WP 3: Service Ontologies and Service Description  
D3.7  
Linking developed ontologies and WSMO to industrial standards and further development of D3.2  

August 1st, 2005
EXECUTIVE SUMMARY

This deliverable relates the DIP service definition framework to current web service related standards. The DIP service definition framework defined in deliverable 3.2 specifies a meta-model dealing with declarative aspects of semantic web service description. The service definition meta-model is realized in DIP with the help of the Web Service Modeling Ontology (WSMO).

In the following sections we outline the relationships between WSMO and commonly accepted WS specifications and standards. First, a short overview of the WSMO description framework is presented and then its position with regard to the WS Architecture Protocol Stack is identified. Then we outline the ways to link WSMO to major protocols from different levels of the stack. A comparison of WSMO to other frameworks for semantic web services such as OWL-S and ebXML is presented and the advantages of WSMO are identified.

The DIP service definition framework also defines a number of extensibility points that can further specify concrete aspects related to business data, business processes and goals. Each extension point enables the definition of an ontology conceptualization. This deliverable also discusses the relationships between the DIP business data ontology and existing conceptualizations of business data.

The relations between WSMO and existing WS standards as well as the comparison of WSMO and the DIP business data ontology to other service related frameworks and ontologies directly contributes to one of the main goals of DIP – namely, producing a standard proposal through SWSI. Such a proposal will highly benefit from a document outlining the position of WSMO and DIP ontologies with respect to existing work in the semantic web service and web service integration domains. This deliverable has a very wide target audience, since it is related to work carried out not only in WP3 but also the other packages of DIP.

Disclaimer: The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.
## Document Information

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>FP6 – 507483</th>
<th>Acronym</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full title</td>
<td>Data, Information, and Process Integration with Semantic Web Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project URL</td>
<td><a href="http://dip.semanticweb.org">http://dip.semanticweb.org</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Document URL</td>
<td>Kai Tullius</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deliverable Number</td>
<td>3.7</td>
<td>Title</td>
<td>Linking developed ontologies and WSMO to industrial standards and further development of D3.2</td>
</tr>
<tr>
<td>Work package Number</td>
<td>3</td>
<td>Title</td>
<td>Service Ontologies and Service Description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date of delivery</th>
<th>Contractual</th>
<th>Actual</th>
<th>18-July-05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>version. 1.0</td>
<td>final</td>
<td></td>
</tr>
<tr>
<td>Nature</td>
<td>Prototype</td>
<td>Report</td>
<td>Dissemination</td>
</tr>
<tr>
<td>Dissemination Level</td>
<td>Public</td>
<td>Consortium</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Authors (Partner)</th>
<th>Zlatina Marinova (Sirma), Jens Lemcke (SAP),</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsible Author</td>
<td>Zlatina Marinova</td>
</tr>
<tr>
<td>Partner</td>
<td>Sirma</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abstract (for dissemination)</th>
<th>This deliverable investigates the links between WSMO and current standards for describing, publishing and interacting with WS as well as a comparison to other frameworks for semantically enabled WS integration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keywords</td>
<td>WSMO, OWL-S, ebXML, WSDL, UDDI, BPEL4WS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue Date</td>
</tr>
<tr>
<td>29-06-2005</td>
</tr>
<tr>
<td>18-07-2005</td>
</tr>
<tr>
<td>01-08-2005</td>
</tr>
</tbody>
</table>

Deliverable 3.7 ii 8/1/2005
## Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Ireland Galway        | NUIG    | Prof. Dr. Christoph Bussler  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Galway  
Ireland  
Email: chris.bussler@deri.org  
Tel: +353 91 512460 |
| Fundacion De La Innovacion.Bankinter         |         | Monica Martinez Montes  
Fundacion de la Innovacion. BankInter  
Paseo Castellana, 29  
28046 Madrid,  
Spain  
Email: mmtnez@bankinter.es  
Tel: 916234238 |
| Berlecon Research GmbH                        |         | Dr. Thorsten Wichmann  
Berlecon Research GmbH  
Oranienburger Str. 32  
10117 Berlin,  
Germany  
Email: tw@berlecon.de  
Tel: +49 30 2852960 |
| British Telecommunications Plc.              |         | Dr John Davies  
BT Exact (Orion Floor 5 pp12)  
Adastral Park Martlesham  
Ipswich IP5 3RE,  
United Kingdom  
Email: john.nj.davies@bt.com  
Tel: +44 1473 609583 |
| Swiss Federal Institute of Technology, Lausanne | EPFL    | Prof. Karl Aberer  
Distributed Information Systems Laboratory  
École Polytechnique Fédérale de Lausanne  
Bât. PSE-A  
1015 Lausanne, Switzerland  
Email: Karl.Aberer@epfl.ch  
Tel: +41 21 693 4679 |
| Essex County Council                          | Essex   | Mary Rowlett,  
Essex County Council  
PO Box 11, County Hall, Duke Street  
Chelmsford, Essex, CM1 1LX  
United Kingdom.  
Email: maryr@essexcc.gov.uk  
Tel: +44 (0)1245 436524 |
| Forschungszentrum Informatik                  | FZI     | Andreas Ahecker  
Forschungszentrum Informatik  
Haid-und-Neu Strasse 10-14  
76131 Karlsruhe  
Germany  
Email: ahecker@fzi.de  
Tel: +49 721 9654 0 |
## Deliverable 3.7: Linking developed ontologies and WSMO to industrial standards and further development of D3.2

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| Institut für Informatik, Leopold-Franzens Universität Innsbruck | UIBK | Prof. Dieter Fensel  
Institute of computer science  
University of Innsbruck  
Technikerstr. 25  
A-6020 Innsbruck, Austria  
Email: dieter.fensel@deri.org  
Tel: +43 512 5076485 |
| ILOG SA | ILOG | Christian de Sainte Marie  
9 Rue de Verdun, 94253  
Gentilly, France  
Email: csma@ilog.fr  
Tel: +33 1 49082981 |
| inubit AG | Inubit | Torsten Schmale  
inubit AG  
Lützowstraße 105-106  
D-10785 Berlin  
Germany  
Email: ts@inubit.com  
Tel: +49 30726112 0 |
| Intelligent Software Components, S.A. | iSOCO | Dr. V. Richard Benjamin, Director R&D  
Intelligent Software Components, S.A.  
Pedro de Valdivia 10  
28006 Madrid, Spain  
Email: rbenjamin@isoco.com  
Tel: +34 913 349 797 |
| NIWA WEB Solutions | NIWA | Alexander Wahler  
NIWA WEB Solutions  
Niederacher & Wahler OEG  
Kirchengasse 13/1a  
A-1070 Wien  
Email: wahler@niwa.at  
Tel: +43(0)1 3195843-11 |
| The Open University | OU | Dr. John Domingue  
Knowledge Media Institute  
The Open University, Walton Hall  
Milton Keynes, MK7 6AA  
United Kingdom  
Email: Jd.domingue@open.ac.uk  
Tel: +44 1908 655014 |
| SAP AG | SAP | Dr. Elmar Dörner  
SAP Research, CEC Karlsruhe  
SAP AG  
Vincenz-Priessnitz-Str. 1  
76131 Karlsruhe, Germany  
Email: elmar.dorner@sap.com  
Tel: +49 721 6902 31 |
## Deliverable 3.7

