue "Innsbruck"
trip#carrier hasValue "OEBB"
trip#reservationNumber
hasValue "BF082AE24B0B463A"

```

Listing 10.4 shows an excerpt of the response that WSMX delivers to the virtual travel agency application. The response contains an instance of type Reservation, as it was indicated by the goal description. This instantiation states that WSMX successfully booked a trip on behalf of the virtual travel agency, and that there is now a seat reserved for the client in an OEBB train from Vienna to Innsbruck with the reservation number “BF082AE24B0B463A”.

**Coordination of WSMX Components** Upon reception of a goal, WSMX takes care of processing the user request. This requires cooperation of various components, and thus the exchange of events. Components that have fulfilled their task are invoking other components by publishing event-related data in the Triple Space that is used for component coordination. Components that are idle, and that are thus waiting for events, query the space for events that match their “signature”. In order to ensure that no two components could process the same request, events are read destructively, i.e., components make sure, while reading, that no other component with the same functionalities would also pick up the information. This is especially important in settings where particular WSMX components are replicated in order to increase the performance and responsiveness of the semantic execution environment.

Listing 10.5 shows an event in RDF that is published by the invoker component and targeted to the choreography component. The event indicates the target of the

```

<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<rdf:Description rdf:about="http://tripcom.org/wsmx/bd863e04-180e-4a84-9a31-75843510d04d">
  <type xmlns="http://tripcom.org/wsmx/">CHOREOGRAPHY</type>
  <context xmlns="http://tripcom.org/wsmx/">achieveGoal1</context>
  <event xmlns="http://tripcom.org/wsmx/">06df500a-f168-4d54-9298-cb509d517706</event>
</rdf:Description>
</rdf:RDF>

```

**Listing 10.5** Coordination of the invoker and the choreography components

event, “CHOREOGRAPHY”, the operation concerned as context of the event and a unique event identifier. Similarly, a user request is passed between various other components until a service or the services are discovered that allow WSMX to react appropriately to the submitted goal.

Once WSMX has terminated the discovery, ranking and possible composition task, it can invoke the necessary service(s). This invocation is subject to the last step that is described below.

**Invocation of External Service** As stated above, after having discovered a suitable Web service, the WSMX invoker takes care of invoking the external service on behalf of the client. As the example services all rely on SOAP messaging, the invoker wraps the service invocation data in a SOAP/RDF graph. The graph is published into a Triple Space that is under observation of the Web service implementation. The service implementation has to be altered in terms of the communication interface, not however, in terms of service implementation. Instead of receiving a SOAP/XML message, the service has to read a SOAP/RDF graph. An excerpt of one such graph is shown in Listing 10.6. This set of triples is published to the Triple Space by the invoker as request message to the Web service.

Upon processing of the request, the Web service puts together a response SOAP message which is again serialized as RDF graph in form of a SOAP/RDF construct. An example response for the booking service example is shown in Listing 10.7. Note that the body of the actual SOAP message is not transformed into RDF but shipped as RDF Literal and thus object value to the <soaps#content> predicate. Prior to delivering the response to the client, WSMX has to extract the booking information from the SOAP/RDF message. From the response in Listing 10.7, we can read that the achieved booking provides the client with a OEBB ticket from Vienna to Innsbruck, as was desired.

Together with the results presented in the course of the first three steps, we can now describe the whole service invocation process over WSMX by means of communication steps over the Triple Space. In particular, comparing the output of Step 2 (Listing 10.4) with the SOAP/RDF message of Listing 10.7 shows nicely how the actual Web service world is matched to the semantic world of WSMX.

**Listing 10.6** Excerpt of the goal description

```

<soapts#urn:uuid:9F125EEF...>
<soapts#type> "soapts#soapMessage" .
<soapts#mep> "http://www.w3.org/ns/wsdl/in-out" .
<soapts#relationship> "soapts#request" .
<soapts#correlationId> "" .
<soapts#destination> "tsc://localhost:8080/OEBB/
    OEBB" .
<soapts#action> "urn:bookTrip" .
<soapts#serviceName> "OEBB" .
<soapts#bindingVersion> "1.0" .
<soapts#isFault> "false" .
<soapts#messageId> "urn:uuid:9
    F125EEF34D3A411DF1237742241441" .
<soapts#unprocessable> "false" .
<soapts#contentType> "application/soap+xml" .
<soapts#content> "<?xml version='1.0' encoding='
    utf-8'?><soapenv:Envelope xmlns:soapenv
    ='http://schemas.xmlsoap.org/soap/envelope
    /'><soapenv:Body><tns:trip xmlns:tns='d45
    '><tns:from>Vienna</tns:from><tns:to>
    Innsbruck</tns:to></tns:trip></soapenv:Body
    ></soapenv:Envelope>"
```

**Listing 10.7** Excerpt of a WSMX response

```

<soapts#urn:uuid:3E7BEBF1...> <soapts#type> "
    soapts#soapMessage" .
<soapts#mep> "http://www.w3.org/ns/wsdl/in-out" .
<soapts#bindingVersion> "1.0" .
<soapts#destination> "tsc://localhost:8080/OEBB" .
<soapts#action> "urn:bookTripResponse" .
<soapts#unprocessable> "false" .
<soapts#isFault> "false" .
<soapts#messageId> "3
    E7BEBF1BA3D41858833ABD5784D52B8" .
<soapts#relationship> "soapts#response" .
<soapts#correlationId> "urn:uuid:9
    F125EEF34D3A411DF1237742241441" .
<soapts#contentType> "application/soap+xml" .
<soapts#content> "<?xml version='1.0' encoding='
    utf-8'?><soapenv:Envelope xmlns:soapenv
    ='http://schemas.xmlsoap.org/soap/envelope
    /'><soapenv:Body><tns:reservation xmlns:
    tns='http://tripcom.org/ns/d45'><tns:from>
    Vienna</tns:from><tns:to>Innsbruck</tns:to
    ><tns:reservationNumber>
    BF082AE24B0B463A</tns:
    reservationNumber><tns:carrier>OEBB</tns:
    carrier></tns:reservation></soapenv:Body></
    soapenv:Envelope>" .
```

## 10.4 Summary

In summary, this example shows nicely how a Triple Space provides a platform for asynchronous communication and coordination of Semantic Web Services and WSMX components. It moreover gives an example of how the Triple Space can transparently integrate with existing technologies and tools, and how space-based approaches limit the storage needs and maintenance redundancy of semantic artefact in the context of WSMX, but also on a much more general basis.

## References

1. Brickley, D., Guha, R.V.: RDF Vocabulary Description Language 1.0: Rdf Schema. W3C Recommendation (2004)
2. Cabri, G., Leonardi, L., Zambonelli, F.: MARS: a programmable coordination architecture for mobile agents. *IEEE Internet Computing* **4**(4), 26–35 (2000)
3. Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: Named graphs. *Journal of Web Semantics* **3**(4), 247–267 (2005)
4. Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: Named graphs, provenance and trust. In: 14th Int. World Wide Web Conference, pp. 613–622. ACM Press, New York (2005)
5. Ciancarini, P., Knoche, A., Tolksdorf, R., Vitali, F.: PageSpace: an architecture to coordinate distributed applications on the Web. *Computer Networks and ISDN Systems* **28**(7–11), 941–952 (1996)
6. Ciancarini, P., Tolksdorf, R., Zambonelli, F.: A survey of coordination middleware for XML-centric applications. *Knowledge Engineering Review* **17**(4), 389–405 (2002)
7. de Bruijn, J., Lausen, H., Polleres, A., Fensel, D.: The Web service modeling language WSML: an overview. In: 3rd European Semantic Web Conference, pp. 590–604 (2006)
8. De Francisco Marcos, D., Toro Del Valle, G., Nixon, L.: Towards a multimedia content marketplace implementation based on triplespaces. In: 7th Int. Semantic Web Conference, pp. 875–888 (2008)
9. Engelmore, R.: Blackboard Systems. Addison-Wesley, Reading (1988)
10. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.-M.: The many faces of publish/subscribe. *ACM Computing Surveys* **35**(2), 114–131 (2003)
11. Fensel, D.: Triple-space computing: semantic Web services based on persistent publication of information. In: IFIP Int. Conference on Intelligence in Communication Systems, pp. 43–53 (2004)
12. Fensel, D., Krummenacher, R., Shafiq, O., Kuehn, E., Riemer, J., Ding, Y., Draxler, B.: TSC—triple space computing. *E&I. Elektrotechnik und Informationstechnik* **124**(1/2), 31–38 (2007)
13. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000)
14. Fielding, R.T., Taylor, R.N.: Principled design of the modern Web architecture. *ACM Transactions on Internet Technology* **2**(2), 115–150 (2002)
15. Franklin, M., Halevy, A., Maier, D.: From databases to dataspaces: a new abstraction for information management. *SIGMOD Record* **34**(4), 27–33 (2005)
16. Gelernter, D.: Generative communication in Linda. *ACM Transactions on Programming Languages and Systems* **7**(1), 80–112 (1985)
17. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): concepts and abstract syntax. W3C Recommendation (2004)
18. Krummenacher, R., Simperl, E., Fensel, D.: Towards scalable information spaces. In: Workshop on New Forms of Reasoning for the Semantic Web: Scalable, Tolerant and Dynamic, ISWC (2007)
19. Krummenacher, R., Simperl, E., Cerizza, D., Della Valle, E., Nixon, L., Foxvog, D.: Enabling the European patient summary through triplespaces. *Computer Methods and Programs in Biomedicine* (2009)
20. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language overview. W3C Recommendation (2004)
21. Mocan, A., Moran, M., Cimpian, E., Zaremba, M.: Filling the gap—extending service oriented architectures with semantics. In: IEEE Int. Conference on e-Business Engineering, pp. 594–601 (2006)
22. Oliver, I., Honkola, J., Ziegler, J.: Dynamic, localised space based semantic Webs. In: IADIS Int. WWW/Internet Conference, pp. 426–431 (2008)
23. Omicini, A., Zambonelli, F.: Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems* **2**(3), 251–269 (1999)
24. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation (2008)

25. Simperl, E., Krummenacher, R., Nixon, L.J.B.: A coordination model for triplespace computing. In: 9th Int. Conference on Coordination Models and Languages, pp. 1–18 (2007)
26. Tolksdorf, R.: Laura—a service-based coordination language. *Science of Computer Programming* **31**(2/3), 359–381 (1998)
27. zur Muehlen, M., Nickerson, J.V., Swenson, K.D.: Developing Web services choreography standards—the case of REST vs. SOAP. *Decision Support Systems* **40**(1), 9–29 (2005)



# Chapter 11

## OWL-S and Other Approaches

**Abstract** WSMO is just one of the frameworks to semantically describe Web services that have been developed in the last years. OWL-S and its extension SWSF, as well as METEOR-S, have proposed alternative approaches to WSMO that, despite sharing some similarities, exhibit significant differences with respect to the technological standards, languages and underlying formalisms that are used. This chapter gives an overview of these approaches together with a comparison based on the criteria referring to the core design principles of Web and Semantic Web development, as well as of distributed, service-oriented computing on the Web. In addition, the chapter introduces IRS-III, a framework and implementation infrastructure that, similarly to WSMX, supports the creation of Semantic Web Services based on WSMO ontology.

### 11.1 Motivation

WSMO is just one of the frameworks to semantically describe Web services that have been developed in the last years. OWL-S and its extension SWSF, as well as METEOR-S, have proposed alternative approaches to WSMO that, despite sharing some similarities, exhibit significant differences with respect to the technological standards, languages and underlying formalisms that are used. Moreover, the WSMO framework has been used and extended by implementation projects such as IRS-III, in addition to WSMX, which was introduced in Chap. 9. The goal of this chapter is to explain the main features of the approaches, to overview the tools supporting the approaches, and to compare the approaches to WSMO in terms of a set of pre-defined criteria which are introduced in the remainder of this section.

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 a information repository for human consumption into a worldwide system for distributed Web computing. Therefore, it comes naturally that in order to achieve these goals a Semantic Web Services framework has to be compliant and integrate the core design principles of the World Wide Web, of the Semantic Web and of distributed, service-oriented computing on the Web. These principles are reflected by the following set of criteria which will be used to compare alternative Semantic Web Services frameworks:

- **Web compliance** An approach must support the foundational Web concepts such as IRIs, namespaces, XML, and decentralization of resources.
- **Ontological foundation** Ontologies must be used as the data model of an approach.
- **Strict decoupling** Resources of an approach such as ontologies, Semantic Web Service descriptions, goals and mediators must be defined in isolation.
- **Centrality of mediation** Handling of the heterogeneities at various levels (data, underlying ontology, protocol and process) must be addressed in a comprehensive fashion by the approach.
- **Ontological role separation** An approach must employ epistemological differentiation of user desires and service offerings.
- **Description vs implementation** Description notions used by an approach must not be bound to the specific executable technologies, in other words, an approach should enable grounding to various implementations.
- **Execution semantics** An approach should support the definition of formal execution semantics of a reference implementation.
- **Service vs Web service separation** An approach should make distinction between the service as a computational entity (i.e., the Web service) and actual value provided (i.e., the service).

## 11.2 OWL-S

OWL-S [12], formerly known as DAML-S, is an ontology for the description of Semantic Web Services expressed in the Web Ontology Language (OWL) [13]. OWL-S defines an upper ontology for services with three major elements (see Fig. 11.1 from [12]):

- **Service Profile** which describes what the service does in terms of inputs, outputs, preconditions, and effects (IOPEs).
- **Service Grounding** which describes how the service can be accessed, usually by grounding to WSDL.
- **Service Model** which describes how the service works in terms of a process model that may describe a complex behavior over underlying services.

The service serves as an organizational point of reference for describing Web services; every service is declared by creating an instance of the service concept. Listing 11.1<sup>1</sup> shows a fictitious airline service called BravoAir.

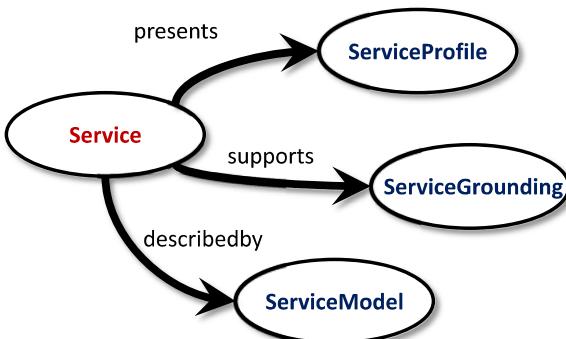
### 11.2.1 Service Profile

In OWL-S, the service profile describes the intended purpose of the service from the perspective of the provider and of the requester. The service profile is designed

---

<sup>1</sup>Listing taken from <http://www.daml.org/services/owl-s/1.1/examples.html>.

**Fig. 11.1** OWL-S service ontology




---

```

<rdf:RDF>
<owl:Ontology rdf:about="">
  <rdfs:comment>
    This ontology represents the OWL-S service description for the BravoAir Web service example.
  </rdfs:comment>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/Service.owl"/>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProfile.owl"/>
  <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl"/>
</owl:Ontology>

<service:Service rdf:ID="BravoAir_ReservationAgent">
  <!-- Reference to the BravoAir Profile -->
  <service:presents rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProfile.owl#
    Profile_BravoAir_ReservationAgent"/>
  <!-- Reference to the BravoAir Process Model -->
  <service:describedBy rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl#
    BravoAir_ReservationAgent_ProcessModel"/>
  <!-- Reference to the BravoAir Grounding -->
  <service:supports rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirGrounding.owl#
    Grounding_BravoAir_ReservationAgent"/>
</service:Service>
</rdf:RDF>
  
```

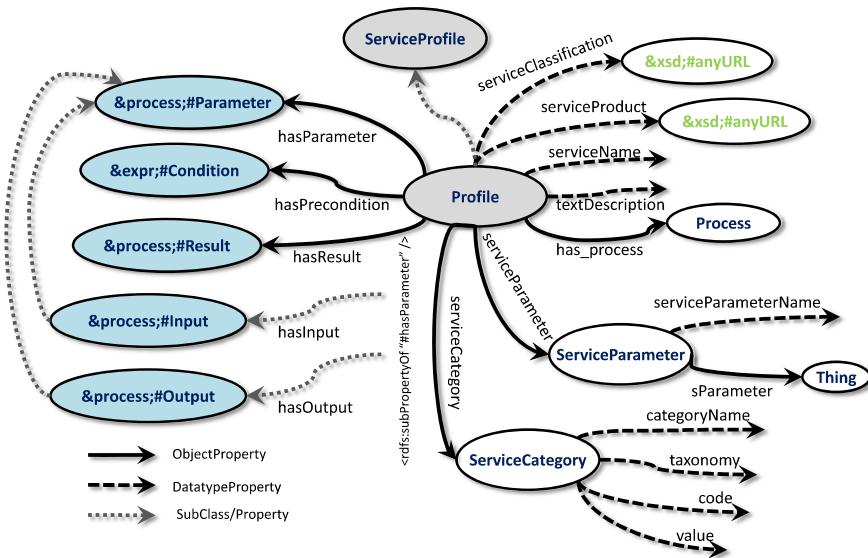
---

**Listing 11.1** BravoAir service instance

to accommodate requirements of a service-seeking agent to determine whether the service fulfills user needs. In order to offer rich modeling capabilities to capture what is accomplished by the service, limitations on service applicability and quality of service, the service profile is further decomposed as presented in Fig. 11.2 from [12].

Through the service profile, users are able to represent functional capabilities of a service, related provider information, and features that specify characteristics of the service regardless of its actual representation. The ability to use OWL sub-classes makes it possible to create specializations of a service representation.

The **provider information** part of the service profile, for instance, aspects such as (*serviceName*, *textDescription*, and *serviceProduct*, yields a human-readable description of the service and its offering, as well as contact information pointing to the service provider (e.g., the service maintenance operator and the customer representative)).



**Fig. 11.2** OWL-S service profile

The **functional description** part captures the transformation produced by the service, that is, the required inputs (*Input*) and the generated outputs (*Output*). Additionally, the execution of the service might be constrained by some external conditions (*Condition*) and have effects (*Result*) that change these conditions.

Finally, the **description of a host of properties** part of the service profile covers the service category (*ServiceCategory*), quality ratings of the service and an unbounded list of service parameters that can contain any type of information (for instance, an estimate of the maximum response time, the geographical availability of the service), given as *ServiceParameter*.

Listing 11.2 gives a Service Profile example of the BravoAir airline site.<sup>2</sup>

### 11.2.2 Service Grounding

The service grounding specifies how the abstract service representations (namely, Service Profile and Service Model) are mapped to the concrete service description elements required for interacting with the service (inputs and outputs of an atomic processes). In this sense, atomic processes represent service communication primitives. In contrast, concrete messages sent to, and received from the concrete service are specified explicitly in a Service Grounding.

Though initially based on the Web Service Description Language (WSDL), OWL-S does not impose any constraints on the choice of a grounding mechanism. The complementary strengths of these two specification languages allow a

<sup>2</sup> Listing taken from <http://www.daml.org/services/owl-s/1.1/examples.html>.

---

```

<rdf:RDF>
  <owl:Ontology rdf:about="">
    <rdfs:comment>BravoAir Example for OWL-S Profile description</rdfs:comment>
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/Service.owl"/>
    ...
  </owl:Ontology>
  <profile:serviceParameter>
    <addParam:GeographicRadius rdf:ID="BravoAir-geographicRadius">
      <profile:serviceParameterName>BravoAir Geographic Radius</profile:serviceParameterName>
      <profile:sParameter rdf:resource="http://www.daml.org/services/owl-s/1.0/Country.owl#UnitedStates"/>
    </addParam:GeographicRadius>
  </profile:serviceParameter>

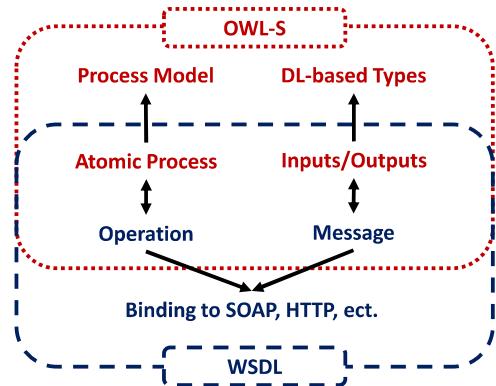
  <profile:serviceCategory>
    <addParam:NAICS rdf:ID="NAICS-category">
      <profile:value>Airline reservation services </profile:value>
      <profile:code>561599</profile:code>
    </addParam:NAICS>
  </profile:serviceCategory>
  ...
  <profile:hasInput rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl#DepartureAirport_In"/>
  <profile:hasOutput rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl#AvailableFlightItineraryList_Out"/>
</rdf:RDF>

```

---

**Listing 11.2** BravoAir service profile

**Fig. 11.3** OWL-S service grounding



service developer to benefit from using both OWL-S' process model and class typing features—as opposed to what XML Schema provides—and to reuse standardized approaches and software support for message exchange related to WSDL.

The WSDL/OWL-S grounding is based on a set of correspondences between the two specifications, as presented in Fig. 11.3 introduced in [12].

- An OWL-S atomic process corresponds to a WSDL (v1.1) operation.
- Inputs and outputs of an OWL-S atomic process correspond to a WSDL message.
- Input and output types of an OWL-S atomic process correspond to the WSDL's extensible notion of abstract type (and can be used in WSDL message-part specifications).

Listing 11.3<sup>3</sup> gives an overview of the OWL-S BravoAir Service Grounding. The listing covers only the WSDL-based grounding definition of the *GetDesiredFlight-Details* atomic process.

### 11.2.3 Service Model

OWL-S defines an interaction with a service in the form of a process. The notion of a process is represented by the OWL-S v1.1 class *Process*, which is conceived as a sub-class of the class *ServiceModel*. The purpose of a process is twofold: it generates and returns some information as a function of the information given and the current world state described by the inputs and outputs of the process, and it can produce a change in the world, captured through the preconditions and effects of the process.

A process represents a specification of the approaches a client may use to interact with a service. OWL-S distinguishes between three classes of processes, as presented in Fig. 11.4 [12]:

- **Atomic processes** which represent directly invocable computational entities. They are executed in a single step and cannot be further refined. An atomic process takes an input message, and returns output message. Additionally, atomic processes have an associated grounding which enables a service requester to construct input and de-construct output messages depending on the concrete service specification and communication language in use.
- **Simple processes** which, similarly to atomic processes, are conceived as having single-step executions, but do not have associated grounding and are not invocable. The simple processes are used as elements of abstraction, offering a specialized way to use an atomic process, or to simplify the representation of a composite process for planning and reasoning purposes.
- **Composite processes** which represent more complex processes decomposable into other (composite or non-composite) processes. The decomposition is defined in terms of control constructs such as sequence and if-then-else (see below). A composite process describes a behavior the client can enact by sending and receiving a set of messages.

For defining composite processes, OWL-S supports the following control structures:

- **Sequence** which is the most basic control construct defining a list of components that must be done in specified order.
- **Split** components represent process components determined for a concurrent execution.
- **Split + Join** which, similarly to split, defines a bag of components that are meant to be executed in parallel but with a precise synchronization point.

---

<sup>3</sup>Listing taken from <http://www.daml.org/services/owl-s/1.1/examples.html>.

---

```

<rdf:RDF>
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/Service.owl" />
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/Grounding.owl" />
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirService.owl" />
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl" />
  </owl:Ontology>

  <grounding:WsdlGrounding
    rdf:ID = "Grounding_BravoAir_ReservationAgent">
    <service:supportedBy rdf:resource = "&ba_service;#BravoAir_ReservationAgent"/>
    <!-- Collecton of all the groundings specifications -->
    <grounding:hasAtomicProcessGrounding rdf:resource = "#WsdlGrounding_GetDesiredFlightDetails"/>
    <grounding:hasAtomicProcessGrounding rdf:resource = "#WsdlGrounding_SelectAvailableFlight"/>
    <grounding:hasAtomicProcessGrounding rdf:resource = "#WsdlGrounding_LogIn"/>
    <grounding:hasAtomicProcessGrounding rdf:resource = "#WsdlGrounding_ConfirmReservation"/>
  </grounding:WsdlGrounding>

  <grounding:WsdlAtomicProcessGrounding rdf:ID = "WsdlGrounding_GetDesiredFlightDetails">
    <!-- Grounding for the Atomic Process GetDesiredFlightDetails -->
    <grounding:owlsProcess rdf:resource = "&ba_process;#GetDesiredFlightDetails"/>
    <!-- Reference to the corresponding WSDL operation -->
    <grounding:wsdlOperation rdf:resource = "#GetDesiredFlightDetails_operation"/>

    <!-- Reference to the WSDL input message -->
    <grounding:wsdlInputMessage>
      <xsd:anyURI rdf:value = "&BravoAirGroundingWSDL;#GetDesiredFlightDetails_Input"/>
    </grounding:wsdlInputMessage>

    <!-- Mapping of OWL-S inputs to WSDL message parts -->
    <grounding:wsdlInputs rdf:parseType = "Collection">
      <grounding:WsdlInputMessageMap>
        <grounding:owlsParameter rdf:resource = "&ba_process;#departureAirport_In"/>
        <grounding:wsdlMessagePart>
          <xsd:anyURI rdf:value = "&BravoAirGroundingWSDL;#departureAirport"/>
        </grounding:wsdlMessagePart>
      </grounding:WsdlInputMessageMap>
      <grounding:WsdlInputMessageMap>
        <grounding:owlsParameter rdf:resource = "&ba_process;#arrivalAirport_In"/>
        <grounding:wsdlMessagePart>
          <xsd:anyURI rdf:value = "&BravoAirGroundingWSDL;#arrivalAirport"/>
        </grounding:wsdlMessagePart>
      </grounding:WsdlInputMessageMap>
      <grounding:WsdlInputMessageMap>
        <grounding:owlsParameter rdf:resource = "&ba_process;#outboundDate_In"/>
        <grounding:wsdlMessagePart>
          <xsd:anyURI rdf:value = "&BravoAirGroundingWSDL;#outboundDate"/>
        </grounding:wsdlMessagePart>
      </grounding:WsdlInputMessageMap>
      <grounding:WsdlInputMessageMap>
        <grounding:owlsParameter rdf:resource = "&ba_process;#inboundDate_In"/>
        <grounding:wsdlMessagePart>
          <xsd:anyURI rdf:value = "&BravoAirGroundingWSDL;#inboundDate"/>
        </grounding:wsdlMessagePart>
      </grounding:WsdlInputMessageMap>
      <grounding:WsdlInputMessageMap>
        <grounding:owlsParameter rdf:resource = "&ba_process;#roundTrip_In"/>
        <grounding:wsdlMessagePart>
          <xsd:anyURI rdf:value = "&BravoAirGroundingWSDL;#roundTrip"/>
        </grounding:wsdlMessagePart>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInputs>
  <grounding:wsdlReference>

```

---

**Listing 11.3** BravoAir service grounding

---

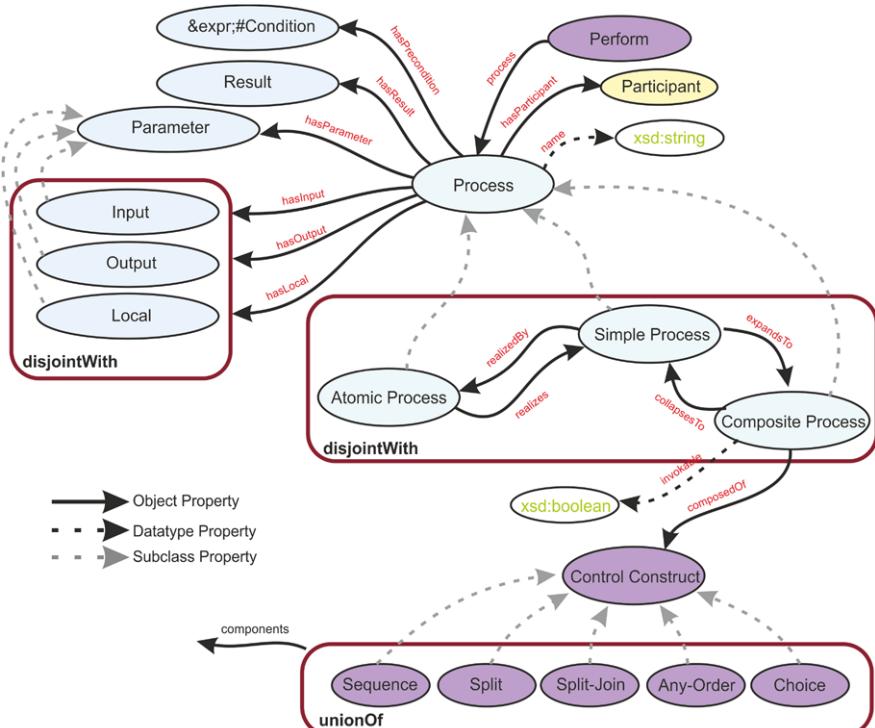
```

<xsd:anyURI rdf:value="http://www.w3.org/TR/2001/NOTE-wsdl-20010315"/>
</grounding:wsdlReference>
</grounding:WsdlAtomicProcessGrounding>

<grounding:WsdlOperationRef rdf:id="GetDesiredFlightDetails_operation">
  <rdfs:comment>
    A pointer to the WSDL operation used for GetDesiredFlightDetails
  </rdfs:comment>
  <!-- locate port type to be used -->
  <grounding:portType>
    <xsd:anyURI rdf:value="#BravoAirGroundingWSDL;#GetDesiredFlightDetails_PortType"/>
  </grounding:portType>
  <!-- locate operation to be used -->
  <grounding:operation>
    <xsd:anyURI rdf:value="#BravoAirGroundingWSDL;#GetDesiredFlightDetails_operation"/>
  </grounding:operation>
</grounding:WsdlOperationRef>
...
</rdf:RDF>

```

---

**Listing 11.3** (Continued)**Fig. 11.4** OWL-S service model

- **Any-Order** construct does not prescribe the execution order but does not allow concurrent execution (neither interleaving, nor overlapping).
- **Choice** specifies any of the control constructs that may be chosen for execution from the given bag of components (given by the component's property).
- **If-Then-Else** constructs have the same semantics as in the modern programming languages—test if the condition holds and execute construct associated to the then part; otherwise execute the one associated to the else part. The condition is viewed as a logical expression.
- **Iterate** enables repetitive execution of a component without any assumption about the number of iterations, repetition initiation, resumption and termination.
- **Repeat-While and Repeat-Until** define the repetition which is constrained, as opposed to the previous construct, by the logical expression represented as a condition. The difference between the two constructs follows the well-known programming language conventions.

Listing 11.4<sup>4</sup> represents the OWL-S BravoAir Service Model. The BravoAir process represents a composite process consisting of a sequence of *GetDesiredFlight-Details*, *SelectAvailableFlight* and *BookFlight*, where the latter is itself a composite process consisting of the *Login* and *ConfirmReservation* atomic processes.

#### 11.2.4 An Extension to OWL-S

Semantic Web Services Framework (SWSF) [3] has its roots in OWL-S and the ISO 18629 Process Specification Language (PSL) [8]. It has been developed with an aim to overcome deficiencies of OWL-S, namely:

- To provide means to overcome the restricted expressivity of Description Logics.
- To provide formal definitions of the semantic aspects of conditions and the dynamic aspects of Semantic Web Services by relying on PSL.
- Providing a single semantic language framework.

Two major components comprise SWSF: the conceptual model Semantic Web Service Ontology (SWSO), and the family of knowledge representation languages Semantic Web Service Language (SWSL). SWSO inherits the three major OWL-S elements—profiles, grounding and service model, further extended and refined in terms of a more expressive underlying language (SWSL) and a richer behavioral process model (based on PSL). Two independent formalizations of the conceptual model are available:

- **FLOWS (First-order Logic Ontology for Web Services)** which relies on the semantics of the SWSL-FOL language, and
- **ROWS (Rule Ontology for Web Services)** which uses the semantics of the SWSL-Rules language.

---

<sup>4</sup>Listing taken from <http://www.daml.org/services/owl-s/1.1/examples.html>.

---

