State of the art of current Semantic Web Services initiatives

Lyndon Nixon and Elena Paslaru (FU Berlin)

with contributions from:
Michal Zaremba (NUIG), Enrica Dente (NUIG), Rubén Lara (UIBK), Walter Binder (EPFL), Ion Constantinescu (EPFL), Radu Jurga (EPFL), Vincent Schickel-Zuber (EPFL), Vlad Tanasescu (EPFL), Mark Carman (UniTn), Loris Penserini (UniTn), Marco Pistore (UniTn)

reviewed by: Emilia Cimpian (NUIG)

Abstract.
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB
Internal Deliverable D1 (WP2.4)
This internal deliverable serves as an introduction to the activities in the fields of Web Services and Semantic Web Services. It provides a survey of the state of the art of current Semantic Web Services initiatives and an analysis of these initiatives to identify semantic needs not covered within existing research efforts. The document shall act as an input to the public deliverables of the Semantic Web Services work package as well as to the other research work packages and to the KnowledgeWeb project as a whole.
Keyword list: Web Services, Semantic Web Services

Document Identifier	KWEB/2004/WP2.4ID1/v1
Project	KWEB EU-IST-2004-507482
Version	v1.0
Date	29.June, 2004
State	final
Distribution	internal

Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

University of Innsbruck (UIBK) - Coordinator
Institute of Computer Science
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

École Polytechnique Fédérale de Lausanne (EPFL)
Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

France Telecom (FT)
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Freie Universität Berlin (FU Berlin)
Takustrasse 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l'Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

National University of Ireland Galway (NUIG)
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

The Open University (OU)
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

University of Karlsruhe (UKARL)
Institut für Angewandte Informatik und Formale Beschreibungsverfahren - AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Liverpool (UniLiv)
Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Manchester (UoM)
Room 2.32. Kilburn Building, Department of Computer Science, University of Manchester, Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

University of Trento (UniTn)
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Amsterdam (VUA)
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to writing parts of this document:

Centre for Research and Technology Hellas
École Polytechnique Fédérale de Lausanne
France Telecom
Freie Universität Berlin
Institut National de Recherche en Informatique et en Automatique
National University of Ireland Galway
Universidad Politécnica de Madrid
University of Innsbruck
University of Karlsruhe
University of Liverpool
University of Manchester
University of Trento

Changes

Version	Date	Author	Changes
0.01	14.04.04	Lyndon Nixon, Elena Paslaru	Proposed bare structure
0.05	23.04.04	Lyndon Nixon, Elena Paslaru	Draft text from FU Berlin
0.06	03.05.04	Ruben Lara	Comments on previous version
0.1	10.05.04	Lyndon Nixon	Revision of v0.05 and contributions from other partners added
0.2	24.05.04	Elena Paslaru	Transformation to Latex
0.3	26.05.04	Lyndon Nixon	Revision of text
0.4	07.06.04	Ruben Lara	Addition of chapter 4
0.45	08.06.04	Elena Paslaru	Added glossary, style changes
0.46	10.06.04	Lyndon Nixon	Edited chapters 2.3 and 3.2
0.5	15.06.04	Lyndon Nixon	Completed conclusion and executive summary
0.7	21.06.04	Enrica Dente, Ruben Lara	Final version of chapter 3.3
0.8	25.06.04	Walter Binder, Ruben Lara	Further editing
0.9	28.06.04	Lyndon Nixon	Minor corrections and cleaning up of citations
1.0	29.06.04	Lyndon Nixon	Final version
	15.07.04	Lyndon Nixon	Post-review version

Executive Summary

The KnowledgeWeb Network of Excellence has as its major aim the promotion of the transferral of Semantic Web technologies from academia to industry. However before these technologies can be strongly promoted to business, research in the Semantic Web and ontologies must reach maturity. New areas of business applicability such as the combination of Semantic Web with Web Services, realizing intelligent ”Semantic Web Services”, require serious new research efforts.

Work Package 2.4 in KnowledgeWeb is focused on supporting the research efforts in bringing Semantic Web Services to maturity. In the course of the initial 18 months of the project, the work package will produce the following deliverables:

- survey on the state of the art of current Semantic Web Services initiatives and analysis of current initiatives to identify semantic needs not covered within existing research efforts;

- semantic requirements for Web Services description;

- definition of semantics for Web Services discovery and composition;

- state of the art on agent-based services;

- guidelines for the integration of agent-based services and web-based services.

This document is the first deliverable in the above list. It is intended to be an internal deliverable distributed within the work package partners and acting as a firm basis for a shared understanding of the current state of the art of Semantic Web Services and the requirements currently arising out of the research initiatives in the field. It will also be distributed within the KnowledgeWeb project as a whole to act as an informative introduction to Semantic Web Services for all project participants including industry partners.

Additionally this document will act as an informed input to the following two deliverables which will explore in more depth the semantic needs for describing Semantic Web Services, particularly in terms of their discovery and composition.

As a groundwork for introducing and analyzing Semantic Web Services initiatives, we have first explained Web Services in terms of their definition, rationale, application and requirements. From this, we could examine the key standards and specifications in the area and evaluate the effectiveness of the Web Services infrastructure for achieving its aims. The findings from this were:

- Web Services are best defined as Web-based self-describing loosely-coupled modular software components which can be published, located and invoked by human or machine clients;

- Web Services arose out of the limitations of conventional middleware and Enterprise Application Integration (EAI);

- the research community proposes the fulfilment of complex business tasks with Web Services but actual use cases tend to be focusing on simpler implementations;

- the Web Services infrastructure consists of a wide range of standards, specifications and tools. While the core standards of SOAP, WSDL and UDDI fundamentally enable Web Services, many other specifications have been required to add functionality. There is overlap in some areas (e.g. in co-ordination and composition specifications) and competing specifications/tools from different vendors;

- complete with the extension specifications Web Services requirements are being tackled in the current infrastructure. However the demands of wide scale Web deployment of application integration solutions are not met by the current specifications and tools.

It is recognized by the Semantic Web Services research community that if Web Services are to achieve their goals, they require mechanization of service recognition, service configuration and combination (i.e., realizing complex workflows and business logics with Web Services), service comparison and automated negotiation. The instability and unreliability of the Web requires from Web Service-based systems an ability to maintain functional capabilities over a dynamically changing environment. The weakness of service descriptions based on semi-formal natural language descriptions is overcome by a combination of Web Services with Semantic Web technology.

In this overview of the state of the art of Semantic Web Services research we have considered four primary initiatives in the area at the present time (June 2004):

- OWL-S : a Web Service framework modelled in the ontology language OWL. Its main goal is to enable a user or a software agent to automatically discover, invoke, compose, and interact with Web Services adhering to requested constraints.

- IRS-II : a framework and implemented infrastructure for accessing, modifying and configuring the functionality of reusable problem-solving resources on the Internet. Web Services are conceived as problem-solving services enriched with semantic descriptions of their functionalities.

- Meteor-S : a project initiated to integrate Web Services standards with the Semantic Web. This has focused on workflow management specifications, service description language and peer-to-peer infrastructure for (semi-)automated publication and discovery.

- WSMF/WSMO : a conceptual model for the development and description of fully functional Web Services and an ontology and formal language for describing aspects of Semantic Web Services based upon that conceptual model.

The analysis of these initiatives is based upon a set of requirements that we derive for Semantic Web Services. The main results of the analysis were:

- Web Service invocation is extended semantically but there still is lacking clear formal semantic models of invocation;

- Web Service discovery has semantic representations of capabilities and goals but the matching operations on those representations are immature;

- Web Service composition lacks a mature semantic description. Such a description must be able to represent dynamic service selection and data and process mediation;

- Web Service interoperation also lacks a mature semantic description. Existing approaches are ambiguous to interpret but a clear conceptual model exists that can be built upon;

- other Web Service aspects still lack an semantically-based approach (e.g. transactionality, security, trust, execution monitoring).

These results will feed into the further activities of the KnowledgeWeb Network of Excellence, not only within the Work Package 2.4 but also orthogonally in the other research work packages and within the future research of the project research partners.

As a result, this internal deliverable forms the beginning of a process whose final goal is the realization of Semantic Web Services.

Contents

1 Introduction **1**

2 Web Services **3**

2.1 General Description of Web Services . . . 3

2.1.1 Definition . . . 3

2.1.2 Rationale . . . 5

2.1.3 Practical Applications and Use Cases . . . 7

2.1.4 Summary of Requirements . . . 11

2.2 Key Standards and Specifications . . . 16

2.2.1 SOAP . . . 17

2.2.2 WSDL . . . 18

2.2.3 UDDI . . . 20

2.2.4 Web Service Composition Languages . . . 21

2.2.5 Web Services Interoperation . . . 24

2.2.6 Other Web Services specifications . . . 26

2.2.7 W3C Working Groups . . . 27

2.2.8 Web Service Tools and Platforms . . . 28

2.3 Evaluation . . . 30

2.3.1 Service Invocation . . . 30

2.3.2 Service Discovery . . . 31

2.3.3 Service Composition and Co-ordination . . . 32

2.3.4 Evaluation Conclusion . . . 33

3 Semantic Web Services **35**

3.1 Definition of Semantic Web Services . . . 35

3.2 Rationale for Semantic Web Services . . . 37

3.3 Current Semantic Web Service Initiatives . . . 42

3.3.1 DAML-S/OWL-S . . . 42

3.3.2 IRS-II . . . 44

3.3.3 METEOR-S . . . 47

3.3.4 WSMF/WSMO . . . 53

4 Analysis of Semantic Needs **57**

4.1 Requirements of Semantic Web Services 57
4.2 Analysis of Current Initiatives 61
4.2.1 WSMO 61
4.2.2 OWL-S 65
4.2.3 IRS-II 70
4.2.4 METEOR-S 72
4.3 Open Issues 76

5 Conclusion 78

A Glossary 84

Chapter 1

Introduction

One of the research aims of the Knowledge Web project is to integrate the European research activity in the field of Semantic Web Services, and to identify open challenges to feed ongoing research in areas such as scalability, heterogeneity, dynamics, and Semantic Web language extensions.

The integration of the twin research fields of Semantic Web and Web Services has been identified as a key application field for both. While Web Services extend the Web from a distributed source of information to a distributed source of services, the Semantic Web adds machine-interpretable information to Web content in order to provide intelligent access to heterogeneous and distributed information. Similarly, Web Services can be defined using Semantic Web concepts so that computer systems can automatically discover, reason about and make use of Web Services. This joint application of Semantic Web and Web Services is usually referred to as Semantic Web Services (Lara et al., 2003).

The Internal Deliverable D1 is intended to provide a survey of the state of the art of current Semantic Web Services initiatives and an analysis of these initiatives to identify a first set of semantic needs not covered within existing research efforts. This is to act as an input to the first Deliverable of the Work Package 2.4, which shall be a draft of semantic requirements for Web Services description. The results of this Internal Deliverable shall make it possible to identify from the determined requirements for Semantic Web Services those requirements not yet satisfactorily being considered in existing initiatives.

As a result it is expected to bring together the existing work with the development of new research tasks for the remaining requirements and hence facilitate co-ordinated research leading to a complete and consistent standardized infrastructure for the implementation of and interoperation with Semantic Web Services. On this basis, Semantic Web Services can be promoted to industry as a practical and viable solution to business needs.

This document is structured in the following way:

The following chapter shall act as a primer to Web Services. This will serve as a brief introduction to Web Services in terms of their rationale, sample use cases (both exemplary and actual), requirements, the key standards and specifications in the field and an evaluation of them.

This serves as a basis for Chapter 3 which introduces why the introduction of semantics is applicable to the field of Web Services and introduces the current initiatives in Semantic Web Service research.

The analysis of these initiatives is found in Chapter 4. This analysis is intended to identify the semantic needs not being covered by the existing work. The basis for the analysis is a set of requirements derived from the rationale for Semantic Web Services, which are what Semantic Web Services are intended to comprehensively enable through the complete standardized infrastructure which is the overall aim of our work.

Chapter 2

Web Services

2.1 General Description of Web Services

Before we introduce the initiatives in Semantic Web Services research and analyze them, it is useful to make clear what is meant by Web Services, which aims we consider Web Services to have, what specifications and technologies define the Web Services field and what issues can be identified which lead us to consider the combined use of the Semantic Web and Web Services.

2.1.1 Definition

While there is a general understanding of what is meant by Web Services, it is nevertheless important at the outset to have a precise definition so that the work that we consider in this field is not too broad, or too narrow. For example, a common brief description of a Web Service would be an application which is available to others over the Web, though this could make into a Web Service any procedural code available through an URL e.g. CGI scripts or Java servlets. This is hence too broad a definition. Likewise the Wikipedia offers "a collection of protocols and standards used for exchanging data between applications". [1]. This could however apply to CORBA, Enterprise Application Integration (EAI) and other technologies pre-dating Web Services.

Several Web Service definitions can be found in the literature. An often quoted description comes from the IBM Tutorial from Web Services (Tidwell, 2000):

"self-contained, self-describing modular applications that can be published, located and invoked across the Web".

The Spencil Group defines Web Services [2] as:

[1]Wikipedia entry on "Web Services", URL: http://en.wikipedia.org/wiki/Web_service

[2]Article "Defining Web Services", URL: http://www.perfectxml.com/Xanalysis/TSG/WebServices.asp

"loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols"

From these descriptions it is possible to identify some required aspects of a Web Service that differentiate it from the more general applications mentioned above:

- "self-contained" and "loosely coupled" refers to the application being complete in itself i.e. the entire minimal functionality to realise the intended service is available and hence the service requester does not have to provide any additional functionality in order to make use of the service.
- "self-describing" and "semantically encapsulate discrete functionality" refers to the application providing a description of itself which is made available for examination by a human or machine.
- "modular" refers to the application being potentially a part of a larger application. Generally this means that the service is capable of being incorporated with other services in order to enable greater functionality (just as different procedures are used together to realise applications).
- Finally, it is emphasized that the service is Web-based in all aspects of its operation with a service requester. It is "published" (made public, e.g. to a service crawler), "located" (referenceable, e.g. through an URL) and "invoked" (executable, i.e. "programmatically accessible", through a call from another system, e.g. over HTTP) on the World Wide Web.

An even more precise definition is provided by the W3C Web Services glossary (Haas and Brown, 2004):

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

This definition is certainly more standards-specific, as well as supplying a clearer indication of how a Web Service should work. While arguably SOAP and WSDL are not the only standards that can be used the definition does indicate how a Web Service should:

- provide a description that is machine-processable and identifies the interface for accessing the service.
- interact with other systems (requesters) using messages with a XML serialization in a format prescribed by the interface description.

These two last definitions are sufficient for determining the common characteristics / base functionality of a Web Service. No restriction is given about the kind of functionality a Web Service can provide. It can provide static or dynamic information or perform real changes in the world, but what is essential in web services is their capability of providing functionality in a distributed manner within or across organizational boundaries using such widely accepted technologies as Internet related technologies.

This functionality should meet at least the following four aspects. Their definitions are based on the W3C Web Services glossary (Haas and Brown, 2004).:

- Discovery - The act of locating a machine-processable description of a Web service-related resource that may have been previously unknown and that meets certain functional criteria. It involves matching a set of functional and other criteria with a set of resource descriptions. The goal is to find an appropriate Web service-related resource.
- Invocation - The act of a message exchange between a client and a Web service according to the service's interface in order to perform a particular task offered by that service.
- Interoperation - defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state (also called co-ordination or choreography).
- Composition - defines the implementation of the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function, i.e. the pattern of interactions that a Web service agent must follow in order to achieve its goal (also called orchestration).

The first two are explicit in how Web Services are defined, while the other two are implicit in how Web Services would be used. In our discussion of Web Services we will relate points back to these four aspects.

A full glossary of terms relating to Web Services can be found in appendix A.

2.1.2 Rationale

An important question of Web Services is not only "what they are" but of course "why they are". On one level they are regarded as merely a Web-friendly progression of long existing ideas in the field of Remote Procedure Calls (RPC) and Enterprise Application Integration (EAI). On the other they are hyped as the future basis of a service-oriented Web and the solution to all business process integration problems.

The interest in and uptake of Web Services arises out of the limitations of conventional middleware and EAI (Alonso et al., 2003).

Middleware includes RPC, object brokers such as CORBA and message brokers for EAI. It facilitates and manages the interaction between applications across heterogeneous computer platforms. This is a significant requirement of business process automation, as such processes tend to rely on functionality available from distributed computer systems. A typical example of such a case is the automatic teller machine (ATM) which communicates over a local network with the information system of the bank.

Middleware handles heterogeneity with the use of specific interfaces to different applications. For different systems, different interfaces must be generated. Fundamentally, interoperability was not possible when different systems integrated non-compatible middleware (e.g. Java based vs Microsoft .NET). On the other hand for compatible platforms which were comparable in functionality, middleware became the accepted means to integrate distributed systems.

Concrete middleware makes implicit assumptions about the nature of underlying systems. EAI is an evolution of middleware to cope with coarse-grained, heterogeneous application integration. Message brokers are used to hide system heterogeneity and distribution from the application integrating those systems. Workflow management systems enable the definition and maintenance of the integration logic. EAI platforms tend however to be expensive, complex and difficult to use and maintain.

Integration does also not just take place within a single company. There is also a desire to automate business processes across several enterprises or make them available to external clients (i.e. persons not accessing the service from the enterprises intranet). Such scenarios are also referred to as B2B (business-to-business) or B2C (business-to-customer). Natural use cases are logistics and e-commerce, where the company has a vested interest in automating the ordering and delivery of supplies, or being able to offer its services to paying customers on the Internet.

Classical middleware is implicitly limited to use on a LAN, yet these scenarios require support for communication over a wide area network. This led to middleware extensions such as the General Inter-ORB Protocol (GIOP) in CORBA. However businesses protect their systems behind firewalls, from where ORBs can not communicate to remote systems. There is also no explicit agreement on the approach on how service discovery and invocation may be realised.

The World Wide Web has become the universal gateway between remote systems as the only communication possibility with firewalls is tunnelling through HTTP, which assumes a Web server on both ends of the communication. While tunnelling could also be used with middleware and EAI, these approaches are not designed with the requirements of the Web architecture in mind.

The implicit lack of trust between autonomous business entities means that each participant in a business process will want to keep control of their own business operations and transaction data. Yet in inter-enterprise communication there is no obvious location for the middleware as the participants are unlikely to agree to a centralised middleware solution hosted by one of the businesses, or by a third party.

Also in EAI there is an assumption that interactions are short-lived. However in the inter-enterprise case an operation could last much longer - hours or even days. Asynchronous interactions could hold up processes for long periods, locking up business data and services. Conventional middleware protocols are not designed for a distributed, peer-to-peer communication scenario.

The lack of standardization at the system and communication protocol levels and the lack of an appropriate infrastructure for B2B integration is a significant problem resulting in costly and complex implementation and deployment of inter-enterprise system integration in an ad hoc point-to-point fashion. Web Services are designed in contrast to be loosely coupled (and hence maximally re-usable) components available over the existing network infrastructure of the Internet. Web Services are intended to provide a common, simple and open framework for enabling the deployment and integration of business processes.

Another benefit of Web Services is that existing (legacy) enterprise systems can be wrapped in a Web Services interface and the system functionality can be made locatable and invocable through the Web in a standardized manner.

The expected benefits of the Web Services approach with respect to middleware and EAI can be summarized thus (Apshankar et al., 2002):

- Simpler: to design, develop, maintain and use compared to technologies like DCOM or CORBA;
- Basis in Open Standards: rather than proprietary EAI solutions;
- Flexible integration: through maximal decoupling between the service provider and service requester;
- Reduced investment: in comparison to EAI solutions such as message brokers;
- Broader scope: than EAI solutions which treat applications as single entities. Web Services can be realised as small logical units and integration made on a granular basis;
- Dynamic rather than static interface.

2.1.3 Practical Applications and Use Cases

The intended benefits of Web Services can be illustrated through the use of scenarios (business use cases) and can be demonstrated in concrete implementations of the use of Web Services as the means to business system integration. Existing scenarios used in the Web Services literature and real examples of the use of Web Services are introduced here. As a result, it can be seen how Web Services are intended to be used (according to the research communities) and are being used (according to industry).

Business Scenarios for Web Services

Scenarios vary from common simple (request/reply) business processes which can be ”outsourced” - rather than making multiple internal implementations, a shared Web service is invoked from different heterogeneous business systems - to higher level implementations which require more complex service architectures for co-ordination, mediation, monitoring etc. It is clear that Web Services are seen as having applicability at both levels. .

A common example of the former is that of a stock quotation service which takes as input the code of the requested stock and returns the current stock price. Another typical illustration is that of a currency conversion tool, which receives as input two strings - currency codes - and returns an integer - the exchange rate between those two currencies.

Many such implementations, which cover a wide range of domains, have been produced by Web Service programmers and can be found and invoked from online Web Service repositories such as the Xmethods site [3]. Other repositories available online include WebserviceList , RemoteMethods , WSIndex , Web Service Resource , Flash-DB and Web Service of the Day . These sites serve as an important demonstration of Web Service functionality at a simple, practical level.

The Java tutorial for Web Services [4] uses as an illustration a coffee company that wants to sell coffee on the Internet. Instead of maintaining a local inventory, the company builds a system based on Web Services that can dynamically respond to customer orders. It uses a discovery protocol to find available coffee distributors, negotiating with their systems to determine the prices of available coffees and storing the distributor and price information locally. It is now able to respond to customer requests by passing the respective coffee orders to the most suitable distributor, arranging coffee delivery and invoice payment and customer billing. This example represents a common business process (order management) which requires interoperation between different systems (the customers’ systems, the suppliers’ systems, the business system negotiating between the customers and suppliers). It also identifies some typical business tasks which can be handled with Web Services supplier discovery, order receipt and processing, inventory and price queries, ordering, invoicing and customer billing.

A similar case is mentioned in the Developer.com tutorial (Ananthamurthy, 2004), where the system generates new orders automatically when inventory is low. In this case a monitoring application is responsible for invoking the Web Service whenever a certain rule is met. The invocation of the Web Service, unlike the customer order in the previous example, is from a computer system without human intervention. This application could even be another Web Service, meaning that a Web Service could integrate operations of other Web Services to realise new functionality.