**Linking developed ontologies and WSMO to industrial standards and further development of D3.2**

<table>
<thead>
<tr>
<th>Company</th>
<th>Contact Person</th>
<th>Address</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sirma Al Ltd.</td>
<td>Atanas Kiryakov</td>
<td>Ontotext Lab, - Sirma AI EAD Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse Sofia 1784, Bulgaria</td>
<td><a href="mailto:atanaskiryakov@sirma.bg">atanaskiryakov@sirma.bg</a></td>
<td>+359 2 9768 303</td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Jeff Eisenberg</td>
<td>Unicorn Solutions Ltd. Malcha Technology Park 1 Jerusalem 96951 Israel</td>
<td><a href="mailto:jeff.eisenberg@unicorn.com">jeff.eisenberg@unicorn.com</a></td>
<td>+972 2 6491111</td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>Carlo Wouters</td>
<td>Starlab- VUB Vrije Universiteit Brussel Pleinlaan 2, G-10 1050 Brussel, Belgium</td>
<td><a href="mailto:carlo.wouters@vub.ac.be">carlo.wouters@vub.ac.be</a></td>
<td>+32 (0) 2 629 3719</td>
</tr>
</tbody>
</table>
**LIST OF KEY WORDS/ABBREVIATIONS**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCC</td>
<td>Association Core Component(s)</td>
</tr>
<tr>
<td>ASM</td>
<td>Abstract State Machine(s)</td>
</tr>
<tr>
<td>B2B</td>
<td>Business to Business</td>
</tr>
<tr>
<td>BCC</td>
<td>Basic Core Component(s)</td>
</tr>
<tr>
<td>BDO</td>
<td>Business Data Ontology</td>
</tr>
<tr>
<td>BIE</td>
<td>Business Information Entities</td>
</tr>
<tr>
<td>BPEL4WS</td>
<td>Business Process Execution Language for Web Services</td>
</tr>
<tr>
<td>BULO</td>
<td>Base Upper-level Ontology</td>
</tr>
<tr>
<td>CCT</td>
<td>Core Component Types</td>
</tr>
<tr>
<td>CCTS</td>
<td>Core Components Technical Specification</td>
</tr>
<tr>
<td>DC</td>
<td>Dublin Core</td>
</tr>
<tr>
<td>ebXML</td>
<td>Electronic Business XML</td>
</tr>
<tr>
<td>IOPEs</td>
<td>Inputs, Outputs, Preconditions and Effects</td>
</tr>
<tr>
<td>KIMO</td>
<td>Knowledge and Information Management Ontology</td>
</tr>
<tr>
<td>MEP</td>
<td>Message Exchange Pattern</td>
</tr>
<tr>
<td>NDRs</td>
<td>Naming and Design Rules</td>
</tr>
<tr>
<td>NFP</td>
<td>Non-functional Property(ies)</td>
</tr>
<tr>
<td>OWL-S</td>
<td>Web Ontology Language for Services</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Service(s)</td>
</tr>
<tr>
<td>UBL</td>
<td>Universal Business Language</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description Discovery and Integration</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifiers</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locators</td>
</tr>
<tr>
<td>WS</td>
<td>Web Service</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Definition Language</td>
</tr>
<tr>
<td>WSMF</td>
<td>Web Service Modeling Framework</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modeling Ontology</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

EXECUTIVE SUMMARY ........................................................................................................1
LIST OF KEY WORDS/ABBREVIATIONS ............................................................................ VI
TABLE OF CONTENTS ........................................................................................................ VII
1 INTRODUCTION .............................................................................................................. 1
2 OVERVIEW OF WSMO .................................................................................................. 1
  2.1 Semantic Web Service Related Descriptions.......................................................... 1
  2.2 Service Capability .................................................................................................... 3
  2.3 Service Behaviour ................................................................................................... 3
  2.4 Service Grounding .................................................................................................. 4
3 RELATING WSMO TO CURRENT WS STANDARDS .................................................. 5
  3.1 WSMO and the Web Service Architecture Stack .................................................... 6
  3.2 Web Service Definition Language ........................................................................ 7
    3.2.1 WSDL 1.1 .................................................................................................... 7
    3.2.2 Overview of WSDL 2.0................................................................................ 8
    3.2.3 WSMO and WSDL....................................................................................... 9
  3.3 Business Process Execution Language for Web Services..................................... 9
    3.3.1 WSMO and BPEL4WS .............................................................................. 11
  3.4 Universal Description Discovery and Integration.................................................. 12
    3.4.1 Creating WSMO Descriptions for Services Published in UDDI Registries.. 13
    3.4.2 UDDI Registries and WSMO Descriptions................................................. 13
  3.5 Other Standards...................................................................................................... 14
    3.5.1 WS-Policy .................................................................................................. 14
    3.5.2 WS-Addressing .......................................................................................... 14
    3.5.3 WS-MEX ................................................................................................... 15
4 COMPARISON OF WSMO AND OTHER APPROACHES TO SWS INTEGRATION..... 16
  4.1 OWL-S ................................................................................................................. 16
    4.1.1 Overview of OWL-S .................................................................................. 16
    4.1.2 Comparison between OWL-S and WSMO.................................................. 17
  4.2 ebXML ................................................................................................................... 19
    4.2.1 Technological Foundations and Business Interactions................................. 19
    4.2.2 ebXML and WSMO ................................................................................... 21
5 BUSINESS DATA ONTOLOGY AND RELATED ONTOLOGIES................................. 21
  5.1 Basic Upper-Level Ontology .................................................................................. 22
5.2 Universal Business Language ................................................................. 22

6 CONCLUSION ............................................................................................. 25

REFERENCES ................................................................................................. 26

LIST OF FIGURES

Figure 1: Web Service Architecture Stack ................................................. 5
Fig. 2: Sample ebXML interaction between two companies ..................... 20
Fig. 3: Core Component Overview .............................................................. 24
1 INTRODUCTION

This deliverable relates the DIP service definition framework to current web service related standards. The DIP service definition framework defined in deliverable 3.2 specifies a meta-model dealing with declarative aspects of semantic web service description. The service definition meta-model is realized in DIP with the help of the Web Service Modeling Ontology (WSMO).

In the following sections we outline the relationships between WSMO and commonly accepted WS specifications and standards. First, a short overview of the WSMO description framework is presented and then its position with regard to the WS Architecture Protocol Stack is identified. Then we outline the ways to link WSMO to major protocols from different levels of the stack. A comparison of WSMO to other frameworks for semantic web services such as OWL-S and ebXML is presented and the advantages of WSMO are identified.

The DIP service definition framework also defines a number of extensibility points that can further specify concrete aspects related to business data, business processes and goals. Each extension point enables the definition of an ontology conceptualization. This deliverable also discusses the relationships between the DIP business data ontology and existing conceptualizations of business data.

An important note should be made, that this deliverable discusses the relationships of WSMO and the Business Data Ontology, in their current state of development, with the current specifications of WS related standards. Future work on the DIP ontologies and WSMO or changes in the discussed web service standards can possibly make sections of this deliverable outdated.

2 OVERVIEW OF WSMO

The Web Service Modeling Ontology (WSMO) is a conceptual model that formalizes and extends the Web Service Modeling Framework (WSMF) presented in [1]. WSMO can be used to describe various aspects of semantic web services. Just like WSMF it discerns four major elements related to semantic web services (SWS) – ontologies, goals, service descriptions and mediators. WSMO is designed as a meta-model for SWS with fully extensible vocabulary.

In this section a short overview of the current stage of development of the WSMO conceptual model is presented. First we present an overview of the top level entities in WSMO (based on [7]), then we outline the current status of the service capability and interface specifications.

2.1 Semantic Web Service Related Descriptions

The top level entities in the WSMO model are:

- Ontologies – providing the shared conceptual representation of the business domain;
- Goals – WSMO explicitly separates the point of view of service requestors and enables descriptions of required capability and interface as Goals;
• Services – capturing all the information related to the capability of a WS and specification of its interfaces;

• Mediators – used to overcome heterogeneities at the data, protocol and process representation levels.

Metadata can be associated with each WSMO element descriptions using non-functional properties (NFP). The set of properties contains the Dublin Core (DC) elements plus some Quality of Service (QoS) related properties. WSMO specifies the basic understanding of what each NFP should represent as well as recommendations on the value types for some properties. NFP are designed to be easily extensible.

In order to accommodate explicit semantic definition of service capability and conditions related to service execution, a language for logical expressions is also specified. The language definition (see [7]) consists of a basic vocabulary (identifying the sets of symbols that are acceptable), the basic terms (denoting objects and providing their names in the universe of discourse) and the ways in which they can be combined to form (correct) simple and complex logical expressions (axioms). The logical language can be extended and customized for the needs of particular application domains.

**Ontologies**

Ontologies have two roles in WSMO- they provide the formal semantics of information as well as present human terminology (common conceptualization of a domain) in machine interpretable format.