```

<rdf:RDF>
  <process:ProcessModel rdf:ID="BravoAir_ReservationAgent_ProcessModel">
    <process:hasProcess rdf:resource="#BravoAir_Process"/>
    <service:describes rdf:resource="http://www.daml.org/services/owl-
      s/1.0/BravoAirService.owl#BravoAir_ReservationAgent"/>
  </process:ProcessModel>

  <process:CompositeProcess rdf:ID="BravoAir_Process">
    <process:composedOf>
      <process:Sequence>
        <process:components rdf:parseType="Collection">
          <process:AtomicProcess rdf:about="#GetDesiredFlightDetails"/>
          <process:AtomicProcess rdf:about="#SelectAvailableFlight"/>
          <process:CompositeProcess rdf:about="#BookFlight"/>
        </process:components>
      </process:Sequence>
    </process:composedOf>
  </process:CompositeProcess>
  <process:CompositeProcess rdf:ID="BookFlight">
    <process:composedOf>
      <process:Sequence>
        <process:components rdf:parseType="Collection">
          <process:AtomicProcess rdf:about="#Login"/>
          <process:AtomicProcess rdf:about="#ConfirmReservation"/>
        </process:components>
      </process:Sequence>
    </process:composedOf>
  </process:CompositeProcess>

  <process:AtomicProcess rdf:ID="Login">
    <process:hasInput rdf:resource="#AcctName_In"/>
    <process:hasInput rdf:resource="#Password_In"/>
  </process:AtomicProcess>

  <process:Input rdf:ID="AcctName_In">
    <process:parameterType rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#AcctName"/>
  </process:Input>

  <process:Input rdf:ID="Password_In">
    <process:parameterType rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#Password"/>
  </process:Input>
</rdf:RDF>

```

---

**Listing 11.4** BravoAir service model

The PSL-based description of the service behavior introduces two fundamental extensions: a structured notion of the atomic process, and an infrastructure for specifying various forms of data flow.

The SWSL family of languages is compliant with the Web principles through the usage of IRIs, the integration of XML built-in types and XML namespaces, as well as import mechanisms. The languages are designed in layers, with increased modeling power. Just like in WSMO, SWSL defines two major variants:

- **SWSL-First-order Logic** which is grounded in first-order logic including HiLog and F-Logic, and
- **SWSL-Rules** representing a Logic Programming language introduced to support service-related tasks such as discovery, contracting and policy specification.

### 11.2.5 Tool Support

Rather than defining and implementing an architecture for fully-fledged tool support of Semantic Web Services using OWL-S, as has been the aim of SESA and WSMX, the community has produced a number of tools that play different roles. Here we are using the SESA architecture as the basis to survey and compare them.

#### 11.2.5.1 Reasoning

OWL-S is an OWL ontology like any other, therefore reasoning support is provided primarily by OWL reasoners. OWL itself distinguishes among three variants of increasing expressivity: OWL-Lite, OWL-DL and OWL-Full. OWL-Full has no decidable reasoning and OWL-Lite provides support for only simple constraints over classification hierarchies, therefore reasoner implementation concentrates on OWL-DL which is based in description logics. Pellet [22] and FaCT/FaCT++ [27] are two such OWL-DL reasoners based on tableau algorithms. Since, however, OWL-DL is not sufficiently expressive in most cases to define all parts of the user models, for instance, defining conditions within the process model, an embedding of another rule language such as SWRL<sup>5</sup>, or rule-oriented extensions are used for this purpose [19]. Consequently, reasoners capable of dealing with OWL-DL and rules, such as RACER [9] and Kaon2 [14] are often utilized. An advantage of the OWL-S model over WSMO and WSML is that, just as WSMO's representation in OCML, the same reasoner can be used to carry out meta-reasoning about the model, and therefore the meta-model itself can be used in user models.

#### 11.2.5.2 Discovery

The process underlying discovery in OWL-S is usually referred to as matchmaking. It heavily relies on the explicit specification of inputs and outputs. The ontological relationships between the inputs and outputs (I/O) of a ‘template’, which in this sense plays the role of a WSMO goal, but is not explicitly conceptualized as such, and a candidate service are compared. A set of five ‘filters’ classify the structural relationships that are accepted between the template and candidate I/Os. Klusch et al. [11] proposes OWLS-MX, a match-maker that implements such an algorithm and combines it with syntactic matching to cover those cases when semantic matchmaking fails—for instance, when two ontologies are disjoint since there is no means for mediation to map these. The survey in this paper also affirms that there are no implementations that offer both this I/O-based approach and also consider the pre-conditions and effects, completing the so-called ‘IOPE’-based matchmaking that was envisioned for OWL-S. To the knowledge of the authors, there exists no discovery approach for OWL-S that also covers the behavior of a service, since they

---

<sup>5</sup><http://www.w3.org/Submission/SWRL>.

are simply considered to be atomic in the model. A second approach we would like to highlight is proposed in [24]. It considers the addition of the OWL-S model to UDDI descriptions and the extension of an open-source UDDI engine, jUDDI, to support semantic matchmaking based on these descriptions.

#### 11.2.5.3 Choreography and Orchestration

The OWL-S model fundamentally assumes that services are invoked as atomic actions, and therefore no model of choreography is contained. On the other hand, the *process model* allows services to be attached either to an atomic process, which is usually grounded to a WSDL operation, or to a composite process. The process model allows for a hierarchical control-flow oriented decomposition of composite processes, ultimately producing some ordering over atomic processes. When such a service is to be executed the OWL-S Virtual Machine [19], formerly the DAML-S Virtual Machine, executes the atomic processes up to completion of the composite process description, returning the outputs just as if the invocation had been atomically carried out. In evaluating a composite process model, the Virtual Machine needs access to OWL-DL and SWRL reasoning provided by RACER. In SESA terms, then the OWL-S Virtual Machine fulfills the basic role of an Orchestration Engine, but OWL-S has no support from a Choreography Engine.

#### 11.2.5.4 Mediation

The OWL-S model does not provide dedicated support for mediators. Data mediation services can, of course, be described using the provision for other services. There are two notable approaches considering data mediation explicitly in OWL-S: Paolucci et al. [20] considers exactly this embedding of WSMO's mediator model into OWL-S, whereas work related with [26] considers mediation services as 'shims', which mix, rather than isolate, the syntactic and semantic representations of data in order to efficiently apply mediation at the syntactic level. Since, as stated above, the OWL-S model treats services as invoked atomically and without choreography, there is no notion of automated process mediation in OWL-S. If the interaction between two services requires process mediation, this must be carried out as part of a composite process, though the process model has deficiencies even in this regard due to its notion of atomic service interaction [16].

#### 11.2.5.5 Composition

A great deal of work in automated service composition has been carried out in OWL-S, not least since its abstraction of service interactions to atomic ones, and its simple model of composite processes are a good fit to existing techniques. Much work has been carried out using AI planning-based techniques, for instance, resulting from [29]. A state-of-the-art tool, which considers the need to apply re-planning during the execution of a composite service, is described in [10].

**Table 11.1** OWL-S summary

Compliance	Criteria	Comments
✓	Web compliance	Supports URI, namespaces and XML
✓	Ontology-based	Based on OWL
✓	Strict decoupling	Ontologies and services are specified independently
	Centrality of mediation	No support for mediation
	Ontological role separation	Doesn't have the notion of user desires
✓	Description vs. implementation	Opened for various grounding
	Execution semantics	Doesn't have the notion of execution semantics
✓	Services vs. Web Services	Designed as a means to describe Web services

### 11.2.6 OWL-S Summary

Table 11.1 lists the main features of OWL-S compared against the criteria introduced in Sect. 11.1.

## 11.3 METEOR-S

The METEOR-S project<sup>6</sup> covers the complete life cycle of Semantic Web processes including annotation, discovery, composition and enactments of Web services:

- **Semantic annotation of Web services** The annotation is supported through the METEOR-S Web Service Annotation Framework (MWSAF) which builds upon the Web Service Semantics (WSDL-S) and its successor Semantic Annotations for WSDL and XML Schema (SAWSDL).
- **Semantics-based discovery of Web services** The Web service discovery mechanisms are offered through the METEOR-S Web Service Discovery Infrastructure (MWSDI) which enhances existing Web service discovery infrastructure by using semantics.
- **Composition of Web services** The METEOR-S Web Service Composition Framework (MWSCF) increases the flexibility of the Web services' composition through semantic process templates.

The approach is strongly founded on existing Web services standards, incrementally extending them with semantic support—a feature which contrasts with approaches such as OWL-S or WSMO.

---

<sup>6</sup><http://lsdis.cs.uga.edu/projects/meteor-s/>.

### 11.3.1 Semantic Annotation of Web services

The METEOR-S project comprises the so-called MWSAF [21], a framework for the semi-automatic annotation of Web services. These annotations address four different aspects of Web services's semantics. First, MWSAF supports including annotations about the semantics of the inputs and the outputs of Web services. Second, the annotation framework supports the definition of functional semantics, i.e., what the service does. Third, MWSAF allows including execution semantics to support verifying the correctness of the execution of Web services. At last, the framework supports including information regarding the quality of service, such as performance or costs associated to the execution of Web services.

Initial research on the framework was devoted to supporting the semi-automatic annotation of XML schemas as part of Web services definitions. This work is based on the transformation of both XML schemas and ontologies into a common representation format called SchemaGraph [21] in order to facilitate the matching between both models (as presented through the examples shown in Figs. 11.5 and 11.6 due to [21]). Once ontologies and XML schemas are translated into this common representation, a set of matching algorithms can be applied to (semi-)automatically enhance the syntactic definitions with semantic annotations.

In a nutshell, the matching algorithm computes a ‘Match Score’ between each element of the WSDL SchemaGraph and the Ontology SchemaGraph. This score takes into account the linguistic and the structural similarity. The best matching element is chosen by taking into account both the Match Score and the specificity of the concepts. Finally, a global matching average is computed to help in selecting the best overall match between Web services and ontologies. Further details about the algorithm can be found in [21].

The MWSAF is composed of three main components: an ontology store, the matcher library and a translator library. The first component stores the ontologies that will be used for annotating the Web services. The matcher library provides different algorithm implementations for linguistic and structural matching between concepts and Web services elements. The translator library consists of the programs used for generating the SchemaGraph representation for ontologies and Web services.

The MWSAF assists users in annotating Web services by browsing and computing the similarity between domain models and Web services elements. The last step in the annotation process is their representation for future reuse for automated processing. To cater for this the METEOR-S project makes use of WSDL-S, which we present in more detail in the following section.

#### 11.3.1.1 WSDL-S and SAWSDL

WSDL-S was proposed as a member submission to the W3C in November 2005 between the LSDIS Laboratory and IBM [2]. In line with the philosophy of the

**Fig. 11.5** Mapping of the XML schema constructs to the SchemaGraph representations

XML schema Construct	SchemaGraph representation
ComplexType	Node
Elementary XML Date Type Element defined under complexType	Node and an Edge between complexType node and this node with name „hasElement”
ComplexType XML Date Type Element defined under complex Type	Edge
SimpleType	Node
Values defined for simple types	Node and an Edge between simpleType and this node with name „hasValue”
Elements	Nodes
<b>Example</b>	
<pre> &lt;xsd:complexType name=„Direction”&gt;   &lt;xsd:sequence&gt;     &lt;xsd:element maxOccurs=„1” minOccurs=„1”       nillable=„true” name=„compas”       type=„xsd1:DirectionCompass” /&gt;     &lt;xsd:element maxOccurs=„1” minOccurs=„1”       name=„degrees” type=„xsd:int” /&gt;   &lt;/xsd:sequence&gt; &lt;/xsd:complexType&gt;</pre>	
<b>SchemaGraph representation of the part of WSDL</b>	

METEOR-S project, WSDL-S is a lightweight approach to associate semantic annotations with Web services using existing Web service standards not related to semantic technologies. Concretely, semantic annotations in the form of URI references to external models are added to the interface, operation and message constructs in WSDL. WSDL-S does not impose any restrictions on the knowledge representation language to be used to define semantic models. WSML, OWL and UML are recommended as potential candidates [2].

WSDL-S provides a set of extension attributes and elements for associating semantic annotations to Web services. Through extension attribute **modelReference** one can specify associations between a WSDL entity and a concept in a semantic model. This extension can be used for annotating XML Schema complex types and

**Fig. 11.6** Mapping of the ontology representations to the SchemaGraph elements

Ontology representation	SchemaGraph representation
Class	Node
Property with basic datatypes as range (Attribute)	Node with edge joining it to the class with name „hasProperty”
Property with other class as range (Attribute)	Edge between the two class nodes
Instance	Node with edge joining it to the class with name „hasInstance”
Class - subclass relationship	Edge between class node to subclass node with name „hasSubClass”

**Example**

```

<daml:Class rdf:ID="WindEvent">
  <rdfs:comment> Superclass for all events
    dealing with wind</rdfs:comment>
  <rdfs:label>Wind event</rdfs:label>
  <rdfs:subClassOf rdf:resource="#WeatherEvent" />
</daml:Class>
<daml:Property rdf:ID="windDirection">
  <rdfs:label>Wind Direction</rdfs:label>
  <rdfs:domain rdf:resource="#WindEvent" />
  <rdfs:range rdf:resource=
    "http://www.w3.org/2000/10/XMLSchema#string"
  />
</daml:Property>
<daml:Property rdf:ID="windSpeed">
  <rdfs:label>Wind speed</rdfs:label>
  <rdfs:domain rdf:resource="#WindEvent" />
  <rdfs:range rdf:resource="#Speed" />
</daml:Property>

```

**SchemaGraph representation of the part of ontology\***

elements, WSDL operations and the extension elements **precondition** and **effect**, which are described below.

The **schemaMapping** extension attribute can be used for specifying mechanisms for handling structural differences between XML Schema elements and complex types, and their corresponding semantic model concepts. These annotations can then be used for what we refer to as lifting and lowering of execution data (i.e., transforming syntactic data into its semantic counterpart, and vice versa).

WSDL-S defines two new child elements for the operation element, namely **precondition** and **effect**. These elements allow defining the conditions that must hold before executing an operation and the effects the execution would have. This information is typically to be used for discovering suitable Web services. Finally, WSDL-S allows using the **category** extension attribute on the interface element in order to

---