The W3C Web Services group give as an illustrative example a travel reservation sys-

[3] Xmethods website, URL: http://www.xmethods.net

[4] Java tutorial for Web Services, URL: http://java.sun.com/webservices/docs/1.3/tutorial/doc/index.html

tem (He et al., 2004). This is representative of a more complex system where several services possibly with quite different implementations on different systems - are being chained together to realise an end-to-end functionality. In this case, it is the co-ordinated booking of a vacation (transportation, hotel, car rental, excursions etc). There is also a payment service infrastructure. The user inputs a destination and travel dates and the service queries across transportation services for availability and fares. When transportation is available, a subsequent stage searches hotel availability and rates (and so on to car rental, excursions etc). Finally, the payment service is invoked to take payment from the user through a credit card. The example notes that the travel agency could have existing agreements with service providers or could dynamically discover appropriate services in response to customer queries.

Looking at the examples used by the research communities, they focus primarily on intra-enterprise communication. The Web service repositories require the use of a discovery mechanism but support solely manual means (i.e. a human must search through the repository for the desired service). The tutorials use examples that make use of the Web Service aspects of discovery, interoperation and composition.

Web Services Real-World Applications

Concrete applications of Web Services can be divided in two categories. In the first category we encounter Web-based services which, prior to their Web Services implementation, have been known and accessible to humans through Web sites. In the second category, real applications use Web Services to solve needs from intra-business and inter-business integration tasks, as well as to support supply chain (B2B) and customer services (B2C) from their internal corporate platforms.

The Web-based Services mentioned here are evolutions of Web-based systems whose internal data and processes were either not available to external systems or only though proprietary client-side tools and integration platforms.

- The Galileo travel reservation system which has moved to Web Services technologies [5] in order to make its travel booking services more easily and openly available. Deployed are services for itinerary viewing and booking, trip planning, processing airport and city codes and real-time flight status.
- The Amazon website has made a Web Service available for accessing the companies' data about stock, prices and availability so that it can be integrated into other applications. Some examples of applications built using the Web Service are given on the Amazon website.
- The search engine Google has also made public a Web Service for performing searches on Google from other applications.

[5] Article ”Galileo travels down Web Services path”, URL: http://www.nwfusion.com/news/2002/0429galileo.html

- TerraService is a Web Service of TerraServer, which serves up US Geological Survey maps and satellite photos of US locations.
- MapPoint offers a Web Service for machine access to location-based services such as maps, driving directions and proximity searches from an address input. It can be integrated on Web Sites to provide store location services, or with mobile devices using GPS for user location to offer directions and proximity to points of interest. A demo shows the service being used with a Web interface to allow users to locate their nearest coffee stores [6].

Web Services functionality is seen as most relevant to the application domains of Enterprise Application Integration (EAI) and E-Commerce. There are business use cases which are not being publicised as businesses see Web Services as a means to gaining competitive advantage and want to protect technical details of their internal business processes. Some reported use cases [7] serve, however, to illustrate where businesses are already applying Web Services technologies.

- Eastman Chemical is using Web Services to offer customers application-based access to their product catalogue in real time.
- The University of California at Berkeley is creating a unified communications system. Web services provide for standardized message exchange to enable communication between the heterogeneous systems (e-mail, voice, fax). This can be done by adding interfaces to the existing systems rather than costly replacements.
- Providence Health Systems is making medical and other records spread across different systems accessible to patients and medical staff through portal-based applications. A service-oriented architecture provides simple, re-usable interfaces for incorporating patient data into an application, cutting development time.
- National Student Clearinghouse uses Web Services to retrieve degree and enrolment verifications from nearly 7,000 educational centres and streamline requests for those verifications from employers or others who need that information.
- The Colorado Department of Agriculture maintains a Captive Elk Facility Web service to track 160 domestic elk herds in the state. The Web Service integrates data from three different divisions and provides a Web portal tool for running reports on that data.
- Hewitt Associates administers employee benefits for nearly 250 large companies. Web Services support those companies with programmatic access to Hewitts employee data. Both Hewitt and its clients have been able to incorporate employee data directly into applications.

[6] MapPoint Demo ”Fourth Coffee Company”, URL: http://demo.mappoint.net/fourthcoffee/MainForm.aspx

[7] Network World Fusion ”Tech Insider: Web Services”, URL: http://www.nwfusion.com/techinsider/2003/0310techinsider1.html

- T-Mobile enables mobile content delivery services to interoperate with its operational services such as user identification, personalization and billing through Web Services. Content providers can re-use T-Mobiles services from their content delivery applications rather than individually implement them. Also, their Service Integration Platform uses Web Services to support access for mobile workers to their corporate applications.

- Lydian Trust, a financial services company, has created an automated car loan processing system called BizCap. It uses Web Services to retrieve car loan applications, integrate data from partners, and perform evaluation tasks such as fraud and credit card checks.

Considering the cases given here, there is an equal spread of inter-enterprise integration scenarios and intra-enterprise (B2B) as well as some cases of use in e-commerce (B2C). However use cases do tend to take on less complex tasks than what is being proposed as examples within the research community. Web Services are being used to extract simple pieces of information out of remote systems (e.g. verify yes/no, return single value), and the complex computational tasks that are required to realize the desired functionality remain as an implementation task for the service requester.

Businesses are not yet ready to commit budgets and development time to more complex Web Service implementation scenarios, which raises the question if Web Services are indeed capable, in their current form, to respond to the sort of complex business processes that they are, at least hypothetically, suited to realizing,

2.1.4 Summary of Requirements

In this final introductory section to Web Services we consider the requirements to be met by the Web Services architecture if that architecture (i.e. standards and tools based on those standards) is to be adequate enough to enable Web Services which achieve their expected potential. We need to consider here also the demands placed upon Web Services as part of end-to-end business solutions where they may realize business-critical processes.

The Web Services infrastructure has these fundamental characteristics (Chitnis et al., 2004):

- a standard way for communication

- a uniform data representation and exchange mechanism

- a standard meta language to describe the services offered

- a mechanism to register and locate Web Services

The first three requirements relate to invocation of services, and the last to discovery. We consider in more detail requirements for both these aspects, before also adding requirements for the other aspects mentioned in the introduction: interoperation and composition. The key requirements are numbered in the text and are summarized in Table 2.1.

Invocation requirements

If a Web Service is to be invocable it must make its services available on the Internet at a given location and through a specified interface which may receive and/or respond with messages conforming to a given standard.

The requester shall be able to know which operations are offered by the service and have a means to understand how they are invoked through a description available in a standardized language (1). This language shall also support free text annotations to ensure a human readable description of services and operations, so that it may also be understood what the operations are meant to do.

The actual binding, including the protocol used and the message encoding shall be independent of the Web Services external implementation (the interface). Likewise, the Web Services internal implementation (the service functionality) shall be independent of the interface in terms of the programming language used or the hardware on which the Web Service is running(2). The W3C is promoting the use of XML (both in terms of the Infoset and the Schema) for uniform data representation and exchange and the use of URIs to reference concepts and resources as part of their requirement that Web Services are integrated into the wider architecture of the World Wide Web.

Another evident requirement is that the Web Service shall function as it would appear to according to its interface. In other words, the service should respond as it is expected to when invoked, in terms of making all operations listed in the interface available and implementing them in the way expressed(3). There should also be a means to express trust in the service, so that service requesters could have a means to be sure that the service will function properly, giving the correct response to their request(4). Other invocation requirements would be reliability, error handling (when the service is, in fact, down) and QoS (e.g. guaranteed response times)(5).

The W3C gives as other requirements (in addition to those mentioned above) for a Web Services description language (Schlimmer, 2004):

- description of abstract policies required by or offered by Web Service;
- description can extend interfaces to define new ones;
- description of application level errors (6);
- message functionality can be given as one-way, request-response, solicit-response or faults(7);

- message exchange can be described as synchronous or asynchronous(8);
- description to use URIs to reference values or services;
- messages can be unambiguously mapped to an operation;
- interfaces are associated to concrete protocols/data formats;
- support Web infrastructure extensibility, e.g. use of XML namespaces.

Discovery requirements

These descriptions are expected to be found through discovery mechanisms. While services could be searched for and located manually using repositories like those mentioned in the previous section, service discovery mechanisms may also be standardized so that applications can use them (as a form of Web Service in itself) to find services programmatically and through their description to invoke them.

A discovery mechanism then would require some form of standardized request and response to be able to accept queries and return the matching services. In order that these services are already indexed by the discovery service, it needs to also support a publishing functionality. This can be expressed as an API(1). To enable matching, a basis must be determined for relating query parameters to service descriptions. Such a mechanism will only be useful if it allows expressive enough searches and returns services with adequate functionality to match the request(2). As part of the response, access to the service description will be necessary so that it is possible to correctly invoke the service(3).

Interoperation requirements

To realise a Web Service often requires more than a single operation execution (such as by e.g. currency conversion). Web Services apply to business practices which are more complex, and require the execution of a number of operations in a particular order (such as a travel booking service that first checks availability, then takes the payment and finally makes the booking). This requires a conversation between the requester and the service that must follow certain rules. These rules must be expressable in terms of linear and sequential execution of operations. Other cases would be rules whose execution is conditional on meeting a stated condition, or finite loops of execution(1).

A result of this model is also that the implementation must know and be able to wait on a response before continuing. This may require monitoring of message state (i.e. to know which response belongs to which conversation, or to track if a response has not been received within a specified time)(2). Another conversational requirement is error handling, and as part of a transactional implementation the facility to compensate for earlier operations (i.e. an error in a later operation may require cancelling the effect of an earlier operation)(3). Finally, a co-ordination infrastructure needs to be in place to make the conversational requirements available to a requester, so that their message exchange with the service conforms to the rules of the conversation(4).

Another requirement that arises in the B2B domain is that of agreements between parties on the execution of services. Individual parties wish to express their specific requirements and constraints on message and data exchange. This is already part of e-business infrastructures (e.g. PIPs in RosettaNet or CPP in ebXML). It ensures that businesses communicate with each other in an agreed way that conforms to their desired business practices. In this case, the differing requirements of multiple business policies must be negotiated to determine a conversation respecting them all, and the conversation manager must monitor conversations that they respect those policies(5).

Additional requirements given by the W3C Choreography Working Group (Austin et al., 2004):

- support for retries and time-outs, with rules for handling them
- error description and support for handling service errors
- support for conditional paths
- hierarchical composition model for re-use of choreographies
- support for recursive behaviour
- reference passing so that intermediaries can handle operations for the client
- demarcation of transactional boundaries to guide conversation implementation
- participants can be statically defined or dynamically bound e.g. through runtime discovery
- ability to generate from conversational description code skeletons or test message interactions

Composition requirements

Another aspect of Web Services is their composition. Web Services are often not executed independently but as part of a larger process which involves multiple services. The travel booking system, for example, will probably use one Web Service offered by a flight booking service, another offered by a hotel reservation service and a third system which validates and processes payment by credit cards.

We require here the means to adequately express a process which executes multiple services to achieve a certain goal from a known input. A composition model must be defined which represents that process in terms of the services to be invoked, how they are to be invoked (operations, parameters, data values), and how they relate to one another (data and control flow)(1). Composition models have been derived from workflow models, which are very similar.

While composition raises a lot of requirements shared by interoperation (conversation), it additionally requires that the internal execution of a composed Web Service can handle the heterogeneity of the participating services (i.e. the differences in their interfaces between which data is being passed). Means to ensure correct data flow must be implemented within a composed Web Service(2). This requires an adequate description of the relationships between interfaces, rather than simply the interfaces themselves. Composition also requires specification of necessary pre- and post-conditions for services (to ensure correct implementation of the execution rules)(3) and a means to monitor the state of messages passing through the composed Web Service (as message state may be part of an execution rule). Error states must also be expressable so that processes can be stopped if an invalid event takes place, and error handling defined to ensure incomplete processes compensate for what has already been invoked (e.g. undoing a reservation)(4).

As part of inter-enterprise Web Service implementations it is required that businesses can hide details of their service implementations (including in terms of compositions) from external requesters. Businesses may wish to provide system functionality to others, but without giving any details of how that functionality is realised (5).

General requirements

The W3C Architecture document working group has produced a set of requirements for the Web Services architecture as a whole (He et al., 2004). In addition to requirements already mentioned above, they state as architectural requirements:

- modular and loosely coupled components
- extensible to allow for future evolution of technology and business goals
- simplicity and no high barriers to entry
- reliable, stable and evolving over time
- consistent and coherent
- a suitable level of alignment with the Semantic Web
- consistent with the architectural principles of the Web
- respecting privacy
- device independence
- support P2P interaction

Web Service invocation	- An standardized interface description
	- Interface abstract from internal implementation details
	- Operations function as described
	- Trust
	- QoS (reliability, response times)
	- Definition of errors
	- Message exchange one-way or with response
	- Message exchange synchronous or asynchronous
Web Service discovery	- An API for publishing and searching for services in a directory
	- Search based on details of service functionality
	- Access from found services to their description files
Web Service interoperation	- Expressive conversation rules (parallel, conditional, repeated operations)
	- Tracking conversation state
	- Transactional operations and compensation (undoing done operations)
	- Co-ordination infrastructure for guiding and proofing valid conversations
	- Enterprise policy description and monitoring to respect that policy
Web Service composition	- Definition of composition (e.g. using a workflow model)
	- Handling data flow between heterogeneous service interfaces
	- Pre- and post-conditions of service invocation (control flow)
	- Specification of error states and compensatory actions if a process is prematurely stopped
	- Hiding of composition details from external entities

Table 2.1: Web Services Requirements

Security will also be a major requirement for Web Services if they are to be widely deployed for important business practices. Though the usage of open Internet protocols (HTTP,TCP/IP) and ports (80) is the best solution to inter-enterprise application integration, network connections built in this way are not secure and businesses have a vested interest in protecting access to their networks. Similarly, XML-based messages being exchanged over the Internet can be intercepted, and require provisions such as encryption if sensitive data is to be protected.

In summary then in Table 2.1 we note the requirements considered to be the most significant for each of the aspects of Web Services.

2.2 Key Standards and Specifications

Web Service standards are often reduced in the literature to a set of three (SOAP, WSDL and UDDI) which represent the functionality of invocation, description and discovery respectively. However, these standards do not cover all Web Service requirements and a glut of other specifications have followed. An overview of these “complementary” specifications is also given. In the future we expect that competing specifications may converge and some functionalities subsumed by future versions of the "core" standards.

2.2.1 SOAP

SOAP (Simple Object Access Protocol) [8] standardizes a lightweight structure for the exchange of information to and from Web Services. These structures, called SOAP messages, are the basis for Web Service invocation.

The messages are serialized in XML to support structured and typed invocation. They are stateless, and realise an one-way communication. Applications can create more complex interaction patterns by combining such one-way exchanges with features provided by an underlying protocol and/or application-specific information. While SOAP is delivery mechanism-neutral, it specifies bindings to HTTP and SMTP (synchronous and asynchronous forms of communication). Finally, SOAP includes a means to indicate how a message should be processed by the entities which receive the message.

SOAP messages are basically XML documents which contain an envelope, which consists of a header and a body. The header contains data to be processed by the nodes receiving the message in its path from the sender to the recipient. SOAP elements can indicate if the message is to be passed on or handled, and if the node must be able to understand the information contained in the header. The body contains data to be processed only by the message recipient. It is generally assumed that the recipient will be able to understand the data contained here. Both header and body data are outside of the scope of the standard, and it is the implementer's responsibility that the data can be correctly handled.

Two aspects will influence how the header and body of a SOAP message are constructed: interaction style and encoding rules. Interaction style can either be document-style (the body contains a XML document to be processed by the recipient) or RPC-style (request/response where the request references an operation and its input parameters and the response returns the result i.e. the output parameters). Encoding rules ensure that the data is represented consistently in the XML serialization. The standard defines a SOAP encoding for data structures as a basis for message consistency. However implementers are free to decide on their own encoding rules (literal encoding).

Some proposals have been made to add to SOAP standardized means to express common functional requirements of Web Service invocation.

WS-Addressing [9] resolves the problem that SOAP relies on the transport bindings to indicate the address of the receiver, i.e. the recipients address is not part of the SOAP message. The proposal specifies an "endpoint reference" - an URI and a set of reference properties - where the URI is normally an application which can resolve the intended

[8]Latest SOAP versions, URL: http://www.w3c.org/TR/soap

[9]Web Services Addressing, URL: http://www.ibm.com/developerworks/library/ws-add/

recipient from the reference properties. By enabling addressing within SOAP messages, delivery can be taken care of independent of any communication protocol.

WS-Routing [10] resolves the problem that while SOAP has the notion of a message delivery path comprising of a set of nodes which may process the message before handing it on, the SOAP message can not in itself specify what the path should be. Rather the path is determined by the underlying delivery protocol. The proposal permits specifying a delivery path as part of the SOAP message, supporting pipeline architectures and ensuring a message is handled by nodes with specific functionalities.

WS-Security [11] is an extension to SOAP to enable integrity and confidentiality. It defines a SOAP header block which can carry a signature, and defines how this header block should be processed by the nodes receiving the message. Unlike HTTPS (which allows the decrypting and re-encrypting of a message at every node), secure SOAP messages should remain encrypted from the sender to the recipient. Additionally encryption can be applied to just the SOAP message body or a section of the body (e.g. the element containing the credit card information). WS-Trust defines extensions to WS-Security for issuing and exchanging security tokens and ways to establish and access the presence of trust relationships, so that business entities can determine if they ”trust” the asserted credentials of another entity with which they wish to communicate.

WS-ReliableMessaging [12] is intended to support the guarantee of reliable message delivery between Web Services and clients in the presence of software component, system or network failures. It specifies basic delivery assurances (AtMostOnce, AtLeastOnce, ExactlyOnce or InOrder) that one entity can give to another. Traditional reliable messaging functionality like message identification and numbering, as well as acknowledgement mechanisms, are specified in the SOAP message.

2.2.2 WSDL

The Web Services Description Language (WSDL) [13] is an XML format endorsed by the W3C for describing Web Services. The language allows the description of a service to be separated in two levels: one specifying interface details like operations and messages and another specifying interoperability details like message encoding, message transport and transport addresses.

In many ways the standard is similar to conventional middleware IDLs (interface definition languages). In constract to IDLs, the WSDL structure is modular in order to allow for the re-use of (parts of) descriptions, and allows for the combining of operations or

[10]Web Services Routing Protocol, URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-routing.asp

[11]Web Services Security, URL: http://www.ibm.com/developerworks/webservices/library/ws-secure/

[12]Web Services Reliable Messaging, URL: http://www.ibm.com/developerworks/webservices/library/ws-rm/

[13]Web Services Description Language, URL: http://www.w3.org/TR/wsdl

groups of operations in one interface. The standard also includes the specification of different interaction paradigms (one-way, request/response, solicit/response, notification).

The interface-level specification captures the more abstract aspects of a service. It is constructed through a four layer component structure (interface, operations, messages, and types). These define respectively:

- Interface - a group of operations
- Operations - a service interaction, defined using a specified interaction paradigm
- Messages- a typed document of message parts, each with a name and a type
- Types - a datatype such as the primitives or complex types defined in XML Schema (a WSDL document can specify another type system if desired)

The interoperability-level specification defines the lower level details required for concretely interacting with the service. It is made up of services, endpoints and bindings. These define respectively:

- Service - a logical grouping of endpoints
- Endpoints - a combination of a binding to a network address (URL) at which an implementation of the interface may be accessed
- Bindings - specification of the message encoding and protocol binding (e.g. SOAP over HTTP) for a given interface, as well as the services interaction style (document or RPC).

The separation of these components in the description supports the flexibility of service description and re-use of components. For example, the same interface could be tied to several different endpoints each with a different binding. Interfaces could also be combined with others in different combinations to form different services. Abstract descriptions could be imported and combined with concrete bindings and endpoints to describe implemented services.

WSDL documents can be seen as a contract specifying formally how the service shall be interacted with, and where the service can be accessed. Implementers can use the WSDL description to correctly implement their applications to call the service and handle the response. As with middleware IDLs, it is also possible to use WSDL to generate automatically the stubs for the Web Service implementation and service requester implementations.

2.2.3 UDDI

Universal Description, Discovery and Integration (UDDI) is an OASIS specification [14] for a framework for publishing and locating Web Services. It aims to enable Web Service discovery in a semi-automated fashion, building on the existing standards of XML, SOAP, and WSDL.

The core idea of UDDI is a ”business registry”. This is the development of a concept that has been around since RPC, that of the name and directory server. Just as the server in RPC was responsible for passing to the client an address of a server which could execute the desired procedure, the UDDI registry is intended to support developers in finding information about services that realise the functionality they require and to enable dynamic binding (allowing clients to query the registry and obtain services of interest).

UDDI fundamentally defines data structures and an API for publishing service descriptions to the registry and querying it for relevant published descriptions. The API is also specified in WSDL using a SOAP binding, so that the registry can be accessed as a Web Service and its characteristics can be described in the registry itself just as with any other Web Service.

The information in an UDDI registry can be categorized into three types:

- White pages - listing of businesses, their contact details and services provided
- Yellow pages - classifications of businesses and services according to taxonomies
- Green pages - description of how a service may be invoked through reference to external description files

In the UDDI data structure there are 4 main entities:

- businessEntities - the business name, address and other contact info
- businessService - the services offered by a businessEntity
- bindingTemplate - the technical information required to use a service
- tModel - a ”technical model”, a generic container for any kind of specification. It could be a classification, an interaction protocol, or a service interface description.

As all of the first three entities make keyed reference to tModels, it follows that tModels really form the core of the UDDI specification. Before anything can be published to the UDDI registry, the tModels that will be referenced need to be defined. Some may well be pre-defined by other standardization efforts (e.g. the UDDI specification includes

[14]UDDI Specifications, URL: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

a tModel for classifying other tModels). Once a tModel is published to an UDDI registry it is assigned an unique key. It is the use of the same tModel by different businesses and services which supports both the design-time discovery and run-time binding of services. tModels can in fact represent anything as the key basis for their use is the shared knowledge of the tModels key and what it represents. The tModel's intended meaning can be determined by a human through the overviewDoc which the tModel internally references and which should contain a human readable explanation of the tModel. For example, if a particular client has been implemented to conform to the service interface referenced by the tModel with the key 327, then the client can also dynamically bind to a suitable service at run time by querying the registry for a service whose bindingTemplate references the tModel with the key 327.

The UDDI API is designed to support the needs of three types of user: a service provider publishing a service, a requester looking for services and other registries that wish to exchange information. It supports fundamentally four types of operation, each of which having a method specific to the data structure it is applied to: find, get, save and delete. The ”find” methods enable the search functionality and ”get” retrieves the information for a specific entity. ”save” and ”delete” permit the addition and removal of entries into and out of the registry. Finally, there is also transfer methods for keeping registries synchronized.