Ontologies define the concepts in the business domain, particular instances of those concepts as well as structural (defining the taxonomy of concepts) and other relationships (formalized as WSMO relations) between those concepts. Instances of concepts and instances of relations are supported.

**Goals**

Goals specify the capability and the interface of the requested service that is needed to satisfy the user’s objective. Explicit separation of goal and service notions, promote strong decoupling between requestor and provider points of view

**Service Descriptions**

Web services are described in WSMO from three different points of view: non-functional properties, functionality and behaviour. As explained above the NFP are used to store metadata annotations of the service based on the DC plus quality of service related characteristics. Functionality is defined by the service capability, while behavioural aspects are captured by the service Interface.

**Mediators**

Mediators describe elements that aim to overcome structural, semantic or conceptual mismatches that appear between the different components that build up a WSMO description. Mediators can be used by all WSMO top level entities to resolve mismatches when reusing ontologies, to link predefined goals with web services that can fulfil them, and to enable seamless communication between services having heterogeneous data and process representations.
2.2 Service Capability

The functional description of a web service is captured by its capability. According to WSMO Primer ([8]), functionality is described in terms of “what the Web service offers to a client (postcondition), when some conditions are met in the information space (precondition), and how the execution of the Web service changes the world (effect), given that some conditions over the world state are met before execution (assumption).”

A service can advertise exactly one description of its capability. Capabilities can also be annotated with NFP, and make use of domain knowledge with the help of ontologies. Ontologies can either be directly imported or reused through mediators. Preconditions, postconditions, assumptions and effects are defined as sets of axioms defined with the help of the WSMO logical expression language.

2.3 Service Behaviour

Choreography and orchestration are the two sides of the behavioural description of the service captured by its interface. WSMO choreography specifies how the client should communicate with a service so that it provides its capability. Thus, WSMO choreography focuses on the interactions of services with their users (it complies with the W3C definition of choreography\(^1\)), and not on multiple party interaction protocols. WSMO orchestration is used to specify how the service can collaborate with other WSMO services, or which goals should be fulfilled, in order to provide its functionality. Hence, WSMO orchestration regards the collaboration aspect of service behaviour.

From a B2B point of view WSMO choreographies can be used to define two types of interactions with a WS - execution choreography which defines a protocol for interaction with the WS during service delivery; and meta-choreography that specifies the interaction protocol used for negotiating an agreed service and for monitoring the agreed service level during execution.

Both choreography and orchestration descriptions in WSMO are conceptually based on Abstract State Machines (ASM). According to [9] this decision enables: minimal ontological commitment (hence, easier standardisation), modelling of any needed behaviour (ASM are expressive enough), and formal semantics. The underlying ASM model also enables reasoning about choreography and orchestration definitions.

A choreography definition consists of a state ontology and rules for the guarded transitions between states. Transition rules fire when the current state matches some conditions (guard), and as a result the state is changed by means of modifying the set of instances and attribute values in the state ontology. State signatures are defined by one or more ontologies and the roles of different concepts and relations with respect to the choreography ASM. These roles express in which way concept or relation extensions can be modified: static (change is not allowed), controlled (extension can be changed only by the choreography ASM), shared (changes both by the choreography and by the environment are possible), out (changes by the service are only possible), in (changes are caused by the environment only). It is a task of WSMO Grounding to describe how the client actually writes or reads instances having in, out or shared role. WSMO choreography further enables annotation of concepts in the state signature with

---

\(^1\) [http://www.w3.org/TR/ws-gloss/](http://www.w3.org/TR/ws-gloss/)
grounding information in the value of the *grounding* non-functional property. Multiple grounding specifications can be associated with a concept to describe the allowed values for this property and what they mean.

The basic form of transition rule specifies state changes by means of adding, updating or removing instance data in the state ontology. More complex transition rules can be recursively defined in one of the following forms:

\[
\text{if } \text{Cond} \text{ then } \text{Rules endif,}
\]

\[
\text{forall Variables with Condition do } \text{Rules endDo,}
\]

\[
\text{choose Variables with Condition do } \text{Rules endDo,}
\]

where Variables are WSML variables, and Condition is a WSML logical expression\(^2\) with all variables being quantified in the scope of the *choose* or *forall* constructs.

Sub-capabilities can be associated with rules to enable modelling of side effects of service execution, complex postconditions as well as to enable structuring the service capability on the basis of user interactions. Sub-capabilities are described as WSML capabilities specifying the postconditions and effects that will hold after the given rule fires, given that the assumptions and preconditions were satisfied.

2.4 Service Grounding

There are two aspects of grounding a semantic web service to concrete WS realization usually described using WSDL:

- Mapping semantically rich data, described in ontologies to XML messages used to interact with the WS, and

- Specifying how the behavioural description of SWS is mapped to a WS interface.

WSMO Grounding (presented in [18]) defines both mapping of data represented in WSMO ontologies to XML (enabling exchanging of messages) and mapping of behavioural service description to WSDL (enabling actual invocation of WSDL operations during WSMO service execution). Thus, WSMO Grounding links the semantic representation of a WS to its syntactical description (in WSDL).

Data transformation between WSMO ontologies and XML is bidirectional: on the one hand, semantic data should be “lowered” to XML representation in order to be able to send messages to a WSDL service, and on the other hand, XML messages received from a WS should be “lifted” back to the semantic level. WSMO Grounding specifies data transformations as conceptual level mappings between WSMO ontologies and XML Schema. This approach is described in details in [18].

Grounding a WSMO service description involves grounding the WSMO choreography description to a WSDL service description. This grounding enables WSMO execution environments to realize a choreography by sending and receiving WSDL messages to the WS implementation. Thus, WSMO Grounding of the choreography specifies how the client of the service can generate concrete messages that are mapped to concepts

\(^2\) Definitions of WSML variable and WSML logical expression can be found at: http://www.wsmo.org/TR/d16/d16.1/v0.2/
marked as *in* or *shared* in the state ontology, as well as how ontology concepts with *out* or *shared* role can be sent back to the client. Details about the actual serialization of the XML messages on the wire are given in the WSDL service description and in order to reuse these details the WSMO service can reference the WSDL service.

A WSMO service can reference a WSDL service, and reuse its on-the-wire serialization and endpoint information, with the help of a designated NFP. The endpointDescription non-functional property can be used to store a URI pointing to the appropriate WSDL service.

In addition to providing a mechanism for grounding WSMO services to existing WSDL services, WSMO Grounding also specifies how WSDL service descriptions can be generated from WSMO descriptions. The rules used in generating WSDL out of a WSMO service actually specify default grounding. The default grounding of WSMO services in WSDL is defined in [18].

### 3 Relating WSMO to Current WS Standards

The Web Services Architecture\(^3\) involves many protocols related to different phases of the WS lifecycle. These protocols can be grouped in layers forming the Web Services Architecture Stack\(^4\) presented on Figure 1.

![Web Service Architecture Stack](http://www.w3.org/TR/ws-arch/)

\(^3\) As defined in the Web Services Architecture W3C Technical Report: [http://www.w3.org/TR/ws-arch/](http://www.w3.org/TR/ws-arch/)

\(^4\) As defined in the Web Services Architecture W3C Technical Report: [http://www.w3.org/TR/ws-arch/](http://www.w3.org/TR/ws-arch/)
While there are many specifications of protocols operating on different levels of the stack, still SOAP\(^5\), WSDL\(^6\), UDDI\(^7\) and BPEL4WS\(^8\) are the de facto standards adopted in the WS community.

The Web Service Definition Language (WSDL) is the standard for describing web services in terms of exposed interfaces (abstract interface) and implementation part (describing how the service can be accessed and what protocols it supports). It defines the set of messages that can be exchanged when interacting with a WS and the message characteristics of end points. Data types are defined by XML Schema specification, which supports rich type definitions and allows expressing any kind of XML type requirement for the application data.

WSDL service descriptions are useful for services participating in stateless message exchanges. Web services participating in longer conversations need means for descriptions of logical and temporal dependencies between messages. That is when choreography languages such as the Business Process Execution Language for Web Services (BPEL4WS) are used. BPEL4WS provides a language for the formal specification of business processes and business interaction protocols. By doing so, it extends the web services interaction model implied in WSDL and enables it to support business transactions. BPEL4WS defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

The focus of Universal Description Discovery and Integration (UDDI) is the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other web services providers, (2) the web services they make available, and (3) the technical interfaces which may be used to access those services. Based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP, UDDI provides an interoperable, foundational infrastructure for a web services-based software environment for both publicly available services and services only exposed internally within an organization.