```

<wsdl:description
  targetNamespace="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
  xmlns="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
  xmlns:wsdl="http://www.w3.org/ns/wsdl"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sawsdl="http://www.w3.org/ns/sawsdl">
  <wsdl:types>
    <xsschema targetNamespace="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
      elementFormDefault="qualified">
      <xss:element name="OrderRequest">
        sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#
          OrderRequest"
        sawsdl:loweringSchemaMapping="http://www.w3.org/2002/ws/sawsdl/spec/mapping/RDFOnt2Request.
          xsml">
      <xss:complexType>
        <xss:sequence>
          <xss:element name="customerNo" type="xs:integer" />
          ...
        </xss:sequence>
      </xss:complexType>
    </xsschema>
  </wsdl:types>
</wsdl:description>

```

---

**Listing 11.5** An example of a WSDL service description extended with the SAWSDL attributes

define categorization information for publishing Web services in registries such as UDDI.

More recently, the WSDL-S proposal has been superseded by SAWSDL [6] which is a W3C Recommendation. SAWSDL is a restricted and homogenized version of WSDL-S in which annotations such as preconditions and effects have not been explicitly contemplated since there is no current agreement about them in the Semantic Web Services community. It is worth noting, however, that SAWSDL does not preclude include this type of annotations as illustrated in the usage guide generated by the SAWSDL Working Group [1].

Besides discarding preconditions and effects, in SAWSDL the construct **Category** has been replaced by the more general **modelReference** extension attribute, which in SAWSDL can be used to annotate XML Schema complex type definitions, simple type definitions, element declarations, and attribute declarations, as well as WSDL interfaces, operations, and faults. Additionally, **schemaMapping** has been decomposed into two different extension attributes, namely **liftingSchemaMapping** and **loweringSchemaMapping**, so as to specifically identify the type of transformation.

Listing 11.5 gives an excerpt of a WSDL service description extended with the SAWSDL attributes which reference the mapping used to lower the instance of the **OrderRequest** element, and to its ontological annotation.

### 11.3.2 Semantics-Based Discovery of Web Services

The Universal Description, Discovery and Integration (UDDI) specification and the Universal Business Registry (UBR) are the main industrial efforts towards the automation of Web service discovery. The METEOR-S Web Services Discovery

Infrastructure (MWSDI) [28] attempts to enhance existing Web services discovery infrastructure by using semantics. MWSDI is a scalable infrastructure for the semantics-based publication and discovery of Web services.

MWSDI aims to provide unified access to a large number of third party registries. Thus, in order to provide a scalable and flexible infrastructure, it has been implemented using Peer-to-Peer (P2P) computing techniques. It is based on a four-layered architecture which includes a Data layer, a Communications layer, an Operator Services layer, and a Semantic Specification layer. The Data layer consists of the Web services registries and is based on UDDI. The Communications layer is the P2P infrastructure which is based on JXTA. The Operator Services layer provides the semantic discovery and publication of Web services. Finally, the Semantic Specification layer enhances the framework with semantics.

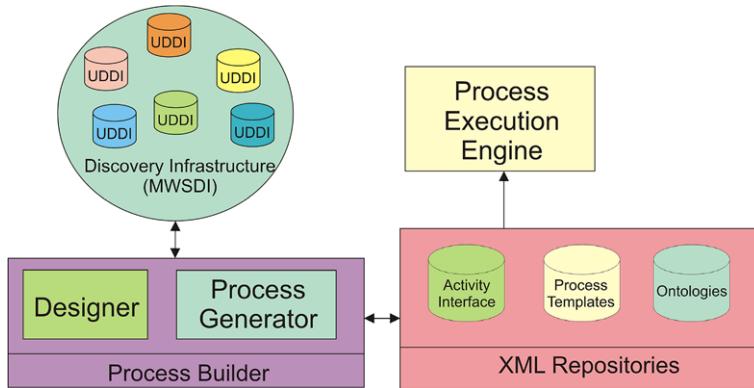
MWSDI uses semantics for two purposes. Firstly, it uses the so-called *Registries Ontology* which stores registries information, maintains relationships between domains within MWSDI and associates registries to them. This ontology stores mappings between registries and domains so that finding Web services for a specific domain can be directed to the appropriate registries. Additionally, the Registries Ontology captures relationships between registries so that searches can be made more selective on the basis of these relationships.

Secondly, MWSDI envisions including domain specific ontologies for registries, so that Web services can be annotated by mapping inputs and outputs to existing domain ontologies. The purpose of defining these mappings is to enable semantic discovery by allowing users to express their requirements as Service Templates which are expressed using concepts from the same ontology.

The semantic publication of services in MWSDI registries uses UDDI tModels for registering the domain ontologies and CategoryBags for categorizing WSDL entities according to one or more tModels. MWSDI provides both a manual and a semi-automatic mechanism for defining the mappings between WSDL elements and the concepts in the domain ontologies [28].

### ***11.3.3 Composition of Web Services***

Semantic Composition of Web services in METEOR-S is supported by the so-called METEOR-S Web Service Composition Framework (MWSCF) [23]. In a nutshell, the composition framework makes use of Semantic Process Templates which conceive processes as a set of semantically represented activities. Based on such templates, executable processes can be generated by binding the semantically defined activities to concrete Web services that conform to the activities' specification. As presented in Fig. 11.7 due to [23], the MWSCF is composed of four components: the process builder, the discovery infrastructure, XML repositories and the process execution engine. The process builder includes a graphical user interface for defining Semantic Process Templates, and a process generator. The process generator



**Fig. 11.7** METEOR-S composition framework

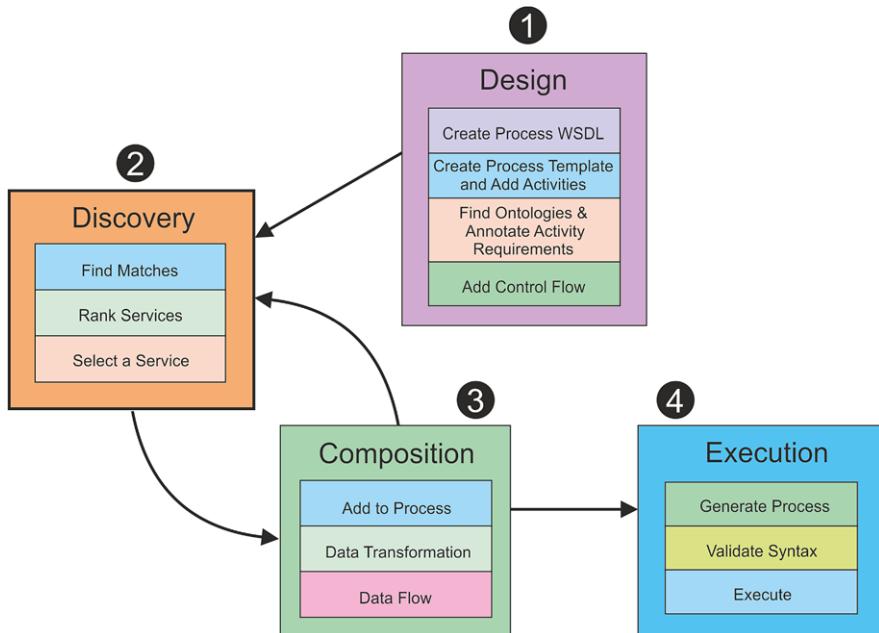
retrieves ontologies, activity interfaces and process templates from the XML repositories, and uses MWSDI for discovering suitable Web services, in order to transform the templates into executable processes. The executable process definitions can then be handed to the process execution engine for the actual execution of the Web service composition.

In MWSCF, Semantic Process Templates are basically a set of Activities connected by means of BPEL control flow constructs. Activities can be defined with a varying degree of flexibility by using a specific Web service implementation, a Web service interface or a Semantic activity template. Specific Web services implementations can be specified for static compositions. Web services interfaces can be applied to gain some flexibility allowing diverse implementations of the same interface to be interchangeable executed. Finally, Semantic activity templates provide a greater degree of flexibility by defining activities semantically in terms of their inputs, outputs and functional semantics, e.g., preconditions and effects.

The creation of an executable process is a multi-step, semi-automated process performed at design time, where the user is assisted in refining the template with concrete Web services and dataflow information (see Fig. 11.8 due to [23]). In order to do so, Web services that implement the specified service interfaces are retrieved from the XML Repository, and MWSDI is used for discovering suitable services when semantic activity templates have been specified. After all the activities have been replaced by concrete Web service instances, the user can map service outputs to other service inputs in order to define the process dataflow. Based on the specified dataflow, the process generator creates the executable representation, which is a BPEL4WS process that can be executed in any BPEL execution engine.

#### 11.3.4 METEOR-S Summary

Table 11.2 illustrates METEOR-S' compliance to the criteria introduced in Sect. 11.1.



**Fig. 11.8** Composition steps supported by MWSCF

**Table 11.2** METEOR-S summary

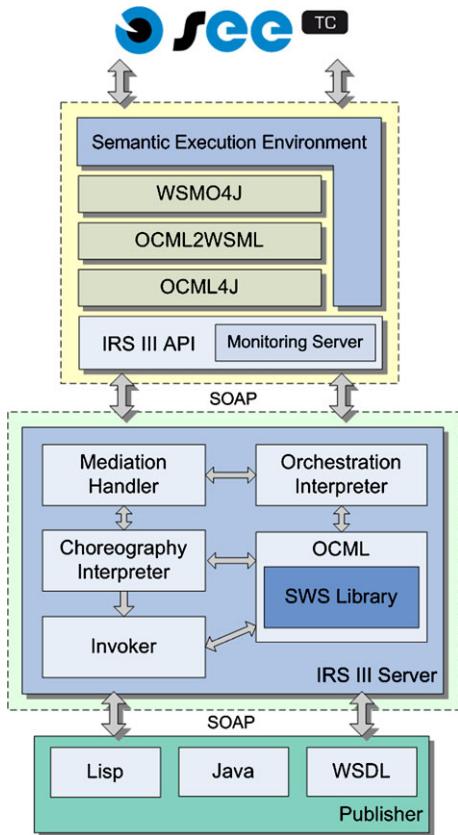
Compliance	Criteria	Comments
✓	Web compliance	Supports URI, namespaces and XML
✓	Ontology-based	Based on DAML and RDF-S
✓	Strict decoupling	Ontologies and services are specified independently
✓	Centrality of mediation	Supported but not as a central concept
	Ontological role separation	Doesn't have the notion of user desires
	Description vs. implementation	Strongly coupled with existing Web service standards
	Execution semantics	Doesn't have the notion of execution semantics
✓	Services vs. Web Services	Designed as a means to describe Web services

## 11.4 IRS-III

The Internet Reasoning Service (IRS) project<sup>7</sup> [4] carried out at the Knowledge Media Institute at the Open University has the overall aim of supporting the automated or semi-automated construction of semantically enhanced systems over the

<sup>7</sup><http://kmi.open.ac.uk/projects/irs>.

**Fig. 11.9** IRS-III architecture



Internet. The latest version of IRS, called IRS-III [4], has incorporated and extended the WSMO ontology so that the implemented infrastructure allows the description, publication and execution of Semantic Web Services.

IRS-III is based on a distributed architecture composed of the IRS-III server, the publishing platforms, and various clients, as shown in Fig. 11.9. The server embeds an OCML [15] interpreter, which includes all the reasoning machinery for manipulating ontologies definitions and invoking Web services in order to satisfy client requests expressed as goals. In order to support the invocation of Semantic Web Services, the reasoning engine is complemented by the Orchestration Interpreter, the Choreography Interpreter, the Mediation Handler and the Invoker.

The server provides access to client applications for creating and editing WSMO descriptions of goals, Web services and mediators and supports the achievement of stored goals by invoking previously deployed Web services. Clients communicate with the server by means of a stack of Java libraries, encapsulated in the yellow box in Fig. 11.9. The libraries provide programmatic access to the server functionality and perform the appropriate transformations between WSMO and OCML code. Furthermore, an implementation of the current version of the Semantic Execution Environment API is also available, so that compatible applications, such as the WSMO

Studio or the Web Service Modeling Toolkit (WSMT), can make use of IRS in an homogeneous way.

Through the publishing platforms service providers can attach semantic descriptions to their Web services; moreover, handlers allow for the invocation of services expressed in various languages and formats—currently, IRS supports WSDL, Lisp code, Java code, and Web applications. When a Web service is published in IRS-III, the information about the publishing platform is also associated with the Web service description in order for the service to be invoked. The Java and WSDL publishing platforms are delivered as Java Web applications, whereas both the Lisp and HTTP publishers have been developed in Lisp.

IRS-III has its own internal representation of the WSMO meta-model in OCML and introduces some specific modifications and extensions which we review next. A first aspect that differs from WSMO is the fact that in IRS-III goals and Web services have an explicit declaration of input and output roles which includes the name and the semantic type. This approach contrasts with WSMO where this information is defined in the **State Signature** as part of the **Choreography** specification.

A second notable feature, closely related to the previous one, comes from the fact that Web services can inherit the input and output roles of goals. As a consequence, roles declaration in Web services are not mandatory although they can be used to extend and refine existing definitions of roles. This feature is enabled as inputs and output roles are defined as part of goals and Web services, as opposed to the approach prescribed in WSMO, and supports further reuse of existing definitions.

In addition to the specific extensions to WSMO included in IRS-III, further distinguishing characteristics arise from the underlying modeling support they use. WSMO is defined using the Meta-Object Facility (MOF) [18] which specifies a metadata architecture based on fours layers. In WSML, the meta-model is separate from the definitions made in the language, which means that it is a separate meta-theory. Domain and application specific definitions span the information and model layers where Web service, ontology, goal and mediator are basically language constructs. Consequently, when modeling any specific domain in WSML, there is no possibility to treat a Web service, ontology, goal or mediator as concepts which could be subsumed or instantiated. This supports the execution of Web services, since existing WSML-based software has embedded knowledge in order to deal with these concepts. However, this separation of layers also comes with some important drawbacks.

A first inconvenience is, as pointed out in [25], the fact that it is important to distinguish between prototypical goals, which represent the specific static definitions, and the actual instantiations of these goals containing invocation data. The authors refer to these as goal templates and goal instances, respectively; the introduction of goal templates proposed in [25] is a way to partially overcome part the limitations introduced by the MOF-based definition of WSMO.

Secondly, since at the model-level a goal is merely a construct, one cannot benefit from subsumption reasoning. Supporting the creation of goal hierarchies would promote and support reusing definitions. Goal achievement could be enhanced by supporting the dynamic selection of refined Goals taking non-functional properties

into account, for example. Similarly, facing situations where a specific goal cannot be achieved, systems could decide to meet at least some of the client requirements by achieving a more generic goal. So far, this is supported in WSMO by means of ggMediators, but using these requires additional ad-hoc reasoning facilities where pre-existing reasoning machinery could be applied (in this case, most prominently subsumption). In fact, ggMediators do not specify whether the relationship between goals is one of equivalence or refinement.

In IRS-III, these inconveniences are overcome in OCML by defining both the meta-model (WSMO) and the user models in the same language, i.e., both the WSMO meta-concept of goal and specific user goals—the previously mentioned templates—are both OCML classes. In this way, there is support for the instantiation of goals, as well as for the definition of relationships between goals.

#### ***11.4.1 Discovery, Selection and Mediation***

In IRS, discovery is based on the existing ontological connection between goals and services expressed via wgMediators. Such connections can be dynamically computed when a new goal is submitted, but the assumption is that IRS enacts a stable broker between pre-existing goals and external deployed Web services. Having found a mediator connecting a goal to some candidate services, the IRS selects the most suitable on the basis of the preconditions and assumptions defined in their capability.

In order to support this model of discovery and selection, IRS has some extensions and differences in interpretation of the WSMO model from that applied by WSMX. Firstly, goals are implicitly assumed to be atomic and explicitly parameterized in inputs, which are therefore all required before execution. For this reason, there is no need for a behavioral part to a goal's choreography in the IRS. Secondly, a WSMO web service is viewed as the application of an external deployed service to a particular atomic task. While this requires only the communication of inputs from the user, and return of an output, IRS as a broker may be required to carry out several interactions with the deployed service to reach this output. The prototype of this interaction is stored in the choreography of the service and is called a ‘client choreography’. This is different from the model of goal choreographies and service choreographies which need dynamic process mediation in WSMX. Process mediation may be used to form client choreographies when service descriptions are added to IRS to make deployed services available for new tasks but, like dynamic discovery, this is outside the scope of the current tool.

Data mediation in IRS is encoded in the wgMediator, which links a goal and Web service and may take one of three forms, each allowing the inputs of the goal to be turned into the inputs required by the Web service, and the outputs provided by the Web service to be turned into the output expected by the goal. Firstly, the IRS model of ooMediator is extended to allow OCML rules mapping between instances of concepts within the ontologies specified. In this way, wgMediators can

be linked to ooMediators including a declarative specification of the data mediation. This is similar to the approach of the abstract mapping language engine described for WSMX, but explicitly attached to the mediator description. The other means to provide data mediation in IRS are via the service abstraction and the ‘usesMediationService’ attribute of WSMO’s wgMediator. This can be used either to attach a service which can convert between goal and service inputs, or service and goal outputs, as described above, or to attach a goal which describes the requirements on such a mediation service. In the latter case, discovery and selection can be applied recursively so that the link between a goal and candidate service may depend on the existence of such a mediation service.

The capability-driven selection mechanism provided by IRS-III has recently been extended to encompass trust-based requirements and guarantees [7]. A Web Services Trust Ontology (WSTO) has been built with which WSMO-based descriptions can be extended and a classification-based algorithm is used to discount and rank candidate services according to the ‘trustworthiness’ of their guarantees on certain non-functional aspects, such as quality of service.

### ***11.4.2 Communication***

The distributed architecture of IRS is composed of three main blocks: the server, the stack of libraries, and the publishing platforms. Interactions between components belonging to the same block take place in the same execution environment, and are based on direct invocation.<sup>8</sup> As a consequence, supporting this does not require any specific communication infrastructure.

Describing the communication mechanisms implemented in the IRS concerns three interactions. The first one corresponds to the communication between clients and IRS as supported by the libraries stack. This stack of libraries provides access to the main IRS functionality such as the manipulation of knowledge models and the invocation of goals. The communication between Java libraries and IRS is centralized in the IRS-III API, which acts as the unique access point to the IRS server as a set of methods that manipulate OCML definitions (see Fig. 11.9). Additionally, it includes a server that enables runtime monitoring of the IRS server tracking all internal processes. Communication between the IRS-III API and the server, both inbound and outbound, is based on SOAP. Each client request/reply interaction is supported by SOAP and a set of pre-defined XML messages formats. Similarly, every tracked activity generates events expressed in terms of an Events Ontology; these events are then serialized and forwarded to the monitoring server via SOAP.

The second communication affects the interaction between the IRS server and the publishing platforms. Four different types of remote services—Lisp code, Java code, WSDL Web services and Web applications—can be invoked via HTTP GET

---

<sup>8</sup>Note that there is no interaction between publishers.

requests. The first three types are handled by the appropriate component of the publishing platform, whereas Web applications are directly invoked by the IRS server as it has built-in HTTP support. The remaining publishing platforms—Java, WSDL and Lisp—are external components that communicate with the server via SOAP. An important aspect of this communication concerns the transition between the semantic world and the syntactic world, usually referred to as lifting and lowering in the Semantic Web Services community. This functionality is provided in IRS by the Invoker. The current implementation of the IRS includes lifting and lowering machinery for dealing with XML-based results. The relevant information for performing the appropriate transformations is stored as a set of Lisp macros which identify the data interchanged, the relevant Semantic Web Service and the corresponding XML element by means of an XPath expression. At runtime the Invoker retrieves information on how to lift or lower the data and applies it right after receiving the results or right before sending the invocation data, respectively. This way, the data sent to the publisher is always in its syntactic representation whereas internally the IRS manipulates semantic data.

The final communication takes place between the different publishing platforms and the actual Semantic Web Service being invoked, and it is therefore platform specific. For instance, Lisp code is executed in the Lisp publishing platform, WSDL services are invoked using the protocol specified, Web applications are invoked through HTTP, and Java services are provided by a Java-based Web application deployed on a Tomcat Web server so as to be invocable.

### ***11.4.3 Choreography and Orchestration***

As discussed in Sect. 11.4.1, choreography in the IRS model consists of the attachment of client choreographies to Web services. The choreography engine is applied to execute a client choreography when a Web service is selected in order to meet a Goal. Whereas the separation of client and service choreographies, together with dynamic mediation between them, in WSMX allows the parties and communication roles in these choreographies to be implicit, a client choreography implicitly includes communications via the broker and a more sophisticated model of communication is required. The choreography model of IRS is detailed in [5].

The IRS offers two models for orchestrations attached to Web service descriptions. In both cases, these models are executable and derive the required atomic behavior from a complex interaction between some number of other artifacts.

In the simplest case, described in [4], control flow primitives are used to link together goals to form a complex behavior. ggMediators are used to form the data flow between these goals; when a ggMediator connects one goal to another, the input will be taken from the source goal and communicated to whichever input of the target goal matches the mediator's output. This may involve data mediation as described for wgMediators above. One consequence of this model is that a given goal may only be instantiated once per orchestration or the data flow is ambiguous.

**Table 11.3** Overall comparison of the SWS frameworks

	WSMO	OWL-S	METEOR-S
Web compliance	✓	✓	✓
Ontology-based	✓	✓	✓
Strict decoupling	✓	✓	✓
Centrality of mediation	✓		
Ontological role separation	✓		
Description vs. implementation	✓	✓	
Execution semantics	✓	✓	
Services vs. Web services	✓	✓	✓

It is also not permitted to directly specify another Web service to be applied during the orchestration.

Both of the stated deficiencies of the simple orchestration model are addressed in the Cashew orchestration model [17]. Here both goals and Web services can be specified in prototypical ‘performances’ which are the units of behavior composed in a hierarchical control flow decomposition. The data flow is unambiguously specified by a new type of mediator between performances, named a ppMediator. As described for wg- and ggMediators, the usual attributes may be used to specify data mediation. The second advantage of the Cashew model is the ability to represent this model in UML Activity Diagrams, and work has been carried out on composition in this model with execution via the Cashew orchestration implementation [17].

## 11.5 Summary

In this chapter, we introduced additional frameworks which aim to fulfill the promise of the Semantic Web Services vision. The first part was devoted to OWL-S where we described its basic building blocks (i.e., service profile, service model and service grounding) used to semantically describe and ground a Web service. We also gave an overview of the OWL-S tool support. The second part of the chapter described METEOR-S as a pragmatic SWS approach founded on heavily used and standardized specifications such as WSDL. As opposed to the other SWS frameworks presented in this book, the METEOR-S promotes a bottom-up approach to semantically enabling Web services technology, which relies on SAWSDL. Again, we paid some attention to the tools used to support semantic annotation, discovery and composition tasks in the METEOR-S environment. Towards the end of the chapter, we presented IRS-III as an environment which supports and further extends WSMO with a strong emphasis on the automated and semi-automated development of semantically enhanced systems.

The SWS frameworks have been assessed with regards to the criteria introduced at the beginning of the chapter. Table 11.3 summarizes the results of this assessment.

## 11.6 Exercises

**Exercise 1** Given the scenario description of the exercise in Chap. 9, try to formalize a set of ontologies and Web service descriptions following OWL-S. Discuss similarities and differences to a WSMO-based solution.

## References

1. Akkiraju, R., Sapkota, B.: Semantic annotations for WSDL and XML schema—usage guide. Working Group Note (2007). <http://www.w3.org/TR/sawsdl-guide/>
2. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T., Sheth, A., Verma, K.: Web Service Semantics—WSDL-S. W3C Member Submission (2005). <http://www.w3.org/Submission/WSDL-S/>
3. Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., Tabet, S.: Semantic Web services framework (swsf) overview. W3C Member Submission 9 September 2005. <http://www.w3.org/Submission/SWSF>
4. Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Norton, B., Tanasescu, V., Pedrinaci, C.: IRS-III: a broker for Semantic Web services based applications. In: Proceedings of the 5th International Semantic Web Conference (ISWC) (2006)
5. Domingue, J., Galizia, S., Cabral, L.: Choreography in IRS III: coping with heterogeneous interaction patterns in Web services. In: 4th International Semantic Web Conference (ISWC2005), 6–10 November 2005 (2005)
6. Farrell, J., Lausen, H.: Semantic annotations for WSDL and XML schema. W3C Candidate Recommendation 26 January 2007. <http://www.w3.org/TR/sawsdl/>
7. Galizia, S., Gugliotta, A., Domingue, J.: A trust-based methodology for Web service selection. In: Proceedings of 1st IEEE Conference on Semantic Computing (2007)
8. Grüninger, M., Menzel, C.: The process specification language (psl) theory and applications. The AI Magazine **24**(3), 63–74 (2003)
9. Haarslev, V., Möller, R.: Racer: a core inference engine for the Semantic Web. In: Proceedings of 2nd International Workshop on Evaluation of Ontology-Based Tools (EON 2003), pp. 27–36 (2003)
10. Klusch, M., Renner, K.-U.: Fast dynamic re-planning of composite OWL-S services. In: Proceedings of 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, pp. 134–137. IEEE Computer Society, Los Alamitos (2006)
11. Klusch, M., Fries, B., Sycara, K.: Automated Semantic Web service discovery with OWL-S-MX. In: Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). ACM Press, New York (2006)
12. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic markup for Web services. W3C, Member Submission (2004). <http://www.w3.org/Submission/OWL-S>
13. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language. W3C, Recommendation (2004). <http://www.w3.org/TR/owl-features>
14. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. In: Proceedings of 3rd International Semantic Web Conference (IWSC 2004). LNCS, vol. 3298, pp. 549–563 (2004)
15. Motta, E.: Reusable Components for Knowledge Modelling. Case Studies in Parametric Design Problem Solving. Frontiers in Artificial Intelligence and Applications, vol. 53. IOS Press, Amsterdam (1999)

16. Norton, B.: Experiences with OWL-S, directions for service composition. In: Proceedings of OWL: Experiences and Directions Workshop (OWLED-2005) (2005)
17. Norton, B., Pedrinaci, C., Henocque, L., Kleiner, M.: 3-level behaviour models for Semantic Web services. International Transactions on Systems Science and Applications (2007, to appear)
18. Object Management Group Inc. (OMG): Meta Object Facility (MOF) Specification V1.4. (2002). <http://www.omg.org/technology/documents/formal/mof.htm>
19. Paolucci, M., Ankolekar, A., Srinivasan, N., Sycara, K.: The DAML-S virtual machine. In: Proceedings of 2nd International Semantic Web Conference (ISWC 2003). LNCS, vol. 2870, pp. 290–305. Springer, Berlin (2003)
20. Paolucci, M., Srinivasan, N., Sycara, K.: Expressing WSMO mediators in OWL-S. In: Proceedings of Semantic Web Services: Preparing to Meet the World of Business Applications (2004)
21. Patil, A.A., Oundhakar, S.A., Sheth, A.P., Verma, K.: METEOR-S Web service annotation framework. In: WWW'04: Proceedings of the 13th International Conference on World Wide Web, pp. 553–562. ACM Press, New York (2004). doi:[10.1145/988672.988747](https://doi.org/10.1145/988672.988747)
22. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. Journal of Web Semantics **5**(2), 51–53 (2004)
23. Sivashanmugam, K., Miller, J.A., Sheth, A.P., Verma, K.: Framework for Semantic Web process composition. International Journal of Electronic Commerce **9**(2), 71–106 (2005)
24. Srinivasan, N., Paolucci, M., Sycara, K.: Adding OWL-S to UDDI: implementation and throughput. In: Proceedings of 1st International Conference on Semantic Web Services and Web Process Composition (SWSWPC 2004) (2004)
25. Stollberg, M., Norton, B.: A refined goal model for Semantic Web services. In: The Second International Conference on Internet and Web Applications and Services (ICIW 2007) (2007). ISBN: 0-7695-2844-9
26. Szomszor, M., Payne, T., Moreau, L.: Automated syntactic mediation for web services integration. In: Proceedings of IEEE International Conference on Web Services (ICWS 2006) (2006)
27. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: system description. In: Proceedings of 3rd International Joint Conference on Automated Reasoning. LNCS, vol. 4130, pp. 292–297. Springer, Berlin (2006)
28. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J.: METEOR-S WSDI: a scalable P2P infrastructure of registries for semantic publication and discovery of web services. International Journal of Information Technology and Management **6**(1), 17–39 (2005). doi:[10.1007/s10799-004-7773-4](https://doi.org/10.1007/s10799-004-7773-4)
29. Wu, D., Parsia, B., Sirin, E., Hendler, J., Nau, D.: Automating DAML-S web services composition using SHOP2. In: Proceedings of 2nd International Semantic Web Conference (IWSC 2003). LNCS, vol. 2870. Springer, Berlin (2003)

# Chapter 12

## Lightweight Semantic Web Service Descriptions

**Abstract** The Web standardization consortium W3C has developed a lightweight bottom-up specification, Semantic Annotation for WSDL (SAWSDL), for adding semantic annotations to WSDL service descriptions. In this chapter, we describe SAWSDL, and then we present WSMO-Lite and MicroWSMO, two related lightweight approaches to Semantic Web Service description, evolved from the Web Service Modeling Ontology (WSMO) framework. WSMO-Lite defines an ontology for service semantics, used directly in SAWSDL to annotate WSDL-based services. MicroWSMO and its basis, hRESTS, are microformats that supplement WSDL and SAWSDL for unstructured HTML descriptions of services, providing WSMO-Lite support for the growing numbers of RESTful services.

### 12.1 Motivation

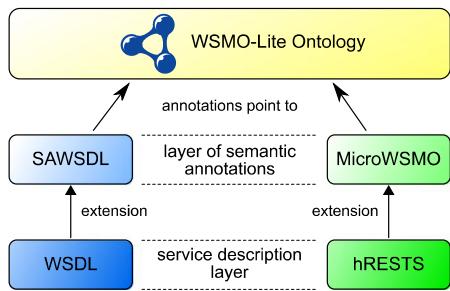
As the first step towards standardizing semantic descriptions for Web services, the Web standardization consortium W3C has developed a lightweight bottom-up specification for annotating WSDL service descriptions. The specification is called Semantic Annotations for WSDL and XML Schema (SAWSDL, [3, 6]).

Technologies such as WSMO (cf. Chap. 7) and OWL-S (cf. Chap. 11) embody the so-called *top-down* approach to semantic Web service modeling: a service engineer describes the semantics of the service independently of the actual realization in the service implementation; then the semantics is *grounded* in an underlying description such as WSDL. This approach separates the semantics from the underlying WSDL, on the principle that the semantics should not be influenced by implementation details.

In contrast, SAWSDL embodies a *bottom-up* modeling approach where the WSDL layer that describes the service on the implementation level forms the basis for adding semantics. With a bottom-up approach, semantics are added to WSDL descriptions in a modular fashion, as needed in a concrete deployment. SAWSDL by itself, however, does not specify any actual types of semantics; it only gives us hooks for attaching semantics to WSDL descriptions. SAWSDL is the ground stone on which semantic Web services frameworks should be built, and a catalyst for building them. We further describe SAWSDL in Sect. 12.2.1.

The standardization of SAWSDL led to the development of WSMO-Lite [4], an emerging technology that realizes the WSMO Semantic Web Services approach in

**Fig. 12.1** WSMO-Lite and MicroWSMO layer cake



SAWSDL and RDF. WSMO-Lite identifies four types of semantics and defines a lightweight RDF Schema ontology for expressing them. Built on a simplified service model, WSMO-Lite applies directly to WSDL services, described in Chap. 4. We present the service semantics defined by WSMO-Lite in Sect. 12.2.2, and in Sect. 12.2.3, we show the specifics of how WSMO-Lite applies to WSDL descriptions.

The WSMO-Lite service model also maps naturally to RESTful services, described in Chap. 5. However, at the time of writing, there is no widely accepted standard for machine-readable descriptions of RESTful services, which are instead simply documented in HTML Web pages. Faced with HTML pages, we employ two *microformats* (an “adaptation of semantic XHTML that makes it easier to publish, index, and extract semi-structured information” [2]) to make the key pieces of the human-oriented service documentation machine-readable (hRESTS), and to add semantic annotations (MicroWSMO). Both microformats, defined in [5], are described in Sect. 12.2.4.

## 12.2 Technical Solution

Figure 12.1 shows the relative positioning of WSMO-Lite, WSDL with SAWSDL, and hRESTS with MicroWSMO. WSDL is the standard description language for Web services; SAWSDL extends it with annotations that can point to semantics, such as WSMO-Lite. hRESTS is analogous to WSDL—it describes the basic structure of RESTful services. MicroWSMO is then a direct realization of SAWSDL annotations over hRESTS. Both SAWSDL and MicroWSMO are annotation mechanisms enabling pointers to semantics; WSMO-Lite specifies a simple vocabulary for four types of semantics that fit in SAWSDL/MicroWSMO annotations.

Below, we first describe SAWSDL, the simple specification that is the cornerstone of lightweight semantic Web services frameworks. Then we proceed to detail WSMO-Lite, with its simple service model and four types of service semantics. Afterwards, we show exactly how WSMO-Lite semantics are applied in WSDL and SAWSDL. Finally, we extend the reach of WSMO-Lite also to RESTful services, using the microformats hRESTS and MicroWSMO.

## 12.2.1 SAWSDL

SAWSLD is a set of extensions for WSDL,<sup>1</sup> the standard description format for Web services. To briefly recap from Chap. 4, WSDL uses XML as a common flexible data exchange format, and applies XML Schema for data typing. WSDL describes a Web service in three levels:

- Reusable abstract *interface* defines a set of *operations*, each representing a simple exchange of messages described with XML Schema element declarations.
- *Binding* describes the on-the-wire message serialization, following the structure of an interface and filling in the necessary networking details (for instance, for SOAP or HTTP).
- *Service* represents a single physical Web service which implements a single interface; the Web service can be accessible at multiple network *endpoints*.

The purpose of WSDL is to describe the Web service on a *syntactic level*: WSDL specifies what the messages *look like* rather than what the messages or operations *mean*. SAWSLD defines an extension layer over WSDL that allows the *semantics* to be specified on the various WSDL components. SAWSLD defines extension attributes that can be applied to elements both in WSDL and in XML Schema in order to annotate WSDL interfaces, operations and their input and output messages.

The SAWSLD extensions are of two forms: *model references* point to semantic concepts, and *schema mappings* specify data transformations between the XML data structure of messages and the associated semantic model.

### 12.2.1.1 Model Reference

A model reference is an extension attribute, `sawsdl:modelReference`, that can be applied to any WSDL or XML Schema element, in order to point to one or more semantic concepts. The value is a set of URIs, each one identifying some piece of semantics.

Model reference has the very generic purpose of referring to semantic concepts. As such, it serves as a *hook* where semantics can be attached. As illustrated in the examples presented later in this chapter, model reference can be used to describe the meaning of data or to specify the function of a Web service operation.

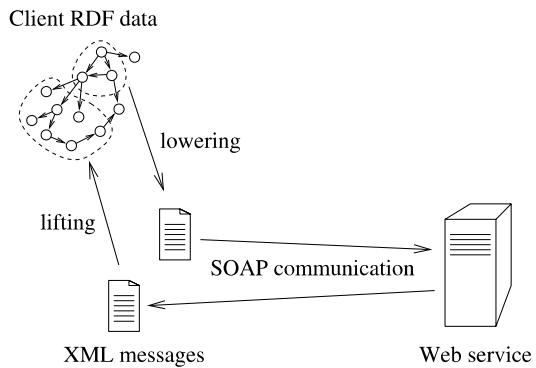
### 12.2.1.2 Schema Mappings

Schema mappings are represented by two attributes, `sawsdl:liftingSchemaMapping` and `sawsdl:loweringSchemaMapping`. Lifting mappings transform XML data from a Web service message into a semantic model (for instance,

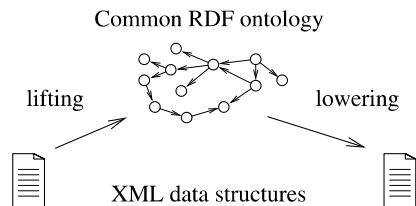
---

<sup>1</sup>Web Services Description Language, <http://www.w3.org/TR/wsdl20/>.

**Fig. 12.2** RDF data lifting and lowering for WS communication



**Fig. 12.3** RDF data lifting and lowering for data mediation



into RDF data that follows some specific ontology), and lowering mappings transform data from a semantic model into an XML message.

Lifting and lowering transformations are required for communication with the Web service from a semantic client: the client software will *lower* some of its semantic data into a request message and send it to the Web service, and when the response message is received, it can be *lifted* for semantic processing. This is illustrated in Fig. 12.2.

Lifting and lowering annotations can also be used for XML data mediation through a shared ontology, as shown in Fig. 12.3. The data in one XML format can be *lifted* to data in the shared ontology and then *lowered* to another XML format, using the lifting annotation from the schema for the first format, and the lowering annotation from the second schema.

### 12.2.1.3 Annotation Propagation

In an XML Schema, the content of an element is described by a type definition, and the name of the element is added as an element declaration. SAWSDL model reference and schema mapping annotations can be both on types and on elements; in fact, the annotations of a type apply also to all elements of that type.

In particular, for a given pair of an element declaration and its type definition, the model references from the type are merged with the model references of the element, and all of them apply to the element. Schema mappings, on the other hand, are only propagated from the type if the element does not declare any schema mappings of its own. This allows a type to provide generic schema mappings, and the

**Table 12.1** SAWSDL syntax summary

Name	Description
<b>modelReference</b> (XML attribute)	a list of references to concepts in some semantic models
<b>liftingSchemaMapping</b> (XML attribute)	a list of pointers to alternative data lifting transformations
<b>loweringSchemaMapping</b> (XML attribute)	a list of pointers to alternative data lowering transformations
<b>attrExtensions</b> (XML element)	used for attaching attribute extensions where only element extensibility is allowed

element to specify more concrete mappings appropriate for the specific use of the type. Table 12.1 provides a summary of the syntactical constructs of SAWSDL.

#### 12.2.1.4 WSDL 1.1 Support

While SAWSDL is primarily built for WSDL 2.0, the older and more prevalent version WSDL 1.1 is also supported. Essentially, both model references and schema mappings apply in the same places in both versions of WSDL. However, the XML Schema for WSDL 1.1 only allows element extensions on operations, so a WSDL 1.1 document with the SAWSDL `sawsdl:reference` attribute on an operation would not be valid. To overcome this obstacle, SAWSDL defines an element called `sawsdl:attrExtensions` that is intended to carry extension attributes in places where only element extensibility is allowed. Instead of putting the model reference directly on the `wsdl11:operation` element, it is put on the `sawsdl:attrExtensions` element, which is then inserted into the `wsdl11:operation` element.

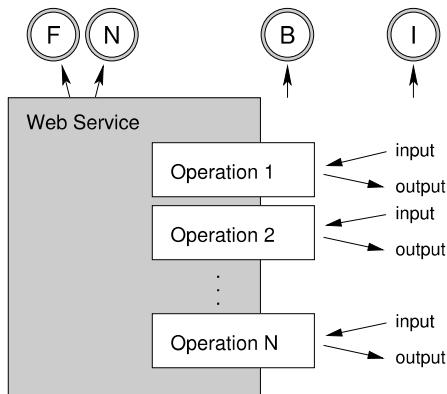
WSDL is a well-known and accepted language for describing Web service interfaces, and virtually all Web service enabled systems use it to advertise and use Web services. SAWSDL is a non-intrusive, simple extension of WSDL that enables semantic annotations in a way that does not invalidate any existing uses of WSDL. This makes it perfectly suitable as the basis for an ontology for describing Semantic Web Services, working towards automation in service-oriented systems.

#### 12.2.2 WSMO-Lite Service Semantics

WSMO-Lite is a lightweight technology for semantic description of Web services. It distinguishes between the following four types of service semantics:

- **Information model** defines the semantics of input, output and fault messages.
- **Functional semantics** defines service functionality, that is, what a service can offer to its clients when it is invoked.

**Fig. 12.4** Web service description model with attached semantics



- **Non-functional semantics** defines any incidental details specific to the implementation or running environment of a service, such as its price, location or quality of service.
- **Behavioral semantics** specifies the protocol (ordering of operations) that a client needs to follow when consuming a service.

These semantics apply to service descriptions, as illustrated in Fig. 12.4: functional and non-functional semantics belong to the Web service as a whole, behavioral semantics belong to the operations, and the information model describes the input and output messages. With SAWSDL, the Web service description can contain pointers to the semantics as appropriate.

The service model shown in Fig. 12.4 is very similar to the model of WSDL, but it also applies to RESTful Web services (as shown in Chap. 5). It can be seen as the minimal service model: a *Web service* provides several *operations*, each of which can have *input* and *output messages*, along with possible *input* and *output faults*.

Figure 12.5 lists the WSMO-Lite ontology. The ontology consists of 3 main blocks: a *service model* that unifies the views of WSDL-based and RESTful Web services, *SAWSLD annotation properties* for attaching semantics, and finally, *classes* for representing those semantics.

Note that instances of the service model are not expected to be authored directly; instead, the underlying technical descriptions (WSDL, hRESTS) are parsed in terms of this ontology, with SAWSDL pointers to instances of the service semantics classes in the third block.

The service model is rooted in the class `wsl:Service`. In contrast to WSDL, this service model does not separate the service from its interface, as such separation does not need to be kept to support SWS automation. A service is a collection of operations (class `wsl:Operation`). Operations may have input and output messages, plus possible fault messages (all in the class `wsl:Message`). Input messages (and faults<sup>2</sup>) are those that are sent by a client to a service, and output messages (and faults) are those sent from the service to the client.

---

<sup>2</sup>Input faults are uncommon but possible in some WSDL message exchange patterns.

```

1 # namespace declarations (this line is a comment)
2 @prefix owl: <http://www.w3.org/2002/07/owl#> .
3 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
5 @prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .
6 @prefix wsl: <http://www.wsmo.org/ns/wsmo-lite#> .
7
8 # WSMO-Lite service model classes and properties
9 # this model is a simplified WSDL-based view of Web services
10 wsl:Service a rdfs:Class .
11 wsl:hasOperation a rdf:Property ;
12   rdfs:domain wsl:Service ;
13   rdfs:range wsl:Operation .
14 wsl:Operation a rdfs:Class .
15 wsl:hasInputMessage a rdf:Property ;
16   rdfs:domain wsl:Operation ;
17   rdfs:range wsl:Message .
18 wsl:hasOutputMessage a rdf:Property ;
19   rdfs:domain wsl:Operation ;
20   rdfs:range wsl:Message .
21 wsl:hasInputFault a rdf:Property ;
22   rdfs:domain wsl:Operation ;
23   rdfs:range wsl:Message .
24 wsl:hasOutputFault a rdf:Property ;
25   rdfs:domain wsl:Operation ;
26   rdfs:range wsl:Message .
27 wsl:Message a rdfs:Class .
28
29 # SAWSDL properties (included here for completeness)
30 sawsdl:modelReference a rdf:Property .
31 sawsdl:liftingSchemaMapping a rdf:Property .
32 sawsdl:loweringSchemaMapping a rdf:Property .
33
34 # WSMO-Lite service semantics classes
35 wsl:Ontology a rdfs:Class ;
36   rdfs:subClassOf owl:Ontology .
37 wsl:FunctionalClassificationRoot rdfs:subClassOf rdfs:Class .
38 wsl:NonFunctionalParameter a rdfs:Class .
39 wsl:Condition a rdfs:Class .
40 wsl:Effect rdfs:Class .

```

**Fig. 12.5** Service Ontology, Captured in Notation 3. (A brief intro to the Notation 3 RDF syntax: comment lines start with “#”, URIs are denoted as namespace-qualified names “a”, is short for `rdf:type`, triples end with a period, and triples with the same subject can be grouped, having the next property-object pair after a semicolon instead of a period. More at <http://www.w3.org/DesignIssues/Notation3.html>)

To link the components of the service model with the concrete description of the functional, non-functional, behavioral and information semantics, as illustrated in Fig. 12.4, we adopt the SAWSDL properties `sawsdl:modelReference`, `sawsdl:liftingSchemaMapping` and `sawsdl:loweringSchemaMapping`, defined by the SAWSDL RDF mapping in [6].

A model reference can be used on any component in the service model to point to the semantics of that component. In particular, a model reference on a service can point to a description of the service’s functional and non-functional semantics; a model reference on an operation points to the operation’s part of the behavioral semantics description; and a model reference on a message points to the message’s counterpart(s) in the service’s information model. The lifting and lowering schema

mapping properties complement the model references on messages, to point to the appropriate data transformations.

A single component can have multiple model reference values, which all apply together; for example, a service can have a number of non-functional properties together with a pointer to its functionality description, or a message can be annotated with multiple ontology classes, indicating that the message carries instances of all the classes.

In the remainder of this subsection, we describe how the four types of service semantics are represented in the WSMO-Lite service ontology. In the example listings, we use the following namespace prefixes:

```

1 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
2 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3 @prefix wsl: <http://www.wsmo.org/ns/wsmo-lite#> .
4 @prefix ex: <http://example.org/onto#> .
5 @prefix xs: <http://www.w3.org/2001/XMLSchema#> .
6 @prefix wsml: <http://www.wsmo.org/wsml/wsml-syntax#> .

```

**Information semantics** are represented using domain ontologies. An ontology intended for information semantics may be marked as `wsl:Ontology` (as shown below on line 2), so that authoring tools can focus annotation suggestions more easily. The following listing is an example domain ontology for a video-on-demand service:

```

1 # ontology example
2 <> a wsl:Ontology.
3
4 ex:Customer a rdfs:Class .
5 ex:hasService a rdf:Property ;
6   rdfs:domain ex:Customer ;
7   rdfs:range ex:Service .
8 ex:Service a rdfs:Class .
9 ex:hasConnection a rdf:Property ;
10  rdfs:domain ex:Customer ;
11  rdfs:range ex:NetworkConnection .
12 ex:NetworkConnection a rdfs:Class .
13 ex:providesBandwidth a rdf:Property ;
14  rdfs:domain ex:NetworkConnection ;
15  rdfs:range xs:integer .
16 ex:VideoOnDemandService rdfs:subClassOf ex:Service .

```

**Functional semantics** are represented with *functionality classifications* and/or *preconditions* and *effects*.

Functionality classifications define the service's functionality using some classification taxonomy (i.e., a hierarchy of *categories*, such as the eCl@ss taxonomy<sup>3</sup>), and the class `wsl:FunctionalClassificationRoot` marks a class that is a root of a classification. The classification also includes all the RDFS subclasses of the root class. Functionality classifications are coarse-grained means for describing service functionality, due to the cost of creating shared taxonomies. An example functionality classification for subscription services is listed below:

---

<sup>3</sup>eCl@ss Standardized Material and Service Classification, <http://ecllass-online.com>.

```

1 # classification example
2 ex:SubscriptionService a wsl:FunctionalClassificationRoot .
3 ex:VideoSubscriptionService rdfs:subClassOf ex:SubscriptionService .
4 ex:NewsSubscriptionService rdfs:subClassOf ex:SubscriptionService .

```

Preconditions and effects can be more fine-grained. They are logical expressions: the *preconditions* must hold in a state before the client can invoke the service, and the *effects* hold in a state after the service invocation. The current set of Web standards does not include a language for logical expressions, therefore the WSMO-Lite classes `wsl:Condition` (for preconditions) and `wsl:Effect` are simply placeholders. In our examples, we use WSML to express the logical expressions (cf. Chap. 8); we expect that in the future, the W3C Rule Interchange Format<sup>4</sup> will be a suitable logical expression language for preconditions and effects.

**Non-functional semantics** are represented using some ontology, semantically capturing non-functional properties such as the price, location or quality of service (QoS). The class `wsl:NonfunctionalParameter` marks a concrete piece of non-functional semantics, such as the example price specification below:

```

1 # non-functional property example
2 ex:PriceSpecification rdfs:subClassOf wsl:NonFunctionalParameter .
3 ex:VideoOnDemandPrice a ex:PriceSpecification ;
4   ex:pricePerChange "30"^^ex:euroAmount ;
5   ex:installationPrice "49"^^ex:euroAmount .

```

**Behavioral semantics** are represented by annotating the service operations with functional descriptions, i.e., with functionality classifications and/or preconditions and effects. Below, we show the preconditions and effects for a subscription operation on a video-on-demand service:

```

1 # precondition and effect examples
2 ex:VideoOnDemandSubscriptionPrecondition a wsl:Condition ;
3   rdf:value """
4     ?customer[ex#hasConnection hasValue ?connection]
5       memberOf ex#Customer and
6     ?connection[ex#providesBandwidth hasValue ?y]
7       memberOf ex#NetworkConnection and
8     ?y > 1000
9     """^^wsm:AxiomLiteral .
10 ex:VideoOnDemandSubscriptionEffect a wsl:Effect ;
11   rdf:value """
12     ?customer[ex#hasService hasValue ?service]
13       memberOf ex#Customer and
14     ?service memberOf VideoOnDemandSubscription
15     """^^wsm:AxiomLiteral .
16
17 # definition of the axiom for WSML language
18 wsm:AxiomLiteral a rdfs:Datatype .

```

---

<sup>4</sup><http://www.w3.org/2005/rules/>.

