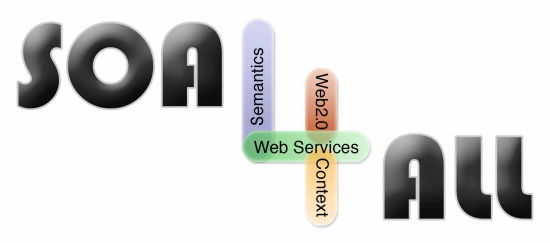


Project Number: **215219**

Project Acronym: **SOA4ALL**

Project Title: **Service Oriented Architectures for All**

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D1.2.1 WSMO grounding in SAWSDL

Activity N:	1 Fundamental and Integration	
Work Package:	1 Service Web Architecture	
Due Date:		M6
Submission Date:		27/08/2008
Start Date of Project:		01/03/2006
Duration of Project:		36 Months
Organisation Responsible of Deliverable:		UIBK
Revision:		1.0
Author(s):	Jacek Kopecký, Adi Schütz	UIBK

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	2008/07/16	Initial version	Jacek Kopecký
0.2	2008/07/24	Filling in more	Jacek Kopecký, Adi Schütz
0.9	2008/07/25	A complete draft, ready for review	Jacek Kopecký, Adi Schütz
0.95	2008/07/19	Review comments incorporated	Elisabetta Di Nitto, Marin Dimitrov, Jacek Kopecký
1.0	2008/07/23	Version to be submitted	Jacek Kopecký

Table of Contents

Glossary of Acronyms

Acronym	Definition
D	Deliverable
EC	European Commission
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
HTTPS	Secure HTTP
MEP	Message Exchange Pattern
OWL	Web Ontology Language
RDF	Resource Description Framework
SAWSDL	Semantic Annotations for WSDL and XML Schema
SEE	Semantic Execution Environment
SOAP	*not an acronym, used to mean Simple Object Access Protocol*
SPARQL	SPARQL Query Language for RDF
URI	Uniform Resource Identifier
WP	Work Package
WSDL	Web Services Description Language
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
WSMX	Web Service (Modeling) Execution Environment
XML	Extensible Markup Language

Executive summary

WSMO is a framework for semantic descriptions of Web services, independent from underlying communication technologies. WSDL is the standard language for describing the syntactical and networking aspects of Web services. Grounding is the mechanism that ties service descriptions in WSMO and WSDL together, so that a semantic tool that processes WSMO descriptions can also invoke the services, as prescribed in WSDL.

In its set of specifications, WSMO provides a direct grounding to WSDL, putting the grounding information in the description of a service choreography. In this deliverable, we provide an alternate grounding mechanism that puts the grounding information in the WSDL document, using SAWSDL, a standard for semantic annotations of WSDL. This way, the low-level infrastructure that deals with WSDL has a more direct link to the semantic description in WSMO.

Also, this deliverable describes a simple way to use a new proposed language, XSPARQL, to describe the data grounding, a problem which was up to now unresolved in WSMO. Beside this manual approach to data grounding, we also discuss a GUI tool which will be developed within Deliverable 1.2.2.

The grounding defined in this deliverable will be used in SOA4ALL WSMO-based service descriptions present in the distributed service bus (Task 1.4) and in test collections built on the SOA4ALL testbeds developed in Task 1.5.

1. Introduction

Web services are, in general, systems that provide certain functionality to their clients and communicate with them by exchanging XML data over computer networks. In order to interoperate successfully, the client and the service must agree on what kinds of messages can be exchanged and when, and what exactly the message exchanges will mean. In other words, the client and the service must agree on a common interface.

In most cases, the service provider decides what the interface will look like, and the client has to understand the interface and comply with it. The interface must specify at least the following components:

- **message types** – what types of messages can be exchanged,
- **message serialization** – how information is serialized into bits and bytes ready for network exchange,
- **message exchange** – who can send messages to whom and when,
- **party obligations** – what are the responsibilities of the parties involved in the message exchange.