Transport level protocols are considered once a web service is discovered and has to be invoked. SOAP defines the basic formatting of a message and the basic delivery options independent of programming language, operating system, or platform. A SOAP compliant web service knows how to send and receive SOAP-based messages.

### 3.1 WSMO and the Web Service Architecture Stack

The Web Service Architecture stack can benefit from introducing semantics at the service description and process levels. As pointed out in [13], adding semantics to service descriptions, service publishing and discovery as well as service process layers enables seamless integration of WS.

---

\(^5\) [http://www.w3.org/TR/soap/](http://www.w3.org/TR/soap/)

\(^6\) [http://www.w3.org/TR/wsdl](http://www.w3.org/TR/wsdl)

\(^7\) [http://www.uddi.org/](http://www.uddi.org/)

WSMO provides means for creating semantically rich descriptions of web services, their functionality and behaviour. Thus, it promotes semantic enabled integration of web services.

Next we are going to present the current de-facto standards for WS descriptions, discovery and processes, and how they can be related to WSMO. Then a short overview of transport and security related standard specifications is presented.

3.2 Web Service Definition Language

The Web Service Definition Language (WSDL) provides a common way for describing web services where the abstract definition of functionality offered by the service is separated from specific data about accessing the service. An abstract service definition can be realized on the wire as SOAP or HTTP message exchange.

Every WSDL document is structured as a description element which serves as container for the other elements. The Types section defines the data types that the WS will be able to send and receive. Message types are defined through a predefined schema that the WSDL processor (service requestor) should be able to interpret.

In the next sections a short overview of WSDL v1.1\(^9\) and v 2.0\(^10\) is presented. Then the relations between WSDL and WSMO are outlined.

3.2.1 WSDL 1.1

A WSDL 1.1 description of a web service consists of the following parts:

- **Types** – a container for data type definitions using some type system (such as XML Schema).

- **Message** – an abstract, typed definition of the data being communicated.

- **Operation** – an abstract description of a message exchange supported by the service.

- **Port Type** – a set of operations supported by one or more endpoints.

- **Binding** – a concrete protocol and data format specification for a particular port type.

- **Port** – a single endpoint defined as a combination of a binding and a network address.

- **Service** – a collection of related endpoints.

Interactions between services are realized through message exchanges (based on four predefined Message Exchange Patterns – MEPs). A message definition specifies the abstract messages that the service can send or receive. A message can be comprised of different parts. Each part format is described by a reference to a type definition or as an element component. Even though WSDL recommends that XML Schema is used for specifying message parts, it is possible to use any other appropriate type system.

\(^9\) [http://www.w3.org/TR/wsdl](http://www.w3.org/TR/wsdl)

Abstract messages can be referenced from other parts of the WSDL document, e.g. port operation definitions and bindings.

Related input and output messages can be grouped in operations. A set of operations can be grouped to form a port type. Each operation actually defines a MEP (one of four types) that characterizes an interaction with the WS. Fault messages can be defined to enable error handling in the interaction protocol.

The port type element abstractly defines a WS as a set of related operations and exchanged messages. It is usually compared to a function library (module or class) in traditional programming languages, while operations correspond to specific functions, and message parts to parameters. WSDL ports define the actual endpoint supported by the service.

Abstract port types are mapped to concrete message format and transmission protocol with the help of bindings. WSDL defines binding operation elements that provide protocol specific information for operations, and binding message references that specify concrete format for the message of an operation. Binding extension elements are used to provide information specific to a particular binding.

A WSDL service combines the abstract and concrete definitions of a WS interface. It specifies the set of abstract port types and the concrete ports where they are provided. Port definitions are local to a service element. They provide information about endpoints where a service can be accessed.

3.2.2 Overview of WSDL 2.0

The major changes introduced in WSDL 2.0 are briefly presented below.

*Interface* element is introduced in WSDL 2.0 to describe the abstract WS interface (instead of the *portType* element used in WSDL 1.1). An interface is defined as a set of abstract *operations*, each operation representing a simple interaction between the client and the service. Inheritance of interfaces is possible, which enables further modularization of service descriptions. Faults can be defined for an interface, and then can be reused by multiple operations. Faults are designed as a special kind of messages that can be reused in an interface.

Operations specify the Message Exchange Pattern (discussed below) which is followed during interactions with the service. In WSDL 2.0 operations have the following properties:

- *pattern* specifies the MEP that is realized by the operation.
- *style* – defines the rules followed when defining a particular operation. Can be RPC style, URI style, multipart style. Value is URI pointing to a particular rule system.
- *safety* – an attribute that has a boolean value indicating whether the operation is safe in terms of the Web Services Architecture\(^\text{11}\). Safe operations would not impose any additional obligations for the client (payments for example).
- Operations can also have *input*, *output*, *input fault* and *output fault* message references specifying the messages exchanged according to the MEP.

\(^{11}\) [http://www.w3.org/TR/ws-arch/](http://www.w3.org/TR/ws-arch/)
WSDL 2.0 predefines eight MEPs and also enables new MEPs to be created and used. MEPs are used to define the sequence and cardinality of the abstract messages in an operation. MEPs are abstract in the sense that they define placeholders for abstract messages and also are abstracting the binding information.

The service section specifies where the defined WS can be accessed. A WSDL 2.0 service specifies a single interface that the WS will support, and a list of *endpoint* locations (equivalent to *ports* in WSDL 1.1) where that service can be accessed. Each endpoint must also reference a binding to indicate what protocols and transmission formats should be used at that endpoint.

### 3.2.3 WSMO and WSDL

WSMO and WSDL are complementary technologies. On the one hand, WSMO can be used to provide the missing semantics in the WS descriptions. It can even be used to describe semantics of WSDL (by an ontology), a semantics that is currently captured by the Component Model definition in WSDL 2.0\(^{12}\). On the other hand, WSMO services can be grounded to WSDL definitions that can be used for invocation of the described WS. Since WSMO does not impose any constraints on the message transmission protocols, the WSDL bindings will simply be reused and need not be mapped.

WSMO also can be used to easily describe relations between different WS interfaces (a currently problematic issue in WSDL). An example is a WS providing two interfaces, one for its clients and one for its management administrators. In WSMO each of these two WSDL interfaces can be described as a separate SWS and then the *Relation NFP* can be used to capture their relation.

The WSMO Grounding defines the mechanisms both for mapping choreography descriptions to existing WSDL services and for generating new WSDL descriptions from WSMO services defined only on the semantic level. Concrete transport protocol and endpoint information from WSDL bindings can be reused by WSMO services that provide reference to a concrete WSDL service element.

Building up a SWS choreography description from WSDL will require a human intervention due to the lack of any explicit ordering of operations, operational semantics or conditions for operation applicability. Such information might be only available in informal human readable text documents accompanying a WSDL description.

### 3.3 Business Process Execution Language for Web Services

Integration of WS requires more than the ability to define and implement simple interfaces for synchronous or unrelated asynchronous message exchanges (as in WSDL). Real business interactions usually are achieved by long-running peer-to-peer conversations between two or more parties. Such long-running stateful message exchanges are formalized in business protocols.

The *Business Process Execution Language for Web Services (BPEL4WS)* enables descriptions of business protocols that involve WS interactions, as processes. Two types of business processes are distinguished – an executable process serves as the model of the implementation of a Web service, while an abstract process describes given business

---

\(^{12}\) [http://www.w3.org/TR/wsdl20/#component_model](http://www.w3.org/TR/wsdl20/#component_model)
A protocol to be followed by interacting parties. Business processes can also be used for modelling the orchestration logic of SWS.

BPEL4WS depends on the following XML-based specifications: WSDL 1.1, XML Schema 1.0, XPath 1.0 and WS-Addressing.

A short overview of BPEL4WS (as specified in [10]) follows. BPEL4WS process definitions consist of three parts:

1. `<variables>` section that defines data variables in terms of WSDL message types, XML Schema simple types or XML Schema elements. Variables are used to store information about state and process history.