```

1 <wsdl:description>
2   <wsdl:types><xs:schema>
3   ...
4     <xs:element name="NetworkConnection" type="NetworkConnectionType"
5       sawsdl:modelReference="http://example.org/onto#NetworkConnection"
6       sawsdl:loweringSchemaMapping="http://example.org/NetCn.xslt"/>
7   ...
8   </xs:schema></wsdl:types>
9   ...
10  <wsdl:interface name="NetworkSubscription"
11    sawsdl:modelReference="http://example.org/onto#VideoSubscriptionService" >
12    <wsdl:operation name="SubscribeVideoOnDemand"
13      sawsdl:modelReference="
14        http://example.org/onto#VideoOnDemandSubscriptionPrecondition
15        http://example.org/onto#VideoOnDemandSubscriptionEffect">
16      <wsdl:input element="NetworkConnection"/>
17    ...
18  </wsdl:operation>
19  ...
20  </wsdl:interface>
21  ...
22  <wsdl:service name="ExampleCommLtd"
23    interface="NetworkSubscription"
24    sawsdl:modelReference="http://example.org/onto#VideoOnDemandPrice">
25    <wsdl:endpoint name="public"
26      binding="SOAPBinding"
27      address="http://example.org/comm.ltd/subscription" />
28  </wsdl:service>
29 </wsdl:description>
```

**Fig. 12.6** Various WSDL components with WSMO-Lite annotations

### 12.2.3 WSMO-Lite in SAWSDL

WSMO-Lite is based on the simplified service model shown in Fig. 12.4. In this section, we describe a set of recommendations on how to annotate actual WSDL descriptions with the four kinds of semantics, and a mapping from the annotated WSDL structure to the WSMO-Lite RDFS service model, which is then the input to semantic automation.

The listing in Fig. 12.6 shows WSMO-Lite annotations on an example WSDL document. WSDL distinguishes between a concrete service (line 22) and its abstract (and reusable) interface (line 10) that defines the operations (line 12). This structure is annotated using SAWSDL annotations with the examples of semantics shown in the preceding section.

The following paragraphs describe how the various types of semantics are attached to the WSDL structure:

Functional semantics can be attached as a model reference either on the WSDL service construct, concretely for the given service, or on the WSDL interface construct (line 11), in which case the functional semantics apply to any service that implements the given interface.

Non-functional semantics, by definition specific to a given service, are attached as model references directly to the WSDL service component (line 24).

Information semantics are expressed in two ways. First, pointers to the semantic counterparts of the XML data are attached as model references on XML Schema el-

ement declarations and type definitions that are used to describe the operation messages (line 5). Second, lifting and lowering transformations need to be attached to the appropriate XML schema components: input messages (going into the service) need lowering annotations (line 6) to map the semantic client data into the XML messages, while output messages need lifting annotations so the semantic client can interpret the response data.

Finally, behavioral semantics of a service are expressed by annotating the service's operations (within the WSDL interface component, lines 13–15) with functional descriptions, so the client can then choose the appropriate operation to invoke at a certain point in time during its interaction with the service.

The mapping of a WSDL document into the WSMO-Lite RDFS service model is mostly straightforward: a WSDL `<wsdl:service>` element becomes an instance of WSMO-Lite `wsl:Service`, the WSDL operations of the interface implemented by the service are attached to the WSMO-Lite service as instances of `wsl:Operation`, with input, output and fault messages as specified in the WSDL operation. Model references from a WSDL service map directly to model references on the WSMO-Lite service instance; and similarly for model references on the operations.

Since WSMO-Lite does not represent a separate service interface, we combine the interface annotations with the service annotations-model references on an interface and added them to the model references of the service that implements this interface. And finally, any annotations (model references and lifting/lowering schema mappings) from the appropriate XML Schema components are mapped to annotations of the messages in our service model.

The listing in Fig. 12.7 shows a generated RDF form that captures the data in Fig. 12.6, using the lightweight WSMO-Lite RDFS ontology. Note the service, its operation and the operation's request message and the appropriate SAWSDL properties; Fig. 12.6 does not contain any additional data, therefore it is not shown in Fig. 12.7.

#### **12.2.4 WSMO-Lite for RESTful Services**

In the case of RESTful services and Web APIs, there is no widely accepted machine-readable service description language. WSDL 2.0 and WADL<sup>5</sup> are two proposals for such a language. However, the vast majority of public RESTful services are described in plain unstructured HTML documentation. In this section, we present two microformats, hRESTs and MicroWSMO, that enhance the HTML service documentation with machine-oriented structure and semantic annotations.

Microformats take advantage of existing XHTML facilities such as the `class` and `rel` attributes to mark up fragments of interest in a Web page, making the fragments more easily available for machine processing. For example, a calendar microformat marks up events with their start and end time and with the event title,

---

<sup>5</sup><https://wadl.dev.java.net/>.

```

1 @prefix ex: <http://example.com/onto#>
2 @prefix gen: <http://example.com/svc.wsdl#>
3 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
5 @prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .
6 @prefix wsl: <http://www.wsmo.org/ns/wsmo-lite#> .
7
8 gen:ExampleCommLtd a wsl:Service ;
9   rdfs:isDefinedBy <http://example.org/svc.wsdl> ;
10  sawsdl:modelReference ex:VideoOnDemandPrice ;
11  sawsdl:modelReference ex:VideoSubscriptionService ;
12  wsl:hasOperation gen:SubscribeVideoOnDemand .
13
14 gen:SubscribeVideoOnDemand a wsl:Operation ;
15   sawsdl:modelReference ex:VideoOnDemandSubscriptionPrecondition ;
16   sawsdl:modelReference ex:VideoOnDemandSubscriptionEffect ;
17   wsl:hasInputMessage gen:SubscribeVideoOnDemandRequest .
18
19 gen:SubscribeVideoOnDemandRequest a wsl:Message ;
20   sawsdl:modelReference ex:NetworkConnection ;
21   sawsdl:loweringSchemaMapping <http://example.org/NetCn.xslt> .

```

**Fig. 12.7** RDF representation of the WSDL/SAWSDL data in Fig. 12.6, using the WSMO-Lite RDFS ontology

and a calendaring application can then directly import data from what otherwise looks like a normal Web page.<sup>6</sup>

As we have shown in Chap. 5, even though the interaction model of RESTful services (following links in a hypermedia graph) differs from that of SOAP services (messaging), the service model is actually the same: a service contains a number of operations with input and output messages. The hRESTS microformat captures this structure with HTML classes `service`, `operation`, `input` and `output` that identify the crucial parts of a textual service description.

In RESTful services, a service is a group of related Web resources, each of which provides a part of the overall service functionality. The interaction of a client with a RESTful service is a series of interactions where the client sends a request to a resource (using one of the HTTP methods GET, POST, PUT or DELETE), and receives a response that may link to further useful resources. While at runtime the client interacts with concrete resources, the service description may present a single operation that acts on many resources (e.g., `getHotelDetails()` which can be invoked on any hotel details resource). An operation description in hRESTS can therefore specify an address as an URI template whose parameters are part of the input data, and the HTTP method that implements the operation. To capture this information, hRESTS defines HTML classes `address` and `method`.

The definitions of the hRESTS HTML classes are summarized below, with pointers into the listing in Fig. 12.8, which shows an example hRESTS and MicroWSMO service description:

---

<sup>6</sup>Further details on how microformats work can be found at <http://microformats.org>.

```

1 <div class="service" id="svc">
2   <h1><span class="label">ACME Hotels</span> service API</h1>
3   <p>This service is a
4     <a rel="model" href="http://example.com/ecommerce/hotelReservation">
5       hotel reservation</a> service.
6   </p>
7   <div class="operation" id="op1">
8     <h2>Operation <code class="label">getHotelDetails</code></h2>
9     <p> Invoked using the <span class="method">GET</span>
10    at <code class="address">http://example.com/h/{id}</code><br/>
11    <span class="input">
12      <strong>Parameters:</strong>
13      <a rel="model" href="http://example.com/data/onto.owl#Hotel">
14        <code>id</code></a> – the identifier of the particular hotel
15        (<a rel="lowering" href="http://example.com/data/hotel.xsparql">lowering</a>)
16      </span><br/>
17    <span class="output">
18      <strong>Output value:</strong> hotel details in an
19      <code>ex:hotelInformation</code> document
20    </span>
21  </p>
22 </div></div>

```

**Fig. 12.8** Example hRESTS and MicroWSMO service description

The **service** class on block markup (e.g., `<body>`, `<div>`), as shown in the example listing on line 1, indicates that the element describes a service API. A service contains one or more operations and may have a label (see below).

The **operation** class, also used on block markup (e.g., `<div>`), as shown in the listing on line 7, indicates that the element contains a description of a single Web service operation. An operation description specifies the address and the method used by the operation, and it may also contain description of the input and output of the operation, and finally a label.

The **address** class is used on textual markup (e.g., `<code>`, shown on line 10) or on a hyperlink (`<a href>`) and specifies the URI of the operation, or the URI template in case any inputs are URI parameters.

The **method** class on textual markup (e.g., `<span>`, shown on line 9) specifies the HTTP method used by the operation.

The **input** and **output** classes are used on block markup (e.g., `<div>` and also `<span>`), as shown on lines 11 and 17, to indicate the description of the input or the output of an operation.

The **label** class is used on textual markup to specify human-readable labels for services and operations, as shown on lines 2 and 8.

The definitions above imply a hierarchical use of the classes within the element structure of the HTML documentation. Indeed, the classes are to be used on nested elements, in a hierarchy that reflects the structure of the service model.

In summary, hRESTS can be used to structure the HTML documentation of a RESTful Web service according to the WSMO-Lite simple service model. With this structure in place, we now describe MicroWSMO, a microformat equivalent to SAWSDL. As shown earlier in this chapter, to annotate a service description with the appropriate semantics, a model reference on a service can point to a description of the service's functional and non-functional semantics; a model reference on an

operation points to the operation's part of the behavioral semantics description; and a model reference on a message points to the message's counterpart(s) in the service's information semantics ontology, complemented as appropriate by a pointer to a lifting or lowering schema mapping.

SAWSDL annotations are URIs that identify semantic concepts and data transformations. Such URIs can be added to the HTML documentation of RESTful services in the form of hypertext links. HTML defines a mechanism for specifying the relation represented by link, embodied in the `rel` attribute; along with `class`, this attribute is also used to express microformats. In accordance with SAWSDL, MicroWSMO defines the following three new types of link relations:

- **model** indicates that the link is a model reference, as shown in the example listing in Fig. 12.8 on lines 4 and 13.
- **lifting** and **lowering** denote links to the respective data transformations, such as in the example listing on line 15.

Similarly to how SAWSDL annotations in WSDL can be mapped into the WSMO-Lite RDFS service ontology (cf. Figs. 12.6 and 12.7), also hRESTS and MicroWSMO descriptions can be transformed into such RDF. Note that hRESTS extends the WSMO-Lite service model with two properties: `hr:hasAddress` and `hr:hasMethod`, applicable to instances of `wsl:Operation`.

For microformats, the W3C has created a standard called GRDDL (Gleaning Resource Descriptions from Dialects of Languages, [1]) that specifies how HTML documents can point to transformations that extract RDF data. In accordance with GRDDL, an XHTML document that contains hRESTS (and MicroWSMO) data can point to an XSLT stylesheet<sup>7</sup> in its header metadata:

```

1 <head profile="http://www.w3.org/2003/g/data-view">
2   <link rel="transformation"
3     href="http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/hrests.xslt" />
4   ... further metadata, especially page title ...
5 </head>
```

This header enables Web crawlers and other parsers to extract the RDF form of the service description data, even if they are not specifically aware of the hRESTS and MicroWSMO microformats. The MicroWSMO description from Fig. 12.8 is embedded in an XHTML document<sup>8</sup> that contains also the GRDDL transformation pointer. Figure 12.9 shows the RDF view of the document.

## 12.3 Extensions

In this section, we complete the picture of our lightweight SWS approach by sketching a number of algorithms for processing semantically annotated service descrip-

<sup>7</sup>The XSLT stylesheet for hRESTS/MicroWSMO is available at <http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/>.

<sup>8</sup><http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/example.xhtml>.

```

1 @prefix hr: <http://www.wsmo.org/ns/hrests#> .
2 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3 @prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .
4 @prefix wsl: <http://www.wsmo.org/ns/wsmo-lite#> .
5 @prefix ex: <http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/example.xhtml#> .
6
7 ex:svc a wsl:Service ;
8   rdfs:isDefinedBy <http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/example.xhtml> ;
9   rdfs:label "ACME Hotels" ;
10  sawsdl:modelReference <http://example.com/ecommerce/hotelReservation> ;
11  wsl:hasOperation ex:op1 .
12  ex:op1 a wsl:Operation;
13  rdfs:label "getHotelDetails" ;
14  hr:hasMethod "GET" ;
15  hr:hasAddress "http://example.com/h/{id}"^^hr:URITemplate ;
16  wsl:hasInputMessage [
17    a wsl:Message ;
18    sawsdl:modelReference <http://example.com/data/onto.owl#Hotel> ;
19    sawsdl:loweringSchemaMapping <http://example.com/data/hotel.xsparql>
20  ];
21  wsl:hasOutputMessage [ a wsl:Message ].

```

**Fig. 12.9** RDF data extracted from Fig. 12.8 (with blank nodes in square brackets “[ ]”)

tions, and thus automating the common tasks which are currently performed largely manually by human operators. Detailed realization of these algorithms remains as future work. Automation is always guided by a given user goal. While we do not talk about concrete formal representation for user goals in this chapter, the various algorithms need the goal to contain certain specific information. We describe this only abstractly, since we view the concrete representation of user goals as an implementation detail specific to a particular tool set.

**Service Discovery** For discovery (also known as “matchmaking”) purposes, our approach provides functional service semantics of two forms: functionality classifications and precondition/effect capabilities, with differing discovery algorithms. With functionality classifications, a service is annotated with particular functionality categories. We treat the service as an instance of these category classes. The user goal will identify a concrete category of services that the user needs. A discovery mechanism uses subsumption reasoning among the functionality categories to identify the services that are members of the goal category class (“direct matches”). If no such services are found, a discovery mechanism may also identify instances of progressively further superclasses of the goal category in the subclass hierarchy of the functionality classification. For example, if the user is looking for a VideoService, it will find services marked as VideoSubscriptionService (presuming the intuitive subclass relationships) as direct matches, and it may find services marked as MediaService, which are potentially also video services, even though this is not directly advertised in the description. For discovery with preconditions and effects, the user goal must specify the user’s preconditions (requirements) and the requested effects. The discovery mechanism will need to check, for every available service, that the user’s knowledge base fulfills the preconditions of the service and that these preconditions are not in conflict with the user’s requirements, and finally, that the effect of the service fulfills the effect requested by the user. This

is achieved using satisfaction and entailment reasoning. Discovery using functionality categorizations is likely to be coarse-grained, whereas the detailed discovery using preconditions and effects may be complicated for the users and resource-intensive. Therefore, we expect to combine the two approaches to describe the core functionality in general classifications, and only some specific details using logical expressions, resulting in better overall usability.

**Offer Discovery** Especially in e-Commerce scenarios, service discovery as described above cannot guarantee that the service will actually have the particular product that the user requests. For instance, if the user wants to buy a certain book, service discovery will return a number of online bookstores, but it cannot tell whether the book is available at these bookstores. Offer discovery is the process of negotiating with the service about the concrete offers pertinent to the user's goal. An offer discovery algorithm uses the behavioral and information model annotations of a Web service to select and invoke the appropriate offer inquiry operations. In the Web architecture, there is a concept of safe interaction, mostly applied to information retrieval. In particular, HTTP GET operations are supposed to be safe, and WSDL 2.0 contains a flag for safe SOAP operations.

**Filtering, Ranking, Selection** These tasks mostly deal with the non-functional parameters of a service. The user goal (or general user settings) specifies constraints and preferences (also known as hard and soft requirements) on a number of different aspects of the discovered services and offers. For instance, service price, availability and reliability are typical parameters for services, and delivery options and warranty times can accompany the price as further non-functional parameters of service offers. Filtering is implemented simply by comparing user constraints with the parameter values, resulting in a binary (yes/no) decision. Ranking, however, is a multidimensional optimization problem, and there are many approaches to dealing with it, including aggregation of all the dimensions through weighted preferences into a single metric by which the services are ordered, or finding locally-optimal services using techniques such as Skyline Queries. Selection is then the task of selecting only one of the ranked services. With a total order, the first service can be selected automatically, but due to the complexity of comparing the different non-functional properties (for instance, is a longer warranty worth the slightly higher price?), often the ordered list of services will be presented to the user for manual selection.

**Invocation** Service invocation involves the execution of the various operations of the selected service in the proper order so that the user goal is finally achieved. To invoke a single operation, the client uses the information model annotations plus the technical details from the WSDL or hRESTS description to form the appropriate request message, transmit it over the network to the Web service, and to understand the response. If multiple operations must be invoked, the client can use artificial intelligence planning techniques with functional semantics, and on RESTful services, the hypermedia graph can guide the client in its invocations, as the client gets links to further operations in the response to the last operation invoked.

## 12.4 Summary

In this chapter, we presented SAWSDL, and then we introduced WSMO-Lite and MicroWSMO, two related lightweight approaches to semantic Web service description, evolved from the WSMO framework. WSMO-Lite defines an ontology for service semantics, used directly in SAWSDL to annotate WSDL-based services. MicroWSMO and its basis, hRESTS, are microformats that supplement WSDL and SAWSDL for unstructured HTML descriptions of services, giving WSMO-Lite support for the growing numbers of RESTful services.

## 12.5 Exercises

**Exercise 1** Referring to the scenario introduced in Sect. 4.3, create a simple taxonomy of travel services in RDF or OWL.

**Exercise 2** Adopting WSML as language, define preconditions for BlueHotelService described in Sect. 4.3.

**Exercise 3** Given the WSDL in Listing 4.13, add annotations from the above taxonomy, the precondition, and an ontology.

**Exercise 4** Transform the annotated WSDL obtained in Exercise 3 into the WSMO-Lite RDFS form (make up identifiers for the service model components).

**Exercise 5** Create from the RESTful version of the BlueHotelService presented in Sect. 5.3 and HTML description and annotate it to obtain the hRESTS.

**Exercise 6** Enhance the obtained hRESTS with MicroWSMO pointers, reflecting the ones used in the SA-WSDL of Exercise 3.

**Exercise 7** Transform the annotated HTML obtained in Exercise 6 into the WSMO-Lite RDFS form.

## References

1. Gleaning resource descriptions from dialects of languages (GRDDL). Recommendation, W3C, (2007). Available at <http://www.w3.org/TR/grddl/>
2. Khare, R., Çelik, T.: Microformats: a pragmatic path to the Semantic Web (Poster). In: Proceedings of the 15th International Conference on World Wide Web, pp. 865–866 (2006)
3. Kopecký, J., Vitvar, T., Bournez, C., Farrell, J.: SAWSDL: semantic annotations for WSDL and XML schema. IEEE Internet Computing **11**(6), 60–67 (2007)
4. Kopecký, J., Vitvar, T., Fensel, D.: WSMO-Lite: lightweight semantic descriptions for services on the Web. CMS WG Working Draft D11 (2009). Available at <http://cms-wg.sti2.org/TR/d11/>
5. Kopecký, J., Vitvar, T., Fensel, D., Gomadam, K.: hRESTS & MicroWSMO. CMS WG Working Draft D12 (2009). Available at <http://cms-wg.sti2.org/TR/d12/>
6. Semantic annotations for WSDL and XML schema. Recommendation, W3C (2007). Available at <http://www.w3.org/TR/sawsdl/>



**Part IV**

**Real-World Adoption of Semantic Web  
Services**



# Chapter 13

## What Are SWS Good for? DIP, SUPER, and SOA4All Use Cases

**Abstract** In this chapter, we present a selection of extensions and use cases for Semantic Web Services from successful European research projects, namely ‘Data, Information, and Process Integration with Semantic Web Services’ (DIP), ‘Semantics Utilized for Process management within and between Enterprises’ (SUPER) and ‘Service Oriented Architectures for All’ (SOA4All). These projects tackle different aspects of Semantic Web Services, and extend the concepts presented so far in the book in different directions. Due to the differing nature of this chapter, rather than following the structure adopted in the previous chapters, we will provide a general overview in the motivation section, and then dedicate a section to each of the aforementioned projects. For each project, we will first introduce the project and provide background motivation behind it; we will then present technology enhancements delivered through the project and contributions to the extension and evolution of Semantic Web Services technologies; finally, we will emphasize one or more use cases adopted in the project to demonstrate how the technology has been adopted. The summary section concludes with a summary of the achievements of these projects.

### 13.1 Motivation

Scrutinizing how technologies can be employed and extended to meet the needs of real-world use cases is an important process that enables researchers and professionals alike to have a better understanding of the potential impact of their work on future ICT developments and the evolution of economy and society in general.

In this chapter, we will discuss results achieved within three EU-funded research project that over the past years had a major contribution to the maturing of Semantic Web Services technologies, demonstrating their usefulness and applicability in various early adopters scenarios. The projects selected are: ‘Data, Information, and Process Integration with Semantic Web Services’ (DIP, 2004–2007), ‘Semantics Utilized for Process management within and between Enterprises’ (SUPER, 2006–2009), and ‘Service Oriented Architectures for All’ (SOA4All, 2008–2011).

In this chapter, we will survey each of these projects in terms of their motivation and objectives, an overview of the technical solution, and, where possible, a concrete example referring to one of the industrial showcases developed throughout the project.

## 13.2 Data, Information, and Process Integration with Semantic Web Services (DIP)

DIP was the first large-scale research project pursuing Semantic Web Services topics, and its main outcomes rapidly became foundations for subsequent Semantic Web Services efforts.<sup>1</sup> In fact, the original research work on WSMO (see Chap. 7), WSM (see Chap. 8) and WSMX (see Chap. 9) were conducted as part of this project. To avoid repetition of discussions from previous chapters, we will simply provide a short overview of the technical solution.

### 13.2.1 Motivation

The DIP project ran from January 2004 to December 2006, and its mission was to make Semantic Web Services a reality by providing the required infrastructure, i.e., an open source architecture and a set of exploitable tools. This first project in this area enabled development of a set of prototypes demonstrating such an infrastructure and showing how Semantic Web Services could potentially change the way electronic co-operation and business is conducted, in the same way that the original Web revolutionized access to electronic information. The main target of DIP research was the investigation of the potential of Semantic Web Service technologies to facilitate more effective and cost-efficient electronic business by drastically reducing the costs of Enterprise Application Integration (EA). Specifically, DIP investigated the automation of integration or mediation activities through Semantic Web Services as a scalable and cost effective solution to existing integration problems.

In the following, we shortly discuss the different research areas addressed by DIP:

- **Information Management** DIP employed Semantic Web technologies and developed a set of new semantic technologies for Web services on top of them to enable intelligent management of service resources. In particular, DIP provided the switch paradigm from keyword matching to intelligent search and from information retrieval to query answering. DIP also provided solutions for enterprise information management as exchanges between departments via ontology translations, definition of views on documents providing personalization and contextualization of information.
- **Enterprise Application Integration** DIP developed an integration strategy that combines the advantages of ad-hoc and global integration strategies. Leveraging on ad-hoc integration enables the adoption of business needs as the driving force for the integration process. Leveraging on global integration means the creation of extensible and reusable integration. In line with this development, DIP adopted the following goals:

---

<sup>1</sup><http://dip.semanticweb.org>.

- *Business needs to drive the integration process.* DIP identified the major integration needs in terms of business processes and available information sources. Moreover, it structured its integration approach around these needs and employed integration techniques that avoid the disadvantages of ad-hoc integration, enabling, for example, extensibility and reusability.
  - *The integration process should be extensible.* DIP adopted ontologies to avoid static ad-hoc integration. Ontologies were used for publishing information regarding data sources and for aligning it with business needs. Ontologies were extensible and thus allowed for the integration process to be easily extended with the change of business needs.
  - *The integration process should be reusable.* DIP, by pushing for standardization, enabled the creation of reusable models and tools for integration. Ontologies were adopted as the key element to enable reusability in integration solutions. The architecture itself was designed to be domain independent.
  - *The integration process should be flexible.* DIP, by exploring Semantic Web Service and advanced techniques for runtime binding of services by means of service discovery, enabled a flexible ad-hoc integration on the fly, in accordance with changing needs. Integration was not longer a static process where partners were always the same. Integration became a flexible process, where according to changes in business needs, new partners could be integrated into the enterprise.
- **e-Commerce** DIP investigated potential of Semantic Web Services for e-Commerce. Web-enabled e-Commerce needs to be available to large numbers of buyers and suppliers. Its success is closely related to its ability to mediate a large number of business transactions. Flexibility of e-Commerce cannot be achieved without multi-standard approaches. In all likelihood, no one standard will arise that covers all aspects of e-Commerce that would be acceptable in all vertical markets and all cultural contexts, nor would such a standard free us from the need to provide user specific views relating to it and to the content it represents. In line with these assumptions, DIP developed mediation support dealing with heterogeneous ontologies and business processes. Furthermore, in business environments, DIP enabled automatic co-operation between enterprises, by supporting automatic discovery and selection of the optimal Web services based on business needs.

### 13.2.2 Technical Solution

As discussed in the previous section, the main technical achievement of DIP is the first implementation of WSMX and its brokerage services. Many WSMX aspects have evolved since then, but most of the principles in the current version of WSMX are still in line with the version that was delivered in DIP. Given that WSMX and related brokerage services were already discussed in Chap. 9, we refer the reader to that chapter for an in-depth discussion on the DIP technical solution.

### 13.2.3 Use Cases

DIP investigated the adoption of Semantic Web Services as infrastructure to provide concrete solutions to real-world business challenges. In particular, DIP applied results in the areas presented in Sect. 13.2.1 in three scenarios:

- **e-Banking** While many financial institutions offer simple, straightforward information portals, only a few offer advanced services such as financial aggregators, where distributed, heterogeneous information is aggregated into a one-stop-shop. While those applications are quite advanced, the challenge lies in the cost of construction and maintenance. DIP developed a solution supporting aggregation of stock market information.
- **e-Government** Nowadays the plethora of available services for citizens and civil servants provided by public bodies is quite broad. Supporting civil servants and citizens in finding the service most appropriate to their needs is not an easy task. DIP investigated how ontologies, service description, discovery and other related tools can be integrated in the delivery of real services to citizens and organizations.
- **Virtual Internet Service Provider (ViSP)** Technology enables service providers and their partners to reach new customers and expand into new markets, while simultaneously reducing the cost of operations. Semantic Web Services can expose functions and information within service providers systems and allow third party companies to creatively configure new bundles of products and services under their own name. DIP investigated how ViSP infrastructures for the B2B market can offer facilities to compose new virtual services and to support the building of new virtual enterprise portals.

In this section, we will provide some details on the e-Banking scenario adopted in DIP and how Semantic Web Services have been adopted in it, a more comprehensive discussion can be found in [3, 8].

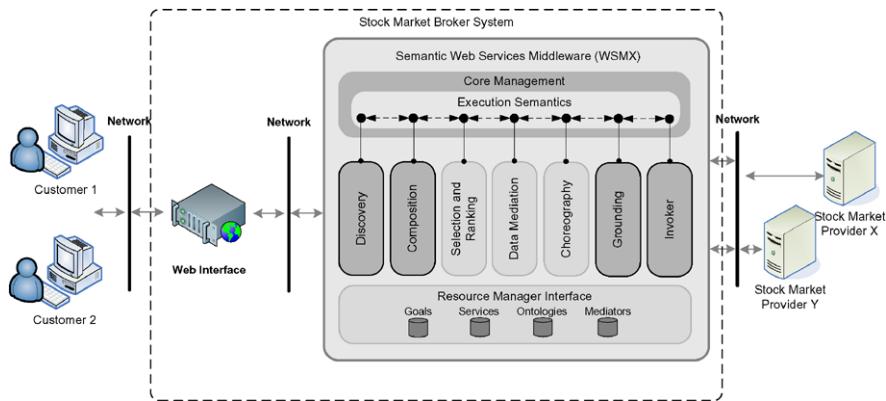
Bankinter,<sup>2</sup> the DIP use case partner for the e-Banking scenario, is centered on financial operations for the stock market.

The area of financial operations is very heterogeneous and dynamic: it includes thousands of products, such as stocks, bonds, commodities and derivatives, which can be combined together. In this area, decision making is influenced by several factors coming from different sources, such factors themselves being very dynamic over time. Thus, this sector is suitable for applying Semantic Web Services technologies to hinder the difficulties its characteristics pose to IT development.

The envisioned application should enable customers of the bank to execute complex operations on the stock market, for instance, to test a condition before executing an action, such as a buy or sell order, in order to select the right stock to buy. Based on the user input, the system formulates a goal template which is forwarded to the

---

<sup>2</sup>Bankinter is a Spanish industrial bank born from a joint venture by Banco de Santander and Bank of America. Bankinter is one of the first bank exploring digital services for Banking. For more information, see <http://www.bankinter.com>.



**Fig. 13.1** The e-Banking scenario in DIP. Most important WSMX services involved in the scenario are portrayed in *dark gray*. The figure adopts the current version of WSMX architecture and not the one available at the time of the project

DIP system for the discovery and execution of appropriate Web services. In the long-term view, Bankinter intends to free customers from the restrictions posed by custom-built applications; the customer should be allowed to formulate whichever goal he/she wishes to achieve.

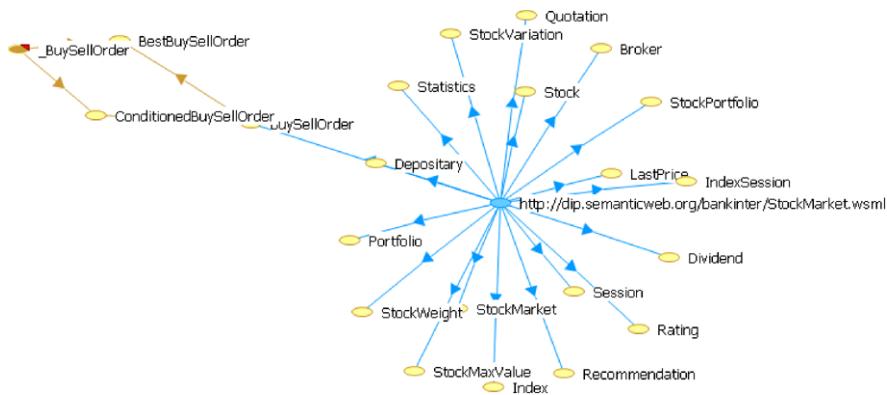
The deployment of WSMX environment as a brokerage system presents a significant added value which lies in its capability of composing existing Web services to achieve more complex functionality. Figure 13.1 presents an overview of the use case.

The scenario discussed involves three actors:

- **Customer** as the end user that needs to perform a complex stock market operation.
- **Broker** as the intermediary between the customer and the stock market services. It discovers and aggregates a set of matching services according to customer needs.
- **Service provider**, that is, a company that provides services to execute operations on the stock market and to access decision-support information for these operations.

According to this classification, WSMX covers the role of a broker. Thus, to support the customer interaction with the broker, the architecture includes a Web interface supporting the customer in the goal specification. The broker is in charge of discovery, composition and execution of Web services. Therefore, the components required to instantiate the scenario are:

- Description of stock market services to be provisioned through the system.
- Service discovery engine that matches services with a given goal.
- Service composition engine that enables aggregation of multiple services in case a single service cannot satisfy the given goal.



**Fig. 13.2** The Stock Market ontology visualized within WSMT

- Service invocation engine that enables the invocation of services returned by the discovery engine.

To support the description of the services, two ontologies are used. The first ontology models the stock market domain. This ontology includes concepts such as stock as well as the related company that issued the stock, the market where the stock is available, the portfolio that may include the stock, and so on. An overview of the concepts is depicted in Fig. 13.2, while a WSML fragment representing the stock concept and stock variation is presented in Listing 13.1. The second ontology models the processes and operations supported in the stock market domain. For example, this ontology describes operations such as ‘obtain details of a given stock’ or ‘execute an action on a stock if its value changes or if its value raises’.

Services and goals are defined on top of ontologies as those just mentioned. Services correspond to the services available through the Bankinter IT infrastructure, while goals correspond to a set of primary customer needs identified within the scenario by Bankinter. Listing 13.2 formalizes the needs of a user that wants to execute some action (action can refer to a buy order, a sell order, or a request for an alert) on a specific stock based on the condition that this stock belongs to top five variations on a specific stock market.

### 13.3 Semantics Utilized for Process Management Within and Between Enterprises (SUPER)

In this section, we introduce the SUPER project,<sup>3</sup> its originating motivation, the technical solution adopted, and an example use case. SUPER ran from March 2006 till March 2009 and built upon the results of DIP by layering business processes

<sup>3</sup><http://www.ip-super.org>.