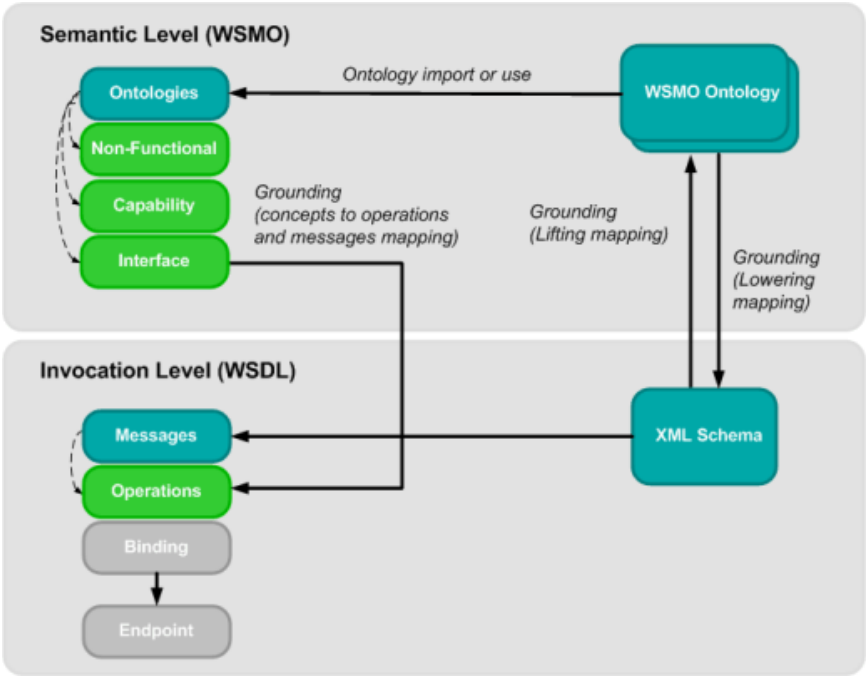
In the area of Web services, XML Schema [XMLSchema] and Web Services Description Language [WSDL] are used together to describe the different aspects of the interface. In particular, XML Schema constrains the message types in XML tree-based structure and WSDL specifies simple message exchanges (operations) and provides the message serialization details. WSDL does not formally specify the ordering and meaning of the operations; this is usually either described by plain text documentation or even only implied by operation names instead. This is sufficient for human operators to create clients that can correctly use the Web services.

To enable automatic discovery and invocation of Web services, the Web Services Modeling Ontology (WSMO) describes functional and behavioural aspects of Web services using an approach based on logics and knowledge representation techniques. First, to enable Web service discovery and composition, WSMO focuses on describing what Web services do using service *capabilities*. Second, to make it possible for clients to determine how to communicate with discovered services, WSMO specifies the *interface* of a Web service using the service *choreography*.

A WSMO choreography is a state machine, with its states described using ontologies, in terms of concepts, their instances and the relations between them. Inputs and outputs of the Web service are represented as instances of certain concepts that can be read or written by the client communicating with the service. The purpose of WSMO grounding is to describe the exact mechanism, using which the client writes or reads the accessible instances.

The state of the art for Web service interface description is WSDL 2.0. It is thus important that infrastructure for Semantic Web services is able to communicate with existing Web services described with WSDL. In the context of WSMO-compliant execution environments, only Web services with a WSMO description are available for the operations of discovery, composition, invocation, etc. Therefore, it is necessary that a service has both WSMO and WSDL descriptions and that they are linked. In WSMO, grounding is this link. The anatomy of the complete service description together with links between WSMO and WSDL is shown in Figure 1, adopted from the original WSMO grounding specification.

Figure 1: Semantic Service Anatomy



In this figure, we distinguish two modelling levels of the service description, namely semantic level and invocation level.

- **Semantic Level** represents the semantic model for services used in various stages of execution process run on middleware. For this purpose, we use the WSMO service and ontology model. WSMO defines service semantics including non-functional properties, functional properties (capability description) and interfaces (behavioural definition) as well as ontologies that define the information models on which the services operate.
- **Invocation Level** describes the physical environment used for service invocation. In our architecture, we use the Web Services Description Language WSDL.

In order to perform invocation of Web services, we must define grounding from semantic descriptions to the underlying WSDL and XML Schema definitions, especially including data transformations between XML (used by the messages of the Web service) and the corresponding semantic data. Definition of such grounding can be placed in the WSMO descriptions (as defined in [WSMOD24.2]) at the WSMO service interface level, or the grounding can be in the WSDL descriptions as semantic annotations.

The standard WSDL 2.0 language has an extension called Semantic Annotations for WSDL and XML Schema (SAWSDL). This specification defines simple hooks for attaching semantics and data transformations to Web service descriptions. In this deliverable, we show how the SAWSDL hooks can be used to implement WSMO grounding.

1.1 Scope and Structure of this Document

The purpose of this deliverable is to describe how WSMO service descriptions can be grounded to WSDL, using SAWSDL annotations. There are two aspects to this problem:

- **Choreography Grounding:** WSMO choreography specifies that the client can access some choreography state data (and thus it can communicate with the Web service), but in WSDL, the client and the service send the data to each other in the form of messages. The grounding must describe what messages are supposed to be sent by the client, and when the client should expect messages from the service. Above that, the grounding must also provide the necessary serialization and networking details, i.e. what underlying protocol (e.g. SOAP, HTTP) should be used for passing the messages, how the XML data is encapsulated in the underlying protocol, and where exactly the data should be sent. Choreography grounding is described in Section 3.
- **Data Grounding:** while the data model of the input and output messages for WSDL services is defined using XML Schemas, the data model for a WSMO service is defined using the conceptual model provided by WSMO ontologies. This leads to the need to map between the ontological data in the state machine and its representation as XML messages (lowering from ontology instances to XML, lifting from XML to ontology instances). Data grounding is discussed in Section 4.