2.2.4 Web Service Composition Languages

This subsection gives an overview over several specifications related to the composition of Web Services. The development of complex Web Services out of other Web Services is not trivial. Composition middleware provides abstractions and infrastructures for the definition and execution of composite services. Fundamentally it supplies a composition language with which a composition schema (a model of the composite service) can be expressed, with modelling constructs for the participating services, the order of their execution and how the execution parameters may be determined. Web Service composition further develops the work in workflow management systems which have been applied to the task of conventional middleware composition. We exemplarily describe one of these composition languages, WSFL, in more detail.

WSFL The Web Services Flow Language (WSFL) [15] is an XML language for the description of the composition of Web Services. WSFL's objective is to specifiy:

- Usage patterns internal to a Web Service composed from other services. This corresponds to the FlowModel defined below.
- External interaction patterns between composed Web Services. This corresponds to the GlobalModel defined below.

[15]Web Services Flow Language, URL: http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

The core concept of WSFL is the FlowModel - a special kind of directed graph which is the basis for the composition model. GlobalModels build then on top of FlowModels by defining how they can be brought together. Recursivity is introduced by allowing FlowModels to refer other sub-FlowModels.

Central for a FlowModel is the concept of an activity. A FlowModel can define zero or more activities. WSFL defines an activity as a business task to be performed as a single step within the context of a business process contributing to the overall business goal to be achieved.

Activities correspond to nodes in the directed graph and they are wired together through controlLinks. A flow model can contain zero or more controlLinks. A controlLink is a directed edge between a source activity and a target activity. All the controlLinks in a FlowModel prescribes the order in which activities will have to be performed. The endpoints of the set of all control links that leave a given activity A represent the possible follow-on activities A1,...,An of activity A.

Which of the activities A1,...,An actually have to be performed in the concrete instance of the business process (that is, the concrete business context or business situation) is determined by so-called transitionConditions. Each controlLink defines exactly one transitionCondition specified as a boolean expression (for now defined using the XPath specification). The formal parameters of this expression can refer to messages that have been produced by some of the activities that preceded the source of the control link in the flow.

When an activity A completes, only the control links originating at A whose transition conditions evaluate to true are followed to their endpoints. This set of activities is referred to as the actual follow-on activities of A in contrast to the full set A1,...,An of possible follow-on activities.

A condition imposed by the WSFL specification is that the activity controlLink graph representation resulting from a FlowModel must be acyclic. However loops are supported as do-until constructs.

An activity is called a fork activity if it has more than one outgoing controlLink. In this case all the follow-on activities spawned-off by the controlLinks with a true transitionCondition will be performed in parallel.

Typically, parallel work has to be synchronised at a later time. Synchronisation is done through join activities. An activity is called a join activity if it has more than one incoming controlLink. By default, the decision whether a join activity is to be performed or not is deferred until all activities that could possibly reach the join are completed. The decision is governed by a joinCondition that can be associated with any activity and that can be a boolean expression.

Sometimes, a weaker semantics of synchronisation is appropriate and supported by the model of WSFL: as soon as the truth-value of a joinCondition is known, the associated join activity is dealt with accordingly (that is, either performed or skipped). Control flow that reaches the corresponding join activity at a later time is simply ignored.

Activities that have no incoming controlLink and that have joinConditions that always evaluate to true are called start activities. Activities that have no outgoing controlLink are called end activities. All start activities are automatically scheduled for execution when a FlowModel is initially instantiated. When the last end activity completes, the output of the overall flow is determined and returned to its invoker.

Each activity has also an associated exitCondition which is again a boolean expression. The purpose of it is to determine whether or not an activity has to be considered completed. If this is the case the next activities to be performed are determined and execution continues otherwise the activity is executed again.

Because WSFL does not allow for cyclic graphs exitConditions are also used for realizing do-until loops. In this case the iteration is done based on the exitCondition of an activity which is then implemented by a sub-FlowModel.

Each activity is performedBy a particular serviceProvider which defines a set of operations. The operation invoked when an activity has to be executed may be perceived as the concrete implementation of the abstract activity.

The ServiceProvider concept allows to specify the role that is expected to be played by a potential business partner. Each ServiceProvider has a type defined by serviceProviderType. A serviceProviderType is just a named set of portTypes. In turn a portType defines one or more operations.

In order to be able to recursively compose FlowModels, WSFL's vision is to perceive them as serviceProviderTypes (that is, a set of portType operations). For that the WSFL model provides a construct to export operations implementing encompassed activities. These exported operations are grouped together to define the public interface of the FlowModel.

A global model defines the interaction between a set of serviceProviders. Interactions are modelled using PlugLinks between operations on the ServiceProviderTypes involved in the composition. Each PlugLink is defined between a source and a target operation. Operations have to be dual, for example, a notification operation on one ServiceProvider can be plug linked to a one-way operation on another ServiceProvider, or a solicit-response operation can be plug linked to a request-response operation. Similar to FlowModels, GlobalModels have associated one or more ServiceProviders and define themselves for the external world as a ServiceProviderType.

BPEL4WS BPEL4WS [16] appears as a candidate specification for defining and managing Web service based business processes. It is a technology for describing the behaviour of Web Services in a business interaction, specifying a XML-based grammar for control logic need to coordinate participating Web Services. BPEL4WS provides a language for the specification of web service processes and interaction protocols. It builds on top of

[16]Business Process Execution Language for Web Services, URL: http://www.ibm.com/developerworks/library/wsbpel/

WSDL to define a process that provides and consumes multiple Web Service interfaces. BPEL4WS is based on traditional business process and workflow technologies, and as a development upon earlier proposals such as WSFL and XLANG it is now the central focus of most software vendors to become the primary Web Service composition language. However, BPEL4WS solely supports the static binding of services in the composition. This limitation results in BEPL4WS currently only being used in semi automatic substitution use cases rather than plain composition.

BPML The Business Process Modelling Language (BPML) [17] is a meta-language for the modeling of business processes (and not specifically web services), just as XML is a meta-language for the modeling of business data. BPML provides an abstracted execution model for collaborative and transactional business processes based on the concept of a transactional finite-state machine. BPML is complementary to WSCI (WSCI is described in the next subsection). WSCI defines the interaction between services deployed across multiple systems, while BPML defines the business process behind each service, mapping business activities to message exchange. Together, they provide an end-to-end view that depicts the role of each individual business process in the overall choreography, and the business activities performed by each role. BPML builds on top of WSCI and is argued to be a strict superset of BPEL4WS.

2.2.5 Web Services Interoperation

The basic Web Services infrastructure available through the standards SOAP, WSDL and UDDI is sufficient for simple interactions with Web Services which involve a single operation invocation. However in real applications communication is typically more complex, involving the invocation of several operations which may be constrained (e.g. in order of invocation, and dependent on which responses are received). The sequence of message exchanges between a client and a service is a conversation, and a specification of the correct and accepted conversations for a particular service is known as the coordination protocol.

WS-Coordination WS-Coordination [18] is a proposal for a framework to support coordination protocols. It is a meta-specification for the standardization of concrete forms of coordination. It supports both central and distributed coordination (i.e. all participants talk to the same coordinator, or each talks of its own coordinator).

Coordination is realized by including a data structure called a coordination context in the SOAP message headers. The structure includes an identification of the coordination protocol and an unique identifier for the actual instance of that protocol (i.e. for the current conversation). The proposal does not however specify how to describe coordination protocols.

[17]Business Process Modelling Language, URL: http://www.bpmi.org/bpml.esp

[18]Web Services Coordination, URL: http://www.ibm.com/developerworks/library/ws-coor/

Whatever coordination protocol is used, and in whatever domain is deployed, the same generic requirements are present:

- Activation of a new coordinator for the specific coordination protocol for a particular application instance and their associated context;
- Registration of the participants with the coordinator such that they will receive that coordinator's protocol messages during the applications lifetime.
- Propagation of contextual information between the Web Services that comprise the application.
- An entity to drive the coordination protocol through to completion.

WS-Transaction WS-Transaction [19] builds on top of the WS-Coordination framework to enable support for transactional interactions. Transactions are a set of operations for which either all the operations in the set shall be successfully executed or none of them shall be executed. If some (but not all) operations in a transaction have been executed before the transaction as a whole fails (due to some error or a cancellation command) then the effect of the completed operations must be undone (rolled back). Transactional guarantees are also important for business processes being handled by Web Services (e.g. if a client makes a reservation and then can not pay, the reservation should be cancelled).

The proposal specifies a standard protocol for long-running transactions (”business activities”) as well as a means for specifying short-duration transactions (”atomic transactions”). Fundamentally, participating Web Services shall update their persistent state at each step of the transaction and be able to undo any step in the case of an abortion of the transaction. The logic for undoing any operation is implemented in a compensation operation, specific to the Web Service and outside of the WS-Transaction standard.

WSCI WSCI [20] is an XML-based interface description language that describes the flow of messages exchanged by a Web Service participating in choreographed interactions with other services.

It describes the dynamic interface of the Web Service participating in a given message exchange by means of reusing the operations defined for a static interface by extending the WSDL interface description. This is expressed in terms of temporal and logical dependencies among the exchanged messages, featuring sequencing rules, correlation, exception handling, and transactions. With the concept of global model, WSCI provides the ability to ”link” operations in a ”collaboration” between the interfaces defined by it.

The focus of WSCI is on tightly coupled application-to-application integration [21] and is not effective for loosely coupled B2B and EAI scenarios.

[19]Web Services Transaction, URL: http://www.ibm.com/developerworks/library/ws-transpec/

[20]Web Services Choreography Interface, URL: http://www.w3.org/TR/wsci/

[21]Analysis of WSCI by Jean-Jacques Dubray, http://www.ebpml.org/wsci.htm

WS-CDL The Web Service Choreography Description Language (WS-CDL) [22] is an effort of the W3C that defines a standard for specifying the required ordered message exchanges between Web Services in order to achieve a common business goal.

It describes the set of rules that explains how different services may act together, and in what sequence. It differs from a composition specification language such as BPEL in that it defines the information being exchanged between all participants rather than just by one participant, models the global message exchange between participants without a specific point of view rather than from the point of view of one participant, and provides reactive rules to allow participants to determine what message exchange will occur next rather than active rules to infer what to do next. Fundamentally, it is proposed as a complement to Web Service composition by providing a shared choreography which services can use at run time to verify that they can participate appropriately in a choreographed process.

2.2.6 Other Web Services specifications

Some other Web Service specifications do not fit into any of the aspects mentioned previously. We consider those specifications in this section.

WS-Policy WS-Policy [23] is a base set of constructs that can be used and extended by other specifications to describe requirements, preferences, and capabilities of service interfaces.

WS-Policy is to be used with other Web Service standards (like UDDI, WSDL, SOAP, etc). The policy is described by an XML-based structure called policy expression which contains domain specific Web Service policy information and a set of grammar elements to indicate how the contained policy assertions apply. A policy contains one or more policy assertions which represent individual preferences, requirements, capabilities or other properties. Choice sets can be specified among policy assertions by operators like ExactlyOne, All, or OneOrMore, which express whether all assertions are to be met or exactly one among them for example (a numeric attribute allows specifying preferences when multiple choice is possible). Assertions are stated according to specific extensions like WS-PolicyAssertions which provide an initial set of assertions to address some common needs of Web Services applications (for example TextEncoding, SpecVersion, or Language), or non specific ones like WS-Security.

WSIF
The Web Services Invocation Framework (WSIF) [24] is an API used to hide from the developer the type of binding used by a Web service. It answers the fact that WSDL is generic and allows the specification of many different bindings, other than SOAP. Since the SOAP binding is dominant for Web Services, most invocation API's simply ignore this fact and provide SOAP as the only possible binding for a WSDL description of service. In WSIF

[22]Web Services Choreography Description Language, URL: http://www.w3.org/TR/ws-cdl-10/
[23]Web Services Policy Framework, URL: http://www.ibm.com/developerworks/library/ws-polfram/
[24]Web Services Invocation Framework, URL: http://ws.apache.org/wsif/

some other Providers (binding specifications) have been developed for EJB's and Java classes. Documentation is given which allow to specify bindings for other applications.

2.2.7 W3C Working Groups

The Web Services Activity of the W3C [25] has as its ultimate goal to develop a set of technologies in order to lead Web services to their full potential.

XML Protocol WG

The XML Protocol Working Group was originally chartered to develop an XML-based messaging framework, which resulted in SOAP - see 2.2.1. The Working Group is now working on attachments and optimization of the transmission of SOAP messages.

Web Services Description WG

The Web Services Description Working Group, chartered to design a language for describing interfaces to Web services and how to interact with them, produced the WSDL standard (see 2.2.2). The working group is now working on WSDL version 2.

Web Services Architecture WG

This W3C Working Group focuses on the description of Web Services components and on their relationships, aiming to give a common defintion of a Web Service architecture. Web Service concepts are approached through the use of four model orientations which are respectively focused on message, service, resource, and policy aspects of Web Services. Models are interdependent and each of them define specific roles, concepts, and relationships. Web Service architecture is further discussed in terms of requirements of common business issues like security, reliability, and management. The Working Group published in February 2004 a set of documents presenting a Web Services Architecture.

Web Services Choreography WG

This W3C Working Group focuses on the description of requirements for the organization of Web Service interactions. This interaction between services is called choreography and defines concepts common to most process organization technologies like roles, activities, interaction types (e.g synchronous or asynchronous) and control structures between activities (like sequence, choice or parallel). The group also focuses on the WS-CDL language (short for Web Services Choreography Description Language) which is intended

[25] W3C Web Services Activity, URL: http://www.w3.org/2002/ws/

to serve as a common interface between companies implementing their choreography requirements in lower level specifications like BPEL4WS or EJBs.

2.2.8 Web Service Tools and Platforms

Since the advent of Web Services many software vendors have embraced the technology and support it in their products. The spectrum of products supporting Web Services reach from database systems over application servers over standard applications to office suites; corresponding support in tools is available too (Leymann, 2003).

Microsoft

Web Services support is a primary concern through the whole .NET framework API implementation and design, as well as in companys the supporting IDE (Visual Studio .NET) and web server (IIS). Support for Web Services is enforced by each new version of these tools (Microsoft Windows Server 2003, Microsoft .NET Framework 1.1, and Microsoft Visual Studio .NET 2003, and IIS 6.0.

The Longhorn SDK is published as a technology preview by Microsoft; it is a thorough refactoring of the actual .NET Framework, keeping backward compatibility but introducing many new features heavily based on XML and distributed services. Under this code-name are held other code-named components which improve for example the presentation and storage layers of the platform. One of them, called Indigo [26] will be the communication subsystem of the platform, allowing applications to better access remote objects or communicate with Web Services.

IBM

IBM provides the ETTK (Emerging Technologies Toolkit) [27] which evolved from the package known as the Web Services Toolkit (WSTK). It is a very rich solution based on WebSphere, the company's application server, but oriented towards many emerging technologies. It contains a tutorial, an embedded version of the IBM Websphere Application Server, a JDK, technology previews of security functions such as WS-Security, and even a offer semantic orientation with what is called Semantic SDO (short for Semantically-Adapted Service Data Objects) which adds the ability to use RDF and OWL (the W3C Web Ontology Language) as a mean of attaching semantic metamodels to object descriptions. One can then use that model to navigate and query the data model independently of any change to it, since it is accessed through the semantic interface.

[26]Understanding "Indigo Web Services, URL: http://longhorn.msdn.microsoft.com/lhsdk/indigo/conunderstandingmessagebuswebservices.aspx

[27]Emerging Technologies Toolkit, URL: http://www.alphaworks.ibm.com/tech/ettk

SUN

SUN provides the JWSDP (Java Web Services Developer Pack) [28]. It is a full solution to be used with the java language, fully supported by SUN's IDE (Sun One Studio) and application server (Sun One Application Server), but provided for download with the free Tomcat J2EE server. Sun also provides a thorough tutorial about using Web Services with Java tools. But the package consists mainly in XML oriented APIs (JAXB, JAXP), dynamic web technologies (Java Servlets, JSP, JSF, JSTL), with a Web Service orientation only represented by:

- The Java API for XML Registries (JAXR) - which allows access to an UDDI registry as a special case.
- The Java API for XML-based RPC (JAX-RPC) - which allows the use of SOAP based RPC in Web Services applications.
- The SOAP with Attachments API for Java (SAAJ) - which enables developers to produce and consume messages conforming to the SOAP 1.2 specification and SOAP with Attachments note. It is a low-level API mainly used by JAX-RPC to access the SOAP packet.

ASF (Apache Software Foundation)

Not a fully featured package [29] but a series of APIs and implementations which are intended to be used with ASF Web Servers (Apache and Tomcat, ASF's JSP compliant web server). The Axis project provides Web Services implementation over Tomcat. WSIF (see more complete description at 2.2.6) provides a way of contacting web services through runtime WSDL inspection. There are also many other projects in incubation (term used by ASF for not completely mature realisations) like WSFX (Web Service Functionality Extensions) providing additional functionality as specified for example in the WS-* specifications (most of them are described above). These functionalities include: Addressing (an implementation of the Web Services Addressing (WS-Addressing), published by the IBM, Microsoft and BEA as a joint specification, on top of Apache Axis), Sandesha (an implementation of WS-ReliableMessaging Specification), WSS4J (an implementation of the Web Services Security (WS-Security) which is a Java library that can be used to sign and verify SOAP Messages with WS-Security information).

Web Services Friendly Tools

Web Service Friendlyness (wsf) is a term stamped to express some level of awareness offered by an application to the concept of Web Services and collateral standards. All

[28]JWSDP, URL:http://java.sun.com/webservices/webservicespack.html
[29]Web Services Project @ Apache, URL: http://ws.apache.org/

application servers of the major IT vendors have taken on Web Services and have quite good “wsf. Open source applications servers like JBoss (in the JBoss.net flavor, which in fact integrates Web Services with J2EE) follow this movement. Most application software doesn’t yet feel the necessity to access Web Services, but wfs seems to be something taken very seriously in all new Microsoft products. For example in Visual Studio .NET 2003 a developer can add a reference to a Web Service like he can do with other kinds of software components. InfoPath 2003, another new Microsoft product, a WYSIWYG XML form builder which allows Web Service discovery through an UDDI registry and to easily build a form-shaped interface to it. Microsoft BizTalk Server 2002/2004 is able to orchestrate calls to Web Services as well as to adapt legacy servers in order to quickly build a Web Service interface. The 2004 version also provides BPEL4WS support for Web Service description and orchestration.

2.3 Evaluation

This introductory chapter has explained what Web Services are, why there is a need for them, illustrated scenarios in which Web Services are deployed to meet that need, and as a result derived what is required of Web Services technologies. The current technologies in the Web Services field have then been introduced: both core standards such as those for invocation, description and discovery, as well as the secondary specifications which are arising from the early identification of missing requirements (e.g. service composition, interoperation, reliability, security).

Do these technologies enable all that Web Services require in order to solve enterprise application integration needs including wide ranging, complex, large scale, business-critical inter-enterprise scenarios?

2.3.1 Service Invocation

Web Services are invoked through messages conforming to an interface described in a document which is accessible to other systems that want to interact with them. The internal implementation of the service is independent of the invocation procedure, so that invoking systems do not need to know in which programming language the service is implemented, on what hardware it is running or in which operating system it is being executed.

The invocation procedure itself may be carred out using different data formats and communication protocols (Leymann, 2003), providing a high degree of interoperability and allowing different parties to interact seamlessly. This is also important to enable several services to be combined to realize a certain functionality.

Using the Web as the network through which invocation takes place also allows services to be accessed irrespective of physical location. However SOAP messages travelling over the Internet, if they are to be used for business critical processes, need to be reliable, secure and trustworthy. The standardization efforts in this area are still in progress and are very important if SOAP based Web Services are to become part of future critical business process infrastructures.

However SOAP is simply a meta-structure for the invocation messages. The required knowledge for the correct processing of the contents of a message header or body could be application specific. While some standards aim to specify commonly required header data, the required message body structure for a service is given in its WSDL file. This is generally restricted to defining a XML structure for a document-style interaction or parameter name and accepted values (as a XML Schema datatype) for RPC-style interaction. This description is insufficient however to precisely and unambiguously define what is meant by certain XML elements or operation parameters, and raises the possibility of erroneous message generation. For example, a parameter ”Temperature” with a datatype of integer is insufficient for a developer to know if Celsius or Fahrenheit values should be sent.

In other words, a WSDL file is lacking semantics and may not sufficiently specify what a service means in order that a developer can pass messages correctly to it, and while human-readable references can clarify issues in a case of manual development, it precludes any possibility of (semi-)automated invocation. For example, an appropriate functionality for a certain service can only be determined from its WSDL description if the name of the operation is precisely known. Other services who offer the same functionality through a differently named operation would not be identifiable as equivalent.

2.3.2 Service Discovery

Service registries can make Web Services available as soon as they are advertised, just as web sites are crawled and indexed in search engines. The UDDI query mechanism is intentionally simple, with fewer constructs than a general query language such as SQL. While this makes querying a rather cumbersome task, it enables simple implementation of query services. While registries can provide a centralised repository of services, distributed searching (i.e. searches that are propagated between registries) is not possible, nor is extending the search criteria to additional characteristics stored outside of the registry (e.g. dynamic characteristics such as current service response time, which would be maintained by the service provider).

The UDDI search criteria can be based on business or service details, with the requested values either being static (e.g. the business entities name) or based on a tModel. In order to make the proper search query, the tModel for the desired criteria must be known in advance, making it necessary to first manually examine the content of the registry and read the relevant overviewDocs. This is a significant effort that discourages the use of more than one UDDI registry, which will use other tModels with different keys.

The syntactic basis of the Web Service publication mechanisms means that discovery is a task which requires human supervision, such as with a Web search engine. The ambiguity of textual descriptions and the non-machine-understandability of the tModels is insufficient to support fully automated discovery. Manual search is complicated by publishers not interpreting consistently the meaning of fields in the UDDI data structure. While Web data is unstructured, Web search engines benefit from years of research while UDDI is a new technology and search mechanisms are still immature.

Another consequence of the syntactic format of Web Service descriptions is the difficulty of matching advertised Web Services to requesters wishes. General service discovery is not likely to be based on which business is providing a service or in which domain the business operates, but a specification of the functionality desired by the client. This specification can at best be given in the form of a service interface and matching services found through interface (e.g. parameter name and data-type) comparison, which is certainly not a conclusive basis to determine if a Web Service has the precise functionality the requester desires.