2. `<partnerLinks>` section specifying the parties that participate in the interaction. Each partner is described by a role and a partner link type.

3. `<faultHandlers>` specify the actions to be taken in case of failure of services signified by WSDL fault message. When a fault occurs during operation execution, normal processing is terminated and control is transferred to the corresponding fault handler.

Additional properties specifying XML query language, expression language, whether a process is abstract or executable can also be used in a process definition.

The BPEL4WS specification ([10]) defines the following activities that can be combined to present the process business logic:

- `<receive>` - allows business processes to wait for a given message to arrive;
- `<reply>` - enables sending a reply to a message that was previously received;
- `<invoke>` - enables invocation of a one-way or request-response operation (in terms of WSDL 1.1) on a partner’s portType;
- `<assign>` - updates the value of variables with new data. An `<assign>` construct can contain more than one elementary assignments.
- `<throw>` - used for fault generation from within a business process;
- `<terminate>` - available only for executable processes.
- `<wait>` - enables waiting for a certain period;
- `<empty>` - enables synchronization of activities through usage of a “no-op” instruction;
- `<sequence>` - defines a collection of activities to be executed in sequential order;
- `<switch>` - enables definition of branches of activities, only one of which will be executed based on given criteria.
- `<while>` - specifies an activity that should be repeated until a condition becomes true;
Activities do not explicitly exchange data, but use globally visible variables for that purpose.

Activities can be synchronized with the used of links. Linking activities is enabled through the optional `<source>` and `<target>` elements.

Messages can consist of two parts: application data and protocol relevant data. Two types of protocol data are supported – business protocol related and infrastructure protocol related, where infrastructure protocols can be specifying security, transaction or reliability constraints.

In order to enable modelling of stateful WS interactions BPEL4WS provides means for defining state variables, data manipulations using data expressions and state update through assignment.

### 3.3.1 WSMO and BPEL4WS

A mapping from BPEL4WS to WSMO would be very useful since there are many existing business process definitions associated with services published in UDDI registries. That is due to the OASIS technical recommendation for using BPEL4WS in UDDI registries ([11]). BPEL4WS process descriptions can be used for generation of choreographies since they provide explicit and structured information about the business process(es) making use of a WSDL description. Here we focus on conceptually relating the business process descriptions to WSMO choreographies. No doubt, more detailed mappings from WSMO to BPEL4WS and vice versa will be needed to enable (semi-automatic) reuse of BPEL4WS descriptions as well as generation of executable BPEL4WS processes out of WSMO choreography and orchestration definitions. Such detailed mappings should also address additional issues such as the representation of BPEL4WS security related elements.

A mapping between WSMO choreography and orchestration models and BPEL4WS can be easily defined as there are direct correspondences between most of BPEL4WS workflow activities and ASM rules. Such mapping can be extensively based on already existing methodologies for representing BPEL4WS processes as ASM (most notably [12]).

Mapping from WSMO to BPEL4WS would also be useful and with its help executable business processes can be generated out of WSMO choreography descriptions (and grounding). Orchestration of WSMO services can be similarly modelled as BPEL4WS.
executable process. Thus, existing BPEL4WS execution environments (such as the BPEL4J\(^\text{13}\) platform) can be used for invocation of WSMO services. Such a translation would also promote better acceptance (and understanding) of WSMO by businesses that are already using BPEL4WS.

### 3.4 Universal Description Discovery and Integration

Universal Description Discovery and Integration (UDDI) enables the storage of data and metadata about web services that is beyond the technical specification of the service interface. Such information is used for classification, browsing and discovering web services in the UDDI registries.

A business entity is the central data structure in UDDI. It is used to represent business structures that act as service providers (these can be companies, departments, etc.). The business entity structure contains information about the business as well as about the service(s) it offers. Descriptions can be presented in multiple languages. Contact and classification information is also kept for business entities. The services that a business entity provides are defined as *businessService* structures. Information about the service name, description and classification is stored there.

Binding templates provide the technical information needed by applications to bind and interact with the web services being described. A binding template must contain either the access point for a given service or an indirection mechanism that will lead one to the access point.

Different types of access points can be specified in a binding template. UDDI defines a list of possible types of access points but this list can be extended. Currently, access points can be: end points specifying actual network address at which the WS can be invoked; binding templates referencing to separately defined binding template entities (enables reuse of bindings for many services), hosting redirectors for WS that are hosted by remote UDDI registries, WSDL deployments referencing a remotely hosted WSDL document containing the necessary binding information.

A binding template also defines technical details about the web service as a set of technical models. Technical model structures (*tModels*) provide representation of additional information about the behaviour of web services and what specifications or standards it complies with. Thus, *tModels* help service requestors to understand how to interact with services. Technical models also enable reuse and uniform usage of concepts or constructs within the modelled framework. Binding templates that refer to precisely the same set of *tModels* are said to have the same "technical fingerprint" and are of the same type. It should be noted that technical description documents are not stored directly in UDDI registries, *tModels* only provide metadata related to the document and links pointing to locations where they can be found.

In UDDI discovery of data is enabled through a couple of mechanisms:

- Every entity can and should be classified according to one or more taxonomies.
- Searching the classified entities is enabled by the UDDI Inquiry API.
- Identification codes can also be specified to enable indexing of the entities.

Our linking of UDDI and WSMO (presented next) is based on the information model that is common for versions 2 and 3 of UDDI, thus enabling usage of both versions. If concrete mapping should be defined, version specific details should be addressed.

### 3.4.1 Creating WSMO Descriptions for Services Published in UDDI Registries

In order to reuse already existing service descriptions, it will be very useful if UDDI business service information can be extracted and reused when creating WSMO service descriptions. Apart from the metadata and service end point details that can be reused, technical models that link WSDL or BPEL4WS descriptions can be used when creating new WSMO service.

As WSMO does not explicitly model service providers, business entity specific information is not mapped to a specific WSMO element, but it will be reused for related services descriptions. In WSMO business-specific data can be captured by the service’s NFPs. More specifically business entity name can be stored in Owner and Publisher properties. The collection of services provided by a business entity can be presented as a set of relations (in Relation property) in each of those service descriptions. Contact information will be lost, since WSMO does not aim at enabling businesses to contact. Discovery URLs will also be discarded when a WSMO description is generated. Categorization information for the business is lost, as well as identification keys.

Mapping business service metadata to WSMO service is easier. A straightforward correspondence exists between the uddi:name and Title, as well as between uddi:description and Description properties. Hence these UDDI element values can be directly transferred in the corresponding non-functional properties. In order not to break the connection between a service and the business entity that publishes it, the Publisher property can be designated to keep the value of the UDDI business key. Service categorization information from the Category bag element can be stored in the Type property (multiple values to be separated by semicolon). The binding template information will be very useful, since so far no direct reference to the actual web service description (WSDL file) or access point is kept in WSMO. Thus, the endpointDescription non-functional property introduced by WSMO Grounding can be used to store references to WSDL files stored in a UDDI registry. These WSDL files can then be used for WSMO grounding and invocation. Similarly, references to associated BPEL4WS documents can be stored and then used when creating WSMO choreography of the service (based on a predefined conceptual mapping).

### 3.4.2 UDDI Registries and WSMO Descriptions

There are two use cases that require a mapping from WSMO to UDDI services. First, UDDI registries can be used to publish WSMO services and second, a relation between WSMO services and already existing UDDI services can be established. Publishing new WSMO services in a UDDI registry will require human intervention, either to establish the business entity to service connection (when the service provider entity is already present in the registry), or to create a new business entity for which a service to be published.

If we want to enable UDDI searching based on information contained in WSMO specific service non-functional properties, they should be defined as tModels and referenced in identifier bags. There are also WSMO NFP (identified in the previous section) that can be directly mapped to UDDI properties. Upon publishing a WSMO service in a UDDI based registry, business entity Discovery URLs can be used to point
to a repository (or multiple repositories) that actually stores this WSMO description, since UDDI only stores metadata and references, but not complete descriptions.