```

wsmVariant _"http://www.wsmo.org/wsmi/wsmi-syntax/wsmi-flight"
namespace { _"http://dip.semanticweb.org/bankinter#",
  dc _"http://purl.org/dc/elements/1.1#"
}

ontology _"http://dip.semanticweb.org/bankinter/StockMarket.wsmi"

concept Stock
  isPartOfCompany impliesType (1 1) _string
    nonFunctionalProperties
      dc#description hasValue "Represents one fraction of one specific company ownership"
    endNonFunctionalProperties
  isContinuous impliesType (1 1) _boolean
    nonFunctionalProperties
      dc#description hasValue "Reflects whether or not the stock can be sell/bought in the
        Continuous Market"
    endNonFunctionalProperties
  hasFaceValue impliesType (1 1) _decimal
    nonFunctionalProperties
      dc#description hasValue "The amount expressed in currency representing the ownership of the
        company"
    endNonFunctionalProperties
  hasPriceValue impliesType (1 1) _decimal
    nonFunctionalProperties
      dc#description hasValue "The price of the last transaction for that stock in an specific
        market"
    endNonFunctionalProperties
  hasStockType impliesType (1 1) _string
    nonFunctionalProperties
      dc#description hasValue "Specific types of rights in the company"
    endNonFunctionalProperties
  hasBestOffer impliesType (1 1) BestOffer
    nonFunctionalProperties
      dc#description hasValue "The best price that can be found in the market at this point of time.
        Two figures are available: best selling price and best buying price."
    endNonFunctionalProperties

concept StockVariation
  hasVariation impliesType (1 1) _decimal
    nonFunctionalProperties
      dc#description hasValue "The variation of the stock"
    endNonFunctionalProperties
  hasDirection impliesType (1 1) _string
    nonFunctionalProperties
      dc#description hasValue "As the variation does not specify the direction of movement, this
        attribute indicates whether it is increasing, steady or decreasing"
    endNonFunctionalProperties
  hasRelativeVariation impliesType (1 1) _decimal
    nonFunctionalProperties
      dc#description hasValue "The relative variation of the stock"
    endNonFunctionalProperties

```

**Listing 13.1** A fragment of the Stock Market ontology summarized in Fig. 13.2

support on top of Semantic Web Services, and by describing business processes with proper semantics.

### 13.3.1 Motivation

Historically, IT systems were mainly used for efficiently storing and retrieving information. Over the past decades, automation gradually shifted from such data-

```

wsmlVariant _"http://www.wsmo.org/wsmi/wsmi-syntax/wsmi-rule"
namespace {
    _"http://dip.semanticweb.org/bankinter/GoalVariationDependentAction.wsmi#",
    dc _"http://purl.org/dc/elements/11#",
    sm _"http://dip.semanticweb.org/bankinter/StockMarket.wsmi#",
    smp _"http://dip.semanticweb.org/bankinter/StockMarketProcess.wsmi#"
}

goal
    _"http://dip.semanticweb.org/bankinter/GoalVariationDependentAction.wsmi"
nonFunctionalProperties
    dc#title hasValue "Goal for variation dependent actions"
    dc#description hasValue "If the value of a specified stock belongs to the top five variations on a
        specified market, then the system execute some action"
endNonFunctionalProperties
importsOntology {
    _"http://dip.semanticweb.org/bankinter/StockMarket.wsmi",
    _"http://dip.semanticweb.org/bankinter/StockMarketProcess.wsmi"}
}