2.3.3 Service Composition and Co-ordination

Business scenarios have identified value in the chaining together of Web Services, whether in the calling of one service from another or many services being called as part of a business workflow. Specifications to define the necessary rules for the service invocation have been reviewed.

Handling multiple operations or Web Services raises issues resulting from the heterogeneity of data in distributed systems. In this case, the issue is how responses in one structure and format may be wrapped into the necessary structure and format for the next request. This must be expressed in the business logic. The mappings must be statically implemented to individual operations and services, with a syntactical representation being insufficient for some form of dynamic mapping.

Another issue is that of differences in service invocation behaviour. Mediation is required to resolve such inconsistencies and form an error-free service coordination or composition. Also service responses or errors may change the state within the workflow and may necessitate ”balancing” mechanisms (such as transactional compensations). As with the dataflow, the service heterogeneity can complicate the task of coordination or composition and presently these problems are handled through process-specific rules in the coordination or composition specification, which makes the specifications difficult to edit or re-use (e.g. in the case of replacing one participating service with another). It is also possible that ad hoc coding in the implementation logic is required to ensure necessary data format conversions or compensation mechanisms.

Composition models also benefit from dynamic service selection i.e. a service is specified in abstract terms and is resolved to an actual service endpoint at execution. However this resolution is based on the WSDL service description, which as noted above, lacks semantics and hence is not expressive enough for service bindings based on a general formulation of a desired functionality.

Coordination and composition are much more complex tasks requiring more complicated models and specifications in comparison with the ”core” Web Services standards of SOAP, WSDL and UDDI. Here the standardization efforts are also in a much earlier phase.

2.3.4 Evaluation Conclusion

Web Services are in a position to fulfil business needs by making functionalities offered by internal business systems widely accessible. The service can then be requested from clients over the Web or integrated with other applications. Once an application has been configured to work with a Web Service, which through intuitive tools and simple open standards can be a cost- and time-saving exercise compared to middleware solutions, it can take advantage of that service’s functionality from within its own application logic.

However the Web Service infrastructure that has been introduced in this chapter has chosen the path of simple functionality support, in order to encourage uptake and development (just as HTML and XML were widely used as a result of being open, simple standards). This is however a mismatch with the apparent goals of Web Services, raising the question of whether they can truly handle the complex real world integration needs of enterprises.

Considering the deployment of Web Services throughout the current systems using RPC and EAI solutions (where Web Services are being promoted as the next generation - see 2.1.2) and eventually in all B2B interactions, Web Services need to demonstrate a high level of scalability, efficiency, reliability, expressiveness, security and manageability (Bussler et al., 2003).

The current core Web Services infrastructure of SOAP, WSDL and UDDI is based on the primitive notion of synchronous remote invocation following a simple client-server model. Complex integration problems, such as B2B interactions based on peer-to-peer communication, are not in the scope of these core standards.

The Web is constantly growing and changing. Web-based services need also to be able to scale to and operate with this fast-moving global environment. While Web Services, as presented in this chapter, have benefits for Web-based access to computational functionality and for the needs of business application integration, the reviewed infrastructure is not able to support the complexity of large scale (Web-based) integration tasks.

The potential of making services available across the Web is to build a services-oriented Web architecture. This is about discovering globally distributed, heterogeneous, standalone software processes and permitting their composition and interoperation to produce re-usable, flexible and dynamic Web Services which can be employed by agents and enterprise systems to realise a required functionality.

The following chapter turns to the proposed Web Service infrastructure that aims to realize this potential. This infrastructure aims to do this by introducing semantic technologies to Web Services i.e. ontologies and knowledge representations.

Chapter 3

Semantic Web Services

3.1 Definition of Semantic Web Services

We introduce the concept of Semantic Web Services as an extension of Web Services as defined in section 2.1.1. To understand how this extension came about, we quote (Lara et al., 2003):

Web Services extend the Web from a distributed source of information to a distributed source of services. The Semantic Web has added machine-interpretable information to Web content in order to provide intelligent access to heterogeneous and distributed information. In a similar way, Semantic Web concepts are used to define intelligent Web services i.e., services supporting automatic discovery, composition, invocation and interoperation. This joint application of Semantic Web concepts and Web Services in order to realize intelligent Web Services is usually referred as Semantic Web Services.

The Semantic Web Services paradigm aims to cope with the need to automatically search, negotiate, select and interact with relevant Web Services among distributed actors in dynamic network environments where information sources, communication links, and actors themselves may appear and disappear unpredictably. An interesting metaphor to model such dynamic scenarios comes from the multi-agent systems area. In this case, a scenario is characterized by agents that provide services, agents that request services, and middle agents. Specifically, the matching process that allows requesters needs to be satisfied is demanded of middle agents, e.g., broker and matchmaker agents (Paolucci et al., 2002),(Tomaz and Labidi, 2003).

The ability of middle agents to autonomously provide the best matching between the service request and the service provider strongly depends on the ways of conceptualizing and organizing semantic information about services. To deliver on such a task, several researchers, e.g. (Horrocks et al., 1999),(Sycara et al., 2002), are dealing with formal languages to support a rich declarative specification of a wide variety of information about Web Services in order to allow agents for automatic service discovery, selection, composition, negotiation and contracting, invocation, monitoring of progress, and recovery from failure.

In order to make clear what we mean when we talk about Semantic Web Services, it may be useful to elicit the requirements of the Semantic Web Services languages (McGuinness et al., 2003)in terms of needed functional capabilities. We identify the following main requirements:

- Advertising and matchmaking. Such capabilities make it possible to discover new services that match the initial service request, e.g., customer needs. At this level, formalisms that allow services for a semantic description can be useful for the matching algorithms, e.g., for details (Li and Horrocks, 2003),(Paolucci et al., 2002), (Tomaz and Labidi, 2003).

- Negotiation and contracting. Even once the matching process has established a correspondence between the request and the service, it may still not be the case that the requester can actually use the service. Therefore, the negotiation and contracting processes are needed, which again rely upon a formal description of the service capabilities.

- Process modeling and composition. This aspect deals with composite services characterized by non-trivial internal structure. At this level, formalisms that describe data type, sequential patterns of operation, or patterns of valid interactions, can be useful in reasoning about the process models of the Web Service and for defining service compositions.

- Process enactment. Since composite services are structured as workflows, the service requester must be able to monitor the service execution. For example, if a service is far from being finished, the requester can decide to stop it.

In order to effectively satisfy the above requirements, which match with the aspects of discovery, interoperation, composition and invocation introduced in 2.1.1, the Semantic Web approach requires that data describing Web Services be not only machine readable, as is currently the case, but also machine understandable. In this way software agents can interact in an automated fashion with the semantically described services (Li and Horrocks, 2003):

Semantic Web Services aim to describe and implement Web Services so as to make them more accessible to automated agents. Here, ontologies can be used to describe services so that agents (both human and automated) can advertise and discover services according to a semantic specification of functionality (as well as other parameters such as cost, security, etc.).

3.2 Rationale for Semantic Web Services

In the previous chapter, it has been noted that Web Services are being used for business process integration as a successor technology to RPC and EAI. However the greatest potential benefits of this approach - the use of the Web as network infrastructure and Web standards such as XML for message exchange - are the root of the key problems of the current Web Service infrastructure.

Web Service deployment requires a significant level of manual effort. A human programmer is required to ensure and maintain correct service operation. There are associated business costs in time and expenditure. If Web Services are to become more attractive to businesses in terms of expenditure and ROI (return of investment) there is a need for greater automation.

Web Services also rely on the Web architecture to function. The Web is a dynamic, variable and changing place. Network connections and resources can change suddenly. The current Web Service infrastructure requires a stability and reliability that is not realistic to expect from the Web. Human oversight must be permanently available to react to changes in other Web services and in the network. This ties the usage of Web Service technologies, whether as a provider or requester, to a long term commitment to maintenance of a Web Service-based system. Hence another need for Web Services is self-maintaining systems.

Finally, the standardization of Web Services has focused on using XML as a common data representation format. This has many benefits in that XML is open, extensible, exchangeable, and is supported by many tools and related specifications. However XML documents are verbose and often not particularly human-readable. Their correct expression depends on the understanding of the XML tags by the human developer. The ambiguities that can arise cause invalid messages or message formulation or failed service discovery. XML Schema types, as pure syntax, have no relationship to the concepts they describe. The range of standards and specifications that have been produced do not share a common conceptual model of what Web Services are, which can impact on Web Service interoperability.

By taking the Web Service infrastructure onto the level of the Semantic Web, semantics will be applied to the issues of automation, self-maintenance (autonomy) and a common conceptual model. We can illustrate the areas which could be improved by the use of semantics by considering the service aspects evaluated in 2.3, namely service invocation, service discovery and service composition and coordination.

Service Invocation

Invocation requires the transmission of a message (e.g. in SOAP) to the service. The service will only execute correctly (i.e. not enter a fault state or return a fault message) if the content of the message respects the formal description of the service interface (i.e.

contains the requisite parameters, each of which has a valid value). To make invocation efficient, the message could be validated according to the interface description before sending.

However as far as the interface description is syntactic, the message validation can only also be syntactic (e.g. check the use of the correct datatypes for parameter values). This means that syntactically correct messages will be considered valid and sent, even when their content is inconsistent. If instead, a service interface specifies permitted values on its parameters in terms of ontological classes, then messages for this service can be checked in terms of ontological axioms (e.g. sub/superclassing). This validation could also be extended by semantic rules for specified classes and the given service. For example, consider the case of a travel agency Web Service, which books flights, accommodation, car hire and so on. In this case semantic rules might include trivial ”consistency” checks (e.g., arrival date should be at least one day before departure date in the case of a hotel), or real application conditions (e.g., apartments can be booked only for periods longer than one week), or even conditions that apply to a particular instance of a service (e.g., a given Web server can book hotels only in Tuscany).

Furthermore, a semantically enriched service interface can support developers in correctly invoking services according to the meaning of the content being passed in the message. For example, a parameter for Temperature could be validly invoked with the Celsius value rather than Fahrenheit (as the syntactic validation checks only for an integer) yet the result from a service expecting a Fahrenheit value would likely be interpreted incorrectly.

A semantic description of the service interface will also support modelling the invocation in the most effective way to meet the requester’s needs. It does this by representing how the service will react to different messages, so that a client can decide the best message structure to send. For instance, the travel agency can specify that a message containing an optional hotel name will result in booking that specific hotel and if the name is not specified then any hotel in the specified town could be selected.

Until now it has still been assumed that the requester has prior knowledge of the service to be invoked and hence knows how to align the internal application data to the data requirements of the service interface. However semantics can support automated invocation, where the service description represents its required data in terms of ontological concepts which are ”understood” by the requester. The requester could then dynamically map internal variables to service invocation parameters.

A further aspect of automated service invocation is the construction of a message from the service interface. While this is already possible from WSDL, the provision of semantics solves the problems of data mapping and consistency checking as already mentioned. In the case of document-style SOAP interaction the requester must construct an XML document from the internal application data. Where the service interface provides only syntactic constraints (in XML Schema), message construction is only possible for services known of at design time. Semantics enable however the dynamic generation of XML from a shared understanding of the application data and the service invocation document. This forms the basis for service negotiation and contracting as mentioned in the previous section.

Service Discovery

Web Service discovery aims to provide a framework for the location of services to meet a specific need. Current discovery approaches can be likened to Web search engines, in that they require individuals to formulate queries and sort through the results for the most relevant resources. This does not scale well to complex business processes which may require the regular location of multiple Web Services with constantly changing requirements.

Businesses require automated service discovery. This is particularly important for Web Service composition. Composite Web Services contain multiple Web Service references which are generally static (fixed within the composition model) or dynamic (in the sense of a formulated query to a UDDI registry). The former approach means that when a specified service fails, the entire composed process fails. The latter approach can not take into account changes in client requirements at run-time or new requirements, which arise during execution. Run-time dynamic matching of services with actual functional requirements solves these problems.

A second benefit of dynamic discovery is the ability to select the most appropriate service at run-time. Selection criteria could be related to QoS issues (e.g. find the service with the currently quickest response time) or contextual issues (e.g. find the service whose response best matches the preferences of the current user).

Dynamic service discovery is made possible using service classification meta-data containing concepts described formally in an ontology, e.g., by means of a description logic based knowledge representation as opposed to UDDI's tModels. Service queries can be as expressive as the underlying service classification meta-data model, and therefore far richer, than would be possible with UDDI/WSDL-based approaches. Queries could include logical expressions and support reasoning across concepts. The discovered services will depend on the matching effectiveness. If the customer's needs and the advertised services are described according to a common ontology, it possible to perform an automatic crossing between customer's requests and supplier's advertisements. In particular, as outlined in several approaches (e.g., (Li and Horrocks, 2003),(Paolucci et al., 2002)), different matching degrees have been considered. Specifically, according to (Li and Horrocks, 2003), the matching can be exact, i.e., the request and the advertisement coincide; plugin, i.e., the request is a sub-concept of the advertisement; subsume, i.e., the advertisement is a sub-concept of the request; intersection, i.e., when the intersection of request and advertisement gives as result a non empty set; and disjoint, i.e., when the intersection of request and advertisement gives as result an empty set.

Service Composition and Co-ordination

In a service-oriented architecture user goals are achieved by locating and communicating with distributed software processes advertised by service providers. In Web Services the advertising, distribution and communication with these services takes place on the Web.

While simple goals may be achieved by a simple request/response interaction (as modelled in SOAP/RPC) it is inconceivable that all potential user goals will be directly implemented by a corresponding service. Rather, complex goals can be achieved by co-ordinating the interaction between two or more services. The functionality to achieve the complex goal can be packaged into a single Web Service (offering a single interface to other clients with the same goal) through service composition. Business processes have complex goals, so there is a business need for both the co-ordination and composition of services. As an example in this section, we return to the travel agency Web Service, which realises the goal of booking a holiday by the invocation of other services for flights, hotels, car hire, etc.

Co-ordination requires a correct understanding of the meaning of the messages in a conversation, if a developer is to implement the rules for a conversation between services correctly. Semantics can formally describe this meaning. Another aspect of this would be supporting an application in carrying out a valid conversation with a service by the semantic interpretation of the services declared choreography. This is required where the service, and hence its choreography, is not known a priori, such as with dynamic service selection.

Co-ordination also involves conversation monitoring, whether this is done by the participating services (checking that their own choreography is being respected) or by a co-ordinator. Monitoring of choreographies expressed syntactically can not flag semantic errors. Where the choreography is using semantics, ontological axioms and additional rules for consistency checking can be utilised to ensure that a conversation between services is ”meaningful”. For example, when conversing with a car hire service, a request-availability operation will be invoked, followed by reservation and payment. Now, if the first operation is invoked by giving “Smart Car as a parameter, and such an automobile is found to be available, then no conversation (process flow) rule will be violated if the booking is completed. However, given a booking party of 5, a semantic process may infer that such an automobile is indeed too small.

The co-ordinator could also become a service mediator with semantic co-ordination. In other words, a mediation process could dynamically determine the valid conversation required between a set of services to realize a certain goal. Dynamic mediation would not only seek to ensure that the resulting conversation respects individual service choreographies but also respects run-time variables such as service state and data content. This would release developers from the need to specifically create co-ordination protocols for realising tasks with a static set of known services. Rather, dynamic discovery could be used and the required conversations determined at run-time. The travel agency Web Service for example may change the services it accesses for bookings based on advertised changes in QoS, or tailor them to the clients' preferences. The different services may require a different choreography for making a booking, in which case the travel agency will need to dynamically alter its conversational process.

Composition also benefits from semantics in that dynamic service selection can be incorporated into the composition model. Effective service retrieval is based on semantically rich queries, and these queries can be formulated within the composition model in terms of internal data and state only when the model itself has an understanding of the semantic concepts represented by that data and state.

The availability of semantic descriptions of service functionality for discovery purposes could also be utilised for composition. Rather than seeking to match a user's goal to a single service in a semantic discovery framework, a user's goal is matched to a composition of services. In other words, service functionalities are treated as sub-functionalities which are combined to form more complex functionality that achieves the user's goal. The resulting combination of services can be expressed using a composition model. Within the model, different services are invoked, each with its own service interface and data structures. From the internal view of the model (the orchestration) the flow of data and state information should be consistent. Hence composition needs interfaces between the internal view of the composition process and the heterogeneous views of the participating services. While composite services using static service selection can be authored with these interfaces explicitly declared, automating the task of inter-operability between heterogeneous services would lessen development time and make composite services more maintainable. Automation becomes a necessity in the case of dynamic service selection. Semantics enable such automation by providing a shared understanding of the execution domain between the internal and external views of the composition model. This is achieved by either sharing or aligning ontologies.

The consistency of a service composition model is measured in the validity of its data and state flow, with the stated goal of a certain result (in terms of data and state) at the conclusion of the execution of the composed service. However, changes in state are not exclusively syntactic as Web Service execution can result in changes in the state of the world, which is not directly measurable within a computer system. Such changes in state are best modelled in ontologies, where concepts from the world can be represented and their relationships expressed, and constraints can be stated which determine if an execution result is meaningful and hence enable the success of a composite service in achieving a goal.

In a semantic composition model, this involves describing service execution not only in terms of the input and output data-types, but also in terms of the preconditions (read as ”constraints on service invocation”) and effects (read as ”repercussions of service invocation”) on a formal model of the situation within which the invocation will take place. As an example, the price of the trip (a service output) may be calculated by summing the cost of the flight, the hotel, the car hire, and the agency servicing fee. This mapping between the input order and the output price can be described among the effects of the service. The payment, meanwhile, can be modeled as an effect, which updates the model of the domain, by reducing the amount of money in the bank account of the user. A reasoning system using the service could then aim to meet user requirements for a travel booking while minimizing the total price.

More sophisticated use of preconditions and effects can be envisaged for handling execution roll-back. For instance, when organizing a trip, we want to make sure that both the hotel and the flights are booked, and that the reservations respect certain constraints (e.g., the check in date for the hotel is also the arrival date at the destination). If we discover that this is not possible (e.g., after booking the hotel we see that there are no available flights left), we want to cancel all reservations (i.e., cancel the hotel reservation). One can see that a model of the domain and of preconditions and effects of Web Service invocations are necessary to define these kinds of conditions on Web Service compositions.

3.3 Current Semantic Web Service Initiatives

Software agents, in order to automatically discover, invoke, compose, and interoperate with Web Services, require a description of the properties and capabilities of such services in a computer-interpretable form, and the means by which the services can be accessed. Current specifications such as UDDI, WSDL, SOAP, BPEL and BPML cannot fully semantically represent the components Web Services make use of in order to support automated service discovery, invocation, composition and interoperability (Rajasekaran et al., 2004). The technologies investigated in this section aim to support these tasks. They attempt to achieve this by augmenting Web Services with the semantic description of various aspects of Web Services.

3.3.1 DAML-S/OWL-S

OWL-S is a Web Service ontology framework based on OWL [1]. Its main goal is to enable a user or a software agent to automatically discover, invoke, compose, and interact with web resources offering a service that adheres to requested constraints. (Polleres et al., 2004)

Previous releases of OWL-S were referred to as DAML-S and were built upon the DARPA Agent Markup Language and Ontology Inference Language, DAML+OIL (Ankolekar et al., 2002). DAML+OIL is a description logic (DL) based Web ontology language which describes the structure of a domain in terms of classes (concepts in DL) and properties (roles in DL).

Partners working on the OWL-S language include BBN Technologies, Carnegie Mel-

[1]OWL-S 1.0 release, URL: http://www.daml.org/services/owl-s/1.0/

lon University, Nokia Research Centre, Stanford University, SRI International, USC Information Sciences Institute, University of Maryland, Baltimore County, University of Toronto, Vrije Universiteit Amsterdam, University of Southampton, De Montfort University and Yale University.

The OWL-S ontology was developed to provide service descriptions with additional semantic information enabling service descriptions to be made and shared. This additional information consists of a set of markup constructs (i.e basic classes and properties) which can be used to declare and describe services.

OWL-S can support complex services as well as simple services. In the OWL specification, complex services are referred to as services which are composed of multiple services. These often require more than one interaction between the user and the service where the user has to provide information based on certain conditions.

According to the OWL-S ontology specification, a service can be defined through the use of core components the Service class is made of: the Service Profile, the Service Model, the Service Grounding and the Resource Ontology.

The Service Profile sub-class of the service class tells agents ”what the service does”. This sub-class aims to support automatic service discovery. Its sub-classes include the organization providing the service, the service name, functional aspects of the service such as inputs, outputs, pre-conditions and effects (”IOPE”), which point to the functionality instances in the Service Model, non-functional aspects of the service such as the contact information of the individuals or organization responsible for the service and additional features used to specify the characteristics of the service (e.g category of the service, quality of service and service parameters such as max response time and geographic availability).

The Service Model sub-class tells agents ”how a service works”. This sub-class aims to support automatic service invocation. In order to control the interaction with a service, it defines the transformation of data from a set of inputs to a set of outputs (e.g from credit card number to a receipt). In order to describe the state change during the execution of a process, it makes use of pre-conditions and effects (e.g. if the card number is valid and the account is not overdrawn, charge the card and organize the physical transfer of goods from the seller to the buyer). The inputs, outputs, pre-conditions and effects should be described as in the Profile Model sub-class. The Service Model sub-class models three types of processes: atomic, composite and simple. The same processes are defined in the Profile sub-class. The Service Model sub-class defines the control flow and the data flow of the constituent processes of a service. The operation descriptions and message descriptions in WSDL can have an OWL-S process attribute mapped to the OWL-S atomic process in the Service Model ontology. WSDL abstract types can also be mapped to different types of OWL-S inputs and outputs of atomic processes. These describe a mapping to an invocable description.

The Service Grounding sub-class specifies ”how agents can interoperate with a service” via messages. Its components enable the transformation from inputs and outputs of OWL atomic processes to concrete atomic process grounding constructs, which can be transmitted in WSDL 1.1 messages. OWL classes of inputs and outputs can be defined as extensions in separate documents and referenced in WSDL descriptions by using the OWL-S parameterType property of the referenced input or output object. Also, the serialization of the message parts of an OWL type can be specified in the encodingStyle attribute. The use of WSDL for grounding presents the following limitations. Each WSDL operation may have an OWL-S process attribute indicating the name of the OWL-S atomic process. However, the current WSDL specification does not make this possible. Also, according to the OWL-S specification, OWL atomic processes can make use of conditional outputs. However, there is only a single output message specification for a given operation in WSDL 1.1. This limits the expression of conditions and the reasoning based on them.