Service functionality as captured by WSMO capability cannot be directly stored in UDDI. A separate $tModel$ has to be defined instead. It can also enable categorization of capabilities if taxonomy for the concrete domain is available.

Binding templates naturally correspond to WSMO service interfaces, but since the interface does not provide any technical information it has to be modelled separately and cannot be used as binding template. On the other hand, a common use of $tModel$ is “to represent a specification that defines wire protocols, interchange formats and interchange sequencing rules”. Therefore, WSMO behavioural descriptions (choreography and orchestration) will be most naturally represented as $tModels$.

In the case when WSMO choreography description is grounded to WSDL or is related to a BPEL4WS process description, choreography $tModel$ can reference the predefined $tModels$ for the WSDL/BPEL4WS description.

3.5 Other Standards

In this section we are going to present some WS specifications that are not yet widely adopted, but are related to WSMO. Our overview is based on the mappings described in [14].

3.5.1 WS-Policy

WS-Policy framework\(^{14}\) defines a model and a grammar for expressing capabilities, requirements, and general characteristics in WS based systems. These web service properties are expressed as policies, where a policy is defined to be a collection of one or more policy assertions. Policy assertions can be used to represent both wire related requirements (such as transport protocol selection) and high level requirements related to service selection and usage (such as QoS characteristics). The WS-Policy grammar enables consistent reasoning about both types of requirements.

As WS-Policy does not specify how policies are attached to web services, WSMO can define a custom link between a service description and a policy. One possibility that is discussed in [14] incorporates policies in WSMO descriptions by defining extensions of the non-functional properties. The namespace-qualified name of the policy attribute is used as name of the new WSMO property, whose value is a list of URIs of the policies attached to a WSMO element. It is also possible to include entire policy descriptions in WSMO if the set of NFP is extended with a new $\text{wsp:Policy}$ property. This property will contain the XML serialization of the policy description.

3.5.2 WS-Addressing

The WS-Addressing specification\(^{15}\) focuses on providing transport-neutral mechanisms to address web services and messages. Currently, it provides XML elements identifying WS endpoints and means to secure end-to-end endpoint identification in messages. WS-Addressing defines two constructs that normalize information underlying in transport protocols into a uniform format that can be processed independently of transport or


\(^{15}\) http://www.w3.org/Submission/ws-addressing/
application. The two constructs are endpoint references and message information headers. A web service endpoint is a (referenceable) entity, processor, or resource where web service messages can be targeted. Endpoint references convey the information needed to identify/reference a web service endpoint. Message information headers allow uniform addressing of messages independent of underlying transport by providing end-to-end message characteristics including addressing for source and destination endpoints as well as message identity.

The WS-Addressing specification permits an endpoint reference to point to a WSDL interface so that the recipient of this reference may choose the appropriate code to use that interface. Similarly, wsml:serviceReference element can be defined. It will store references to WSMO service descriptions to be used when the recipient of an endpoint reference acts in accordance with the semantic description of the referenced endpoint. As pointed out in [14], a WSMO ontology of endpoint references can be defined. It will be used in the case when communicating SWS need to exchange messages containing WS references.

The WS-Addressing specification defines a family of message information headers that augment messages with some abstract properties such as:

- Destination – the address of the intended receiver of the message;
- Source endpoint – providing reference to the endpoint from which the message originates;
- Reply endpoint – containing a reference to the intended receiver of replies to the current message;
- Fault endpoint – specifies the receiver of faults related to this message;
- Action – a URI that uniquely specifies the semantics of the message. It is recommended that it points to a WSDL input, output or fault message;
- Message id – uniquely identifying the message;
- Relationship – relating current message with other messages.

Since WSMO does not mandate a message representation, WS-Addressing message information headers need not be mapped.

3.5.3 WS-MEX

WS Metadata Exchange\footnote{http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-metadataexchange.pdf} enables exchanging all sorts of metadata associated with a WS in SOAP-based communication. It defines two request-response interactions enabling either requesting for a specified type of metadata, or requesting of all available metadata about a service. Metadata can be represented in different languages which is specified in the Dialect attribute. Metadata available on the Web can be associated to a WS using the wsx:Location element. The wsx:MetadataReference element, when present, can be used to specify a WS-Addressing endpoint reference which identifies a service which will return the metadata upon a simple SOAP request.
WS-MEX can be used for exchange of WSMO descriptions as well. To enable this, a dialect language has to be specified and a mechanism for optional assignment of URIs to WSMO descriptions. Concrete syntax and example can be found in [14].

4 COMPARISON OF WSMO AND OTHER APPROACHES TO SWS INTEGRATION

This section presents two other frameworks for semantic web services – the Web Ontology Language for Services (OWL-S), which builds on Semantic Web technology developed at W3C, and the OASIS ebXML initiative that aims at standardizing B2B interactions. Even though ebXML does not restrict B2B interactions to Web services, they also propose means for web service integration.

4.1 OWL-S

OWL-S is an OWL based upper ontology for services, including sub-ontologies for profiles, processes, and groundings. The ontology is still evolving, and making connections to other development efforts, such as those building ontologies of time and resources. The OWL-S ontology can also serve as a language for describing services, since it provides a standard vocabulary enabling creation of service descriptions.

4.1.1 Overview of OWL-S

The OWL-S ontology is based on four top-level concepts: Service, Service Profile, Service Model and Service Grounding. The Service concept is linked to the other 3 by the properties: presents, describedBy and supports.

Service Class

Each web service is modelled in OWL-S with the help of the Service class. The Service class enables description of three distinct aspects of web service related knowledge: what functionality does the service provide, how the web service can be used and how the client can actually interact with the service. Service functionality is represented using the Service Profile class, and a service can publish different descriptions of its functionality. It is the task of the Process Model to describe how the service can be used. At most one process model can be associated with each service using the describedBy property. The perspective of how to interact with a service is captured by the Service Grounding which provides the details of how to access the service, what communication protocol and message formats are used, as well as how each semantically described input or output of the service can be serialized when exchanging messages with this service.

Service Profile

The OWL-S Profile class specifies three types of information related to the service functionality – service provider information, what is the actual function that the service realizes and other characteristics of the service. The provider properties of the service description enable definition of information pointing to the actual organization providing the service. Service functionality is expressed in terms of the inputs required by the service, the outputs it provides and the actual transformation that is produced. Additional properties enable service profiles to be categorized based on a classification system, rated based on quality of service properties, geographical availability of the service and other custom tailored information.
Process Model

The Process model in OWL-S describes the service functionality and how to interact with the service in order to achieve this functionality. The service is described as a process that can be composed of other processes (services). It is in the Process Model where the functionality description is actually defined (in terms of IOPEs) and then it can be reused by the Service Profile. Consistency between the two definitions is not ensured although it is specified that the IOPEs in the profile are a subset of the ones defined in the model.

OWL-S differentiates between two types of processes- atomic processes which model an interaction comprised of a single message being received and a single message being sent back in response, and composite processes that specify more complex forms of interaction. OWL-S composite processes can make use of other atomic and composite processes whose functionality is combined and described as workflow. Explicit workflow definition constructs for sequential, parallel and conditional execution of constituent processes are supported, as well as a very simple dataflow mechanism.

Grounding

OWL-S Grounding ontology specifies how to map the abstract level (semantic, metadata annotated) SWS to a concrete WS (defined as WSDL). In OWL-S service to grounding relation is 1:n (where atomic services are grounded). The main purpose of OWL-S grounding (as declared in [17]) is to “show how the (abstract) inputs and outputs of an atomic process are to be realized concretely as messages, which carry those inputs and outputs in some transmittable format.” OWL-S grounding relies on WSDL and SOAP for the specification of concrete transmission protocols and details.

Two cases of grounding of OWL-S inputs and outputs to WSDL messages can be distinguished—when the type of the message is an OWL type, and when it has another type. OWL message types either be defined in the WSDL types section, or in a separate document related to the WSDL description using the owl-s-parameter extension. Another WSDL extension defined for the needs of OWL-S Grounding is the owl-s-process attribute that can specify that a WSDL operation is mapped to the referenced OWL-S atomic process.