capability
    _"http://dip.semanticweb.org/bankinter/GoalVariationDependentAction#capability"
sharedVariables {?marketISIN, ?stockISIN}
/*
?marketISIN – the name of the market
?stockISIN – the identifier of the stock
*/
precondition
nonFunctionalProperties
    dc#description hasValue "The stock specified by the user has to belong to the top five
        variations on a specified market"
endNonFunctionalProperties
definedBy
    ?stock [hasName hasValue ?stockName, hasISIN hasValue ?stockISIN] memberOf
        sm#Stock and
    ?market [hasISIN hasValue ?marketISIN, hasTopVariations hasValue ?stock] memberOf
        sm#StockMarket.
postcondition
nonFunctionalProperties
    dc#description hasValue "Result of the web service is execution of the desired action"
endNonFunctionalProperties
definedBy
    ?action[actionType hasValue ?actionType, number hasValue ?number,
    customerID hasValue ?customerID, portfolioID hasValue ?portfolioID,
    stockISIN hasValue ?stockISIN, market hasValue ?marketISIN] memberOf smp#action.

```

**Listing 13.2** A goal for variation dependent actions

driven objectives towards process-oriented support of specific business activities. This trend led to the so-called Workflow Management Systems (WFMS) by which pre-defined processes can be automatically enacted. WFMS evolved into Business Process Management (BPM) techniques that aim at enabling ‘business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information’ [16]. BPM has made more evident the difficulties for obtaining automated solutions from high-level business models, and for analyzing the execution of processes from both a technical and a business perspective. The SUPER project motivation starts from these needs [5]; mediating between the business level and the IT level requires a significant amount of manual work, an approach which is not only costly, but also error-prone.

Semantic (Web Service) technologies can be applied to achieve a feasible level of automation in the BPM area. SUPER thus proposes to combine Semantic Web Services and business processes in order to create one consolidated technology, termed Semantic Business Process Management (SBPM), which would support both agile process implementation and sophisticated process management through knowledge-driven queries issued to the business process space in the form of logical expressions, useful, for instance, to identify activities relevant for compliance with financial or environmental regulations or in emergency situations.

### **13.3.2 Technical Solution**

A in-depth discussion of the SUPER approach is given in [13], here we provide a short overview of the main components of its technical solution.

Semantic Business Process Management aims at accessing the process space of an enterprise at the knowledge level so as to support reasoning about business processes, process composition, and process execution. SBPM builds upon the use of ontologies as a core component, capturing the required domain information in a formal way to enhance the composition, mediation and discovery of Web services via Semantic Web services' techniques implemented in platforms such as WSMX.

SUPER covers all main phases of business process management—design, configuration, execution and analysis—increasing the degree of automation and reuse with the help of ontology-based semantics. The architecture that supports these phases is presented in Fig. 13.3. In the following, we will discuss each phase of the life cycle and how they are supported at the technical level.

#### **13.3.2.1 Design**

The fundamental approach adopted in SUPER is to represent both the business perspective and the IT system perspective of enterprises using a stack of ontologies, and to use automatic reasoning for navigating across the typical layers of business processes:

- **Business level (strategic)** provides a general overview, and contains a functional breakdown of a domain usually subdivided into multiple layers which allow navigating through the problem space.
- **Business level (operational)** is organized into multiple layers which depict concrete processes within the enterprise.
- **Technical level (processes)** is the first level for which well-defined semantics are provided since the models are directly executable by workflow engines.
- **Technical level (services)** is concerned with the actual implementation of processes decomposed into activities that can hopefully be automated.
- **Technical level (implementation)** covers actual implementation artifacts, i.e., business functions.

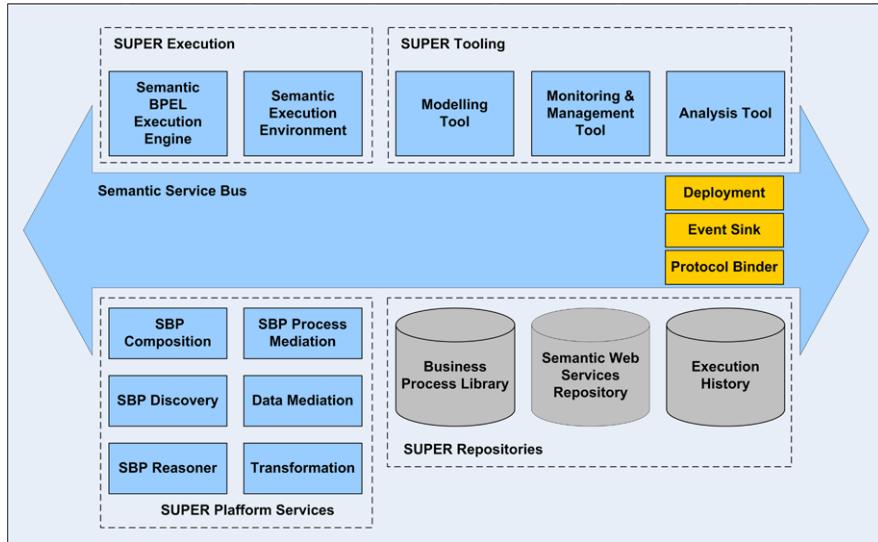


Fig. 13.3 SUPER architecture

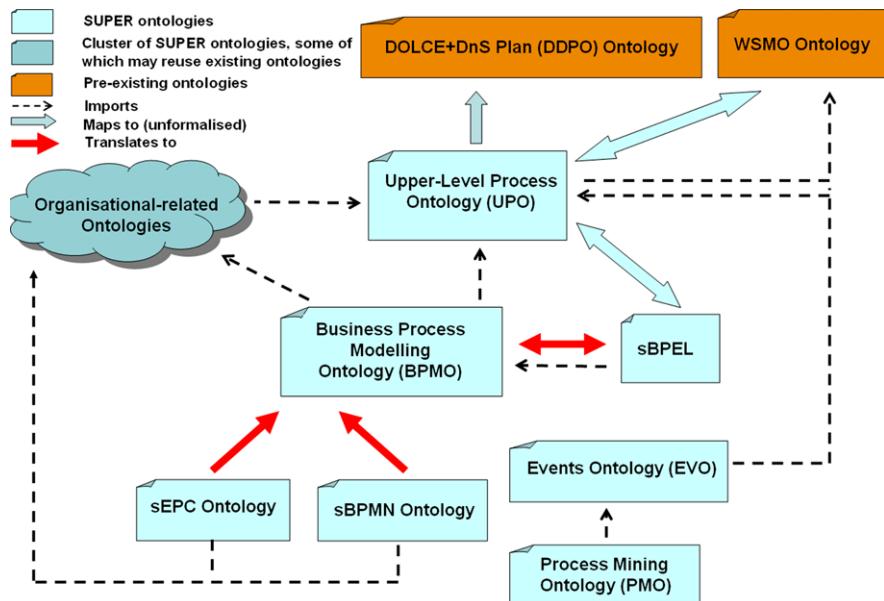


Fig. 13.4 SUPER stack of ontologies

The stack of ontologies depicted in Fig. 13.4 builds upon the use of the Web Service Modeling Ontology (WSMO) (see Chap. 7), as the core Semantic Web Services conceptualization and Web Service Modeling Language (WSML) (see Chap. 8), as

the family of representation languages supporting the specification of ontologies, goals, Web services and mediators.

The integration between the different conceptualizations is provided by the Upper-Level Process Ontology (UPO) that captures general concepts such as process, activity, agent and role which are extensively reused across the subsequent ontologies. UPO plays an integrating role by linking high-level business conceptualizations to lower-level ones concerning both technical and business details. High-level business aspects are formalized in a set of ontologies which are represented in Fig. 13.4 as the Organizational Ontologies cloud. It consists of ontologies capturing resources, organizational structures, business functions, business policies, business strategies, etc.

The lower levels of abstraction upon which we have defined the Semantic EPC (sEPC) and Semantic BPMN (sBPMN) ontologies, which conceptualize EPCs and BPMN, respectively, incorporate, additionally, the appropriate links to WSMO concepts. These ontologies therefore provide support for two of the main modeling notations currently used in BPM. The Business Process Modeling Ontology (BPMO) provides a common layer over both sEPC and sBPMN and links them to the rest of the ontologies from the SUPER stack. BPMO links process models to organizational information as conceptualized in the Organizational Ontologies. It is also linked to the Behavioral Reasoning Ontology (BRO) whose aim is to support the composition of processes by reasoning about their behavior. Finally, BPMO enables the transformation of business processes modeled using different notations in an executable form.

### 13.3.2.2 Configuration

The outcome of the design phase is in the form of business process models that define a control flow between activities, but usually lack specific data flow information and references to particular implementations of business functions. SUPER uses WSMO as the ontological framework to describe the semantics of services in terms of their capabilities, data models and behavioral interface, i.e., how they need to be invoked. Such descriptions can be used to define services and to retrieve implementations thereof that match the requirements formulated by a client as a WSMO goal. Similar concepts can be used to describe the functionality required by the tasks composing a process, and the functionality services provide. Finally, in order to support the execution of business processes, WSMO is complemented by BPEL [6], given its extensive support and use within the industry. Several approaches exist within the literature for the translation of conceptual models expressed in notations such as BPMN to an executable model such as BPEL. They are supported by ontologies through the definition of semantic business process models (sEPC, sBPMN, and BPMO ontologies). The Semantic BPEL (sBPEL) ontology [11] formalizes BPEL and includes additional constructs linked to WSMO so as to support the mediation between heterogeneous data and processes, and the invocation of goals as opposed to explicitly specified Web services. Different transformations have been defined between these different conceptualizations.

SUPER investigated two strategies for binding implementations to executable process models during configuration:

- **WSMO goals as activity implementations** Using WSMO goals as activity implementations requires a middleware that implements the WSMO model following the SEE reference architecture (e.g., WSMX, see Chap. 9). Goals can be used to query for suitable service implementations, following an approach similar to query-by-example (QBE) in databases. Depending on the completeness of the formulated goal the result of such a query can be either a matching service (static binding), or a ranked list of services implementing the requested functionality. In the latter case, the actual binding is delayed until the execution phase, when the execution engine picks the best matching service implementation during runtime (late binding).
- **WSDL services as activity implementations** Instead of connecting activities with WSMO Goals, BPEL4SWS also supports static binding to conventional Web services.

### 13.3.2.3 Execution

The use of Semantic Web Services provides the flexibility required in the BPM execution phase. At runtime goals can be bound to specific Semantic Web Services selected on the basis of the existing conditions and informed by contextual knowledge, which includes monitoring data. Furthermore, since services are described semantically, both functional and non-functional properties are formally represented using domain ontologies.

In a nutshell, the use of Semantic Web Services provides the following benefits from a process execution perspective:

- Process models are independent of the invoked partner services. If a particular partner is not available, the execution middleware chooses a different functionally-equivalent service without the need to change the process model and its deployment information.
- Process models are independent of the internal data model of the partner. The process model has its own semantically annotated data model. If the partner's data model differs from that, semantic models help to bridge this gap.
- Partner services can be selected based on business aspects. Non-functional information about cost, quality of service, trust, legal constraints, etc. can be taken into account so that the selected service is most suitable from a business perspective.

### 13.3.2.4 Analysis

SUPER defined a stack of ontologies that connect log information to higher-level conceptual models about business processes, organizational structures, business goals and strategic aspects. In particular, the Events Ontology provides a sharable

conceptual model for capturing business processes logs, whereas Core Ontology for Business pRocess Analysis (COBRA) connects it to the business level.

Important efforts have been devoted to enhancing mining techniques within SUPER [4], and the uses of semantic technologies have already proven their benefits. So far, the techniques developed as part of the ProM framework [17] have focused on enhancing existing ones with semantics.

### 13.3.3 Use Cases

SUPER developed two different showcases on the topic of managing IT processes in the telecommunication sector.

- **Telecoms Framework and Ontology** Within this use case, SUPER developed a series of domain and process ontologies relevant to the telecommunication sector, together with a semantic framework for telecommunication. The major outcome of this endeavor is the telecom framework YATOSP (Yet Another Telecom Ontology, Service and Process framework) which builds upon established Telecommunications Business Area artifacts such as TMF'S NGOSS.
- **Telecoms Semantic Business Process Applications** Three telecommunication use cases covering the field of fixed telephony, traffic routing and management (through VoIP communications) and mobile environments with innovative devices and technologies have been selected for this particular use case and the project developed several demonstrators showcasing the core technology.

SUPER use cases material are protected by confidentiality, interested readers should contact the use case owners or the project coordinator, via the project Web site to gain access to additional material.

## 13.4 Service Oriented Architectures for All (SOA4All)

In this section, we present the European research project SOA4All<sup>4</sup> launched in 2008 whose aim is to provide Semantic Web Services technology that is feasibly applicable in large-scale, open and dynamic environments where an arbitrarily high number of parties are providing and consuming functionality as services. SO4All is a key contributor to the research on lightweight semantics approaches presented in Chap. 12. In this chapter, we will focus on the technical infrastructure required to realize the SOA4All Service Web vision, and exemplify its application through one of the three use cases of the project.

---

<sup>4</sup><http://www.soa4all.eu>.

### 13.4.1 Motivation

One of the most prominent IT trends of the last decade is the shift towards service-oriented IT architectures (SOA) [7]. SOA specifies a set of loosely coupled services whose interfaces are published, discovered and invoked over the Internet, along with their interactions. It has been adopted by the industry not only as a technological but also a business trend, promising to close the gap between what companies require and what IT is able to deliver. The aim is to achieve a more flexible IT infrastructure that can react to business changes faster than the classic monolithic IT systems. Similarly, using Web standards it is easier for companies to interact with each other, forming business ecosystems. Surveys and experience show, however, that SOA is primarily used for internal integration and much less for external integration, in particular, only very few companies offer their (internal) services to other parties such as their customers. This mind set is now changing as some, predominantly U.S.-based companies such as Yahoo, eBay, Amazon, and Google are exposing their IT services to the Web, allowing third parties to integrate these services to build new applications in the so-called ‘mash-ups’.

Parallel to the internal SOA transformation within large enterprises, the Web has continued its path of success, becoming the dominant information medium for consumers and enterprises alike. It has helped many small and medium enterprises become globally visible in a world dominated by global corporations. Consumers have been dazzled by the new means of participation brought forward by the advent of Web 2.0. Technologies that further simplify user contributions such as blogs and tagging have unleashed the power of communities with initiatives such as Wikipedia convincingly demonstrating the ‘wisdom of the crowds’ in creating the world’s largest encyclopedia.

Advances in communication technologies provide bandwidth that allow users to interact with Web applications similarly to how they interact with desktop applications, facilitating the formation of new business models such as ‘software as a service’ with minimal installation costs and rich Internet applications (RIA).

Early-adopters like Amazon have picked up on these trends and started to expose and offer their internal applications as Web services to other businesses. Nevertheless, today, the Web contains only around 30,000 Web services<sup>5</sup>—a minuscule amount compared to the 30 billion Web pages constituting its content. In consequence, SOA is still largely an enterprise specific solution exploited by, and located within, large corporations and their supply chains. As mobile, wireless, optical and broadband communication infrastructures become ever bigger and more interdependent, we expect the number of Web services to grow exponentially in years to come. In particular:

- More companies will publish their offerings as services accessible through the Web, inspired by the success of examples such as Amazon, eBay and Yahoo.

---

<sup>5</sup><http://seekda.com>.

- Web 2.0 has popularized concepts such as mash-ups and RSS and thereby illustrated comparatively simple means for business networking and business flexibility.
- Efforts to turn the Web into a general platform for accessing and interconnecting arbitrary devices and services are maturing and may turn billions of diverse devices such as household appliances into accessible Web services.
- Following the ‘mechanical turk’ paradigm, humans may act on behalf or as part of a Web service to acquire the input data required to perform a certain task.

Hence, there is a need to master the very large—we must be able to handle the complexity of these systems confidently as well as provide them with learning capabilities and self-organizing functions. Crucially, systems and software must be secure, robust, dependable and optimized in terms of functionality in order to cater to multiple audiences.

SOA4All aims to realize a world where billions of parties are exposing and consuming services via advanced Web technology. The outcome of the project will be a comprehensive framework and infrastructure that integrates four complementary technical advances into a coherent and domain-independent service delivery platform:

- Web principles and technology as the underlying infrastructure for the integration of services at a world wide scale.
- Web 2.0 as a means to structure human–machine cooperation in an efficient and cost-effective manner.
- Semantic technologies as a means of abstracting from syntax to semantics as required for meaningful service discovery.
- Context management as a way to process, in a machine understandable way, user needs that facilitates the customization of existing services for the needs of those users.

### 13.4.2 Technical Solution

SOA4All realizes a service delivery platform that is optimized and tailored to the needs of Web services (both traditional WS-\* stack-based services, as much as REST APIs), which can be expanded to support arbitrary types of exposable functionalities, such as mobile services, sensors and aggregators, and hardware resources. As such, the service delivery platform will provide the core provisioning and consumption functionalities for emerging XaaS (“Everything as a Service”) infrastructures. A more comprehensive discussion can be found in [9].

In the remainder of this section, we present the overall architecture of SOA4All. This architecture comprises the following main components: SOA4All Studio, as user front-end, Distributed Service Bus, Platform Services, and Business Services (third party Web services and light-weight processes). Figure 13.5 shows a high-level depiction of these core elements and their relationships.

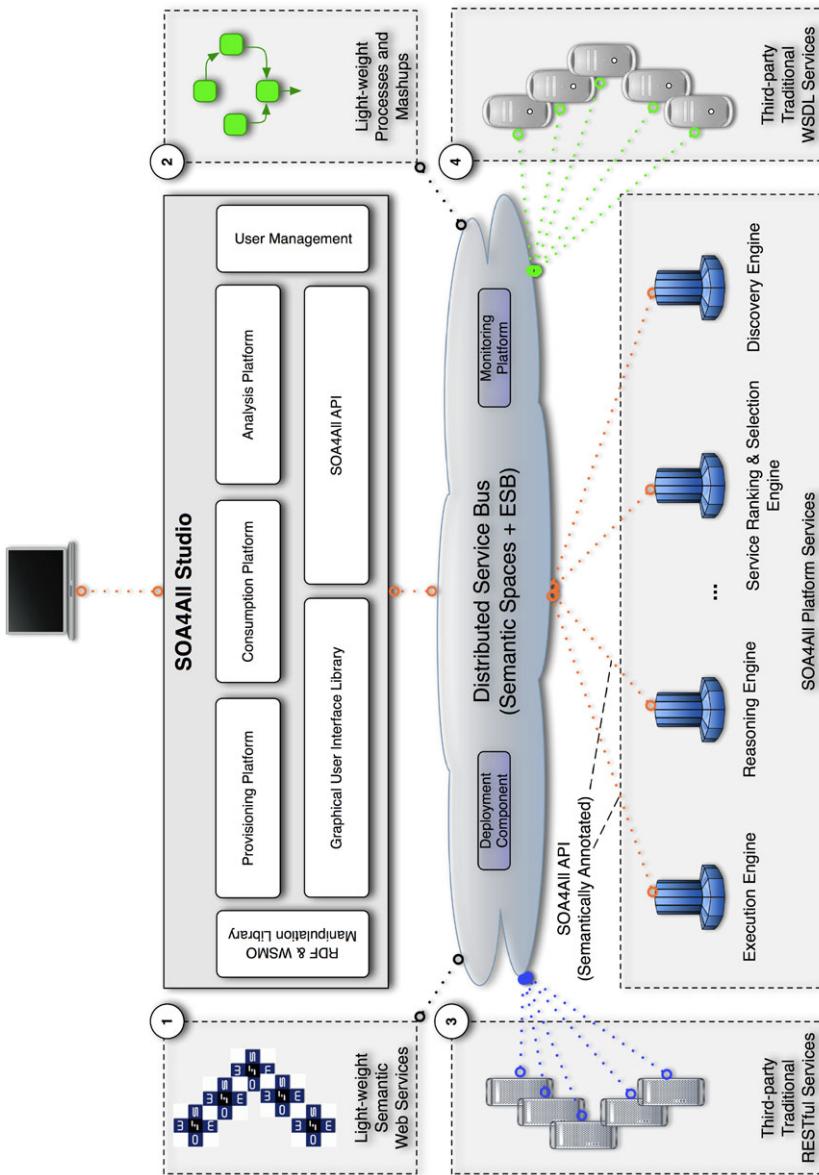


Fig. 13.5 SOA4All overall architecture [9]

### 13.4.2.1 Service Bus

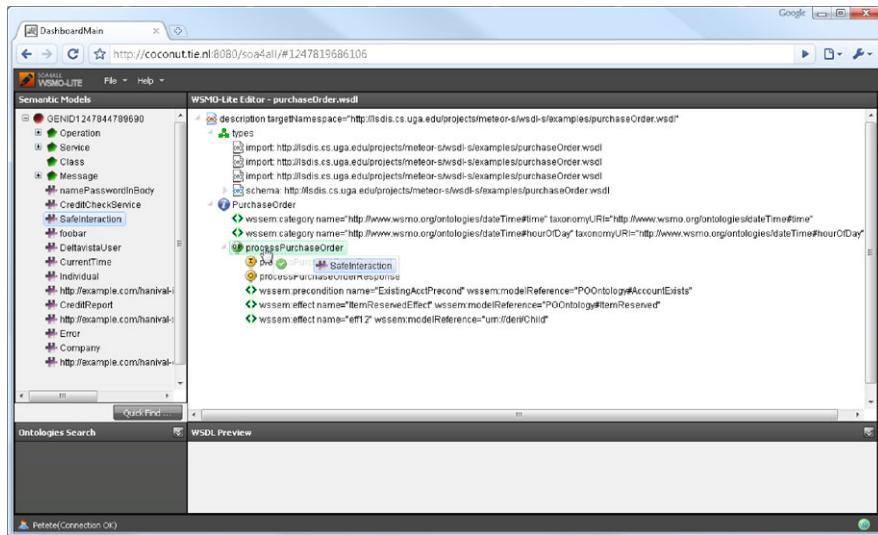
The Distributed Service Bus is at the core of the service delivery platform, as the core communication and integration infrastructure. The DSB augments traditional enterprise service bus technology (ESB [15]) towards large-scale, open, distributed, and hence Web-scale, computing environments. The extensions of the service bus infrastructure include, for this purpose, the implementation of distributed service registries, the integration of the bus realization with established Grid-aware middleware for decentralized deployments, and the enhancement of the communication and coordination protocols by means of semantic spaces. From a deployment perspective, the problems to be solved become of similar nature to those raised when programming, deploying, securing, monitoring, and adapting a distributed application on a computing grid infrastructure.

The enterprise service bus core adopted in SOA4All is based on PEtALS.<sup>6</sup> And finally, the Web-style publish-and-read infrastructure is provided by the so-called semantic spaces that significantly increase the scalability of the bus in terms of communication and coordination of distributed and autonomous services [12]. Semantic spaces, introduced in Chap. 10, are a novel type of communication platform that has recently gained momentum in the middleware community, as a response to the raising challenges of data sharing and service coordination in large-scale, distributed, open and highly dynamic Web environments.

In SOA4All, semantic spaces are realized as a virtualization layer on top of distributed but tightly coordinated semantic repositories. The semantic spaces offer a simple but powerful set of operations for publishing RDF statements, querying SPARQL [14] end-points, and coordinating actions via pattern-based notification services. More pragmatically spoken, service prosumers can subscribe to particular patterns that they would like to match within the wealth of semantic data (RDF graphs). Semantic patterns are triple patterns or graph patterns, as they are supported by most RDF query languages, including SPARQL. Upon publication of matching sets of statements, the subscriber is notified and can proceed with its own workflow. In terms of data management, semantic spaces moreover provide a means for creating virtual containers in the form of the so-called subspaces or federations. The latter are temporary read-only views over multiple subspaces, comparable to views in relational databases across different tables. Subspaces and federations are used to create dedicated interaction channels that increase local scalability by naturally grouping collaborating services and related data. The implementation of the semantic space infrastructure is based on well-established P2P technology and is deployed on the same grid-aware middleware as the service bus nodes. In that way, the space nodes and bus nodes are co-existent and coordinated resources in the same grid infrastructure. From a usage point of view, semantic spaces are exploited to realize many of the large-scale data sharing scenarios for which the volumes of semantic data that has to be processed is likely to exceed the numbers that semantic repositories can currently handle. SOA4All relies on semantic spaces to incorporate various types of repositories for service annotations, goals or process descriptions, as

---

<sup>6</sup>PEtALS [10] is an open source ESB, more information is available at <http://petals.ow2.org/>.



**Fig. 13.6** SOA4All Studio: WSMO-Lite Editor (provisioning platform)

memory infrastructure for shared access to semantically described monitoring data and user profiles, and most of all as alternative communication channel that enhances the traditionally message-oriented service bus with features for anonymous and asynchronous service coordination based on RDF statements; this aspect will be described in more detail in the section on processes that follows, in particular in the context of mash-ups. It is important to note that the complementary communication facilities of the semantic spaces are offered to all services and users as an integrated service of the DSB, without requiring the maintenance of additional access points. In other words, the DSB is conceptualized to provide the core infrastructure services (integration, communication and storage) of SOA4All in an all-in-one solution. This is an important principle for a scalable service delivery platform, as it allows any type of communication efficiently and transparently by means of sharing or exchanging any type of data in between any type and number of distributed parties.

### 13.4.2.2 SOA4All Studio

The SOA4All Studio delivers a fully Web-based user front-end that enables the creation, provisioning, consumption and analysis of services that are published to SOA4All (Fig. 13.6). It consists of three subcomponents that target the three different service management tasks: provisioning at design time, consumption, and analysis at runtime.

- The Provisioning Platform has two main purposes. First, it provides the tools to annotate services, either WSDL services via WSMO-Lite, or REST APIs via

MicroWSMO (cf. Chap. 12 for more details). Second, it incorporates a Process Editor that allows users to create, modify, share, and annotate executable process models based on a light-weight process modeling language. SOA4All provides such a language as a considerably simplified subset of the Business Process Modeling Notation (BPMN1.2, [18]) for the abstract parts, and a subset of the Business Process Execution Language (WS-BPEL2.0, [6]) for executable parts. In this way, SOA4All hides much of the service composition complexity whilst providing sufficient notational semantics to understand the interactions between services being composed, and sufficient expressive power for the users to construct useful compositions.

- The Consumption Platform is the gateway for users to the service world when they act as consumers. The platform allows them to formalize their desires in several ways, defining and refining goals that can be used to discover and invoke the services that fulfill their needs. The platform stresses personalization by making use of contextual factors to offer a more suitable service consumption to the users, and to adapt the services and the platform based on past use, e.g., by recommending goals for different users in varying situations.
- The Analysis Platform obtains information (monitoring events) from the monitoring subsystem of the bus and performs processing in order to extract meaningful information. Monitoring events should come from data collectors that perform basic aggregation from distributed sources in the service delivery platform. Data collectors are installed at the level of the bus (message exchange monitoring), the grid middleware (node monitoring in terms of storage or processing load), and the execution engine (service access and process invocation monitoring).

### 13.4.2.3 Platform Services

The platform services deliver discovery, ranking and selection, composition and invocation functionality, respectively. These components are exposed via the SOA4All Distributed Service Bus as Web services and hence consumable as any other published service. The distinguishing factor between platform services and other Web services is the fact that they provide the core functionality required to realize the service platform. A priori, the set of platform services and their roles and interfaces is aligned with recent research in the area of Semantic Execution Environments and Semantic Web services (e.g., the OASIS SEE Technical Committee).<sup>7</sup> Platform services are mostly used by the SOA4All Studio to offer clients the best possible functionality, while their combined activities (i.e., discovery, selection, composition and invocation) are coordinated via the Distributed Service Bus. The ensemble of DSB, SOA4All Studio and platform services delivers the fully Web-based and Web-enabled service experience of SOA4All: global service delivery at the level of the bus, Web-style service access via studio, and advanced service processing, management and maintenance via platform services.

---

<sup>7</sup><http://www.oasis-open.org/committees/semantic-ex>.

### 13.4.2.4 Business Services (Web Services) and Processes

In the corners of Fig. 13.6, semantic service descriptions and processes (composed services) are depicted. They are created and processed by means of the SOA4All infrastructure. First, SOA4All enables available Web services that are exposed either as RESTful services, or as traditional WS-\* stack-based services in the form of WSDL endpoints [2] (marked with (3) and (4) in Fig. 13.6). Such services are referred to as invocable third-party business services, and are enhanced by SOA4All in terms of automation, composition and invocation via the lifting to the semantic level. Second, Fig. 13.6(1) depicts the semantic annotations of the business services, the so-called Semantic Web services. The semantic descriptions are published in the service registry that is shipped as a platform service, and used for reasoning with service capabilities (functionality), interfaces and non-functional properties, as well as context data. These semantic descriptions are the main enablers of the automation processes related to Semantic Web services. Third, marked as (2), light-weight processes and mash-ups are shown. They are the basis for the realization and execution of composed services. Processes in SOA4All are orderings of Semantic Web services, goal descriptions with associated constraints, data, and control flow information, respectively. As stated earlier in this paper, a goal is a formal specification of the objective that a user would like to have performed, and as such it yields an implicit abstraction over the Web services (REST or WS-\* stack-based) that needs to be executed. Semantic descriptions of processes are published in form of RDF graphs in the shared semantic space infrastructure, and as such become a public common and building block for service computing. In contrast to fully-fledged processes, a mash-up is a data-centric specification over a collection of REST services. A further characteristic of mash-ups is the fact that they are almost entirely executable in semantic spaces. By being data-driven, their composition is enabled by the coordinated access to shared data in semantic spaces. Although being comparably simple, mash-ups thus provide a very promising approach to Web-style service computing, a pre-requisite for large-scale service economies.

### 13.4.3 Use Cases

SO4All investigated the adoption of SOA in infrastructure to provide concrete solutions to real-world business challenges. In particular, SO4All applied results in the areas presented on three scenarios:

- **End-User Driven Enterprise Integration** To investigate potentials and challenges of the integration between new, dynamic, end-user driven service construction and delivery platforms, on the one hand, and existing Enterprise SOA infrastructures, on the other. To demonstrate the feasibility of the end-user service construction in the established area of Enterprise SOA.
- **Twenty First Century Telecom Services** To evaluate how the discovery, integration and use of the capabilities of Telecom operators can be done much more

effectively, reducing the cost and time of creating mash-ups and allowing a much more flexible approach to integration with other services.

- **Customer-to-Customer e-Commerce** To provide a reference service application that supports the setup of e-Commerce applications in customer-to-customer (C2C) exchange settings, by integrating services over the Web.

In this section, we will provide some details on the e-Commerce scenario adopted in SOA4All and how SOA4All structures have been adopted in it. A more comprehensive discussion can be found in [1]. The scenario is centered around the provision of end-customers means to create flexible and highly customizable e-Commerce Web sites. In order to do this, users will use a C2C e-Commerce framework, which operates on an ISP's infrastructure and is accessible to its customers. This framework will add e-Commerce-specific functionality such as a product catalogue, order management, invoicing, and basic payment services. Other functionalities, such as finding a service and composing of process, will be more generic and supported by the SOA4All service delivery platform. A dedicated service broker, offering a search engine and a service marketplace for value-added (composed) services, will provide access to third-party services.

The e-Commerce application framework will enable consumers to conduct business with other customers. This approach is similar to popular Web 2.0 platforms (e.g., Facebook or Google Checkout) where the company provides a framework that allows customers to create solutions for C2C business. What is different in the presented scenario compared to existing solutions is that the scenario will entirely focus on providing an easy way of using third party services (solutions) offered through the framework.

End-customers are able to use various SOA4All-enhanced tools offered through the framework to build their own end customer applications. For example, the framework will include typical Web shop functionalities such as a shopping cart feature and an access to payment providers using the SOA4All service orchestration and communication facilities.

The scenario is developed around three actors (see Fig. 13.7 due to [1]):

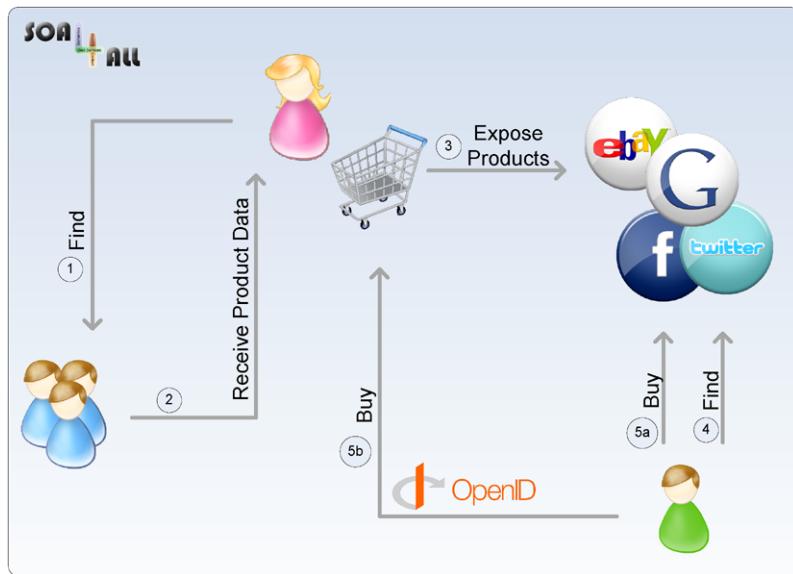
**Reseller** who wants to generate income on various popular Web 2.0 sites and uses SOA4All to connect services of various partners in order to expose product information to Facebook, Twitter, her own Web shop, etc.

**Seller** who wants to increase sales by offering Web services, allowing resellers to retrieve a product list and to order a specific product.

**Buyer** who visits the reseller's Facebook page, sees some personalized products and decides to buy one of them.

In the following, we describe how the scenario can be enabled by SOA4All technologies and how the different actors interact with them. Figure 13.8 depicts the scenario.