The Resource Ontology contains classes of attributes to express pre-conditions of processes and sub-classes for resource composition. However, due to limitations in the Resource ontology and limitations in the expression of rules in OWL, it is not clear in the OWL-S specification how post-conditions can be used to support automatic service composition and interoperability.

3.3.2 IRS-II

IRS-II, which stands for Internet Reasoning Service, is a framework and implemented infrastructure for Semantic Web Services, conceived as reusable online problem-solving resources. (Motta et al., 2003)

Partially supported by the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration, the group of researchers working on the IRS-II project includes the University of Aberdeen, Edinburgh, Sheffield, Southampton, the Open University and the Dipartimento di Scienze dell'Informazione at the University of Bologna.

The primary goal of the IRS-II project is to support the discovery and retrieval of knowledge components (i.e. services) from libraries distributed over the Internet and their semi-automatic configuration in order to realize specific tasks according to user requirements. The benefits IRS-II claims to provide are scalability and flexible mediation between problem and services. (Crubzy et al., 2003)

IRS-II has two key features. Firstly, it is built on knowledge modelling research on reusable components for knowledge-based systems. Secondly, it makes use of the Unified Problem-Solving Method-Development Language (UPML) framework for knowledge annotation of problem solving resources, built within the IBROW project. (Omelayenko et al., 2000) IRS-II separates the problems to be solved (i.e. task model) from the different ways to solve them (i.e Problem Solving Methods) from where these problems can be solved (i.e. domain model). The domain-independent problem solving methods (PSMs) can be used to address stereotypical knowledge intensive problems such as diagnosis, design and classification. The UPML framework in IRS-II enables flexible n:m mappings between problems and methods. Hence matchmaking agents can reason about the task-method competence matching of a method in a given scenario. In the UPML framework problem solving is modelled as finding a solution (i.e. an output class) that best explains a set of known facts (i.e. input observables) about an unknown object. PSMs are selected if their post-condition fulfills the goal expression of the task and if their pre-conditions do not contradict the assumptions of the task. IRS-II uses four UPML classes of components: domain models, task models, PSMs and bridge adapters. A description of each of these components follows.

Domain models are classes used to identify groups of similar objects.

Task models are achievable functions. Each task is specified in terms of input roles (e.g observables, match criteria, solution criteria and score comparison criteria), output roles (i.e solutions), pre-conditions (e.g observables and candidate solutions should be non-empty), assumptions (e.g the output of the task will include at most one solution) and default goal (e.g find a solution which is admissible with respect to the given admissibility criterion). Symbol level requirements (i.e input and output roles) are mapped to XML Schema types in the SOAP bindings.

Problem Solving Methods (PSMs) are abstract descriptions of reasoning processes which can be used to solve tasks in different domains. Each PSM includes the following sub-tasks: data abstractions, solutions abstractions and solutions refinement. Data abstractions are used to collect, select or apply abstraction mechanisms (e.g has a high temperature). Solutions abstractions are used to match abstractions to possible explanations and ranking with respect to a set of observed features or according to a match criterion (e.g has fever). Solution refinements are used to collect, select or apply abstraction mechanisms to generate more specialized solutions according to domain knowledge about the solutions. (e.g has yellow fever) Each sub-task is defined using specific input and output roles (e.g has-observables, has-solutions) specified by the user or provided as defaults as well as general pre-conditions and assumptions on domain knowledge and pre-conditions (i.e logical goal expressions). (Motta and Lu, 2000)

At the practical level both task models and PSMs make use of ontologies (i.e a library of resources formally modelled and pragmatically annotated), metadata annotations in RDF format (e.g Dublin Core) and costs. Bridge adapters map relations between ontologies of two types of components for the components to be configured, connected together and executed in different systems. The adapters need to be defined when PSMs don't comply with both the ontology and with the SOAP encoding of inputs and outputs. The adapters can map PSMs (e.g classification) to domain models (e.g identification of apples). They can also map PSM abstractions (e.g sweetness) to tasks (e.g sugar-level classification) providing a solution for dealing with ontology mismatches. Finally, they can adapt existing resources (e.g introduction of fuzziness) to generate specialized solutions.

The IRS process model consists of eight phases: task selection, task configuration, domain selection, task verification, PSM selection, PSM configuration, PSM verification and application execution.

To enable task selection and configuration, the solution space of classes of similar objects (i.e set of possible solutions) is mapped to domain knowledge (e.g hierarchy). The input observables (i.e set of possible features) are also mapped to domain knowledge (e.g attribute). The tasks are selected as refinements of high level tasks that match with the requirements of the domain knowledge. So the attributes are transformed by way of mapping relations into task-complaint inputs. The same mappings can be used for the outputs to conform to the domain ontology. Task-domain ontology mapping relations, which are the architectural elements of domain-specific configuration knowledge, are stored in a task-domain bridge as reusable resources. IRS-II provides structured mapping templates which help users select the tasks.

In the domain selection phase IRS-II checks the input roles and the assumptions that the task defines on domain knowledge and notifies the users about inputs that do not satisfy the assumptions of the task. If required, it directs them back to the configuration step. According to the specification, not all assumptions can be verified.

During task verification users can select a domain model from a UPML complaint library with reusable mappings. Alternatively, they can provide their own knowledge base with mappings.

The PSM selection and configuration phase is carried out as follows. PSM descriptions that match the configured task are computed using a set of PSM-task bridges. The input and output roles of the PSM are mapped to the inputs and outputs of the configured task. Alternatively, matches can be computed with a competency matching process based on first-order logic, functionality that the current IRS implementation does not provide yet. Users can select the PSM that they think better matches the task. The PSM configuration can be mapped to a domain and stored as a PSM-domain bridge.

During the PSM verification phase IRS-II checks the goal of the task against the preconditions of the PSM and the assumption of the task against the preconditons of the PSM and notifies the users about any domain input that does not satisfy the PSM requirements and, if necessary, it directs them back to the configuration step.

During the application execution phase IRS-II acquires user case data and interprets the domain-task, task-PSM and domain-PSM mapping relations. It transforms the user specified attributes into the inputs of the configured PSM and invokes this to realize the configured task. This way it retrieves the location and type of PSM code. Finally domain outputs are filled in with the results of the PSM execution.

Finally, in addition to the features described so far, IRS-II can publish standard Web Services from programming code such as Java. It achieves this by adding wrappers to the Java code.

IRS-II supports RDF so PSMs can be published as RDF resources on the Web. In WebOnto all the components of the UPML framework, including their operational contents, are modeled in the Operational Conceptual Modeling Language (OCML), an Ontolingua-derived language. OCML provides a relation mapping mechanism to enable the matching of PSMs to queries in Web Service discovery and to tasks in Web Service invocation and composition. According to the UPML framework, each PMS library can use a specific knowledge-modeling language to describe its resources. (e.g OIL, OWL) However, the need for interoperability requires the use of shared ontologies and common modeling languages or standard knowledge-level APIs (e.g Protégé supports OKBC).

The IRS-II architecture includes brokering and registry mechanisms and consists of three components which use the SOAP protocol to communicate with each other: the IRS server, the IRS publisher and the IRS client. The IRS server contains the Semantic Web Services descriptions (i.e domain models, tasks and PSMs) and the mapping rules to connect knowledge descriptions to specific Web Services. In one implementation a plug-in extension to the ontology modeling tool Protégé 2000 is integrated to enable library providers to define their own PSMs with properties that can be browsed. In a second implementation it is integrated with the WebOnto ontology server, with the OCML modeling and reasoning platform and with a library of problem solving resources accessible online through an API. The IRS publisher can link Web Services to their semantic descriptions in the IRS server and can generate wrappers to transform standalone Java code into a Web Service described by a PSM. The IRS client takes user case input (e.g attributes and types) and invokes Web Services. This can also be done programmatically through a set of client APIs which are task-centric. On selection of a task to be achieved the broker locates an appropriate PSM and then invokes the corresponding Web Service.

3.3.3 METEOR-S

Started in 2002 at the LSDIS Lab at the University of Georgia as the follow on project of Meteor (i.e Managing End-To-End Operations), a project focused on techniques from the Semantic Web, Semantic Web Services and workflow management, Meteor-S aims to integrate Web Services standards such as Business Process Execution Language for Web Services (BPEL4WS), Web Service Description Language (WSDL) and Universal Description, Discovery and Integration (UDDI) with Semantic Web technologies. (Verma et al., 2004)

The Meteor-S project contributors include Faculty Members, active students and Alumni from the Computer Science Department based at the LSDIS lab at the University of Georgia. [2]

The main goal of the Meteor-S project is to provide Web Services with enhanced dynamism and scalability through the application of semantics in the Annotation, Discovery, Composition, Quality of Service and Execution of Web Services. In order to achieve this goal, it has added semantics to the following layers of the Web Services conceptual stack: Publication and Discovery Layer, Description Layer and Flow Layer. In the Publication and Discovery Layer the input and output data of Web Services are mapped to ontolo-

[2]METEOR-S: Semantic Web Services and Processes. URL: http://lsdis.cs.uga.edu/Projects/METEOR-S/index.php

gies. The Description Layer includes annotations of operational data of Web Services, pre-conditions and effects. The Flow Layer is based on BPEL4WS and uses State Machines, Petri nets and activity diagrams to formally represent the flow of a service in a process or the operations in a service for verification, simulation and exception handling of the process model. It also uses the QoS model for Web Services to describe operational metrics of services and processes. (Sivashanmugam et al., 2003)

The Meteor-S project was developed in three phases. The first phase consists of the Semantic Discovery Infrastructure (MWSDI), which leverages peer-to-peer networking as a scalable infrastructure for automated and semi-automated Web Service publication and discovery. The Semantic Annotation of WSDL (WSDL-S), which was developed in the second phase, also supports service discovery. The Semantic Web Process Composition Framework (MWSCF),which represents the third phase of the Meteor-S project, consists of tools supporting the automatic service discovery and composition, control flow and data flow based on process and business constraints. (Sivashanmugam et al., 2004) (Aggarwal et al., 2004) Here is a detailed description of each of these phases.

The Semantic Discovery Infrastructure (MWSDI) is an infrastructure of registries for the semantic publication and discovery of Web Services. It makes use of ontologies and it categorizes registries based on domains. Its architecture is divided into four layers: the Data layer which is a P2P network of registries providing Web Services, the Operator Services layer which maintains the services provided by the operator peers, and the Semantic Specifications layer (i.e ontologies spread across all the layers) which supports service publication. The Data layer consists of UDDI registries mapped to one or more domains classes in the Registry ontology. It provides access to one or more services depending on the search query. The Communications layer consists of a P2P infrastructure for distributed components to interoperate. The Gateway Peer is the only peer that controls access to the network for new operators to join the network and updates the Registry Ontology for all peers. The Operator Peer acts as provider of operator services and the Registry Ontology. The auxiliary peers make sure the Registry Ontology is available. The client peers can connect to the network to discover Web Services. They can achieve this by creating and sending a template to the Operator Peer and by invoking the Operator Service.

The P2P network was implemented using the JXTA framework to enable interoperability and support for all platforms and devices. It makes use of the following protocols: the Peer Discovery Protocol, to enable peers to discover each other, the Pipe Binding protocol to bind a pipe (i.e communication channel) to more than one peer at runtime, the Operator Peer Initiation protocol to define the process for adding a new registry to the Registry ontology and the Client Peer Interaction protocol to enable a client to enter the network and request the Registry ontology for either discovery or publication of Web Services. The Client Peer Interaction protocol enables the client to enter the network as a guest peer and makes a request for the Registry ontology. One of the peers in the network provides the ontology. The ontology is displayed by the user client interface. The user selects a specific domain. The user sends the Web Service publication (or discovery) details to the corresponding Operator Peer. The Operator Peer executes the relevant Operator Service to publish the Web Service in the registry it maintains (or to query the registry and return the services published in the registry to the client).

The Operator Services layer maintains the services provided by the operator peers such as the semantic discovery and publication of Web Services in UDDI. The client peer uses this layer to communicate with the registries. The users can select registries, discover services and publish services in registries through the use of templates. This layer translates the templates to a registry specific format in order to perform specific functions. It can also provide additional value-added services (e.g wrapper service to translate registry entry details in UDDI data structures specifications and vice versa during the SOAP message processing)

The Semantic Specifications layer, which consists of the ontologies used to support the semantics based publication of Web Services, enables a number of operations. It allows adding domain-specific semantics operations, and pre-conditions and effects to WSDL descriptions, registering the descriptions in domain specific registries and locating registries and services based on user requirements. This is achieved by matching the inputs and outputs of the service requirements provided by the user with the semantic annotations of the services stored in the registries. The input and output types in the WSDL descriptions are mapped to input and output taxonomies (i.e tModels) and grouped in a domain specific ontology (e.g departureCity and arrivalCity are mapped to airportCity in outputs AirTravel taxonomy), which contains the terminology used by Web Services in a domain. The tModels names are mapped to domain ontology concepts (i.e key-value pairs stored in UDDI registries). CategoryBags are also used for the categorization. As a result, a query can be restricted to one group of related partners.

Web Service functional properties such as operations, inputs, outputs, pre-conditions and effects are mapped to an ontology. Web Service non-functional properties such as name, version, API supported, operator details, QoS and constraints are also mapped to an ontology. The mapping is provided as a service in the service layer. Two types of mapping were implemented: manual mapping in WSDL files stored in UDDI data structures and semi-automatic mapping stored in UDDI.

The Semantic Annotation of WSDL (WSDL-S) project (Rajasekaran et al., 2004) focuses on adding semantic descriptions to WSDL and to UDDI in order to provide semantic discovery. Its tools are the Semantic Web Service Designer, the Semantic Descriptor Generator, the Publishing Interface, the Discovery Engine and the XML repositories.

The Semantic Web Service Designer is used to add semantic descriptions to the source code. The services are designed by providing associations between service inputs, outputs, port types, exceptions and ontological concepts and constraints for pre-conditions and pre-conditions. For each operation the tags describe input and output parameters, exceptions and constraints, which themselves include pre-conditions and post-conditions, interface-specific annotations and service parameters (e.g ”location”, ”quality of service” and ”reliability”). The constraints are represented as Boolean expressions.

The Semantic Descriptor Generator is used to add semantics to the WSDL descriptions to specify ”what the service does”. It takes as input the annotated source code (e.g Java) and maps Java types into XML Schema datatypes by means of a hash table. It makes use of an interface and can be called by an application through an API. In this last case it performs more stringent matching. The descriptor generator uses other components: the Document Generator, the Type Converter and the Validator. It can generate annotated WSDL1.1, WSDL-S and OWL-S files associated with the annotated source code. Each operation in the port type of a WSDL file can be mapped to a concept. The message types can be described using XML Schema Definition (XSD), OWL or UML/XMI.

As part of the descriptor generator, a WSDL-S Meta-Model was developed. According to this model, which presents extensions to WSDL 1.0, the namespace feature can reference classes and properties from the RosettaNet ontology using OWL for example. Each operation can have an action attribute. Exceptions tags can be used to represent the exceptions thrown by an operation. The constraints (i.e. Boolean expressions) are converted into SWRL rules specified in WSDL-S documents. WSDL inputs, outputs, exceptions and constraints tags have element attributes. Pre and post condition tags can also be used to depict the conditions for an operation to be executed. Java data types can be transformed into XSD types. It appears that type system round-tripping (i.e transformation of XSD types into Java types, Java types into OWL types, OWL types into XSD types and the other way round) is not simple.

The Publishing Interface advertises services (i.e annotated files) in enhanced UDDI, a layer above UDDI which can handle semantic data. Four enhanced UDDI data structures are used to accommodate semantic information: the Business Entity structure, which holds the provider information, the Business Service structure, which provides the service description and the Binding template, which provides service access information with references to domain and location of T-Models. T-Models provide service parameters descriptions (i.e each KeyName is associated to a KeyValue which points to a concept in the ontology). Category bags are used within the first three UDDI data structures for categorizing services based on semantics.

The Discovery Engine, which contributes to both the discovery and composition of Web Services, queries semantically enhanced UDDI data structures and compares concepts and properties of concepts to produce effective ranked responses. The results benefit from the UDDI categorization based on domain and locality of service, and from the operation-ontology and message part-ontology mappings. The services are selected by computing the degree of similarity of the input and output semantics of the candidate services to the requirements template. The match can also be determined by computing the similarity of the pre-conditions and effects semantics of the candidate services to the requirement templates. This can be achieved by using preconditions and effects tags. It can also use additional ranking criteria. These include cost, availability, reliability, domain specific QoS and analysis of constraints on operation. It can also use heuristics based on subsumption-relations, data-type matching between requestor specified constraints and provider specified concepts, common ancestor, property or child, in order to compare ontological concepts specified in the query with those published in the registry. Finally, in order to rank relevant services, it can use metrics based on properties comparison, data types and cardinality matches and distance from the common parent. When the discovery engine is called by the execution engine, it needs to perform more stringent matching because there is no human intervention to adjust it.

A number of repositories are used during process design, service discovery and process generation. There are ontologies to map inputs and outputs (i.e data types) and operations. These are used in service description and in the annotation of semantic templates. They are published in UDDI and use T-Models ID to be identified. There are also semantic Process Templates that can be edited, categorised and stored. Each of these templates is stored in a particular collection that represents a category based on the identifier. Finally, the Activity/Services interface definitions in WSDL syntax are stored in a different repository. They can be browsed and selected in order to link to an activity during the design of a process.

The Semantic Web Process Composition Framework (MWSCF), which supports the automatic discovery and composition of services, makes use of the following tools: the WSDL files with the descriptions of the process operations of the participating Web Services, a BPEL file containing the process definition and the BPEL4WS orchestration server. It also uses the MWSDI and data in repositories (i.e ontologies, activity interfaces and process templates) to convert the process templates into executable format. The MWSCF adds four types of semantics to the description of Web Services: Data semantics, which enable agents to communicate with Web Services; Functional semantics, which enable agents to invoke a Web Service; Quality of service semantics, which enable agents to select Web Services that satisfy their requirements; and Execution semantics, which enable agents to interoperate with Web Services. Data semantics consist of inputs, outputs and exceptions mapped to OWL classes or standard vocabularies such as RosettaNet. Functional semantics consist of a set of related operations, pre-conditions and postconditons mapped to concepts in an ontology. Quality of service semantics consist of metrics based on service time, cost, availability and reliability. Execution semantics consist of the process flow information and the data dependencies between Web Services.

The dynamic composition of Web Services in Meteor-S is achieved by automatically binding Web Services to abstract processes, based on process and business constraints. The descriptions of constraints, cost estimation and service optimization are used to support the constraints analysis and quality selection of services. The Semantic Web Process Composition Framework uses WSDL4J for processing WSDL files, the Jena toolkit and DAML API for building and processing ontologies and the BPEL4WS industry standard for modeling workflow patterns.

The MWSCF architecture makes use of the following components: the Abstract Process Builder, used to design and abstractly represent the control flow between the services, the Discovery Engine, used to support semantic publication and discovery, the Constraint Analyser, used to rank services based on business and process constraints, and the Process

Execution Engine, used to bind the matched services to their abstract process and execute them.

The Abstract Process Builder enables programmers to use Semantic Process Templates (SPT) to design the semantics of each activity and the control flow used to form an executable process. This feature supports dynamic discovery of the services required to carry out each activity and the generation of the appropriate executable process based on the discovered services. The functional semantics of an activity (i.e inputs, outputs, pre-conditions and effects) are specified using a Semantic Activity Template describing the requirements of a service. The internal distributor services for the operations that don't change are statically bound to the process while the services that are decided at run time (e.g supplier services) are dynamically bound to the process. The semantic activity template needs to be designed for the services that are bound to the process dynamically. During discovery the semantic requirements of the activity and QoS requirements specified in the semantic activity template are compared with candidate semantically annotated WSDL service operations linked to QoS specifications (to be specified in WSEL - the Web Services Endpoint Language - a proposal from WSFL for the description of non-functional characteristics of services) for ranking and selection purposes. The process builder, in order to compose the activities, specifies the activities and control flow constructs (e.g sequence, flow, switch) in a semantic process template in XML format. This process template contains a collection of activities linked to BPEL control flow constructs. Receive and reply constructs in the template are mapped to an operation in the WSDL file which captures the messages received and returned by the process. The process template specifies the discovery QoS criteria and ranking details for each activity so that the relevant services to carry out activities of type interface or semantic template can be discovered and selected for the composition of the services. The most relevant service is retrieved by performing three operations: automated service discovery, constraint analysis and constraint optimization, based on user constraints.

Given a template, the Discovery Engine returns the services that best match the template. The service descriptions in the WSDL files, which are mapped to a domain specific ontology, are semantically compared to the requirements and ranked.

The Constraint Analyzer, in order to enable the dynamic selection of services, uses constraint representation and cost estimation. The Constraint representation module, which represents the required business and technological constraints and partnerships, can be derived from the RosettaNet ontology. The Cost estimation module queries the Constraint representation module to estimate the cost of various factors affecting the selection of a service for the processes. These include querying and cost estimation (i.e cost of procurement, delivery time, compatibility with other suppliers, relationship with supplier, response time of supplier, reliability of service), service dependencies (ie the selection of one service will affect choices for other services) and process constraints set on actual values or estimated values on QoS specifications (e.g time, cost, availability, reliability). The process level quality QoS is calculated as the aggregation of QoS of all the services in the process. The values of the parameters can be pre-set or estimated. In order to set the priorities between these factors (e.g time, cost and supply time), the Cost estimation module assigns costs to them based on one or more of the following: the results obtained from querying the Constraint Representation module, the results obtained from querying an internal database or the results obtained from querying the suppliers' Web Services.

The Constraint Optimizer converts the Process constraints (e.g time, cost, cost partner preference) into constraints for an Integer Linear Programming Solver called Lindo. It uses an equation to weight the cost and preference of a service according to the process designer. The function for the optimization is extracted from the service template with operations and constraints defined by the user. This function will rank the constraints from optimal to near optimal solutions. The Process Designer can then select the optimal constraints and send them to the run time module.