The WsdlGrounding class in OWL-S contains all grounding related information. It provides properties for specifying the particular WSDL version used, a URI of the WSDL document, the operation within this document to which the atomic process is grounded, access point can be optionally specified as well, and mappings for all inputs and outputs of the atomic process to corresponding WSDL messages or message parts.

4.1.2 Comparison between OWL-S and WSMO

In this section we present a high level comparison between OWL-S and WSMO which is based on [15]. Detailed mapping between these two service ontologies (as well as the logical languages) can be found in [16].

The Service Profile in OWL-S is used to represent both the provider and the requester perspectives on the WS functionality. In WSMO the two perspectives are decoupled so that the requestor objectives are described as goals, while providers describe their service functionalities as web services. Hence the Service Profile has two correspondents in WSMO. While in OWL-S the cardinality of the presents property of a Service is not restricted (zero, one or multiple profiles can be associated with a service),
WSMO prescribes that no more than one capability description should be present for a service. On the other hand, WSMO allows a web service to be linked to many goals (which it presumably fulfils) via \textit{wgMediators}. That can be regarded as corresponding to the multiple OWL-S profiles. The difference in these two approaches to semantic descriptions of service functionality stems from the fact that OWL-S enables classification based discovery, while WSMO sets the way for logically enhanced discovery.

Service profiles in OWL-S can be arranged in a hierarchy and liked to an existing taxonomy. Such taxonomies can be used for category based discovery. Mediators can be used in WSMO to enable such categorization based on existing taxonomy of (predefined, generic) goals. Goal taxonomies can be created with the help of \textit{ggMediators} that define more specific goals as refinements of more generic ones. Service capabilities can then be linked to goals in this taxonomy with the help of \textit{wgMediators}. An advantage of WSMO enabled functionality taxonomies is that they can be entirely heterogeneous, since the hierarchy is not based on a predefined shared ontology of profiles, but rather each goal or capability can use its own domain dictionary. Differences can then be resolved with the help of \textit{ooMediators}.

OWL-S service profiles have associated properties that present contact information, service title and human language description of service functionality, as well as service category. In WSMO such information is encompassed in NFP (based on Dublin Core). Straightforward mapping between \textit{profile:serviceName} and \textit{dc:title}, \textit{profile:textDescription} and \textit{dc:description} can be defined. In addition the DC properties creator, publisher and contributor can be used to represent \textit{profile:contactInformation}. The categorization in WSMO can be enabled through the \textit{dc:type} property. It is an advantage of WSMO that it provides more NFPs and enables such metadata to be associated with all kinds of descriptions (not only of services).

The OWL-S profiles also provide a \textit{serviceParameter} property which enables definition of additional (custom defined) parameters of the profile. In WSMO the set of NFPs is freely extensible (not requiring a designated property).

Functionality in the OWL-S profile is described as information transformation (as inputs and outputs) and state change (through preconditions and effects). The Inputs, Outputs, Preconditions and Effects (IOPEs) are defined in the process model and referred to by the profile. WSMO also describes how the service transforms the information space, but with the help of preconditions and postconditions. As suggested by their names, they specify conditions (in terms of the WSMO logical language) over the inputs and outputs of the service. When a requested service (user goal) is described, only definition of postconditions is feasible, as the user is only interested in the results achieved by the service (and most often cannot “guess” what the expected input is).

The state of the world is also related to the service description both in OWL-S and in WSMO. OWL-S describes the state of the world through preconditions and effects, while WSMO defines assumptions and effects. OWL-S effects are defined as part of a structure called result. The result of a service encompasses a condition for the delivery of this result (\textit{inCondition}), the binding of the output to the appropriate value \textit{(outputBinding)}, the state of the world that will be achieved as result, and the local variables defined in the result (\textit{ResultVariables}). Conditions are specified as expressions in a logical language of choice. WSMO also relates service functionality to the state of
the world – **assumptions** describe necessary conditions for service execution not related to the input, while **effects** describe the state of the world after service execution.

The interaction-related part of the *Service Model* can be mapped to WSMO interface. A difference is that multiple interfaces can be associated with a WSMO service, while OWL-S allows a single model for a service. OWL-S process is comprised of its participants (but so far this is underspecified) while WSMO does not support the notion of participants.

OWL-S atomic processes can be mapped to WSMO service without orchestration and composite services are represented by full-fledged WSMO services (with complete interface definition).

OWL-S and WSMO have similar approaches of grounding semantic web services. The OWL-S grounding specifies how each semantic type of input or output defined in the Service Model can be serialized as XML message and which WSDL operation corresponds to each atomic process in the *Service Model*. The current specification of WSMO Grounding (presented in [18]) also concentrates on mapping data represented in WSMO ontologies to XML (enabling exchanging of messages) and mapping of behavioural service description to WSDL (enabling actual invocation of WSDL operations during WSMO service execution).

### 4.2 ebXML

The Electronic Business XML (ebXML) standard enables enterprises of arbitrary size and location to meet and conduct business in a global, electronic marketplace. Data interchange is based on XML messages. So far, a huge amount of initiatives have been started, to define specialized XML-definitions for data exchange. Amongst them are initiatives like CIDX, Rosetta Net, and UBL. The target of ebXML is to provide a unified, modular, electronic business framework to harmonize the different integration approaches in a single methodology.

Different companies first publish a profile of their capabilities and business interests into a central ebXML Registry. Then, based on the submitted profiles, the ebXML framework allows businesses to search for potential business partners as well as to agree on a process and format for conducting their business and transmitting their data. This process is explained in little more detail in Sect. 4.2.1. The information in these sections is a condensed overview from the following sources: [2] and [http://www.ebxml.org](http://www.ebxml.org).

#### 4.2.1 Technological Foundations and Business Interactions

Fig. 2 shows a sample interaction between two companies in the ebXML framework willing to engage in joint business. To enliven the scenario depicted, certain aspects need to be addressed by specific technologies. While stepping through the sample business partner interaction, this section gives a rough overview over the most important technologies at each point they are needed.
Fig. 2: Sample ebXML interaction between two companies

1. The central access point for an ebXML scenario is provided by the ebXML Registry. The Registry is a central server hosting a variety of data needed by industrial partners to conduct joint business. The first step in the sample business interaction from Fig. 2 thus shows “Company A” querying the contents of the ebXML Registry in order to retrieve particular business details.

2. Based on the results from reviewing the contents of the ebXML Registry, “Company A” builds or buys an own implementation of an ebXML system. This system would be capable of facilitating the ebXML transactions needed for this party’s business interaction requirements.

3. Next, “Company A” publishes its interaction desires and requirements in the Registry. This is done by submitting a Collaboration Protocol Profile (CPP)\(^\text{17}\). The CPP defines the Business Processes and Business Service Interfaces supported by the submitting business partner. A Business Service Interface specifies the way a business partner expects business transactions to occur in order to be handled by its Business Process. The Business Service Interface therefore contains Business Message definitions as well as the protocols used to transport these. Business Messages are used to transport the actual information transmitted during a business transaction. Business Messages are built in a layered way. The Simple Object Access Protocol (SOAP)\(^\text{18}\) is recommended by the ebXML initiative as the message envelope. The outside layer must be an actual communication protocol (like HTTP or SMTP), while other layers could for example deal with encryption and authentication.

\(^{17}\) [http://www.ebxml.org/specs/ebCCP.pdf](http://www.ebxml.org/specs/ebCCP.pdf)

\(^{18}\) [http://www.w3.org/TR/soap/](http://www.w3.org/TR/soap/)
4. Once “Company A” published its CPP in the ebXML Registry, “Company B” could retrieve it from the Registry and decide whether it is able and willing to collaborate within this business interaction.

5. It is intended by the ebXML initiative, that based on the conformance of both parties’ CPPs, a Collaboration Protocol Agreement (CPA)\(^\text{19}\) can automatically be negotiated by agreement protocols defined as ebXML standards. The CPA is used to express a contract between two business parties, who defined their collaboration desires and requirements in CPPs. The CPP is to be seen as capability and intent description. In contrast, the CPA is to be seen as a mean to express commitments on intents.

6. Finally, after agreement on the terms of collaboration engagement, both business partners “Company A” and “Company B” start actual transactions in order to implement the agreed business interaction. The transactions might be based on the Business Message definitions exchanged before as well as on other ebXML related standards.