Sellers are affiliated to different companies which are all producing products from the textile industry, reaching from footwear t-shirts for all seasons and covering both male and female apparel. They are responsible for the e-Commerce part of their companies and need simple means to provide and consume services. To enable



**Fig. 13.7** Roles and process steps of the e-Commerce use case scenario

interaction between buyers and the e-Commerce back-end, they provide a simple service that allows to

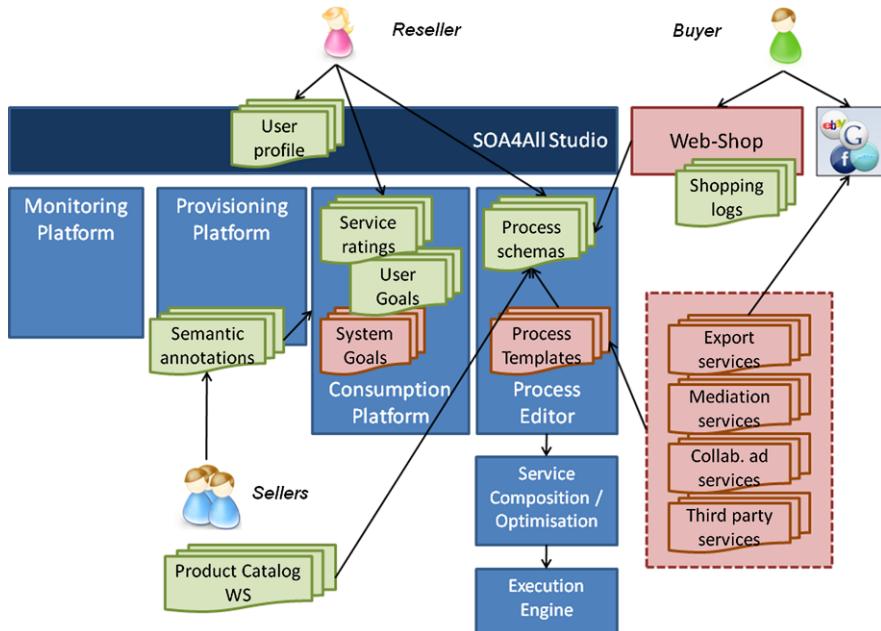
- Retrieve a list of products and product descriptions.
- Request information about the price and availability of a product.
- Place a product order.

Services can be both WSDL and REST-based. Once created, they can be registered in the SOA4All platform through the SOA4All Studio. Sellers use the SOA4All tools to annotate their services and describe them graphically with some semantic information based on common e-Commerce ontologies, such as eCl@ss<sup>8</sup> for product categorization and the GoodRelations<sup>9</sup> for details on their product definitions and order services. They can do all of this using the graphical components of SOA4All—without any knowledge of the technologies behind the scenes—just by using drag & drop to annotate their service elements. Afterwards they click the save button to add the service to SOA4All.

The reseller registers a business to the framework, which allows her to buy and sell products. The reseller is skilled in IT and Web technologies: she uses various Web 2.0 platforms including Facebook, Twitter and even an alpha version of Google Wave and even owns a small Web shop where she adds textile products manually from time to time. However, via her Web shop she is only making a limited number of sales and the product descriptions are usually outdated.

<sup>8</sup><http://www.eclasse-online.com>.

<sup>9</sup><http://purl.org/goodrelations>.



**Fig. 13.8** Graphical representation of the customer-to-customer e-Commerce scenario [1]

She spends most of her time that she would like to instead invest into her Web shop updating prices and availabilities or to remove and add products. Also she has no way of automatically aligning her offered products with the Web 2.0 platforms she is using. For example, she also has created an eBay shop to sell and auction some of her products, but needs to manually synchronize the two shops. The reseller wants to change this and to do this more efficiently, thus saving time and being able to invest it in more sales opportunities. She wants to be able to automate product listings, being convinced that adding her products to her Web 2.0 appearances would significantly increase her sales. However, automating things is a highly technical work, and even if she develops a piece of software to connect her shop to the product data of her supplier, she would make herself bound to this supplier and would not be as flexible any more as things tend to be highly connected.

The reseller visits the SOA4All Studio and creates a user profile with the SOA4All Profile Editor. She then starts searching for suitable services using the SOA4All Discovery Platform and finds many services related to products and product catalogs such as a service from Amazon and other sellers. She has the chance to refine her search and to filter the result set to those services related to the textile domain. The reseller also has the chance to see comments and feedback left by other users who tried them before.

The SOA4All recommendation system is capable of automatically recognizing her interests by analyzing her past behavior and relating it to other SOA4All users

within the Consumption Platform. In this way, SOA4All is able to ‘recommend’ her services that are appreciated by users with a similar profile. She looks at the product service result list and gets an ordered list of four different services, in which the fourth one was rated not so positively, while the first three have received good feedback. She finally decides to use the three different services coming from the sellers introduced before, and bookmarks them to her favorite list.

Having selected some services she wants to achieve, the reseller wants to connect them to her Web shop. One of the services offers more available end points. She selects all possible services at design time and leaves it to SOA4All to select the best one based on the current context. For example, SOA4All will select one of the service end points automatically during runtime.

Alternatively, the reseller can also select goals instead of services. She uses a pre-defined template from another SOA4All user, which allows her to take advantage of existing process templates without having to start from scratch. She is recommended to use a mediation & aggregation service for merging and aggregating the three product services including some descriptions. Once she has done so, she uses the process editor again to create a second process which forwards an order from her Web shop to the product manufacturer service, as soon as an order arrives. She executes her process to test it and she can directly see the results of her execution.

The reseller can use also SOA4All to show her product information on some well known Web 2.0 platforms, thus presenting her products to a large number of people. She also wants to ensure that people that visit her Facebook profile see the products belonging to their current context. For example, she does not want her Facebook profile to advertise a winter jacket in the summer time, but she might want to advertise clothes with special Facebook logos.

Although the buyer does not realize that SOA4All is used under the hood, he benefits greatly from the new functionalities of SOA4All.

**Context Awareness** The buyer will see products that are ‘made for him’, or are more precisely matching his profile. He will not see any winter dresses for women in the summer time.

**Always up-to-date** There is nothing worse than ordering a product, paying and then getting a notification that the product is not available any more. With SOA4All, the buyer can be sure that all product data is up-to-date and available.

**Process Optimization** The buyer indirectly benefits from the SOA4All template generator which will optimize the order process over time to match the requirements of the users. This will shorten the time needed for him to buy products and will increase his shopping experience.

**Reliability and Trust** As the reseller used the rating facilities of SOA4All, she only added product sellers that are reliable. Furthermore, in case the services fail for some reasons, the execution engine is able to substitute them to complete the execution. The buyer therefore directly benefits from this increased reliability.

## 13.5 Summary

In this chapter, we gave an overview of extensions and use cases for Semantic Web Services from three successful European research projects, namely Data, Information, and Process Integration with Semantic Web Services (DIP), Semantics Utilized for Process management within and between Enterprises (SUPER) and Service Oriented Architectures for All (SOA4All). Other interesting extensions and results have been provided in projects such as SWING,<sup>10</sup> SemanticGov<sup>11</sup> and SHAPE.<sup>12</sup> Interested readers can refer to these projects Web sites to gather further information.

## References

1. Abels, S., Schreder, B., Villa, M., Zaremba, M., Sheikhhasan, H., Puram, S.: eCommerce framework infrastructure design. SO4All Project Deliverable D9.2.1. Tech. rep. (2009)
2. Chinnici, R., Haas, H., A.L.J.J.M.D.O., Weerawarana, S.: Web services description language (wsdl) Version 2.0 Part 2: adjuncts. W3C Recommendation (2007)
3. Cobo, J.M.L., Losada, S., Cicurel, L., Bas, J.L., Bellido, S., Benjamins, V.R.: Ontology Management in E-banking Applications. In: Hepp, M., Leenheer, P.D., de Moor, A., Sure, Y. (eds.) Ontology Management. Semantic Web and Beyond Computing for Human Experience, vol. 7, pp. 229–244. Springer, Berlin (2008)
4. de Medeiros, A.K.A., Pedrinaci, C., van der Aalst, W.M.P., Domingue, J., Song, M., Rozinat, A., Norton, B., Cabral, L.: An outlook on semantic business process mining and monitoring. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM Workshops (2). Lecture Notes in Computer Science, vol. 4806, pp. 1244–1255. Springer, Berlin (2007)
5. Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel, D.: Semantic business process management: a vision towards using semantic web services for business process management. In: Lau, F.C.M., Lei, H., Meng, X., Wang, M. (eds.) ICEBE, pp. 535–540. IEEE Computer Society, Los Alamitos (2005)
6. Jordan, D., Evdemon, J.: Web Services Business Process Execution Language Version 2.0. OASIS Standard (2007)
7. Josuttis, N.M.: Soa in Practice: The Art of Distributed System Design, 1st edn. O'Reilly Media, Sebastopol (2007). <http://www.worldcat.org/isbn/0596529554>
8. Kleczek, D., Losada, S., Bas, J.L., Bellido, S., Contreras, J., Montes, M.M.: WSMO descriptions of application 2. DIP project deliverable D10.8. Tech. rep.
9. Krummenacher, R., Norton, B., Simperl, E.P.B., Pedrinaci, C.: Soa4all: enabling Web-scale service economies. In: ICSC, pp. 535–542. IEEE Computer Society, Los Alamitos (2009)
10. Louis, A.: Build an SOA application from existing services—Put heterogeneous services together with the Petals ESB. JavaWorld.com (2006). <http://www.javaworld.com/javaworld/jw-10-2006/jw-1011-jbi.html>
11. Nitzsche, J., Wutke, D., van Lessen, T.: An ontology for executable business processes. In: Hepp, M., Hinkelmann, K., Karagiannis, D., Klein, R., Stojanovic, N. (eds.) SBPM. CEUR Workshop Proceedings, vol. 251. CEUR-WS.org (2007)
12. Nixon, L.J.B., Simperl, E.P.B., Krummenacher, R., Martín-Recuerda, F.: Tuplespace-based computing for the semantic web: a survey of the state-of-the-art. Knowledge Engineering Review 23(2), 181–212 (2008)

<sup>10</sup><http://www.swing-project.org>.

<sup>11</sup><http://www.semantic-gov.org>.

<sup>12</sup><http://www.shape-project.eu>.

13. Pedrinaci, C., Domingue, J., Brelage, C., van Lessen, T., Karastoyanova, D., Leymann, F.: Semantic business process management: scaling up the management of business processes. In: ICSC, pp. 546–553. IEEE Computer Society, Los Alamitos (2008)
14. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for Rdf. W3C Recommendation (2008)
15. Schmidt, M.T., Hutchison, B., Lambros, P., Phippen, R.: The enterprise service bus: making service-oriented architecture real. IBM Systems Journal **44**(4), 781–797 (2005)
16. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business process management: a survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) Business Process Management. Lecture Notes in Computer Science, vol. 2678, pp. 1–12. Springer, Berlin (2003)
17. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The prom framework: a new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN. Lecture Notes in Computer Science, vol. 3536, pp. 444–454. Springer, Berlin (2005)
18. White, S.: Business process modeling notation. OMG Standard (2009)

# Chapter 14

## Seekda: The Business Point of View

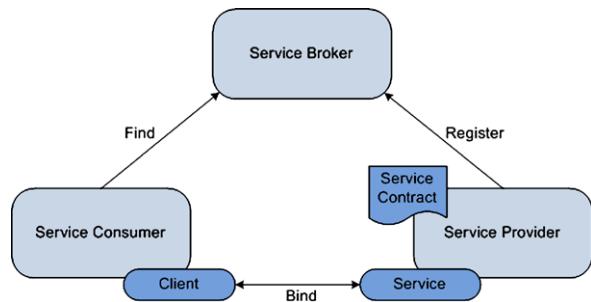
**Abstract** Industry is slowly picking up on the use of semantic technologies within their systems. In this chapter, we describe how these technologies are employed by seekda, a company focused on Web services. The aim of seekda is to ease the search, interoperability and bundling of services and make them readily usable in various areas, for example, in e-Tourism. This is done through the use of a crawler, a search portal and a service bundle creator. An application of these technologies within a seekda product—*seekda! connect*—is also described.

### 14.1 Motivation

Since its inception, the Internet grew very rapidly. From a Web of simple html pages, it has evolved into a Web of multimedia content, followed by a Web of services. These services range from simple ones, such as sending text messages over mobile phones, to more sophisticated ones, like booking a flight and purchasing items. In parallel to this evolution, software development companies moved from creating monolithic applications to more service-centric architectures where the focus is given on the reuse of software artifacts rather than rebuilding them from scratch. This led to the birth of Service Oriented Architectures (SOA) [3]. The basic principle is that of having services that are readily available for organizations. Such organizations are not interested in the implementation of the services they require, but rather the descriptions of the services which define what can be provided by the service. Furthermore, the combination of such services leads to the fulfillment of a more complicated requirement that cannot be obtained by a simple atomic service—here is where service processes come into play. Processes combine services together into a workflow to achieve some predefined functionality. Generally, an organization has what is called a *Business Process* which is enacted as a *Workflow Process* with specific (automated or non-automated) services that fulfill higher-level business needs. Automated services need not be legacy services within the organization itself, but may also be provided by some external entities. In a nutshell, the basic idea is that services are *registered* by a provider to a service broker. Organizations then *search* for services through the service broker and eventually *bind* the needed services with the provider. This scenario is depicted in Fig. 14.1.

Even though this looks like a simple scenario, various problems arise in realizing it:

**Fig. 14.1** The *Register, Find* and *Bind* paradigm



- There are still no effective means of finding services over the Internet—existing search engines are aimed at exploiting textual descriptions in Web pages which are not available for services.
- Services are not transparent, and semi-formal or formal descriptions are not available for them.
- Combinations of services that fulfill more complicated requirements of businesses are not available.
- In most of the cases, data models underlying the service descriptions do not match, and transformation between such data models is required.
- Service Level Agreements (SLA) have to be taken into account and this is traditionally a manual task.

All these aspects pose severe obstacles in creating a true Web of services and hamper interoperability thereof. The mission of seekda is to ease the search, interoperability and bundling of services and thus achieve a true Web of services. seekda provides a dedicated Web services search engine, featuring monitoring and invocation facilities. Interoperability and bundling of services are controlled by a *Bundle Configurator and Assistant* tool designed to semi-automatically assist the creation of service bundles.

In this chapter, we describe how these technologies are developed and illustrate their benefits. In Sect. 14.2, we describe the technologies employed by seekda to address the above problems; in Sect. 14.3, we show the application of these technologies by one of the seekda products.

## 14.2 Technical Solution

Searching and discovering services is the first step towards bundling and using services. This section describes how seekda, through the use of a search engine for services, eases the process of finding, monitoring and also using services over the Internet. To enable such a search engine, two basic components are needed: a crawler to discover, classify and index services [8] and a portal that enables end-users to search for these services.<sup>1</sup> Section 14.2.1 describes the crawler, followed

<sup>1</sup><http://webservices.seekda.com>.

by Sect. 14.2.2 which describes the search portal. To enable the combination of services together (called service bundles), seekda provides a *Bundle Configurator and Assistant* tool—this is described in Sect. 14.2.3.

### 14.2.1 Crawler

The main purpose of the crawler is to crawl the Web in order to find, classify and index services. In this section, we describe the technologies employed by this component in order to achieve these functions. Traditionally, searching for services requires queering on specific platforms such as StrikeIron or ProgrammableWeb. However, these platforms require service providers to manually register their services in order for them to be found by requesters. This approach does not scale and a more automated approach is required. Another approach is to provide full semantic descriptions for services and discover them using complex reasoning tasks. The drawback of this approach is that in the current state of the Web and technologies, very little semantic annotations for services are available. Furthermore, manually annotating such services is close to impossible and there currently exist no tools that automatically extract semantic information.

The crawler developed at seekda detects services over the Web and classifies them in an internal ontology that is maintained by seekda. Discovered services can then be annotated with semantic descriptions. The aim is to detect as many public services as possible. To achieve this goal, the crawler is focused on both WSDL-based and RESTful services. The search is not just restricted to pure technical service descriptions but also encompasses information surrounding the service, for example, HTML documents that talk about the services. This information will help in a two-fold way: to discover the actual service (and to automatically classify it) and to further annotate the service (given that the extra information about the service is available). The semantic information is then used by the front-end search engine that seekda also develops and provides to users (more in Sect. 14.2.2).

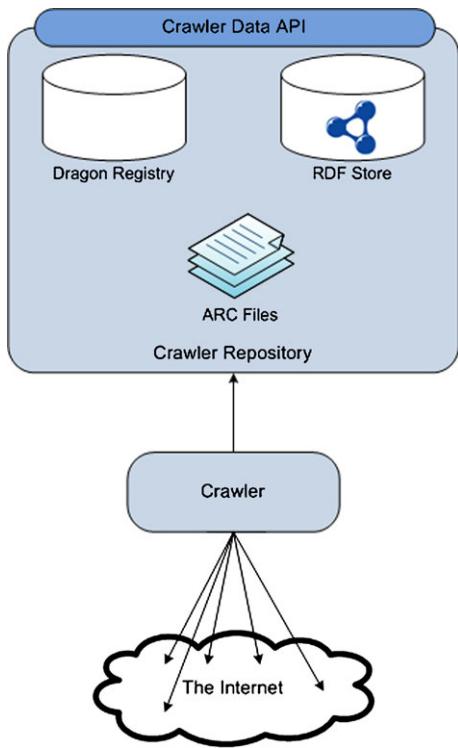
The crawler is based on the open source Internet crawler Heritrix<sup>2</sup> and it is designed to produce snapshots of a large part of the Internet. There are some default components and also a *processor chain* in this system that allow different crawl configurations. The processor chain is extended by new processors in order to focus the crawl on services over the Web, retrieve the related documents and store the metadata in an RDF store.

A high level overview of the crawling components is given in Fig. 14.2. The crawler stores fetched documents—including service descriptions—in the *Crawler Repository* which contains a set of *ARC files*, an *RDF store* and the *Dragon registry*. The *crawler data API* is an interface that can be used by other components in order to access the data stored in the crawler repository. The descriptions that are fetched are compressed and stored in ARC format (a file format for compressed files) and

---

<sup>2</sup><http://crawler.archive.org>.

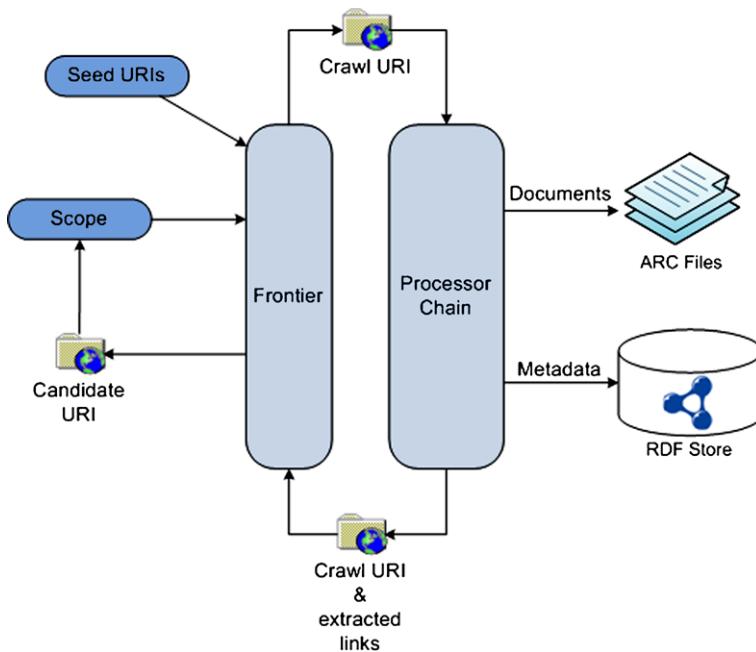
**Fig. 14.2** High level overview of the crawling components and their layout



the metadata is stored in an RDF store. The Dragon registry is used to index WSDL files that are compressed in the ARC files.

Figure 14.3 shows how URIs are handled by the crawler. The starting point is a set of URIs, which is called a seed. The *Frontier* is responsible for determining the next URI to be crawled. It hands over the URI to a worker thread which executes a set of processors on this URI. Processors are responsible for enforcing preconditions, fetching the content, extracting links and writing documents to the file system. Fetched documents are first normalized and then written to files in the ARC format. All metadata which has been generated by different processors is sent in the form of RDF triples to an RDF store. Every extracted link gets a cost assigned, which is done by one of the non-default Web service specific processors. This is done for keeping the crawl focused, as links with a lower cost will be preferred in the queue. The crawl URI—which now includes the extracted links with the cost assignment—is returned to the Frontier which adds the links to the Frontier queues.

As described earlier, the crawler can be configured for specific needs and also uses ontologies in the background for classifying the services that are found. In Sect. 14.2.1.1, the configuration options are described followed by the metadata information in Sect. 14.2.1.2.



**Fig. 14.3** Architecture of the crawler

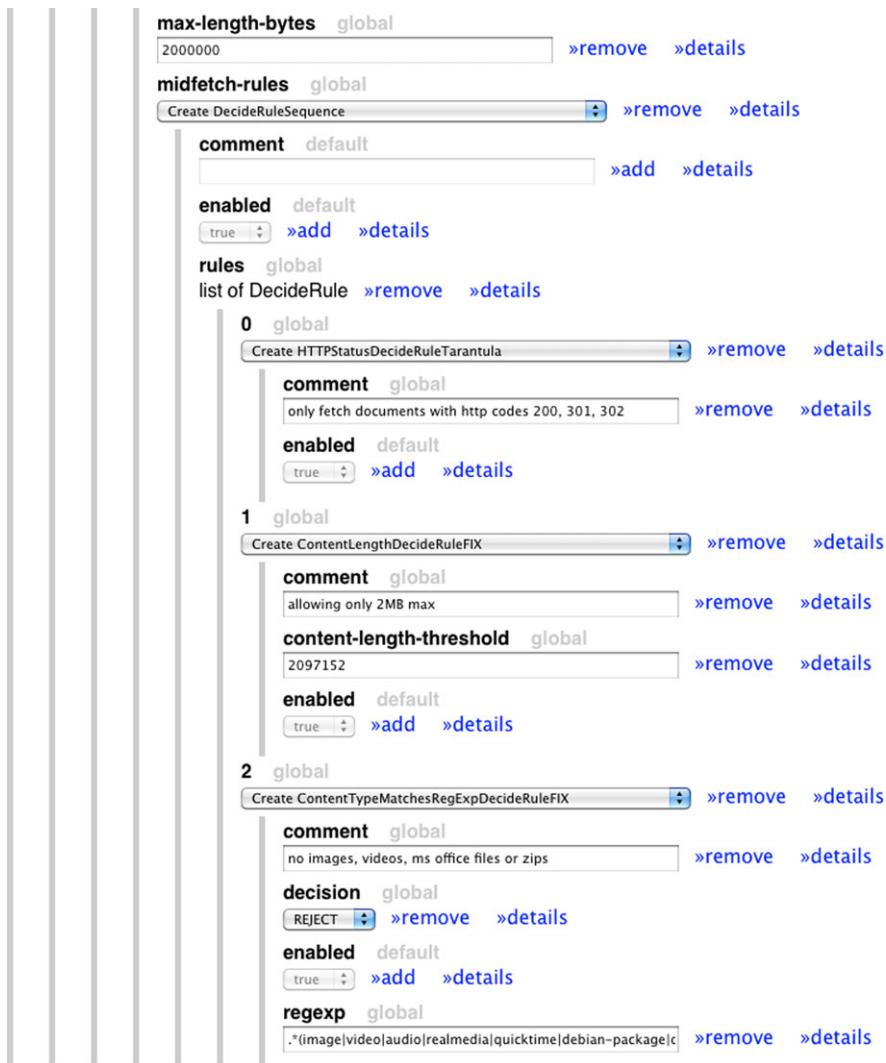
#### 14.2.1.1 Crawl Configuration

A crawl configuration consists of hierarchical structured modules and its individual settings. Figure 14.4 shows the Web interface for defining these settings, specifically it shows settings for the module which is responsible for fetching content via HTTP.

We will now describe a simple configuration setup for the crawler. The user supplies some metadata about the crawl, like a description and information about the operator of the crawl. In addition, a Web crawler should respect other server operators and follow the rules specified in *robot* files as much as possible. The rules in the robot exclusion standard specify which user agents are allowed to access which parts of a Web site. For this reason, the crawl operator specifies how the crawler should deal with those files, where the possibilities are to obey the relevant directives, namely, to completely ignore them or to obey only the most liberal restrictions available. A custom set of directives can also be provided for specific hosts.

During crawling, events are logged so that results are understood and configuration flaws detected. These events include URIs that are handled at a particular point in time, used system resources and runtime errors. This is all taken care by the logger module.

The starting point of a crawler is the so-called seed, containing a set of URIs. These URIs are crawled first and depending on the existence and configuration of other modules, the links on these sites will be added to a queue for further processing. The rules for following such links are called *scope rules*. Since the crawler is



**Fig. 14.4** Configuration interface of the crawler

focused exclusively on services, content that is not relevant is excluded (such as images, audio, movies, etc.).

Different filter implementations are available in order to exclude processed URIs. The nature of these implementations is determined by whether processed URIs are managed in-memory or on disk. For example, the BDB filter uses a BerkeleyDB JE database to hold its records of known hosts (queues), while the Bloom filter works in-memory over hashing functions. The BDB implementation can be used for smaller crawls with less than ten million URIs. When the number of URIs gets

higher, the performance decreases and the Bloom filter is the best choice. This filter demands, however, substantial memory.

The run of a crawl is controlled by the crawl controller. This component can be configured with the following parameters:

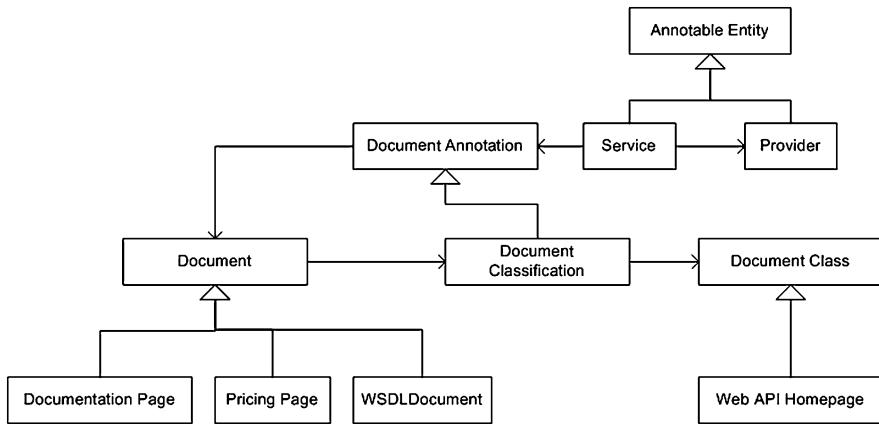
- Maximum response time when crawling over a URI.
- Number of attempts the crawler makes to get the contents of a URI.
- Uptime of the crawler.
- How many working threads should be created.

The main part of the crawl configuration is the list of processors (i.e., the number of working threads). All URIs that are in the scope are handled by each of the processor on this list. PDF files are converted to HTML in order to ease further processing and an HTML extractor is used to get new URIs from the translated document. Information about the structure of the document can also be extracted. This information is used by other processors to take decisions (for example, about how likely it is that a service is available). Other types of URI extractors are used to extract links from other formats such as XML, Javascript and flash/shockwave files. Custom processors generate a link graph which describes how links are connected together from a source to the next. All these documents are stored as ARC files. WSDL documents, however, are handled by a different specific processor. All processors can add RDF triples—which are collected by another processor in the list—and commit these triples to an RDF store. Such statements use concepts from the specific ontologies, described in Sect. 14.2.1.2.

Given these pieces of information about the document and the outgoing links, a cost assignment for all the new URIs is made. As mentioned, URIs with the lowest cost will be processed first. The value of this assignment depends on the nature of the crawl. For example, when crawling for Web services, the value depends on keywords.

The described configurations are stored as *profiles* and can be reconfigured for every single job. Each crawl produces specific types of outputs. When crawling for WSDL descriptions, two types of archives are produced: one containing all the WSDL files and another containing the related documents. When crawling for REST services, two other types of archives are produced: one containing the related documents and another containing the metadata (in the form of RDF triples) that is extracted from the service.

The problem here is what *related* exactly means. The approach is to use the link graph as the basis for this decision: all documents that have a direct link to the WSDL are considered to be related to the service. Such link connections are either *outgoing* links, i.e., links from the WSDL to other document, or *incoming* links, i.e., links that lead from other documents to the WSDL. Another approach is to use vector similarities. This is done by comparing the term vectors of WSDL files with the term vectors of web documents gathered from the same domain of the WSDL files. This might help, for example, in finding pages on a service provider's site that talk about the service without being directly interlinked with it.



**Fig. 14.5** Crawler ontology

The related documents assemble information in different formats, such as HTML pages, text files, PDF files. All the documents are normalized, i.e., they are all encoded in UTF8; PDF files are transformed to HTML, and HTML files are cleaned from unnecessary markup and stripped to avoid lengthy pages. Together with the two archive types, an index is delivered. This index maps services to WSDL descriptions and to related information. Furthermore, it allows random access to the archive data by indicating the archive number and offset of all documents. Along with the textual representation of the documents, some other information about its retrieval is stored. This includes the URL, the date of archiving, the content-type and the offset of the record in the ARC file. Other information (such as the service provider data) which is extracted from the WSDL file, is then represented as RDF triples based on the ontologies described in Sect. 14.2.1.2.

#### 14.2.1.2 Metadata

All information about the services, such as their WSDL description(s), their related documents, their provider, their operations, etc., is structured according to a generic *Service Ontology* (Fig. 14.5). This ontology has been developed in the scope of the European project Service-Finder<sup>3</sup> which is building a platform for Web service discovery embedded in a Web 2.0 environment. The Service-Finder Service Ontology<sup>4</sup> is publicly available under the Creative Commons Attribution-Noncommercial-ShareAlike 3.0 Unported License. This ontology has been extended to support supplementary information coming from the crawler. An additional feature is the classification of documents. Each document is tested for containment in a specific class and a score is assigned. This score determines how certain the containment is.

<sup>3</sup><http://www.service-finder.eu>.

<sup>4</sup><http://www.service-finder.eu/ontologies/sfo.rdfs>.

### 14.2.2 Search Engine

The crawling information is used by the front-end service search engine portal which is also developed and maintained by seekda. Services that are indexed also have their behavior monitored; such information can also be accessed through the search engine. Furthermore, services can be directly invoked through this search engine. This section describes such features and gives a brief walk-through on how to use the search engine.

Figure 14.6 shows the portal for the Web services search engine provided at <http://webservices.seekda.com>. The interface resembles that of a normal search engine but the layout of the search results is slightly different and constructed in a way to show a high level overview of the services' characteristics. Figure 14.7 shows a snapshot of the results obtained when searching using the keyword *SMS*. Results can be sorted by other criteria, such as provider and country. For each result entry, the following information is provided:

- Name of the service.
- Provider of the service.
- Country where the service is available.
- A short description of the service.
- Tags associated with the service.

When selecting a specific service, more information is given, which includes the location of the WSDL file (if applicable), the availability of the service (in a percentage form) and the user rating for the service. Users can also write their own textual descriptions to the service. These aspects are shown in Fig. 14.8.

When clicking on the *Availability* tab in Fig. 14.8, a graph similar to that shown in Fig. 14.9 is observed. What the graph shows is a more detailed description of the availability of the service. Among other things, the following are shown:

- *Average/connect time* in milliseconds until connection with the service could be established.
- *Average/response time* in milliseconds until response was completely received.
- *Connect errors and timeouts*.
- *Read errors and timeouts*.

As described earlier, services can also be directly invoked from the search engine interface. When clicking on *Use Now*, more details about the available operations are made available. When clicking on one of the operations, the user is presented with a form with input fields. These fields are actual parameters that are given to the operation (Fig. 14.10, left). Once invoked, the result of the operation is again presented to the user (Fig. 14.10, right).

### 14.2.3 Bundle Configurator and Assistant

As described earlier, one of the missions of seekda is to ease the creation of service bundles. This is done through the adaptation of existing technologies and through the use of intelligent problem solving methods and leveraging between static and

The screenshot shows the Seekda service search engine portal. The header features the Seekda logo (a grid of colored squares) and navigation links for login, register, home, help, contact, seek Services, News, Consumers, Providers, About, and a search bar.

**What is seekda?**

seekda™ is the free search engine for Web API (Web Services at this moment) and their providers.

We run a focused crawler to gather info about services from the Web. We daily monitor known services. But we also allow anyone to edit selected data such as details regarding providers or their services.

**seekda! connect**

**seekda! connect - the next generation multi booking solution for e-tourism**

**Get in Touch**

Questions or suggestions about seekda Web Services Search Engine? Contact us via mail. (no free agent available to chat)

**Service Tags**

business science USA development **free onsale** unknown **unkown** bioinformatics commercial what are tags?

**Web Services**

seekda's Web Services portal provides a direct access to a robust technology platform enabling search for public Web Services, and in the near future a one-stop-shopping marketplace for services offered by Providers from all around the world.

With Web Services technologies you enable, convert and enhance your applications to truly Web-based solutions. By using Web Services, you can plug-in and immediately use new components as easily as you already use software libraries today. The only difference in comparison to the

Fig. 14.6 Service search engine portal

The screenshot shows a web-based search interface titled "Web Service Search". The search bar contains the query "SMS". Below the search bar are links for "Advanced Search" and "Top 100 Tags". The main area is titled "Search Results" and displays "results 1 to 10 of 367" sorted by relevance. A navigation bar at the top right allows sorting by relevance (default), with a dropdown arrow icon. Below the navigation bar is a page number indicator from 1 to 9, with "1" highlighted.

**SMSVersand**

by [info-messenger.de](#)

**SMS Versand** Wichtig! Es müssen immer alle Felder übergeben werden sonst liefert der Service einen Fehler 500. (provider description)

Tags: [dsads](#), [company](#), [mobile messaging](#), [sms](#), [commercial](#), [dsaaa](#), [telecommunication](#), [commercial](#), [sms germany](#), [dsqdas](#)

**Send**

by [smscreator.de](#)

**SMS WebService** (provider description)

Tags: [sms](#)

**SendSMSWorld**

by [webservicex.com](#)