The Process Execution Engine uses the run time module (i.e Process generator) to convert the abstract process and service templates into a BPEL4WS Web process which can be executed in any BPEL execution engine. Users can specify the process flow and data dependencies between the Web Services using BPEL4WS. They also need to specify the location of the WSDL files. The Binder takes the optimal set of services bound to the abstract process and builds an abstract executable BPEL file (i.e with placeholders for service details to be filled in). The BPEL placeholders (i.e the invoke elements in the generated process) are filled in with the service description and location details (i.e portType and namespace) extracted from the WSDL file. The input and output container details of the invoke elements are generated from the data flow details provided by the user. WSDL data, data flow provided by the user and the flow constructs in the template generate a BPEL executable process. An in-memory model of the process is written in a BPEL file. This is validated by the BPEL4WS orchestration server, it is deployed and it can then be invoked by another Web Service.

3.3.4 WSMF/WSMO

WSMF

The Web Service Modeling Framework (WSMF) defines a rich conceptual model for the development and the description of Web Services in order to bring this technology to its full potential. (Fensel and Bussler, 2002a)

In order to achieve the objective stated above, WSMF establishes the use of the principle of maximal decoupling and strong mediation. (Fensel and Bussler, 2002b) It also defines a number of components which should be used in order to model an ontology and formal language for describing Semantic Web Services. These components are as follows: Goals, Ontologies, Web Services and Mediators.

Goals are problems that Web Services need to solve. They include pre-conditions (i.e what a service expects to receive) and post-conditions (i.e what a service returns in response to an input). Ontologies are used to link machine processable content with human meanings based on consensual terminology. Web Services are black box and grey box descriptions of various Semantic Web Services aspects. They include different levels of acknowledgement (i.e message reception, understanding and exchange layer) and support the declarative specification of Web Service composition through the use of constraints. The key role of Mediators is to bypass interoperability problems and provide a human-understandable description of the goal. They include adaptors enabling the combination of objects that differ in syntactical input and output descriptions and specific mediators used to mediate between data structures, message exchange protocols and business logics (i.e compensation for data mismatches and process sequencing mismatches).

WSMF does not define a concrete syntax or the semantics for the framework. According to the WSMF specification, this could be achieved through the use of WSFL or OWL-S and the Process Specification Language (PSL). On the contrary, WSMF defines the conceptual model for the development and description of Web Services and its key elements.

WSMO

The Web Service Modeling Ontology (WSMO), developed as a refinement and extension of WSMF, is an ontology and formal language for describing various aspects of Semantic Web Services. (Roman et al., 2004a). These include the components defined above. The WSMO specification, which is currently under development, uses a layered approach, which includes the definition of three different ontologies: WSMO Lite (i.e a basic ontology), WSMO Standard (i.e a mature set of concepts) and WSMO Full (i.e the full ontology). [3]

The WSMO working group is an initiative from the SDK Cluster, which includes the Knowledge Web project contributions. The Knowledge Web project deliverables will contribute to the development of WSMO. Other partners are HP Galway, British Telecommunications, Open University, Tiscali and many other partners from industry and research.

WSMO aims to meet three goals: to describe the domain of Semantic Web Services, to define a conceptual model for a formal language design and to define an execution environment which can enable the complete definition and execution of Web Services' interactions. (Sinuhe et al., 2004) Also, it has been designed based on the following domain-specific principles. Firstly, the external interfaces/behavior should be different from the internal interfaces/behavior. Secondly, it proposes the use of a peer to peer approach instead of a client-server approach. Thirdly, it makes use of instance description instead of type description.

Non functional properties and value constraints are classified in two categories: core properties (i.e Dublin Core Metadata Element Set plus the version element), which are

[3]WSMO Working Drafts, URL: http://www.wsmo.org/2004/

defined globally and Web Services-specific properties, also related to the QoS, the quality aspect of a Web Service.

Ontologies define formal semantics to enable information processing by a computer and to link these semantics with meanings agreed by humans (i.e real-world semantics). WSMO aims to achieve this by providing concepts, relationships among the set of concepts and their semantic properties captured by logical expressions. In WSMO these logical expressions, which consist of non functional core properties defined by a logical constraint, are named axioms.

Concepts have attributes with names and types (i.e range), can change overtime and can have super-concepts specified by an ”is_a relation”. They also have methods that can be invoked on each instance of a concept and that can take parameters. The possible values that instances can take are the range. The possible values that the parameters can take are the domain. Instances of concepts can be defined through the use of the ”instance_of” relation.

Relations are used to define the interdependencies between several instances of concepts as a set of n-tuples (i.e links in UML) with respect to a domain. Relationships that are used in more than one domain (e.g symmetry, transitivity, reflexivity) can be defined implicitly through the use of axioms. A relation uses parameters to specify the interrelated concepts. The semantic properties of relations can be constrained to different levels.

In WSMO the Goals specify the objectives a requestor has when consulting a Web Service. Goals are defined through the following elements: non-functional properties, used mediators, pre-conditions and effects.

The Mediators define a number of elements: non-functional properties (i.e. core properties), a source and a target component, a mediation service pointing to a goal or a Web Service with some mapping and a reduction, which consists of a logical formula to describe the difference between the functionality described in the goal and the Web Service functionality. WSMO divides mediators into two classes: refiners and bridges. Refiners can be used to generate new components as a refinement of existing ones. They include goal mediators (i.e ggMediators) and ontology mediators (i.e ooMediators). Goal mediators can link a source goal to a target goal. They can also use and combine existing goals. Ontology mediators can be used to import ontologies for merging or transforming these into different ontologies. They can also resolve representation mismatches between ontologies. Bridges, the second class of mediators, enable two components with interoperability problems (i.e functionality mismatch, data mismatch, protocol and/or process mismatch) to work together. Bridges include wgMediators and wwMediators. The wgMediators link Web Services to goals and use ontology mediators to map different ontologies to each other while the wwMediators can be used to link two Web Services to enable them to interoperate with each other.

Web Services in WSMO are described using the following elements: non-functional properties, used mediators, service capability, service interface, choreography and orchestration.

Non functional properties are Web Service specific properties. The Service capability (i.e the defined goals) consists of non-functional properties (i.e core properties), used mediators for importing ontologies with ooMediators and for linking a capability to a goal by using wgMediators, pre-conditions which define the conditions over the input, post-conditions which define the relation between input and output, assumptions which define the conditions before the input and also the effects which describe the status of the service after the execution.

The Service Interface component describes how the service operates. It consists of the following elements: non-functional parameters (i.e core properties) and used-mediators (i.e ooMediators) importing ontologies to make concepts and relations accessible to the Interface.

The Choreography component (i.e the service requester point of view) tells a service requester how to communicate with a Web Service in order to consume its functionality. It is the service that advertises its capabilities to potential service requestors. It decomposes a Web Service capability into sub-capabilities. The inputs and outputs define a message exchange pattern (MEP) to describe the specific capability of a Web Service. The MEP specifies a sequence of conditional speech acts that send or receive messages. State-less MEPs model stimulus-response patterns and state-based MEPs model conversations. The collection of these messages defines the state description of a MEP.

The Orchestration component (i.e the service provider point of view) determines how a service makes use of other service providers to achieve its capability. Like the Choreography component, it decomposes a Web Service capability into sub-capabilities. These sub-capabilities are used to define the activities in a problem solving pattern (PSP) for the Web Service capability, which consists of state-less or state-based descriptions. The PSP specifies a sequence of conditional activities. Each time a Web Service needs to be invoked, a proxy needs to be declared by declaring a goal or by linking it to a wwMediator. The collection of these messages defines the state description of a PSP.

In WSMO many attributes of goals, mediators, ontologies and Web Services consist of axioms and rules defined by formal logical expressions. The WSMO use cases make use of F-Logic (Kifer et al., 1995) to represent these logical expressions. This choice offers the advantages of a conceptual high-level approach typical of a frame-based language and the expressiveness, the compact syntax, and the well defined semantics from logics.

The next stated goal of WSMO is to build a layer to support fully flexible eCommerce and eWork.

Chapter 4

Analysis of Semantic Needs

4.1 Requirements of Semantic Web Services

Web Services have added a new level of functionality to the current Web, making the first step to achieve seamless integration of distributed components. Nevertheless, current Web Service technologies only describe the syntactical aspects of a Web Service and, therefore, only provide a set of rigid services that cannot adapt to a changing environment without human intervention. The human programmer has to be kept in the loop and scalability as well as economy of Web Services are limited (Fensel and Bussler, 2002a).

The vision of Semantic Web Services is to describe the various aspects of a Web Service using explicit, machine-understandable semantics, enabling the automatic location, combination and use of Web Services. The work in the area of Semantic Web is being applied to Web Services in order to keep the intervention of the human user to the minimum. Semantic markup can be exploited to automate the tasks of discovering services, executing them, composing them and enable seamless interoperation between them (McIlraith et al., 2001), thus providing what are also called intelligent Web Services.

The description of Web Services in a machine-understandable fashion, enabling the automatic location and configuration of distributed functional components based on the requester needs, is supposed to have a great impact in areas of e-Commerce and Enterprise Application Integration. Semantic Web Services can constitute a solution to the integration problem, as they aim at enabling dynamic, scalable and reusable cooperation between different systems and organizations. These great potential benefits have led to the establishment of an important research area, both in industry and academia, to realize Semantic Web Services.

Due to the big efforts devoted to research on the description and effective use of Semantic Web Services, and due to the potential impact these new technologies can have in industry, it is essential to analyze the state and achievements of current initiatives in order to guide future research in the area.

In this section, we provide an analysis of the most prominent initiatives in the area, namely: WSMO, OWL-S, IRS-II and METEOR-S. The aim of this analysis is to contrast current initiatives against a set of requirements for Semantic Web Services in order to provide a reference evaluation for these four initiatives.

In order to perform the analysis, we define in the following a set of requirements the semantic description of a Web Service must fulfill. In (McGuinness et al., 2003), a first draft of the requirements for a Semantic Web Services description language is provided. However, some of these requirements are not clear enough and others go into too much detail for the purpose of this document. As a complete and detailed set of requirements for the description of Semantic Web Services will be provided within the Knowledge Web deliverable 2.4.1, we here start by providing a set of general requirements that can serve to provide a first analysis of current initiatives and can considerably help to identify open issues in current initiative. The deliverable 2.4.1 will go deeper into these requirements and will revisit (McGuinness et al., 2003) as a point of reference for the deliverable. Here we group some of these requirements and skip some others that are too specific at this point.

We group the requirements these initiatives must fulfill into the following categories:

- **Discovery**: Web Services have to be located after they are made available by service providers. Semantic Web Services should enable the automatic location of Web Services that provide a particular functionality and that adhere to requested properties (McIlraith et al., 2001). The discovery process should be based on the semantic match between a declarative description of the service being sought, and a description of the service being offered. This problem requires not only an algorithm to match these descriptions, but also a language to declaratively express the capabilities of services (Paolucci et al., 2002). Therefore, the following requirements should be fulfilled in order to enable the effective and dynamic location of Web Services:

  – 1) A standard language to express the capability i.e. functionality of a given service and the goal i.e. request of a given requester.

  – 2) The description of the service functionality must be independent of the underlying implementation, service binding, message exchange pattern, etc.

  – 3) An efficient and well-defined mechanism to locate the Web Services matching the requester needs.

  Notice that most of the requirements fall in the functional requirements defined in (McGuinness et al., 2003), although we skip some more detailed (and arguable) requirements found there.

- **Composition**: Web Services can be composed in order to provide a new functionality based on the functionality of the constituent services. An initiative that aims at realizing Semantic Web Services must provide a way to define how a Web Service is composed:

  - 4) A language for defining the orchestration of the service i.e. the composition of such service has to be provided, and this language needs to have defined formal semantics.
  - 5) The orchestration must allow the use of both statically defined i.e. hard-wired Web Services and dynamically located Web Services.

  Notice that these requirements fall into the process modelling requirements in (McGuinness et al., 2003). The details of the type of constraints (e.g. ordering constraints, state constraints, resource constraints) that are required is out of the scope of this document.

- **Interoperation**: One of the main purposes of Web Services is the automation of application integration within and across organizational boundaries. This implies necessarily the need for interoperation between services. This interoperation can be between services in an organization or crossing different organizational boundaries. To ensure automatic interoperation, description means must be defined declaratively using explicit semantics. Therefore:

  - 6) A formal language for defining the choreography i.e. the conversational behaviour of the service must be provided.
  - 7) The composition of the service i.e. the internals of how the service achieve its offered functionality must be clearly separated from how the service interoperates with the outside world. That means that the orchestration of the service must be clearly separated from its choreography.

  These requirements include requirements on process modelling and process enactment from (McGuinness et al., 2003), although in a lower level of detail.

- **Heterogeneity**: The use of distributed and independent Web Services implies to deal with heterogeneous terminologies, data formats, and interaction models. Therefore, mediation is required to bypass the inherent heterogeneity of Web Services in an automatic way:

  - 8) Data mediation is needed to bypass data heterogeneity.
  - 9) Process mediation is needed to bypass different interaction. patterns

  These requirements are not explicitly considered in (McGuinness et al., 2003). However, we consider them as essential due to the inherent heterogeneity of distributed sources.

- **Invocation**: The semantic description of the Web Service must include a link to a grounding i.e. an invocable description of the service:

  - 10) Grounding information must be given for every service.

  This requirement is included in (McGuinness et al., 2003) under the enactment requirements category.

- **Support**: A viable proposal for Semantic Web Services must include a supporting implementation and the modelling of some realistic use cases:

  - 11) Existence of supporting implementation.
  - 12) Realistic use cases modelled.

  Requirement 11) can be related to the metaproperties that a Semantic Web Service language must fulfill, defined in (McGuinness et al., 2003). However, 12) is not mentioned in the list of requirements given, although we consider it of great importance to evaluate the applicability of a given proposal.

- **Other requirements**: Besides the requirements listed before, other requirements have to be met by any Semantic Web Services initiative in order to provide an effective and ready-to-use solution:

  - 13) Error definition and handling.
  - 14) A human-readable description of the service must be available in order to allow not only machines but also humans to browse and search available services.
  - 15) Other non-functional properties such as contact information must be included in the description of the Web Service.
  - 16) Transaction support, including compensation mechanisms in case of failure. Transaction contexts have to be defined both in the orchestration and the choreography of the service.
  - 17) Definition of the reliability of the service.
  - 18) The Web Service must include trust information and trust policies.
  - 19) The Quality of Service (QoS) must be defined.
  - 20) The Web Service must include security mechanisms and define security policies for the provision of the service.
  - 21) Monitoring of the execution of the service.
  - 22) Extensibility of the approach to adapt potential future requirements and use cases.

Most of these various requirements are also found in (McGuinness et al., 2003), although some of them in greater detail. As explained before, we will limit ourselves in this document to general requirements.

The details of the analysis of the different existing initiatives, based on the requirements above, can be found in the following subsections.

4.2 Analysis of Current Initiatives

4.2.1 WSMO

The Web Service Modeling Ontology - (WSMO Standard, currently at version 0.2) (Roman et al., 2004b) is an initiative to create an ontology for describing various aspects related to Semantic Web Services, with a defined focus: solving the integration problem. WSMO takes into account specific application domains (e-Commerce and e-Work) in order to ensure the applicability of the ontology for these areas. WSMO is led by the Semantic Web Services working group of the SDK cluster [1], which includes more than 50 academic and industrial partners.

WSMO follows a layered approach. There are three WSMO species envisioned: WSMO-Lite (Roman et al., 2004c), WSMO-Standard (Roman et al., 2004b)and WSMO-Full (Priest and Roman, 2004). WSMO-Lite represents a minimal yet meaningful subset (in the context of Web service integration) of WSMO-Standard for which an execution environment is easily implementable, whereas WSMO-Full aims to extend WSMO-Standard and to incorporate a B2B perspective. For the analysis, we will consider WSMO-Standard and refer to the WSMO-Full extensions where necessary.

Discovery requirements

1) WSMO distinguishes the provider point of view and the requester point of view, providing different elements to express the service capability and the requester needs. A goal in WSMO specifies the objectives that a client may have when she/he consults a Web Service. A Web Service capability defines the service by means of what the service offers to the requester i.e. what functionality the service provides.

The elements used to capture the functionality requested in the goal are post-conditions and effects. Post-conditions are defined as the state of the information space that is desired. Effects are defined as the state of the world that is desired. Goals in WSMO contain neither pre-conditions nor assumptions, capturing only the results (both in terms of information and state change) that the requester requires.

The capability defines both pre-conditions (what the service expects for enabling it to provide its service, defining conditions over the input) and post-conditions (what the service returns in response to its input, defining the relation between the input and output).

[1]http://sdk.semanticweb.org/

It also defines assumptions (similar to pre-conditions, but referencing aspects of the state of the world beyond the actual input) and effects (the state of the world after the execution of the service). In this way, the capability of the service models the state change caused by the execution of the service both in the information space and in the world.

Pre-conditions, post-conditions, assumptions and effects, both in the goal and in the capability, are described as an axiom, which is an arbitrary logical expression. To describe these logical expressions, F-Logic (Kifer et al., 1995) is used.

We only find one missing element in the description of the functionality requested:

- Goals in WSMO completely capture the functionality desired by the requester, but there is not a defined way to express what information the user is willing to disclose in order to achieve such functionality, which is also a relevant aspect of the user request. A first approach to describe this information can be found in (Olmedilla and Lara, 2004).

2) In WSMO, the functional description of the service is independent of the implementation details. The grounding of the service is specified as part of the choreography and orchestration descriptions.

3) In WSMO, the proof obligations necessary to perform discovery of Web Services are being described (Keller et al., 2004), and some services are being modelled and matched (Stollberg et al., 2004). However, this work is not finished yet, so future versions of this work and potential additional work will further clarify if an efficient and well-defined discovery mechanism can be established for WSMO Semantic Web Services.

Composition requirements

4) A language for defining the orchestration of the service will be defined (it can be found in the work plan for WSMO). However, there is no available information about this language yet.

5) The WSMO orchestration specifies a set of proxies that the service uses in order to fulfill its functionality. Proxies can be goals or wwMediators. This way, both dynamic (on the fly) composition (by declaring proxies consisting of goals descriptions) and static composition (by linking proxies to wwMediators) are supported (Roman et al., 2004b).

Interoperation requirements

6) A formal language to describe the choreography of a Web Service is being defined in WSMO (Roman et al., 2004d). However, this definition is at a preliminary stage and it has to be further specified and tested.

7) A service in WSMO has an interface, which describes how the functionality of the service can be achieved (i.e. how the capability of a service can be fulfilled) by providing a twofold view on the operational competence of the service: Choreography decomposes

a capability in terms of interaction with the service (service user's view), and orchestration decomposes a capability in terms of functionality required from other services (other service providers' view). It can be seen that in WSMO composition and interoperation are clearly separated, as the choreography defines how to communicate with the Web Service in order to consume its functionality, and the orchestration defines how the overall functionality is achieved by the cooperation of more elementary service providers (Roman et al., 2004b).

Heterogeneity requirements

8) WSMO includes mediation components that are used to bypass heterogeneity problems. Different types of mediation are distinguished, as described in chapter 3. These mediators point to a goal that declaratively describes the mediation service needed or to a Web Service that actually implements the mapping. This mediation service is in charge of performing the necessary data mediation. However, an issue is open:

- It is not defined how this data mediation will work, as no examples of data mediation are available yet.

9) The mediators describe before will also apply to process mediation. But:

- No examples of process mediation are available yet.

Invocation requirements

10) The grounding information of WSMO services will be included in WSMO in the choreography and orchestration definitions (Roman et al., 2004b), although the grounding is not defined in the current version of WSMO.

Support requirements

11) Three implementations are being developed that will support WSMO. The first one is WMSX, that will create an execution environment for the dynamic discovery, selection, mediation, invocation and inter-operation of WSMO Semantic Web Services. This platform is going to be a sample implementation of the Web Services Modelling Ontology (WSMO). The second one is an adaptation of the IRS-II platform to use WSMO descriptions of Web Services. The third one will be the Semantic Web FRED[2].

12) The WSMO working group is devoting a considerable part of his work to the modelling of real use cases. At the moment, the modelling of a B2B and a B2C use cases is being done (Stollberg et al., 2004). WSMO will take use cases from the projects participating in the SDK cluster and model them to test the applicability of the approach (see (Bussler, 2004), (Lara, 2004) and (de Bruijn, 2004) for details). However, complete modelled examples are not available yet.

[2]http://www.deri.at/research/projects/swf/

Other requirements

13) The definition of errors is not explicitly consider in WSMO, although it will be included in the choreography of the service. However, this definition is not given yet. Error handling is also not explicitly considered at the moment.

14) WSMO introduces a set of core non-functional properties that are defined globally and that can be used by all the modelling elements of WSMO. These properties include the Dublin Core Metadata Element Set plus a version element. The Dublin Core elements include human-readable description of all the elements of WSMO, including the service.

15) Extensions for core non-functional properties specific for the service such as security, trust, etc. are also provided. In addition, these core properties can be extended if required in order to capture specific non-functional properties of any WSMO element.

16) Transaction and compensation is not considered in WSMO at the moment, although it will be included in the orchestration and choreography definitions.

17) In the extension of the WSMO core non-functional properties to include Web Service specific non-functional properties, the reliability of the service is defined.

18) Trust information is also included in the Web Service specific non-functional properties. However, trust policies are not defined. A first approach to include trust policies in WSMO can be found in (Olmedilla and Lara, 2004).

19) QoS is included as part of the Web Service specific non-functional properties.

20) Security is also part of the Web Service non-functional properties, although security policies are not treated in detail.

21) Monitoring of the execution of the service is not present in WSMO.

22) WSMO is extensible in every direction due to the loose-coupling of all its elements. All the modelling elements are defined independently and the core elements of WSMO are linked via mediators, which guarantees the extensibility of the ontology.

Summary

Our conclusion from the analysis of WSMO is that this initiative is appropriately covering the discovery requirements with the exception of the information the requester is willing to disclose to achieve the desired functionality. A first approach to add this information is related in (Olmedilla and Lara, 2004). Composition requirements are well addressed at the conceptual level, but a concrete language to describe the orchestration has not been provided yet. The same applies to interoperation, where the separation between the choreography and the orchestration is well-defined, but a language to define the choreography of the service is still to come. Regarding mediation, we find a similar situation; the conceptual elements to bypass heterogeneity problems are in place, but a more concrete view (mainly via examples) on how the mediation will actually take place would help to better evaluate the conceptual approach. The tool support for WSMO Web Services is being developed and it is heading on the right direction. However, the implementations are not available at this moment, so they cannot be evaluated in detail. The modelling of real use cases is started and there is an explicit intention to model a meaningful set of use cases, but this task is still in progress. Some of the other requirements we have used for the analysis are not covered at the moment. Some of these requirements will be fulfilled when the choreography and orchestration in WSMO is defined, and some others have to be addressed separately.