4.2.2 ebXML and WSMO

While the target of ebXML is to provide a unified framework for B2B interactions based on XML standards for electronic interchange, it does not provide explicit semantics for its CPP or CPA definitions. Since it does not mandate a language for definition of collaboration protocols and profiles, WSML can be used as such language. Hence WSMO service descriptions can be used by businesses using ebXML based solutions. WSMO descriptions will add the missing explicit semantics and enable the automation of tasks related to service discovery, contracting and interaction in the ebXML framework. On the other hand WSMO does not mandate architecture for service operation and B2B integration as the one specified by ebXML. Therefore, WSMO and ebXML are complementary initiatives and further research should be performed to identify the synergies between the two and how they can be exploited.

5 BUSINESS DATA ONTOLOGY AND RELATED ONTOLOGIES

As pointed out in DIP deliverable D3.2 [1], the Semantic Web Service (SWS) Usage Process will be based on a set of ontologies. Each of the steps of this process relies on the information given by specific ontologies.

So far, in the scope of DIP the Business Data Ontology (BDO) has been created (described in more details in DIP deliverable D3.3, [4]). BDO builds the top-level basis for defining more specific domain ontologies to be actually used. It thus supports interoperability (data integration) between the communicating peers.

The structure of BDO basically consists of two layers. On the highest level, the Base Upper-Level Ontology (BULO) builds the backbone of BDO. Where appropriate, the concepts defined in BULO are extended on a lower layer by their adequate counterparts in the Universal Business Language (UBL). Thus, BDO is based on the BULO ontology and the UBL standard (still an OASIS Committee Draft). As currently there are no standards related to business-data ontologies, in the next sections we will discuss the relationships of the BDO with BULO and UBL.

\(^{19}\) \url{http://www.ebxml.org/specs/ebCCP.pdf}, Section 8
5.1 Basic Upper-Level Ontology

The Basic Upper-Level Ontology (BULO\textsuperscript{20}) is being developed in the SEKT\textsuperscript{21} project (EU IST IP 2003-506826). It is derived from an ontology built for the Knowledge & Information Management (KIM [3]) platform, called KIM Ontology (KIMO [5]). Base Upper-Level Ontology provides high-level definitions of general terms; like time, agent and location. Defining these high-level terms is essential for their later consistent usage and reference by extending concepts of derived ontologies.

According to [5], the KIM Ontology was built in order “to provide a minimal but sufficient ontology, suitable for the open-domain, general-purpose semantic annotation”. Although built from scratch, the development of KIMO was amongst others inspired by OpenCyc\textsuperscript{22}, WordNet 1.7\textsuperscript{23}, DOLCE\textsuperscript{24}, and EuroWordnet Top\textsuperscript{25} which are widely used and accepted ontologies. The layout of KIMO was intentionally designed to be compliant with Dublin Core\textsuperscript{26}, the ACE annotation types\textsuperscript{27}, and the ADL Feature Type Thesaurus\textsuperscript{28}.

The fact that the BDO is based on BULO ensures that domain specific ontologies (extending BDO) will use widely accepted vocabulary. Thus, the time for development of domain specific ontologies is reduced (because the general concepts are already defined) while the potential for interoperability with other ontologies and schemata is increased.

5.2 Universal Business Language

The UBL-FAQ\textsuperscript{29} describes UBL as follows:

“UBL is the first true standards body implementation of the ebXML Core Components Technical Specification (CCTS 2.01, aka ISO/TS 15000-5). The UBL library consists of ebXML CCTS Business Information Entities (BIEs). UBL XML schemas are defined through the application of UBL Naming and Design Rules (NDRs) to an underlying data model mapped to the Core Component types.”

More than just providing a data model for \textit{statically} describing data integration, ebXML also contains a methodology for \textit{dynamically} mapping different communicating peers’ data models on each other. An overview over the ebXML framework was already given

\textsuperscript{20} The latest version of BULO is called PROTON, more information can be found at http://proton.semanticweb.org
\textsuperscript{21} http://www.sekt-project.com/
\textsuperscript{22} http://www.opencyc.org/
\textsuperscript{23} http://wordnet.princeton.edu/wn1.7.shtml
\textsuperscript{24} http://www.loa-cnr.it/DOLCE
\textsuperscript{25} http://www.illc.uva.nl/EuroWordNet/corebcs/topont.html
\textsuperscript{26} http://dublincore.org/
\textsuperscript{27} http://www.itl.nist.gov/iad/894.01/tests/ace/
\textsuperscript{28} http://www.alexandria.ucsb.edu/gazetteer/FeatureTypes/ver070302/
\textsuperscript{29} http://www.oasis-open.org/committees/ubl/faq.php
in section 4.2. Here, we take a little closer look at the CCTS which acts as the enabling data structure of ebXML, and which was also used to design UBL.

The Core Components Technical Specification (CCTS) aims at providing the technical basis to facilitate data interchange between different business partners. In contrast to existing standards, this approach conducts standardization in a syntax-independent manner. Two business partners using different syntaxes are ensured to talk about the same Business Semantics as far as their syntaxes both ground on the same abstractly defined Core Components (CC).

Core Components are the smallest elements to build larger electronic Business Messages.

**Definition of Core Component:**

“A building block for the creation of a semantically correct and meaningful information exchange package. It contains only the information pieces necessary to describe a specific concept.” [6]

The CCTS defines the four following types of Core Components. Their interrelation is depicted on the left-hand side of Fig. 3.

- **Aggregate Core Components** combine related pieces of business information. They are built by respectively pooling BCCs and ASCCs (see below).
- **Basic Core Components (BCC)** are singular CCs that therefore have a specific Data Type defining their respective set of values. Data Types are defined as restrictions on Core Component Types (see below).
- In contrast to BCCs, an **Association Core Component (ASCC)** entails a complex structure described by an Aggregate Core Component.
- **Core Component Types (CCT)** consist of exactly one Content Component and one or more Supplementary Components. While the Content Component defines an actual content (like a value of “12”), a Supplementary Component is used to specify essential extra definitions for the content (like a monetary unit of “EUR”).
Based on the Core Components, the CCTS defines the so-called Business Information Entities (BIE). The key differentiator to Core Components is their relation to a specific “Business Context”. In contrast to the CCs, which can uniquely be used across Business Scenarios, BIEs reflect the specialties of their assigned Business Context. Thus, CCs define a Controlled Vocabulary which is able to interconnect different business domains.

Definition of Business Information Entity

“A piece of business data or a group of pieces of business data with a unique Business Semantic definition. A Business Information Entity can be a Basic
Business Information Entity (BBIE), an Association Business Information Entity (ASBIE), or an Aggregate Business Information Entity (ABIE).” [6]

In analogy to the four different kinds of Core Components, the CCTS defines three types of BIEs which essentially replicate the purpose of their respective CCs, but which are enriched by a Business Context (Fig. 3, right-hand side).

As stated in the quote above, for building up the UBL library, the BIE structure is applied to the underlying data model of UBL containing the actual business terms. Again, for the definition of these business terms, UBL utilizes the naming convention introduced by the CCTS. The specification defines a consistent schema to name all Core Components, Business Information Entities, and Data Types. By applying these naming rules, as well as the specified BIE structure, derived extensions of UBL would be able to conform with the ebXML framework and therefore be used within a ebXML business interaction scenario as described in Sect. 4.2.1.

As stated before, BDO will be the base for building domain ontologies. Incorporating UBL into BDO aims at empowering users to align their personal business domain ontologies with those of other parties compliant to the ebXML methodology.

6 Conclusion

In the course of this document we have outlined how WSMO can be linked to existing WS standards in order to promote its acceptance and positioning as the DIP service description framework realization. An overview of the WSMO description framework was presented and then its position with regard to the WS Architecture Protocol Stack was identified. We have outlined the conceptual ways in which WSMO can be linked and make use of WSDL, UDDI and BPEL4WS. WSMO was also compared to other frameworks for semantic web services. In addition we have explained the relations between the Business Data Ontology created as an extension of the DIP service definition framework and other ontologies providing conceptualizations of business data descriptions.
REFERENCES