Send free SMS to a limited set of providers in certain countries which you can find here: <http://www.webservicex.net/WCF/ServiceDetails.aspx?SID=20> (community description)

Tags: [send](#), [company](#), [SMS](#), [global](#), [free](#), [country](#), [worldwide](#), [unlimited](#)

**SendSMSWorld**

by [webservicex.net](#)

Send unlimited free SMS to following countries which you can find here:

**Fig. 14.7** Example of search results

dynamic composition of services. At the static level, widely known notations (such as BPMN [7] and BPEL [1]) are adapted; at the dynamic level, seekda develops and implements composition algorithms inspired by the parametric design methodology [2, 5]. The bundling solution (called a *Service Bundler*) is characterized by dynamic discovery and binding of third-party Web Services. As new services emerge, consumption criteria for the existing ones change, and services already used can be

The screenshot shows the 'Web Service Details' page for 'SendSMSWorld'. At the top, there's a navigation bar with tabs: Overview (selected), Use Now, Availability, Comments, and Wiki History. A 'Bookmark' button is also present. Below the tabs, there's a list of service details:

- Country: United States (with a small USA flag icon)
- Provider: [webservicex.net](#)
- WSDL File: <http://www.webservicex.net/sendsmsworld.asmx?WSDL>
- WSDL Cache: XML or HTML
- Monitored since: Apr 13, 2007
- Server:
- Availability: ||||| (66.43% since Apr 2007) ([view details](#))
- Documentation: none (within WSDL)
- Description:
- User Rating: ★★★★★ (cast your vote by clicking the corresponding star)

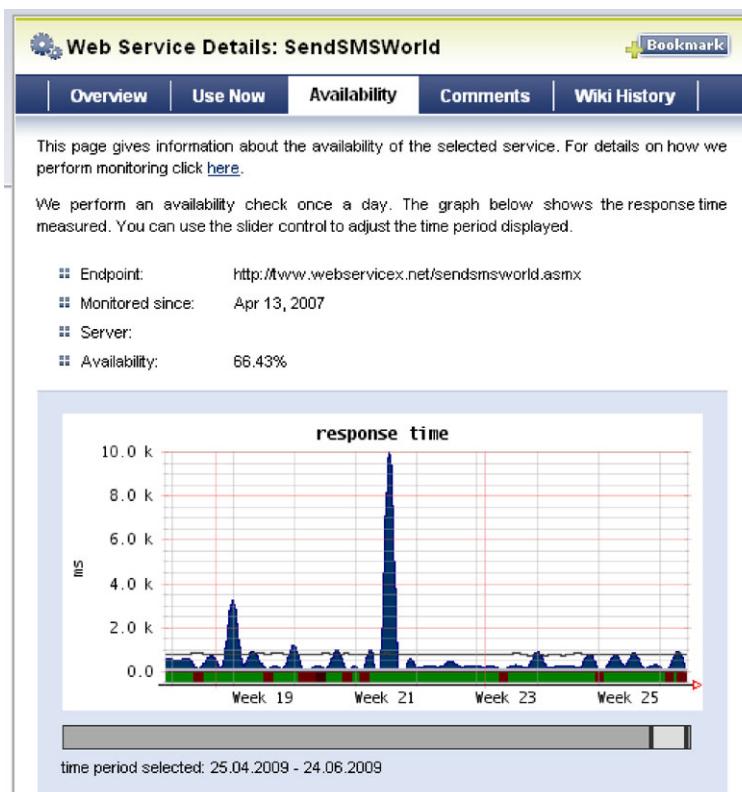
Below this, there's a section titled 'Service Description by seekda Users' with an 'edit' button. It contains a short description: 'Send unlimited free SMS to following countries which you can find here: <http://www.webservicex.net/WS/WSDetails.aspx?CATID=4>'. Under 'User Tags', there's a list of tags: [sly](#), [free](#), [company](#), [global](#), [\\_free](#), [unlimited](#), [worldwide](#), [SMS](#), [country](#). An 'add' button is located next to the tag list.

**Fig. 14.8** An example of a selected service from the search results

disposed of. By reconfiguration, the new criteria and constraints are met and new services are discovered and used.

The tool responsible for this is called the *Bundle Configurator and Assistant* and adapts a graphical process notation and plugs in with the parameter and constraint language. The graphical notation is simple enough to be used by business designers, but it also allows for the ability to express richer functionalities for product engineers. With respect to task fulfillment in the service bundles, current industry practices rely on keyword-based (manual) searching for available services from a given pool. Through the use of a parametric design methodology, the tool assists the designer to find adequate services that fulfill particular tasks given a specific set of parameters. This dramatically reduces the effort required to create service bundles, and consequently reduces cost and increases efficiency of manpower.

One of the main advantages of a template based approach is reusability [4, 6]. Templates are stored and reused in similar domains. The changes for templates within the same domain are generally minimal, and in many cases this boils down to different settings for parameters and peculiarities for the specific context in which the process has to be deployed. Not only does the provider of the service platform benefit from this (therefore, seekda itself) but so does the client since skeletons of processes (in the form of templates) are readily available and only need to be tailored for specific needs. Again, this results in less effort, and consequently, in a decreased burden of cost. Furthermore, the tool itself is generic enough to handle different



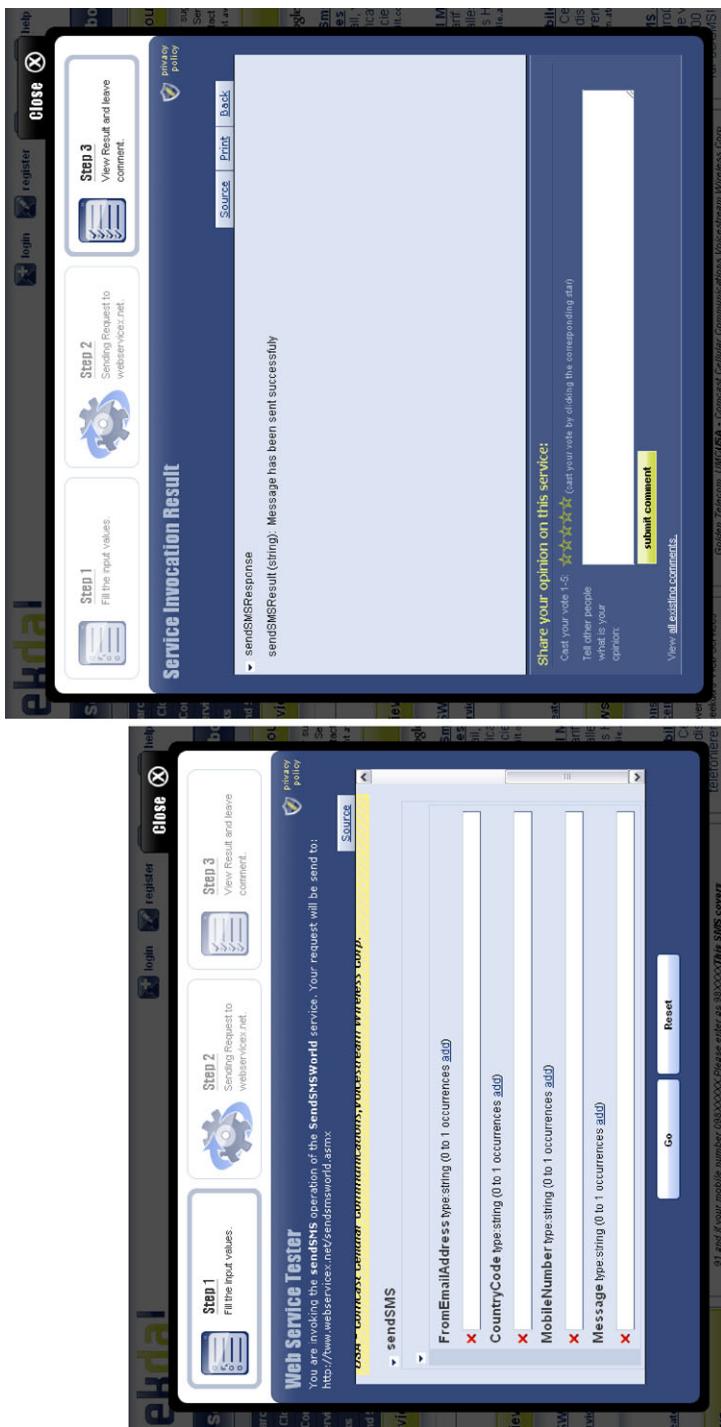
**Fig. 14.9** Availability of a service in graphical form

kinds of scenarios—beyond e-Tourism, further application of these methodologies in the area of e-Conference and e-Messaging are envisioned. This is different from current approaches in the industry where tools usually are tailored for specific domains. All-in-all, a high degree of flexibility is expected to be achieved by making the generation of service bundles efficient.

Considering the template based approach, a stepwise approach for creating service bundles is taken. These phases are as follows:

1. Template Design
2. Template Refinement
3. Enactment
4. Execution

Figure 14.11 shows a graphical representation of the different service bundling phases. The process starts with the *Template Design* phase during which the Business Person identifies the main components needed to perform a particular task. Along with the main components of the bundle, parameters must be identified and properly defined. Various individuals in the company have access to different types



**Fig. 14.10** Invoking a service operation (*left*) and result of the operation (*right*)

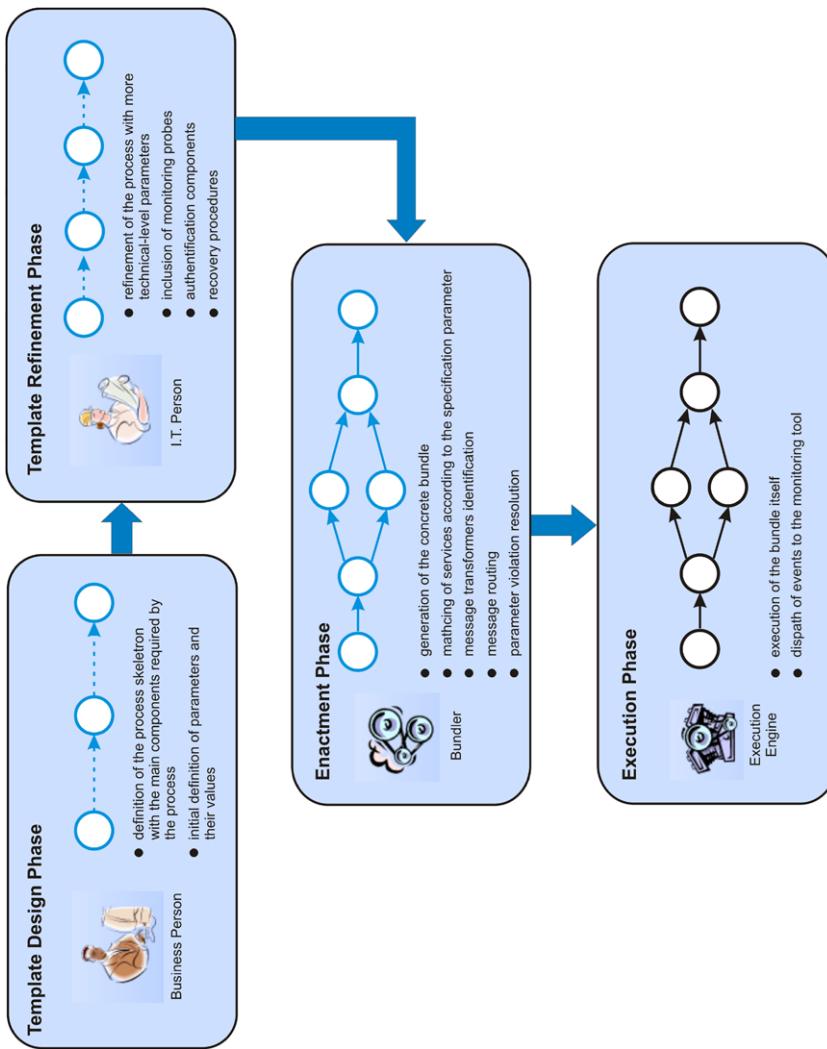


Fig. 14.11 Phases of the bundling of services

of parameters. For an e-Tourism process (described in more detail later in this section), we require publishing information about availability of rooms in a hotel across several booking channels. The business person is interested in defining with which channels this hotel should be registered, which is not really the case for the technical (or IT) person. The system therefore also defines access rights at the level of categories of parameters which can be edited when creating templates.

The following is the *Template Refinement* step. During this phase, the template is further refined with more technical details—these usually must be edited by an IT person. Such details include the definition of specific related parameters to allow for satisfactory execution of the bundle (for example, fault handling operations). Specific types of parameters—accessible by a technical person—are also further defined during this phase.

The *Enactment Phase* is at the heart of the whole bundle creation process, during which templates are fulfilled with actual services. Apart from identifying the required services that satisfy the specified parameters, the appropriate message mappings must be identified together with routings for these messages through the different control constructs of the process. Another important aspect is the resolution of violated parameters. Typically, a violation of a parameter does not necessarily imply that the potential service is completely neglected, but that there might rather be a specific action that can be done on the data to make it acceptable by the service. In an e-Tourism scenario, a specific channel may require that a picture of the hotel is of some particular size. If the size is greater than that required, a possible action can be to crop the picture appropriately—this can also be done by a specific service available on the Web.

Finally, the last step is the *Execution Phase*. This phase takes the concrete instantiated bundle (in the form of some specific execution language like BPEL) and executes it.

Before looking into the details of the Bundle Configurator and Assistant, a few words need to be said on mediation. As described earlier, current seekda products tackle mainly an e-Tourism scenario (more in Sect. 14.3). In this setting, seekda acts as a broker between the hoteliers and the advertisement and search services (for example, Expedia). Service data models vary, consequently mediation and transformation between such data models have to be catered for. Figure 14.12 gives an overview of how mediation is approached in seekda.

The datamodel used by seekda (and its products) is OTA—Open Travel Alliance.<sup>5</sup> OTA defines a schema that describes elements in the traveling domain, such as hotels, flights and tickets. With this schema as the underlying datamodel for the configurator tool, the datamodels of the services must be mapped to it. Therefore, as a first step, a domain expert must map the OTA elements to target channel services (for example, Expedia) and store these mappings. When a hotelier would like to publish the service on different advertising platforms, all the hotelier needs to do is to provide a mapping from its data model to OTA—the rest is taken care of by the bundle configurator (or other tools). At runtime, the incoming message is therefore

---

<sup>5</sup><http://www.opentravel.org>.

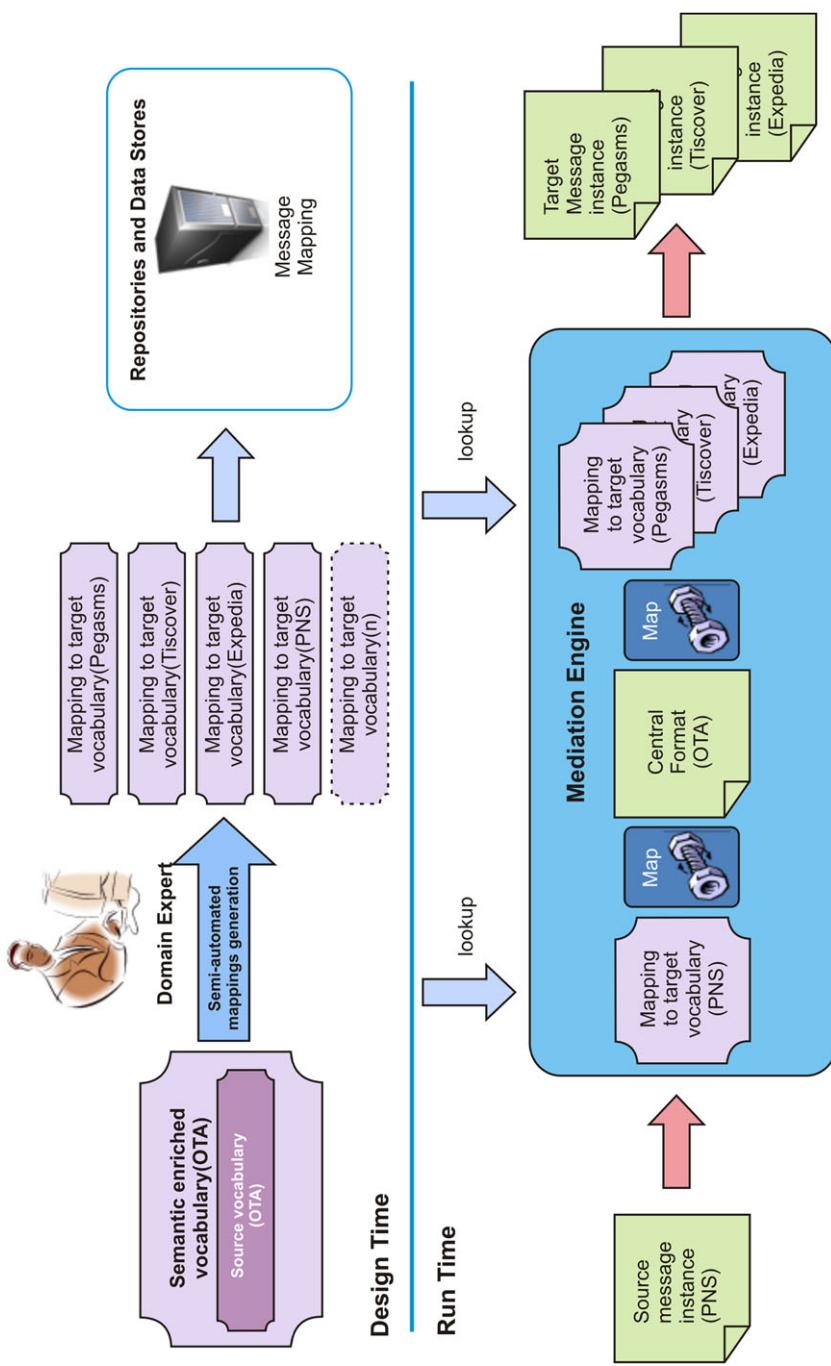


Fig. 14.12 Mediation architecture

transformed to OTA which, in turn, is transformed to the data model of each specific advertising platform (depending upon where the hotelier wants to make the advertisement). In terms of architecture, the key idea here is to store the mappings and reuse them. Furthermore, when a new Hotelier asks for such a similar service, only one mapping is needed. Business-wise, this reduces the costs for hoteliers, speeds up advertisement and updating of room availability, and makes it easier to make use of more advertising services. More about this is discussed in Sect. 14.3.

The main components of the Bundle Configurator and Assistant tool are illustrated in Fig. 14.13. The *Configurator* comprises two main components, namely the *Graphical Interface* and the *Enactment Engine*. The component has access to the *Template*, *Message Mappings* and *seekda Service* repositories.

The *Graphical Interface* is the main entry point for the creation of templates. The *Template Editor* enables graphically defining a classical process (using the *Process Editor*) and additionally specifying parameters that are required to be fulfilled by the desired services (through the *Parameter Editor*). The former typically resembles a normal workflow editor, with additional elements required for the system's needs; the latter allows the creation of parameters based on a specific language, allowing also actions to be taken in case of parameter violations.

The *Enactment Engine* consists of the *Composer* and the *Message Mapping Finder*. The Composer is responsible for finding appropriate services that match the requested parameters; this is done through the *Parameter Resolution* component. If services are required to be connected with one another, then the *Message Router* comes into play to route the appropriate messages between them through the required control flow constructs. The *Message Mapping Finder* is used to identify the required mappings between the different types of the schema of the services.

### 14.3 Illustration by a Larger Example

In this section, we describe one of the main products provided by seekda, namely *seekda! connect*. As briefly pointed out in Sect. 14.2.3, the core idea is for seekda to act as intermediary between a hotelier and an advertising service. The domain is therefore the same, but each hotelier has specific requirements and constraints which determine, amongst other things, which advertising channels are to be used. Figure 14.14 shows an overview of the basic idea for seekda! connect. Consider a hotelier who would like to advertise the hotel on different channels. Traditionally, the hotelier is required to go and connect with every distributing channel in order to advertise the hotel. Hence, a lot of effort is required:

- A translation between the data model of the hotelier and the data model of every channel is needed.
- Given a change in the hotelier's data model, every translation has to be changed (consequently, this approach is not so scalable).
- Different service level agreements must be maintained.
- More costs are incurred to maintain these connections between different channels leading to more drawbacks:

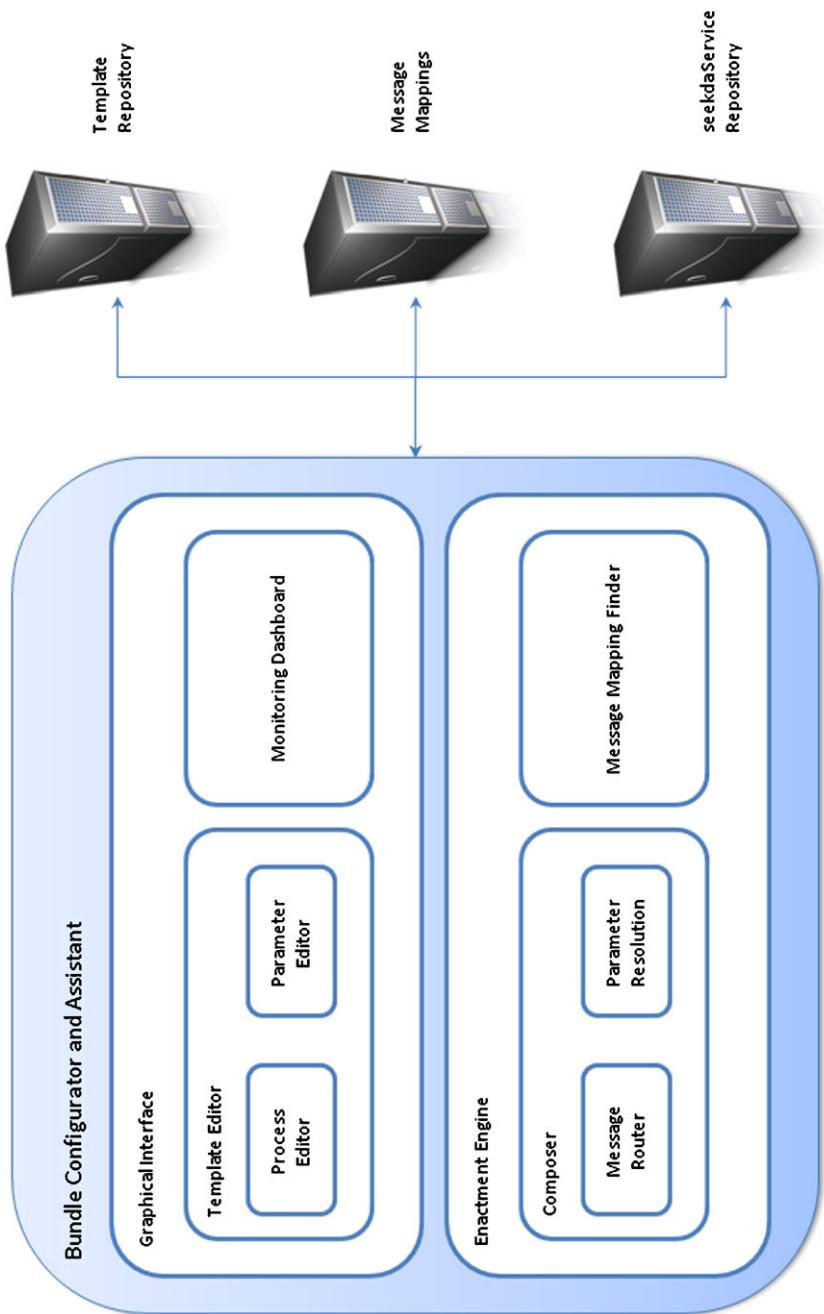
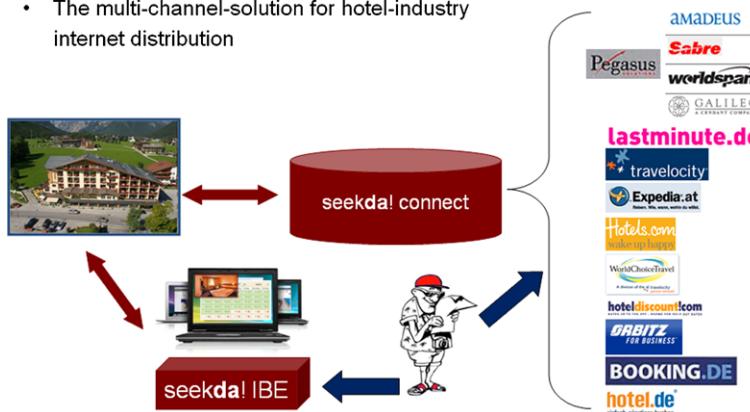


Fig. 14.13 High level architecture of the bundle configurator and assistant tool

- The multi-channel-solution for hotel-industry internet distribution



**Fig. 14.14** *seekda! connect* overview

- Due to the substantial degree of costs, it is very likely that the number of distributing channels is low, and consequently the hotel is less reachable by end users.
- Due to the higher costs and less reachability by end users (and consequently, fewer customers), the profit margin is lower.

*seekda! connect* aims at overcoming the above bottlenecks. A hotelier is no longer required to go to every distributing channel, but rather uses *seekda* to connect to them. The system therefore serves as a broker between the hotelier and the different distributing channels. Different distributing channels—and their associated options—are provided to the hotelier in one single stop. The system uses OTA as the underlying data model, and consequently, only one mapping between the hotelier's data model and OTA is needed. The rest of the mappings are provided by *seekda*. Furthermore, hoteliers are no longer required to go through every channel and compare options therein. Rather, it is all conveniently presented and compared—the hotelier simply chooses which channels are desired, in a similar fashion as that of using a shopping cart on a typical e-Commerce Web site. During this process, the system—thanks to intelligent methods described in previous sections—assists the hotelier in configuring the components of the advertising process and the channels.

Given this approach, the costs for the hotelier are drastically reduced, a higher degree of distribution over the channels is achieved, and consequently, more end-users can be reached. This is also beneficial for the distribution channels themselves as more hoteliers are attracted to use their services. We will now go step by step through this system and point out some of the technical peculiarities in the process. We will describe the setup of a hotel called *Hotel Erfolg*. Initially, the hotelier logs in to *seekda! connect* and sets up the general properties of the hotel (Fig. 14.15). There are various other aspects that can be described at this stage, for example, amenities, rooms, policies, and seasons of operation. We will look at the rooms setup.

First, the rooms are created and given an identifier. After this, their particular features are described (Fig. 14.16).

**seekda!** connect

Welcome, cybage@cybage.com | Change Password | Logout

Manage Hotel - Hotel Erfolg

Hotel Rates Availability Channel Reservations IBE Manager User manager Web Page

General Amenities Description Pictures Policies Rooms Seasons Terms & Conditions

### Hotel - General

#### Accommodation details

Hotel name *	Hotel Erfolg Austria	
Hotel short name	Hotel Erfolg	
Web address		
Segment	First class	
Selected property types	Conference Center	
Year of construction	2000	
Year of renovation	2009	
VAT number	ATU 64922999	
Currency code	EUR	
Latitude	47.26	NORTH
Longitude	11.39	EAST
Awards	-Select-	

**Fig. 14.15** General properties for Hotel Erfolg

Following the room creation step is the setup of the rates. This is done through a calendar system. Amongst the properties are the setup of rates (for each possible number of guests in the room), adult and child pricing, as well as minimum and maximum number of guests (Fig. 14.17). Note that all these configuration parameters are possible input to the bundle configurator as described previously—an end user may use these types of configurations to find specific hotel stays.

In the availability view, the hotelier is able to access the rooms that are available in the hotel (Fig. 14.18). Note that these are automatically updated with reservations and cancellations.

Note that all of the information shown above is stored and described in XML using the OTA data model. Furthermore, all of this information can be mediated and exported to different distributing channels. This is one of the core parts of *seekda! connect*, integrating with other distributing channel systems such as the hotelier who does not have to go through the burden to replicate all the information on different channels; this is done automatically by the system. In our particular example, *Expedia* and *Hotel.de* are two distributed channels that the hotelier has enabled (Fig. 14.19).

Having this data setup, normal end users can now simply look for rooms either on the distributing channels, or through the hotelier's Web site. Figure 14.20 shows how the search widget of the system is embedded within the hotel's Web site. From

View Room type name - SGDELUXE

### Hotel - Rooms

Room Code *	SGDELUXE	
Occupancy	Standard * 1	Maximum * 1
Room size (square meter)	27	
Total number of rooms	50	
Room website		
Non-smoking room	<input checked="" type="radio"/> Yes	<input type="radio"/> No
Bed type	Single	
Selected amenities	227 Premium movie channels , Television	

### Room Name

Room type name *	Single Room Deluxe
------------------	--------------------

### Room Description

Room description	EN	Suite with 27 sqm for 1 person: bedroom with living area, bath or shower, toilet,
------------------	----	-----------------------------------------------------------------------------------

Fig. 14.16 An example of a room setup

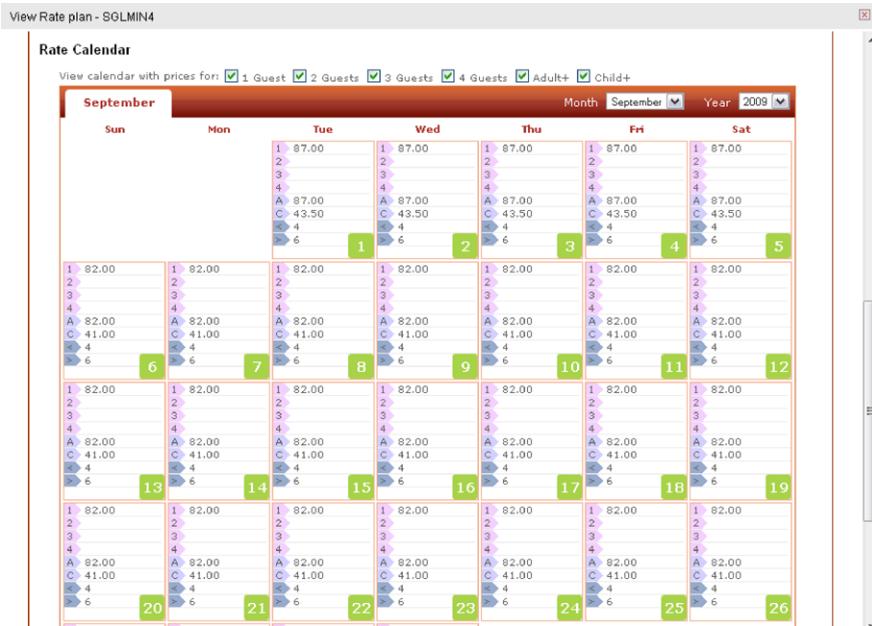


Fig. 14.17 Rates setup for a room in Hotel Erfolg

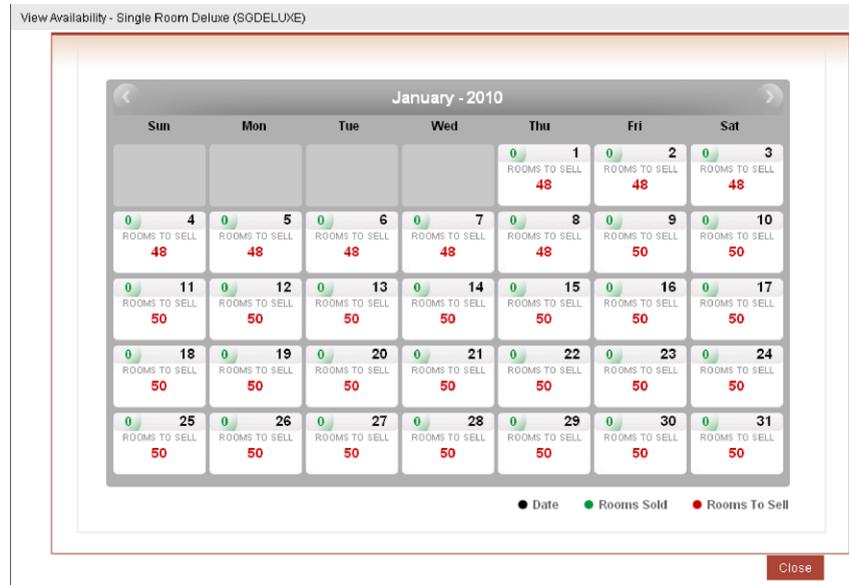


Fig. 14.18 Availabilities view for a room

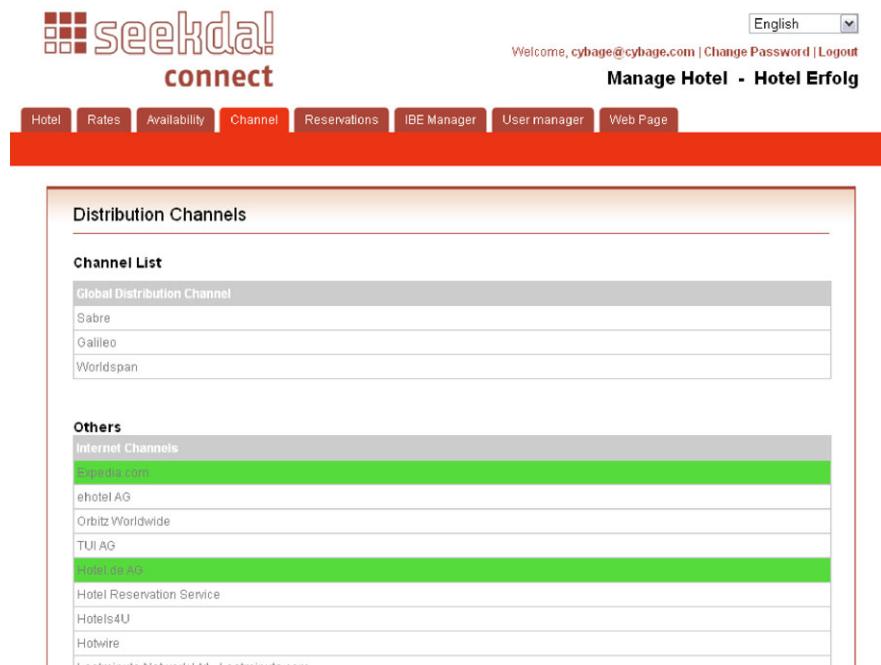
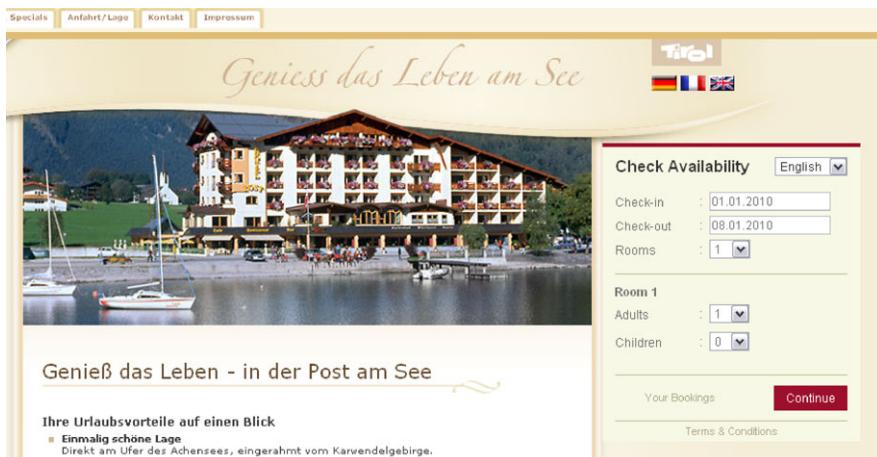


Fig. 14.19 Enabled channels for Hotel Erfolg



**Fig. 14.20** Embedded *seekda! connect* search widget on Hotel Erfolg

that point on, the end user simply selects an available room in a standard fashion (Fig. 14.21).

Our future work is aimed towards more flexibility in using the aforementioned techniques. Our vision is depicted in Fig. 14.22 which shows how an e-Tourism template is constructed using *seekda! connect*. The example already contains a few of the parameters and constraints that could be part of this template. These may include:

- Setting variable number of rooms to advertise could depend on the channel.
- Advertising availability of room categories might depend on the specific channel.
- Dealing with proactive notifications when a room is booked.
- Handling faults.
- Transforming data.

At the template level, the process is very simple. As a start, a *Product Engineer* describes the process in its primitive form. For example, the start and end nodes, and the checking and advertise tasks, as well as additional services (booking the car, booking the skip pass, paying for a service). These tasks provide a very high level description of what the process should do. Additional constraints may be specified by the product engineer or by other entities. For example, a process engineer specifies that these actions must be performed in an authorized way. Next, a *customer* (a hotelier, in this case) may further specify constraints and other options. This idea will be developed in the form of a dynamic shop, whereby third party service providers will also be able to register with the system and have their services configured to work with specific hotels.

The screenshot shows a hotel booking interface. At the top, there's a banner with the text "Geniess das Leben am See" and the Tirol logo. Below the banner, the page title "Room Selection" is visible, along with language options "English" and "ch". A navigation bar includes tabs for "Check availability", "Room selection" (which is selected), "Personal information", "Payment", and "Booking confirmation".

Search parameters are listed: Check-in 01.01.2010, Check-out 08.01.2010, Room 1, Stay 7 Nights, Guest 1 Adult. Below this, a sorting option "Sort result by" allows choosing "Alphabet" or "Lowest price" (which is selected).

A heading "The following rooms meet your search criteria" is followed by "(Adult 1 : Child 0)".

The first result is "Doppelzimmer typ D", shown with a thumbnail image of a double room, a "View room details" link, and a price of €770.00.

The second result is "Doppelzimmer typ C", shown with a thumbnail image of a double room, a "View room details" link, and a price of €1,638.00.

Navigation arrows at the bottom of each room listing allow for viewing more results.

**Fig. 14.21** A search for a hotel stay using the embedded widget on Hotel Erfolg

## 14.4 Summary

In this chapter, we described the technologies employed by seekda to enable a true Web of services. As a first step, seekda provides a crawler that looks for services (both WSDL and REST), classifies them into an ontology, creates RDF metadata triples (if possible) and indexes them. Next, a search portal enables, to look for such services, to view monitoring data related to the services, and possibly invoke them. Finally, a bundle configurator tool enables the creation and mediation of service bundles. Such bundles are configured in a stepwise manner and automatically enacted to create an executable process that can be deployed and used by a third-party. These technologies have been illustrated by an example from the *seekda! connect e-Tourism* product.

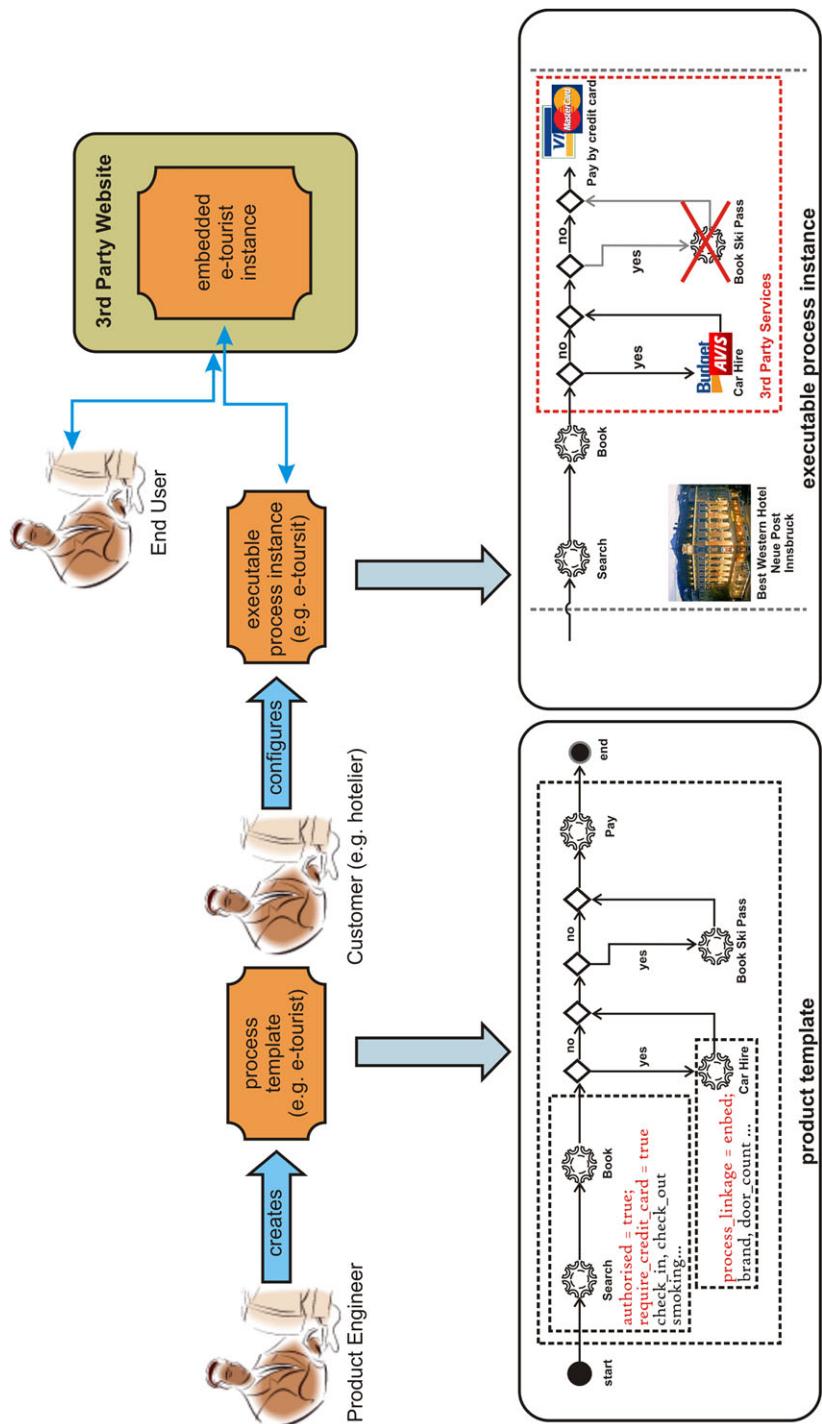


Fig. 14.22 seekda! connect template-based composition

## 14.5 Exercises

**Exercise 1** Describe the *Register*, *Find* and *Bind* paradigm and the entities involved.

**Exercise 2** Give at least three bottlenecks that are hampering a true Web of services in the current Internet.

**Exercise 3** List and briefly describe the 3 main components that are provided by seekda to ease the find and composition of services.

**Exercise 4** Describe the Crawling process.

**Exercise 5** Apart from searching, what are the two other features provided by the seekda Web service search engine?

**Exercise 6** Which methodology does the service bundler use for composition?

**Exercise 7** List and describe the four phases for creating and using service bundles.

**Exercise 8** List at least two advantages created by the mediation architecture and the *seekda! connect* product.

## References

1. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbura, F., Ford, M., Goland, Y., Guzar, A., Kartha, N., Liu, C.K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language, Version 2.0 edn. OASIS (2006). <http://www.ibm.com/developerworks/webservices/library/ws-bpel/>. OASIS. Public Review Draft, <http://docs.oasis-open.org/wsbpel/2.0/>
2. Fensel, D.: Problem-solving Methods: Understanding, Description, Development, and Reuse. Lecture Notes in Computer Science, vol. 1791. Springer, Berlin (2000)
3. MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R.: Reference model for service oriented architecture 1.0. Tech. rep., OASIS (2006)
4. Mandell, D., McIlraith, S.: Adapting BPEL4WS for the Semantic Web: the bottom-up approach to web service interoperation. In: Proc. of ISWC'03 (2003)
5. Motta, E.: Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving. IOS Press, Amsterdam (1999)
6. Narayanan, S., McIlraith, S.: Simulation, Verification and automated composition of Web services. In: Proc. WWW'02 (2002)
7. OMG: Business Process Modeling Notation—Bpmn 1.0. <http://www.bpmn.org/>. Final Adopted Specification, 6 February 2006
8. Steinmetz, N., Lausen, H., Brunner, M.: Web service search on a large scale. In: Proceedings of the 7th International Joint Conference on Service Oriented Computing (ICSOC 2009) (2009)



# Index

## A

A2A Integration, 174  
Abstract State Machines (ASM), 115, 134, 179, 205  
Adaptation, 163, 173  
Adoption, 238  
Advanced Research Projects Agency (ARPA), 11  
American Standard Code for Information Interchange (ASCII), 90  
Anonymous identifier, 137  
Application Programming Interface (API), 39, 48, 165  
Asynchronous JavaScript and XML (AJAX), 77, 78  
Atom Entry Document, 78  
Atom Feed Document, 78  
Atom Publishing Protocol (AtomPub), 78, 79  
Atom Syndication Format, 78  
Axiom, 113

## B

B2B, 16, 20, 37, 55  
B2B Integration, 174, 200  
B2C, 16, 20, 37  
Blogosphere, 5  
Business Process Execution Language (BPEL), 39, 164, 191, 269  
Business Process Execution Language for Web Services (BPEL4WS), 167

## C

Cascading Style Sheets (CSS), 77  
Choreography, 172  
in IRS, 273, 275  
Choreography Engine, 179, 275  
Closed-World Assumption (CWA), 132

## Communication

Composition, 163, 173, 180, 238  
Customer Relationship Management (CRM), 200, 202

## D

DARPA Agent Markup Language (DAML), 120  
Datalog, 135, 153  
DBpedia, 99  
Delicious, 88  
Description Logic Programs (DLP), 135  
Description Logics (DLs), 97, 132–135, 141, 144, 158, 223, 261  
SHIQ, 135  
Discovery, 163, 172, 176, 185, 207, 238  
in IRSIII, 273  
Document Object Model (DOM), 74, 77  
Document Type Definition (DTD), 91, 101  
Dublin Core (DC), 109, 110, 138

## E

E-Commerce (Electronic Commerce), 16, 19, 20, 37  
E-Service, 26–34  
E-Tourism (Electronic Tourism), 119  
EBusiness, 5  
EbXML, 50, 53, 59  
Registry, 54, 55  
Registry Information Model (RIM), 54  
Registry Services and Protocols (RS), 54  
Electronic Data Interchange (EDI), 16  
Electronic Funds Transfer (EFT), 16  
Enterprise Application Integration (EAI), 37, 174  
Entity-Relationship (ER), 98

- Execution Management, 172  
 Execution Semantics, 192  
   achieveGoal, 198  
   getWebServices, 196  
   invokeWebService, 196  
 Extensible HyperText Markup Language (XHTML), 77  
 Extensible Markup Language (XML), 39, 43, 45, 51, 54, 67, 68, 70–73, 76–78, 83, 89, 91, 92, 94–96, 101, 108, 119, 135–137, 164, 168  
 External Communications Manager, 182
- F**  
 F-Logic, 135  
 Facebook, 88  
 Fault Handling, 173  
 File Transfer Protocol (FTP), 11  
 First Order Logic (FOL), 132  
 Flickr, 72, 80  
 Future Internet, 3, 238
- G**  
 Geonames, 99  
 Goal, 179, 261, 272, 273  
 Google, 88  
 GRDDL, 93  
 Grounding, 173
- H**  
 Horn Logic (HL), 135, 144  
 HRESTS, 6, 118, 119  
 HyperText Markup Language (HTML), 12, 72, 76, 78, 88, 99  
 Hypertext Transfer Protocol (HTTP), 12, 39, 44, 45, 47, 57, 68–70, 78, 88, 164
- I**  
 Internationalized Resource Identifier (IRI), 110, 136, 145  
   Compact IRI, 136  
 Internet Information Services (IIS), 34  
 Internet Reasoning Service, 270  
   Architecture, 271  
   IRS-III, 271  
 Invocation, 163  
 IRIS, 153
- J**  
 JavaScript, 73–75, 77, 78, 83, 84  
 JavaScript Object Notation (JSON), 68, 72–75, 77, 81, 83
- K**  
 KAON2, 153
- L**  
 Linda, 221, 223, 225, 227, 228, 242  
 Linked Data, 5, 98, 99, 101  
 Linked Open Data (LOD), 99, 101  
 LinkedIn, 88  
 Logic Programming (LP), 133–135, 142, 143, 158
- M**  
 Mediation, 109, 173  
   Data Mediation, 164, 177, 185, 208, 273  
   Process Mediation, 164, 186, 208, 209, 273  
 Mediator, 139  
   GGMediator, 117, 120, 124, 275  
   OOMediator, 117, 124, 273  
   WGMediator, 117, 124, 150, 273, 275  
   WWMediator, 118, 124  
 Message Exchange Pattern (MEP), 43  
 Meta-Object Facility (MOF), 109–112, 272  
 METEOR-S, 263  
   MWSAF, 264  
   MWSCF, 268  
   MWSDI, 268  
   WSDL-S, 264  
 MicroWSMO, 5, 6, 118, 119, 155  
 MINS, 153  
 Mobile Web, 3  
 Monitoring, 163, 173
- N**  
 Negotiation, 163  
 Netscape, 88
- O**  
 OCML, 271  
 Ontology, 96, 108–110, 112, 116, 117  
 Ontology Mapping, 206  
 Open World Assumption (OWA), 132, 133  
 Orchestration, 115, 179, 204, 275  
   in IRS, 275  
 Orchestration Engine, 179  
 Order Management System (OMS), 200, 202, 209  
 OWL-S, 5, 6, 109  
   Process Model, 262
- P**  
 Partner Interface Processes (PIPs), 200  
 PELLET, 153

**Q**

Qualified Name ( QName ), 136, 137  
Quality of Services ( QoS ), 204

**R**

Ranking, 177, 238  
    Ranking Interface, 177  
RDF N-Triple, 93  
RDF Notation 3 ( N3 ), 93  
RDF Schema, 94–96, 101, 119, 134, 135, 138, 139, 223  
RDF Turtle, 93  
RDF/XML serialization, 93  
RDFa, 6, 93  
Reasoner, 185  
Reasoning, 168, 173, 238  
Reification, 93  
Resource Description Framework ( RDF ), 92–96, 98, 99, 101, 135, 136, 142, 221, 223, 225, 227, 229–231, 234, 236, 237, 244–246  
REST, 6, 68, 69  
RESTful Service, 68–70, 72, 77–80, 83, 84, 118, 220, 240  
RosettaNet, 179, 200, 209  
Rule Interchange Format ( RIF ), 95, 98, 101, 155, 158  
    RIF-BLD, 98  
    RIF-Core, 98  
    RIF-FLD, 98  
    RIF-PRD, 98

**S**

SA-REST, 6  
Secure Sockets Layer ( SSL ), 34  
Security, 172  
Service, 242  
Selection, 207, 238  
    in IRSIII, 273  
Semantic Annotations for WSDL ( SAWSDL ), 5, 6, 118, 119, 263, 267  
Semantic Execution Environment ( SEE ), 231  
Semantic Web, 6, 90, 91, 94–96, 98, 101, 108, 109, 133, 220, 222, 223, 225, 242  
Semantic Web Layer Cake, 90, 101  
Semantic Web Rule Language ( SWRL ), 133  
Semantic Web Services ( SWS ), 5–7, 107–110, 118, 119, 124, 127, 128, 131, 153, 163, 231, 241, 243, 247  
Semantically Enabled Service Oriented Architecture ( SESA ), 128, 163, 175, 200, 238  
    Middleware Layer, 172  
    Base Layer, 173

Broker Layer, 172

Vertical Layer, 172  
Problem Solving Layer, 171  
Service Providers' Layer, 173  
Service Requesters' Layer, 172  
Stakeholders' Layer, 171  
Service, 26, 41, 109  
Service Cloud, 238, 240–242  
Service Delivery Platform, 29, 237  
Service Level Agreements ( SLA ), 204  
Service Oriented Architecture ( SOA ), 30, 38, 39, 55, 56, 163, 237, 240, 241  
Service Oriented Computing ( SOC ), 4, 38, 39, 56, 164  
Service Science, 25–27, 29, 34, 35  
Service Web, 238, 242  
SharePoint, 34  
Simple API for XML ( SAX ), 77  
Simple Mail Transfer Protocol ( SMTP ), 47, 57  
Simple Object Access Protocol ( SOAP ), 6, 39, 44–47, 50, 53, 56, 57, 67, 68, 107, 164, 167, 168, 184, 220, 231, 246  
SPARQL, 93–96, 101, 227  
Standard Generalized Markup Language ( SGML ), 76, 91  
Storage, 173  
SWS Challenge, 200  
System Behavior Modeling, 192

**T**

Transmission Control Protocol/Internet Protocol ( TCP/IP ), 11  
Triple Space, 220–234, 236–238, 240–242, 244–247  
Triples Space, 247  
Tuple Space, 190, 220–223

**U**

Unicode, 90, 91  
Unified Modeling Language ( UML ), 98, 193  
    Activity Diagrams, 193  
Unified Service Description Language ( USDL ), 6  
Uniform Resource Identifier ( URI ), 12, 13, 67, 68, 70, 88, 91–93, 98, 99, 101, 108, 136, 220, 225  
Uniform Resource Locator ( URL ), 91, 226  
Uniform Resource Name ( URN ), 13, 91  
Unique Names Assumption ( UNA ), 133  
Universal Description Discovery and Integration ( UDDI ), 42, 47, 48, 56, 57, 67, 107, 267, 268

- Universal Description Discovery and Integration (UDDI) Registry, 42, 56, 57
- V**
- Virtual Travel Agency (VTA), 55, 81, 119, 155, 242
- Vocabulary of Interlinking Datasets (voiD), 101
- W**
- Web 2.0, 22, 88, 89, 238
- Web Application Description Language (WADL), 68, 70, 71, 81, 83
- Web Architecture, 14, 15
- Web Ontology Language (OWL), 94–98, 101, 132–135, 138, 141, 153, 158, 223
- OWL 2, 97
  - OWL DL, 97, 135, 139
  - OWL Full, 97
  - OWL Lite, 97
  - OWL2EL, 97
  - OWL2QL, 97, 98
  - OWL2RL, 97
- Web Science, 9, 10, 22, 25
- Web Science Research Initiative (WSRI), 9, 10
- Web Service, 5, 6, 37, 38, 40, 41, 45, 51, 55, 56, 67, 68, 107, 109, 131, 134, 179, 220, 221, 225, 231, 238, 240, 241, 272
- Web Service Description Language (WSDL), 6, 39, 42, 43, 45, 47, 52, 56, 57, 67, 70, 107, 118, 147, 164, 203
- Web Service Description Language (WSLD), 231
- Web Service Execution Environment (WSMX), 6, 108, 109, 128, 163, 187, 231, 233, 234, 236, 237, 242–247
- Communication and Coordination Component, 189
- Core Component, 187, 234
- Grounding Component, 232
- Invoker Component, 232
- Management Component, 188
- Manager Component, 236
- Resource Manager Component, 186, 237, 243
- Web Service Modeling Framework (WSMF), 6
- Web Service Modeling Language (WSML), 6, 108, 119, 128, 133–139, 141, 147, 158, 202, 236
- Annotations, 138, 148
- Atom, 143
- Attribute, 139
- Axiom, 142
- Capability, 142
- Concept, 139
- Conceptual Syntax, 136
- Datatypes, 137
- Header, 138
- importsOntology, 138, 150
- Instance, 141
- Logical Expression, 142
- Logical Expression Syntax, 136
- Message Exchange, 147
- Molecule, 143
- Prologue, 137
- Relation, 141
- State Signature, 147
- Term, 143
- Transition Rule, 147
- usesMediator, 139
- Variable, 142
- WSML-Core, 135, 136, 141, 144, 155
- WSML-DL, 135, 136, 141, 144, 155
- WSML-Flight, 135, 137, 144
- WSML-Full, 135, 144
- WSML-Quark, 155
- WSML-Rule, 135, 144, 155
- Web Service Modeling Ontology (WSML), 237
- Web Service Modeling Ontology (WSMO), 5, 6, 108–111, 116, 119, 120, 124, 127, 128, 131–134, 153, 155, 202, 231, 243
- Assumption, 146, 150, 151
  - Capability, 120, 145, 150, 203
  - Choreography, 115, 147, 151, 179, 204, 272
  - Effect, 146, 150, 151
  - Goal, 109, 116, 117, 120, 149
  - Interface, 115, 145, 147, 151
  - Mediator, 109, 111, 116, 152
  - Non Functional Properties, 114, 148, 177, 204
  - Ontology, 139, 142
  - Orchestration, 147, 208
  - Postcondition, 146, 150, 151
  - Precondition, 146, 150
  - Web Service, 109, 111, 113, 116–118, 120, 145, 146, 149, 152
- Web Service Modeling Toolkit (WSMT), 152, 153, 175, 203
- Web Services Programming Model, 39, 165
- Web Tools Platform (WTP), 153
- World Wide Web (WWW), 12, 14, 68, 76, 87, 108, 221, 222, 224

WS-Addressing, 50, 59  
WS-Policy, 50–52, 59  
WS-Security, 50, 53, 59  
WSML2Reasoner, 153  
WSMO-Lite, 5, 6, 118, 119, 155  
WSMO4J, 185, 186, 243

**X**

XML Schema, 43, 91, 92, 101, 136, 137, 203,  
220

XPath, 94  
XQuery, 95, 136  
XSL Transformation, 206  
XSL Transformations (XSLT), 77, 89

**Y**

YouTube, 88