Summarizing, WSMO is a conceptually strong approach that provides a good base for the realization of Semantic Web Services. However, and due to the still short life of the initiative, some aspects are still undefined and some of the results of the WSMO working group [3] are still in progress. WSMO appears as a promising initiative, that has to still fill some gaps and show in the near future its real applicability to real use cases. Although our first analysis is positive, a really accurate evaluation of WSMO is not completely possible at this point in time, as it is still at its first stage.

4.2.2 OWL-S

As outlined in section 3.3.1, OWL-S (OWL Services Coalition, 2003) is a collaborative effort by BBN Technologies, Carnegie Mellon University, Nokia, Stanford University, SRI International and Yale University to define an ontology for semantic markup of Web Services. OWL-S, currently at version 1.0, is intended to enable automation of Web Service discovery, invocation, composition, interoperation and execution monitoring by providing appropriate semantic descriptions of services.

The purpose of OWL-S is to define a set of basic classes and properties for declaring and describing services i.e. an ontology for describing Web Services that enable users and software agents to automatically discover, invoke, compose and monitor Web resources offering services, under specified constraints.

In order to determine to what extent OWL-S initiative is applicable in a real setting, we will go through the general requirements outlined in the previous section and will describe to what extent they are met.

Discovery requirements

1) In OWL-S, the Service Profile describes the intended purpose of the service, both describing the service offered by the provider and the services needed by the requester. The profile specifies what functionality the service provides, the specification of the conditions that must be satisfied for a successful result, and the results of the service execution. This is described by using four different elements: inputs, outputs, preconditions and effects (IOPEs).

In OWL-S, a Web Service is viewed both as a data transformation process i.e. a transformation from a set of inputs to a set of outputs, and as a state transition in the world i.e.

[3]http://www.wsmo.org/

some effects in the world emerge after the execution of the service. Information transformation is expressed in OWL-S using inputs and outputs, and state change is expressed using preconditions and effects. OWL-S also include the possibility of defining conditional outputs and effects i.e. outputs and effects that will actually be delivered by the service only if the defined conditions are met.

The functionality of the Web Service is also described in the Service Model using again IOPEs. The consistency between the service profile and the service model is not imposed in OWL-S. The descriptions contained in the profile and in the model can be inconsistent without affecting the validity of the OWL expression (although it may result in a failure to provide the service functionality). However, the OWL-S specification envisions that the set of inputs, outputs, preconditions and effects (IOPEs) of the service profile are a subset of the ones defined by the service model.

It can be seen that OWL-S defines, mainly via the profile and service model concepts, a standard way to express the capability of a service and the goal of a requester. However, the following problems are encountered:

- OWL-S conditions are limited to OWL class expressions at its current state. Extensions towards a fully fledged rule language like SWRL (Horrocks et al., 2004) or DRS (McDermott, 2004) are under discussion. However, at the moment there is not a language available to describe pre-conditions, effects, and conditions for conditional outputs and effects.
- OWL-S does not distinguish between the provider point of view and the requester point of view, as it uses a single concept (the profile) to model both the service capability and the requester needs. This unification is conceptually not clear and, furthermore, it can introduce problems if different aspects have to be expressed for the goal and for the service capability.
- The relation between IOPEs is not captured in OWL-S i.e. the relationship between the inputs, outputs, preconditions and effects of the service are not described. As OWL-S does not provide means for arbitrary logical expressions or rules, the description of the functionality of the service and of the requester needs remains limited.

2) In OWL-S, the description of the service functionality is decoupled from the underlying implementation, service binding, message exchange pattern, etc. The grounding of the service, which links the semantic description of the service to an invocable interface, is defined separately. Therefore, this requirement is fulfilled by OWL-S.

3) Some work has been done to define discovery algorithms for OWL-S Semantic Web Services, such as the work described in (Paolucci et al., 2002) and (Li and Horrocks, 2003), among others. These approaches mainly rely on subsumption reasoning over IOPEs to determine if a given OWL-S service meets the requester requirements. However, several limitations are found:

- Efficient subsumption reasoning in Description Logics is at the moment limited to T-Box reasoning i.e. reasoning about concepts. As A-Box reasoning i.e. reasoning about instances is not feasible in an efficient way, current approaches model every service profile as a new class. Although this solution leads to a computationally effective solution, it is conceptually arguable. Different service profiles should be modelled as instances of the service profile concept define in the OWL-S ontology.
- The matchmaking algorithm is based on inputs and outputs, while preconditions and effects are usually not considered. Therefore, the applicability of the proposed mechanisms to dynamically discover OWL-S services is not yet proved for general descriptions of Web Services.
- Discovery in OWL-S inherits the limited description of functionality commented in requirement 2). Assuming that OWL-S descriptions should be improved to capture more accurately the functionality of the service, the validity of current discovery approaches in a real setting is not clear.

Composition requirements

4) OWL-S allows to define in the process model composite processes, including the definition of what other processes constitute the composite one. It also includes control constructs to specify the control flow of the constituent processes and mechanisms to define the data flow between these processes. Nevertheless, we have not found a definition of an explicit formal semantics for the modelling of the orchestration.

5) In OWL-S, the processes or services fulfilling the subgoals of a given service are hard-wired in the orchestration of the service. It is not possible to include a service profile (a user request) and automatically locating an appropriate service when defining how a given Web Service will be composed. Therefore, the dynamism of OWL-S compositions is limited.

Interoperation requirements

6) In OWL-S, only the service model can be seen as defining the conversational behaviour of the service. However, it is not clear whether that is the purpose of the service model or if the real purpose of such model is to define the orchestration i.e. composition of the Web Service.

7) As stated before, there is no separation between the orchestration and the choreography of a given service, that is, the internal details of how a Web Service achieves its functionality and the external behaviour of such service are messed up. Furthermore, it is not clear what is the role of the OWL-S service model. This issue should be clarify and a clear separation between the orchestration and the choreography of the service should be defined in OWL-S.

Heterogeneity requirements

8) Data mediation is not considered in OWL-S, leaving this issue unresolved.

9) Process mediation is not considered either.

Invocation requirements

10) OWL-S links a Web Service to its grounding by using the property supports. A Web Service can have multiple groundings (although an atomic process can have only one grounding) and a grounding must be associated with exactly one service. OWL-S does not dictate the grounding mechanism to be used. Nevertheless, the current version of OWL-S provides a pre-defined grounding for WSDL, mapping the different elements of the Web Service to a WSDL interface. Although a grounding mechanism is provided in OWL-S, the following problems arise:

- Groundings are associated to the atomic processes defined in the service model, but this association is not described in the model but only in the grounding. Therefore, the groundings for the atomic processes of the model can only be located by navigating from the service model to the service (via the describes property), and from there to the service grounding (via the supports property).
- OWL-S imposes that an atomic process must have exactly one grounding, which limits the number real implementations an OWL-S Web Service can have. This leads to some problems when defining real services, as described in (Sabou et al., 2003).

Support requirements:

11) Some work has been done to exploit the semantic information of OWL-S Semantic Web Services. This work has concentrated and the dynamic discovery of services. After a service has been selected, the grounding is used to invoke the service as a conventional Web Service. However, little work has been done to exploit additional OWL-S descriptions besides the functionality description found in the profile or the service model. One of the few examples of a (limited) exploitation of the OWL-S process model can be found in (Paolucci et al., 2003)). There is not an implementation that fully exploits the OWL-S semantic description of services. Taking into account that the initiative has been running for around 3 years, this can be interpreted as a symptom of problems and caveats in the OWL-S approach to semantically describe Web Services.

12) Some realistic use cases are modelled using OWL-S, such as the interaction with Amazon.com (Paolucci et al., 2003). However, there is not a set of available use cases in different application domains that help to prove the applicability of OWL-S in a real setting and, as mentioned in the previous point, the available use cases only make a limited use of OWL-S descriptions, only exploiting some of the aspects described by the OWL-S ontology.

Other requirements:

13) OWL-S does not model error information and does not consider any error handling mechanism. Although errors can be captured by using OWL-S conditional outputs, this characterization of errors is not explicit, as the definition of a conditional output does not necessarily imply that one of the possible outputs is an error.

14) The OWL-S service profile includes human-readable information, contained in the properties serviceName and textDescription.

15) Non-functional properties are explicitly modelled in OWL-S. The contact information of the service is included as a property of the service profile. In addition, an expandable list of non-functional properties and a service category (using a categorization external to OWL-S) is given.

16) No transaction support is present in OWL-S.

17) The reliability of the service can be modelled using the expandable list of non-functional properties defined in the profile. However, this has not been done at the moment.

18) Trust information could be again modelled using non-functional properties. However, the modelling of trust policies may require an explicit and more elaborated modelling beyond non-functional properties. Furthermore, information disclosure policies might be necessary in the definition of preconditions.

19) The Quality of Service can also be modelled using non-functional properties.

20) Security is not explicitly considered in OWL-S, although there is some work done in the area by Denker et al. (Denker et al., 2003), which proposes some ontologies for security annotation of OWL-S services and mechanisms for matchmaking of the requester and provider security capabilities and requirements.

21) Monitoring was mentioned in earlier versions of OWL-S. However, at its current version the monitoring of the execution of Web Services is not considered.

22) OWL-S is extensible through the use of explicit expandable properties (such as non-functional properties in the profile) and through OWL subclassing. However, the tight coupling of some elements in OWL-S (such as the use of the Service class and the expected reference from the service profile to the IOPEs of the service model) can slightly limit the extensibility of the OWL-S ontology.

Summary

OWL-S aims at providing an ontology for semantically annotating Web Services in order to enable Web Service discovery, invocation, composition, interoperation and execution monitoring. Nevertheless, it can be seen from the analysis above that the tasks that should be enabled by OWL-S are not successfully performed. Discovery is one of the most mature areas, although current approaches are still limited and the OWL-S ontology itself is under-defined in several aspects. Invocation is achieved though the definition of the OWL-S grounding and there are some examples of its use e.g. (Richards and Sabou, 2003), (Sabou et al., 2003) or (Paolucci et al., 2003). However, some problems, specially regarding the cardinality of the grounding, are still encountered. Regarding composition and interoperation, they are not clearly separated and so far only a limited use of the service model has been accomplished. Monitoring is not even considered in the last version of OWL-S. Tool support is very limited despite the relatively long life of the initiative, and the modelling of real use cases has not been shown to fully exploit or being able to make use of the complete OWL-S semantic descriptions. Regarding other type of requirements, while some of them are reasonably covered, issues such as transactions or trust have not been solved. In a nutshell, OWL-S has shown its capability to perform certain tasks exploiting semantic annotations, while it is lacking essential features to make Semantic Web Services viable in a real setting. Therefore, some issues has to be solved or corrected in order to make OWL-S realize Semantic Web Services.

4.2.3 IRS-II

IRS-II (Internet Reasoning Service) (Motta et al., 2003) is a framework to support the publication, location, composition of execution of heterogeneous Web Services. The framework relies on the semantic description of the service functionality. The IRS-II framework can be seen as an adaptation of the UPML (Fensel et al., 2003)framework to the Web Services domain.

Discovery requirements

1) IRS-II distinguishes between tasks (what to do) and methods (how to achieve tasks) (Motta et al., 2003). Both tasks and methods are described by means of inputs, outputs, preconditions and postconditions, that provide the functional description of the service in terms of information and state of the world change. Tasks, Problem Solving Methods (PSMs) and domain models (Fensel et al., 2003) are represented in OCML, an Ontolingua-derived language (Motta et al., 2003). Therefore, IRS-II provides a language to express the service capabilities and the user goals.

2) The semantic descriptions of available Web Services is independent of the underlying implementation, service binding, etc.

3) IRS-II aims to support a dynamic, knowledge-based service selection (Motta et al., 2003). The user specifies the capability he requires (the task) and a PSM that performs this task has to be selected. The semantic description of the task is used as a query that returns the available PSMs that satisfy such query. However, in the available examples IRS-II tasks are tied to PSMs via the ”Tackles-Task-Type” property, which gives an explicit link between PSMs and a type of tasks. Therefore, it is not clear if the discovery algorithm used can work without such explicit links.

Composition requirements

4) IRS-II does not provide a language to describe the service orchestration.

5) As the orchestration of the service is not defined, this requirement cannot be met by IRS-II.

Interoperation requirements

6) The conversational behaviour of the service is not described in IRS-II.

7) As neither orchestration nor choreography are described in IRS-II, this requirement does not apply to the framework.

Heterogeneity requirements

8) In the UPML framework, bridges are introduced to bypass heterogeneity problems. However, such bridges are not used at the moment in IRS-II and, as a consequence, data mediation is not available.

9) Process mediation is missing in IRS-II.

Invocation requirements

10) Competence specifications i.e. the semantic description of services functionality is mapped to specific Web Services that can be invoked. The IRS Publisher links Web Services to their semantic descriptions within the IRS server. It is possible to have multiple services described by the same semantic specifications (i.e., multiple implementation of the same functionality), as well as multiple semantic specifications of the same service (Motta et al., 2003). The association between the PSM and the Web Service endpoint is stored in the IRS server.

Support requirements

11) The IRS-II framework counts with an implemented infrastructure. Such infrastructure consists of several tools that communicate using SOAP:

- The IRS-Publisher links Web Services to their semantic descriptions within the IRS server. It also supports publishing of existing programming code, automatically transforming programming code into a Web Service (Motta et al., 2003). At the moment, the IRS-Publisher supports this functionality for Java and Lisp programs.
- The IRS server stores the descriptions of Semantic Web Services. Both the method descriptions (PSMs) and the concrete implementations are registered in the IRS server.
- The IRS broker locates PSMs fulfilling a given task.
- The IRS client serves as an interface for the user to the IRS infrastructure.

12) An scenario from the health-care domain is presented in (Motta et al., 2003).

However, a wider and more representative set of modelled use cases is missing.

Other requirements

13) Error handling is very basic and its improvement is planned as part of the future work in IRS-II (Motta et al., 2003).

14) A human-readable description of the service is not provided.

15) Other non-functional properties are missing in the description of the IRS-II elements.

16) IRS-II does not provide transaction support.

17) The definition of the reliability of the service is not given.

18) Trust information is also missing.

19) The Quality of Service of the service is not defined.

20) IRS-II does not provide security support.

21) Monitoring is not considered.

22) The approach is based on UPML, a solid and extensible conceptual model. Therefore, IRS-II can be extended to meet missing requirements and future use cases. In fact, there will be an extension of IRS-II to follow the direction of WSMO.

Summary

IRS-II is very close to UPML, a solid conceptual approach to the reuse of software components. Therefore, it provides a good basis to realize Semantic Web Services. However, it is a too direct application of UPML to the Web Services domain, without considering special requirements of this domain. Although it provides a good infrastructure for the storage, location and execution of IRS-II Web Services, it presents several limitations and does not fulfill many of the requirements analyzed.

4.2.4 METEOR-S

As described in section 3.3.3, the main goal of the Meteor-S (**?**) project is to provide web services with enhanced dynamism and scalability through the application of semantics in the Annotation, Discovery, Composition, Quality of Service and Execution of Web Services, building on top of existing standards and initiatives such as WSDL, UDDI and BPEL4WS.

In the following, we will analyze the results of Meteor-S against the requirements we have introduced for Semantic Web Services.

Discovery requirements

1) In METEOR-S, Web Services are annotated semantically in order to enhance the discovery process. The METEOR-S Web Services Discovery Infrastructure (MWSDI) adds semantics (using ontologies) at two levels (Verma et al., 2004): at the level of individual Web Services and at the level of the registries that store the services.

For the annotation of individual Web Services, a bottom-up approach is followed i.e. WSDL message types, both inputs and outputs, are mapped to the appropriate concepts in domain specific ontologies.

In addition to the annotation of WSDL inputs and outputs, WSDL operations are also mapped to ontological concepts from an operations domain ontology (Sivashanmugam et al., 2003). In the annotation of WSDL operations, preconditions and effects are also added, interpreted as logical conditions that must hold for performing the operation, and as changes in the world caused by the execution of the operation, respectively.

The user goals are expressed using service templates based on the concepts from the domain ontologies. In such template, information about the operation being sought and their inputs and outputs are given, and optionally preconditions and effects can also be specified.

The aim of annotating registries is to enable the classification of Web Services based on their domain. Services are specialized in a given domain, and will store Web Services related to that domain. A specialized ontology, the registries ontology, is used to annotate registries, mapping the registries to a given domain and giving additional information about the registry, relations with other registries and relations to other domains.

It can be seen that METEOR-S provides a way to specify service capabilities and user goals. However, some limitations are found. These limitations are similar to the ones identified for OWL-S, as the approach both initiatives follow is quite similar:

- METEOR-S does not provide a expressive language to describe preconditions and effects, thus limiting the description of the service capability.
- The relation between the different elements that describe the service template for the requester and the service functionality is not captured i.e. the relationship between the inputs, outputs, preconditions and effects of the service are not described. As METEOR-S does not define a way to express arbitrary logical expressions or rules, the description of the functionality of the service and of the requester needs remains limited.

2) METEOR-S is based on annotating WSDL descriptions of services. Therefore, the description of the service functionality is independent of the implementation, service binding, etc. but it does impose the use of WSDL, thus limiting the applicability of such description to WSDL.

3) An algorithm has being developed for the discovery of METEOR-S Semantic Web Services (Sivashanmugam et al., 2003). First, the operation specified in the service template is matched against the operations described in the available Web Services. Then,

inputs and outputs are matched. Finally, preconditions and effects are checked. As the approach followed for the mapping is similar to the one followed for OWL-S descriptions, it suffers from the same problems:

- Efficient reasoning is limited to T-Box reasoning.
- The discovery algorithm inherits the limited description of functionality discussed in requirement 2).

Composition requirements

4) In METEOR-S, a framework for defining composition of Web Services have been defined. This framework is based on BPEL4WS, with extensions to support the specification of the so-called semantic process templates (Sivashanmugam et al., 2004). These templates use BPEL4WS constructs together with semantic activity templates to enable dynamic location of suitable Web Services. Based on these templates, executable BPEL4WS business processes are generated. It can be seen that BPEL4WS is the underlying orchestration language in METEOR-S. As BPEL4WS lacks defined formal semantics, the METEOR-S composition language inherits this caveat.

5) In the specification of a semantic process template, three possibilities are present in METEOR-S: specifying an activity in the process using a concrete Web Service implementation, specifying an activity using a Web Service interface, or specifying an activity using a semantic activity template. The second possibility allows the dynamic discovery of an appropriate service to perform the process activity based on mere syntactical aspects (WSDL interface), which is very limited. The second possibility allows the dynamic discovery of services based on their METEOR-S semantic annotations. Therefore, this requirement is met by the METEOR-S composition language.

Interoperation requirements

6) No choreography language is provided in METEOR-S.

7) This requirement does not apply to METEOR-S because of 6).

Heterogeneity requirements

8) Data mediation is not addressed.

9) Process mediation is not addressed.

Invocation requirements

10) Grounding information is given in METEOR-S, although this grounding is limited to WSDL. In fact, as METEOR-S annotates WSDL documents, the WSDL grounding is obviously an element always present in METEOR-S Semantic Web Services.

Support requirements

11) The exists an implementation of the METEOR-S Web Services Composition Framework (MWSCF), including a process builder, XML repositories and a process execution engine. In addition, a discovery infrastructure (MWSDI) has been implemented.

12) We have not found in our analysis any complete real use case modelled, which is crucial to test the applicability of METEOR-S.

Other requirements

13) Error definition and handling is not present in METEOR-S, besides the error definition and handling included in BPEL4WS as the underlying composition language for METEOR-S.

14) No human-readable information is explicitly modelled in METEOR-S.

15) Non-functional properties are not considered.

16) As in 13), the transaction support in METEOR-S is the one provided by BPEL4WS.

17) Reliability issues are not addressed in METEOR-S.

18) Trust information and trust policies are lacking.

19) Quality of Service is very well addressed in METEOR-S. A predictive QoS model to compute the QoS for workflows automatically based on the specification of atomic tasks QoS is provided. A model to specify QoS, and algorithm to compute, analyze and monitor workflow QoS metrics, as well as a simulation system, have been developed (Cardoso, 2002).

20) Security mechanisms and security policies are not defined in METEOR-S services.

21) The monitoring of the execution of the service is provided only at the QoS level (Cardoso, 2002).

22) The extensibility of METEOR-S is not clear. Due to the bottom-up approach followed, starting from the annotation of WSDL service descriptions, a well-defined ontology to define all the aspects of a Web Service is not provided. We believe that its too tight coupling to WSDL descriptions and to BPEL4WS makes its extension to consider other languages quite difficult.

Summary

METEOR-S provides a bottom-up approach to the semantic description of Web Services. By building on top of WSDL, UDDI and BPEL4WS, METEOR-S is more likely to provide short-term results but its potential to provide a complete description of Semantic Web Services is rather limited. It is of special value the work done in the modelling of quality of service aspects, both at the level of individual Web Services and at the level of workflows, and the development of prototypes to support the discovery and composition of Web Services. However, many of our requirements are not fulfilled by METEOR-S, and its bottom-up approach can be a benefit in the short term but might prevent it from providing a complete formalism to describe Semantic Web Services.

4.3 Open Issues

The analysis conducted in this section of the current initiatives to semantically describe Web Services shows that none of them have realized the Semantic Web Services vision yet. However, we can see that the different approaches present strengths and weaknesses, and that some of the requirements are met by certain approaches while other requirements are not fulfilled or not addressed at the moment.

Discovery requirements are not completely fulfilled by any of the initiatives. WSMO provides a good description of the service capabilities and requester goals, but a well-defined discovery mechanism that makes use of these descriptions is still in progress. OWL-S presents serious limitations to capture the service capabilities and does not offer a differentiation between service capabilities and user goals and, as a result, the discovery mechanisms proposed that make use of OWL-S descriptions are not sufficient to be applicable in a real setting. IRS-II discovery mechanism has to be further elaborated, and METEOR-S presents the same problems as OWL-S, as their service descriptions are quite similar. Therefore, WSMO and IRS-II seem to be the most promising proposals regarding discovery, but they have to further work in providing a well-defined and efficient discovery mechanism.

Regarding composition, WSMO has not provided a language to describe orchestration yet, although dynamic discovery of the services to be used is identified as a requirement that has be met by such language. OWL-S lacks a formal semantics for its composition model and does not allow dynamic use of services in the composition. IRS-II does not provide a composition language. METEOR-S bases its composition in BPEL4WS and allows dynamic composition. However, BPEL4WS has not defined formal semantics. Therefore, it can be seen that composition is an important open issue not covered by any of the proposals analyzed.

Interoperation requirements are also not covered at the moment. WSMO has not defined its choreography language yet, although it is stated that it will provide a clear separation from the orchestration of the service. In OWL-S, it is not clear if the choreography of the service is provided by the service model and, in fact, the purpose of the service model is not completely clear. Neither IRS-II nor METEOR-S define the choreography of the service.

Mediation to bypass heterogeneity problems, although it is an essential issue to be addressed in the Web Services domain, is ignored by most of the initiatives. Only WSMO addresses it, although how the data and process mediation will be realized is not fully defined yet.

All the initiatives provide a grounding mechanism, although it is under-defined in WSMO, it presents some problems in OWL-S, and it is limited to WSDL in the case of METEOR-S.

Support requirements are also not completely covered. IRS-II and METEOR-S provide a good set of tools that support some aspects of Semantic Web Services, although they do not offer the full support needed to use Semantic Web Services in a real setting. Real use cases modelled are missing in all the initiatives. Only WSMO explicitly considers this modelling as an essential requirement to test the applicability of the solution.

Many of the other requirements considered in our analysis are not covered. Transaction support, trust, security and execution monitoring are not properly addressed. QoS is only addressed in detail by METEOR-S. Regarding extensibility, WSMO and IRS-II are the most easily extendible proposals. They are similar in this regard, as they both have their roots in the conceptual model provided by UPML.

In conclusion, important requirements are to be met if Semantic Web Services want to show its viability as a solution for the integration problem. We perceive WSMO as the, in some aspects and due to its short life, less mature initiative, but also the most promising. It has a good focus on the problems to solve and a good conceptual model as starting point. Its cooperation with IRS-II, which will provide a WSMO-compliant implementation of its infrastructure, also contributes to this perception. METEOR-S can have faster practical applications due to its bottom-up approach based on existing industry standards or proposals, but it is less likely to provide a solution for the integration problems that considers all the aspects that have to be solved, as its extensibility is fairly limited. OWL-S will need to address some of its unsolved problems and accomplish the modelling of real and complex use cases to show that its approach is really viable.

Chapter 5

Conclusion

This document has provided an overview of the field of Web Services and Semantic Web Services, and has described and evaluated the current initiatives in terms of enabling the goals of Semantic Web Services, that is, automatic discovery, composition, invocation and interoperation of Web Services through the use of Semantic Web technologies.

The final aim of the internal deliverable has been to identify the needs of Semantic Web Services which are not being adequately considered by the current research. Our analysis has found that:

- in Web Service invocation, initiatives extend the interface description semantically and reference a service endpoint (e.g. in WSDL). This still needs models with clearer formal semantics for the implementation of semantically based invocation frameworks.

- in Web Service discovery, some initiatives have developed complete models for expressing service capabilities and user goals. However dynamic discovery mechanisms are still an open area which may lead to changed demands upon those models.

- in Web Services composition, some initiatives define workflow model-based approaches which do not have any formal semantics. Only OWL-S defines composition at a semantic level but its model does not support dynamic service selection. The heterogeneity of services is a vital issue to be resolved yet data and process mediation is only mentioned in the WSMO initiative.

- in Web Services interoperation there is not yet a clear semantic model. Choreography is ambiguous to interpret in OWL-S and WSMO does include choreography as part of its conceptual framework but development of a choreography language is currently at a very early stage. Co-ordination of services is only mentioned in WSMO (wwMediators).

- other Web Services requirements apply also to Semantic Web Services such as transactionality, security, trust and execution monitoring. Significantly no initiative has yet considered these (solely QoS issues which have been well defined in Meteor-S).

From this identification of semantic needs, the subsequent public deliverables of the

Semantic Web Services work package in KnowledgeWeb will focus on comprehensively describing these semantic needs that are required to be expressed in Semantic Web Service models if automatic discovery, composition, invocation and interoperation is to be realizable for concrete business use cases.

Furthermore, Semantic Web Services require conceptual models and execution frameworks which support the needs of scalability, heterogeneity and dynamics. These are the focus on the other work packages in the Research track of KnowledgeWeb, and there will be on-going synergy between these work packages and the Semantic Web Services group, so that the findings of the other groups may eventually contribute to Semantic Web Service models which are scalable, mediated and dynamic enough for real-world application.

Semantic Web Services are part of an ultimate vision to enable automated, self-maintaining software agents which can infer and perform tasks for human and machine clients. There is still large hurdles to jump over before this vision can be thought of as being a realistic and reachable goal. However, with the current initiatives which are moving towards this goal one step at a time and the co-ordination being promoted as part of the KnowledgeWeb Network of Excellence to ensure research results can be shared and efforts not needlessly duplicated, the research communities have an opportunity to move even more effectively forwards.

Bibliography

Aggarwal, R., Verma, K., Miller, J., and Milnor, W.: 2004, Dynamic web service composition in METEOR-S, in *Proceedings of the IEEE International Conference on Services Computing, 2004*

Alonso, G., Casati, F., Kuno, H., and Machiraju, V.: 2003, *Web Services*, Springer

Ananthamurthy, L.: 2004, *Introduction to Web Services*, http://www.developer.com/services/article.php/1485821

Ankolekar, A., Burstein, M., Hobbs, J. R., Lassila, O., Martin, D., McDermott, D., McIlraith, S. A., Narayanan, S., Paolucci, M., Payne, T., and Sycara, K.: 2002, *DAML-S: Web Service Description for the Semantic Web*, http://www-2.cs.cmu.edu/ terryp/Pubs/ISWC2002-DAMLS.pdf

Apshankar, K., Clark, M., Chang, H., Fernndez, E., Fletcher, P., Hankinson, W., Hanson, J., Irani, R., Mittal, K., et al.: 2002, *Web Services Business Strategies and Architectures*, Expert

Austin, D., Barbir, A., Peters, E., and Ross-Talbot, S.: 2004, *Web Services Choreography Requirements*, http://www.w3.org/TR/ws-chor-reqs/

Bussler, C.: 2004, *WSMO in DIP*, http://www.wsmo.org/2004/d19/d19.1/

Bussler, C., Maedche, A., and Fensel, D.: 2003, Web services: Quo vadis?, *IEEE Intelligent Systems* **18(1)**, 80–82

Cardoso, J.: 2002, *Quality of Service and Semantic Composition of Workflows*, *Ph.D. thesis*, University of Georgia

Chitnis, M., Tiwari, P., and Ananthamurthy, L.: 2004, *Introduction to Web Services Part 2: Architecture*, http://www.developer.com/services/article.php/1495021

Crubzy, M., Motta, E., Lu, W., and Musen, M. A.: 2003, *Configuring Online Problem-Solving Resources with the Internet Reasoning Service*, http://www-smi.stanford.edu/pubs/SMI_Reports/SMI-2003-0958.pdf

de Bruijn, J.: 2004, *WSMO in SEKT*, http://www.wsmo.org/2004/d19/d19.3/

Denker, G., Kagal, L., Finin, T., Paolucci, M., and Sycara, K.: 2003, Security for daml web services: Annotation and matchmaking, in *Proceedings of the Second International Semantic Web Conference (ISWC)*, Sanibel Island, Florida, USA

Fensel, D. and Bussler, C.: 2002a, The web service modeling framework WSMF, *Electronic Commerce Research and Applications* 1(2)

Fensel, D. and Bussler, C.: 2002b, *WSMF in a Nutshell*, http://informatik.uibk.ac.at/users/c70385/wese/wsmf.iswc.pdf

Fensel, D., Motta, E., Benjamins, V., Decker, S., Gaspari, M., Groenboom, R., Grosso, W., Musen, M., Plaza, E., Schreiber, G., Studer, R., and Wielinga, B.: 2003, The unified problem-solving method development language UPML, *Knowledge and Information Systems (KAIS): An international journal* 5(1)

Haas, H. and Brown, A.: 2004, *Web Services Glossary*, http://www.w3.org/TR/ws-gloss/

He, H., Haas, H., and Orchard, D.: 2004, *Web Services Architecture Usage Scenarios*, Technical report, W3C

Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., and Dean, M.: 2004, *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, http://www.daml.org/rules/proposal/

Horrocks, I., Sattler, U., and Tobies, S.: 1999, Practical reasoning for expressive description logics, in A. V. H. Ganzinger, D. McAllester (ed.), *LPAR'99*, pp 161–180, Springer Verlag

Keller, U., Lara, R., Polleres, A., and Lausen, H.: 2004, *Inferencing Support for Semantic Web Services: Proof Obligations*, http://www.wsmo.org/2004/d5/d5.1/

Kifer, M., Lausen, G., and Wu, J.: 1995, Logical foundations of object oriented and frame-based languages, *Journal of the ACM* **42(4)**, 741–843

Lara, R.: 2004, *WSMO in Knowledge Web*, http://www.wsmo.org/2004/d19/d19.2/

Lara, R., Lausen, H., Arroyo, S., de Bruijn, J., and Fensel, D.: 2003, Semantic web services: description requirements and current technologies, in *International Workshop on Electronic Commerce, Agents, and Semantic Web Services, In conjunction with the Fifth International Conference on Electronic Commerce (ICEC 2003)*, Pittsburgh, PA

Leymann, F.: 2003, Web services: Distributed applications without limits - an outline, in *Proceedings Database Systems For Business, Technology and Web BTW 2003*

Li, L. and Horrocks, I.: 2003, A software framework for matchmaking based on semantic web technology, in *12th international conference on World Wide Web (WWW03)*

McDermott, D.: 2004, *DRS: A Set of Conventions for Representing Logical Languages in RDF*, http://www.daml.org/services/owl-s/1.0/DRSguide.pdf

McGuinness, D., Parsia, B., Payne, T., Tate, A., Martin, D., Kifer, M., Gruninger, M., and Grosof, B.: 2003, *Semantic Web Services Language Requirements*, http://www.daml.org/services/swsl/requirements/swsl-requirements.shtml

McIlraith, S., Son, T., and Zeng, H.: 2001, Semantic web services, *IEEE Intelligent Systems* 16(2)

Motta, E., Domingue, J., Cabral, L., and Gaspari, M.: 2003, *IRS-II: A Framework and Infrastructure for Semantic Web Services*, http://www.cs.unibo.it/ gaspari/www/iswc03.pdf

Motta, E. and Lu, W.: 2000, *A Library of Components for Classification Problem Solving*, http://kmi.open.ac.uk/projects/ibrow/Documents/Class_Libr_Dv1.0.pdf

Olmedilla, O. and Lara, R.: 2004, Trust negogiation for semantic web services, in *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*

Omelayenko, B., Crubzy, M., Fensel, D., Ding, Y., Motta, E., and Musen, M.: 2000, *Meta Data and UPML, UPML Version 2.0*, http://www.cs.vu.nl/ upml/upml2.0.pdf

OWL Services Coalition: 2003, *OWL-S: Semantic Markup for Web Services*, http://www.daml.org/services/owl-s/1.0/owl-s.pdf

Paolucci, M., Ankolekar, A., Srinivasan, N., and Sycara, K.: 2003, The DAML-S virtual machine, in *Proceedings of the Second International Semantic Web Conference (ISWC)*, pp 209–305, Sanibel Island, Florida, USA

Paolucci, M., Kawamura, T., Payne, T., and Sycara, K.: 2002, Semantic matching of web services capabilities, in I. Horrocks and J. Handler (eds.), *1st Int. Semantic Web Conference (ISWC)*, pp 333–347, Springer Verlag

Polleres, A., Lara, R., and Roman, D.: 2004, *D4.2v01 Formal Comparison WSMO/OWL-S*, http://www.wsmo.org/2004/d4/d4.2/v0.1/

Priest, C. and Roman, D.: 2004, *Web Service Modeling Ontology - Full (WSMO - Full)*, http://www.wsmo.org/2004/d12/

Rajasekaran, P., Miller, J., Verma, K., and Sheth, A.: 2004, *Enhancing Web Services Description and Discovery to Facilitate Composition*, Technical report

Richards, D. and Sabou, M.: 2003, Semantic markup for semantic web tools: A DAML-S description of an RDF-store, in *Proceedings of the Second International Semantic Web Conference (ISWC)*, Sanibel Island, Florida, USA

Roman, D., Lausen, H., and Keller, U.: 2004a, *D2v02. Web Service Modeling Ontology - Standard (WSMO - Standard) WSMO Working Draft 06 March 2004*, http://www.wsmo.org/2004/d2/v02/20040306/

Roman, D., Lausen, H., and Keller, U.: 2004b, *Web Service Modeling Ontology - Standard (WSMO - Standard)*, http://www.wsmo.org/2004/d2/

Roman, D., Lausen, H., Oren, E., and Lara, R.: 2004c, *Web Service Modeling Ontology - Lite (WSMO-Lite)*, http://www.wsmo.org/2004/d11/

Roman, D., Vasiliu, L., Stollberg, M., and Bussler, C.: 2004d, *Choreography in WSMO*, http://www.wsmo.org/2004/d14/

Sabou, M., Richards, D., and Splunter, S.: 2003, An experience report on using DAML-S, in *WWW 2003 workshop on E-services and the Semantic Web (ESSW03)*, Budapest, Hungary

Schlimmer, J.: 2004, *Web Services Description Requirements*, http://www.w3.org/TR/ws-desc-reqs/

Sinuhe, A., Stollberg, M., and Ding, Y.: 2004, *D3.1v01. WSMO Primer, DERI Working Draft 19 April 2004*, http://www.wsmo.org/2004/d3/d3.1/v0.1/

Sivashanmugam, K., Miller, J. A., Sheth, A. P., and Verma, K.: 2004, *Framework for Semantic Web Process Composition*, Technical report

Sivashanmugam, K., Verma, K., Sheth, A., and Miller, J.: 2003, Adding semantics to web services standards, in *Proceedings of the International Conference on Web Services (ICWS'03), 2003*

Stollberg, M., Lausen, H., Keller, U., Lara, R., and Polleres, A.: 2004, *WSMO Use Case Modeling and Testing*, http://www.wsmo.org/2004/d3/d3.2/

Sycara, K., Widoff, S., Klusch, M., and Lu, J.: 2002, Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace, *Autonomous Agents and Multi-Agent Systems* **5(2)**, 173–203

Tidwell, D.: 2000, *Web Services: The Web's Next Revolution*, http://www.ibm.com/developerWorks

Tomaz, R. F. and Labidi, S.: 2003, Increasing matchmaking semantics in intelligent commerce system, in *IEEE/WIC International Conference on Web Intelligence (WI'03)*, Halifax, Canada

Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., and Miller, J.: 2004, A scalable P2P infrastructure of registries for semantic publication and discovery of web services

Appendix A

Glossary

Sources for the glossary were the W3C Web Services Glossary [1] and the Semantic Web Services Languages Requirements document [2].

advertisement A service description or capability request that is submitted to some individual or intermediary (e.g. a discovery service), with the intention of being ”discovered” in response to (or matched with) a request.

agent An agent is a program acting on behalf of a person or organization.

architecture The software architecture of a program or computing system is the structure or structures of the system. This structure includes software components, the externally visible properties of those components, the relationships among them and the constraints on their use. A software architecture is an abstraction of the run-time elements of a software system during some phase of its operation. A system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture.

authentication Authentication is the process of verifying that a potential partner in a conversation is capable of representing a person or organization .

authorization The process of determining, by evaluating applicable access control information, whether a subject is allowed to have the specified types of access to a particular resource. Usually, authorization is in the context of authentication. Once a subject is authenticated, it may be authorized to perform different types of access.

binding An association between an interface, a concrete protocol and a data format. A binding specifies the protocol and data format to be used in transmitting messages defined by the associated interface. The mapping of an interface and its associated

[1]http://www.w3.org/TR/ws-gloss/
[2]http://www.daml.org/services/swsl/requirements/swsl-requirements.shtml

operations to a particular concrete message format and transmission protocol. See also SOAP binding.

capability A capability is a named piece of functionality (or feature) that is declared as supported or requested by an agent.

capability request A description of a desired service. This may be in terms of functional and/or non-functional properties, that define the salient properties of a required service, and desired characteristis that assist in the filtering and ranking of matching service descriptions.

choreography A choreography defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state. Web Services Choreography concerns the interactions of services with their users. Any user of a Web service, automated or otherwise, is a client of that service. These users may, in turn, be other Web Services, applications or human beings. Transactions among Web Services and their clients must clearly be well defined at the time of their execution, and may consist of multiple separate interactions whose composition constitutes a complete transaction. This composition, its message protocols, interfaces, sequencing, and associated logic, is considered to be a choreography.

contract Result of a negotiation process (if successful)

conversation A Web service conversation involves maintaining some state during an interaction that involves multiple messages and/or participants.

discovery The act of locating a machine-processable description of a Web service-related resource that may have been previously unknown and that meets certain functional criteria. It involves matching a set of functional and other criteria with a set of resource descriptions. The goal is to find an appropriate Web service-related resource.

discovery service A discovery service is a service that enables agents to retrieve Web services-related resource description.

dynamic state constraint Condition involving predicates on any combination of these states: initial, final, intermediate.

dynamic action constraint Condition on the order and/or occurrences of actions (e.g., action A before action B, if A and B execute then C executes, if A executes then B before C).

effect of an action The actual change to the initial state that the action does. Note that the effect can be specified as a formula that involves the final state, but, unlike the postcondition, such a formula is always true in any final state of the action.

functional description A description of a service in terms of (a) what is required by the service in order that it can execute successfully, and (b) what is generated by the successful execution of the service. This includes data elements (i.e. data that is consumed by the service, or produced by it) and knowledge states (i.e. states that should be asserted prior to invocation, or that are generated by the service during execution).

general constraint A formula involving any of the above types of predicates.

negotiation process Execution of a negotiation protocol.

non-functional description A description of a service in terms of its descriptive meta-data, such as a reference to a classification type, or characteristics that are not directly related to the functional description of the service.

orchestration An orchestration defines the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function. I.e., an orchestration is the pattern of interactions that a Web service agent must follow in order to achieve its goal.

precondition of action A constraint that expresses a condition on the initial state of action execution.

postcondition of action A constraint that expresses a condition on the final state of action execution (what must be true in the state immediately after the action is over). Note that a postcondition is a constraint. The action definition might not imply that the postcondition is true in every possible final state of the action.

query A description of the desired products of a service. This may require the synthesis of a capability request that is then matched to an advertised service description, and subsequently invoked, possibly by an intermediary.

request A service description or capability request that is submitted to some individual or intermediary (e.g. a discovery service), with the intention of finding or locating a matcing capability request or A service description already advertised.

service description A service description is a set of documents that describe the interface to and semantics of a service.

service interface A service interface is the abstract boundary that a service exposes. It defines the types of messages and the message exchange patterns that are involved in interacting with the service, together with any conditions implied by those messages.

A logical grouping of operations. An interface represents an abstract service type, independent of transmission protocol and data format.

service intermediary A service intermediary is a Web service whose main role is to transform messages in a value-added way. (From a messaging point of view, an intermediary processes messages en route from one agent to another.) Specifically, we say that a service intermediary is a service whose outgoing messages are equivalent to its incoming messages in some application-defined sense. See SOAP intermediary.

service provider Provider agent or provider entity.

service request A description of a desired service. This may be in terms of functional and/or non-functional properties, that define the salient properties of a required service, and desired characteristics that assist in the filtering and ranking of matching service descriptions.

service requester Requester agent or requester entity.

service role An abstract set of tasks which is identified to be relevant by a person or organization offering a service. Service roles are also associated with particular aspects of messages exchanged with a service.

service semantics The semantics of a service is the behavior expected when interacting with the service. The semantics expresses a contract (not necessarily a legal contract) between the provider entity and the requester entity. It expresses the effect of invoking the service. A service semantics may be formally described in a machine readable form, identified but not formally defined, or informally defined via an out of band agreement between the provider and the requester entity.

service-oriented architecture A set of components which can be invoked, and whose interface descriptions can be published and discovered.

SOAP The formal set of conventions governing the format and processing rules of a SOAP message. These conventions include the interactions among SOAP nodes generating and accepting SOAP messages for the purpose of exchanging information along a SOAP message path .

SOAP application A software entity that produces, consumes or otherwise acts upon SOAP messages in a manner conforming to the SOAP processing model.

SOAP binding The formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange. Examples of SOAP bindings include carrying a SOAP message within an HTTP entity-body, or over a TCP stream.

SOAP body A collection of zero or more element information items targeted at an ultimate SOAP receiver in the SOAP message path.

SOAP envelope The outermost element information item of a SOAP message.

SOAP feature An extension of the SOAP messaging framework typically associated with the exchange of messages between communicating SOAP nodes. Examples of features include ”reliability”, ”security”, ”correlation”, ”routing”, and the concept of message exchange patterns.

SOAP header A collection of zero or more SOAP header blocks each of which might be targeted at any SOAP receiver within the SOAP message path.

SOAP header block An element information item used to delimit data that logically constitutes a single computational unit within the SOAP header. The type of a SOAP header block is identified by the fully qualified name of the header block element information item.

SOAP intermediary A SOAP intermediary is both a SOAP receiver and a SOAP sender and is targetable from within a SOAP message. It processes the SOAP header blocks targeted at it and acts to forward a SOAP message towards an ultimate SOAP receiver.

SOAP message The basic unit of communication between SOAP nodes.

SOAP message exchange pattern (MEP) A template for the exchange of SOAP messages between SOAP nodes enabled by one or more underlying SOAP protocol bindings. A SOAP MEP is an example of a SOAP feature.

SOAP message path The set of SOAP nodes through which a single SOAP message passes. This includes the initial SOAP sender, zero or more SOAP intermediaries, and an ultimate SOAP receiver.

SOAP node The embodiment of the processing logic necessary to transmit, receive, process and/or relay a SOAP message, according to the set of conventions defined by this recommendation. A SOAP node is responsible for enforcing the rules that govern the exchange of SOAP messages. It accesses the services provided by the underlying protocols through one or more SOAP bindings.

SOAP receiver A SOAP node that accepts a SOAP message.

SOAP role A SOAP node’s expected function in processing a message. A SOAP node can act in multiple roles.

SOAP sender A SOAP node that transmits a SOAP message.

transaction Transaction is a feature of the architecture that supports the coordination of results or operations on state in a multi-step interaction. The fundamental characteristic of a transaction is the ability to join multiple actions into the same unit of work, such that the actions either succeed or fail as a unit .