First, however, Section 2 presents the underlying technologies – WSMO, WSDL, SAWSDL and XSPARQL.

2. Underlying Specifications

In order to describe WSMO grounding in SAWSDL, we need to first review the underlying specifications; namely WSMO, WSDL, SAWSDL and XSPARQL.

2.1 WSMO

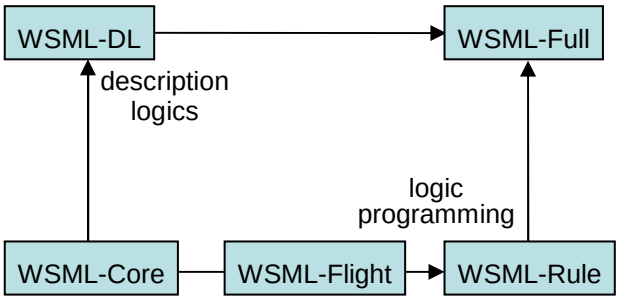
The Web Service Modeling Ontology [WSMO] is a conceptual model for describing the semantics of Web services and related entities, for the purpose of automating some aspects of service discovery and usage. WSMO is complemented by the Web Service Modeling Language [WSML], a concrete language that implements the conceptual model and fleshes out the details. In the following, all statements about WSMO also apply to WSML.

WSMO has four main building blocks:

- *Ontologies* for knowledge representation.
- *Web services* represent the semantics of services.
- *Goals* represent user requests that can be satisfied with services.
- *Mediators* represent components that bridge any heterogeneities.

Ontologies and ontological instances (data) are used in all the other parts of WSMO. For representing ontologies with varying levels of expressivity and reasoning complexity, WSML provides several *fragments*: WSML-Core allows basic modeling supported by most knowledge representation technologies. WSML-Flight and WSML-Rule extend WSML-Core with techniques of Logic Programming for advanced reasoning with axioms and rules. WSML-DL extends WSML-Core with modeling constructs coming from Description Logics. And finally, WSML-Full unifies both branches (logic programming and description logics) and thus provides the most expressive language. The following figure illustrates the relationships between the various fragments of WSML:

Figure 2. WSML Ontology Language Fragments



Mediators are used where different descriptions express similar meaning differently. For instance, ontology mediators can be used to import OWL ontologies into WSML, or to map between different, yet semantically overlapping terminologies.

Finally, goals and Web services describe what users want and what Web services provide. In WSMO, descriptions of goals and Web services have the same structure, therefore in the following, we will only talk about Web service descriptions, in terms of what a service *provides*, and the reader may also read it in terms of what a client *requests*.

A Web service description captures the *functional semantics* (the *capability*) of a Web service, i.e. what the service does, and the *behavioral semantics* (the *interface*), i.e. how the

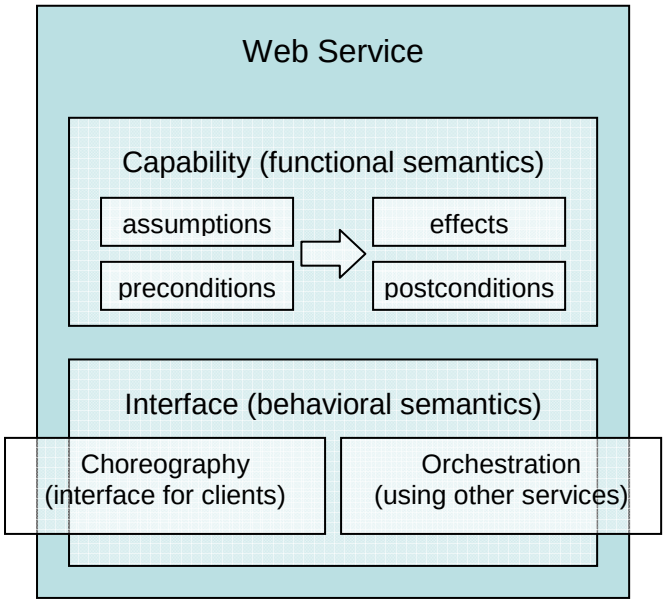
service communicates with other parties.

Functional semantics are expressed with a *capability* construct that specifies the preconditions and assumptions that must be valid before the service can be used, and the postconditions and effects which are expected to be valid after the service is successfully invoked.

The interface of a Web service (the behavioral semantics) has two aspects: external behavior called *choreography*, i.e. how the clients talk to the service, and internal behavior called *orchestration*, i.e. how the service uses other services to implement its functionality.

The following figure illustrates the structure of WSMO Web service (and goal) descriptions:

Figure 3. Structure of WSMO Web Service Description



A WSMO choreography is a state machine, with its states described using ontologies, in terms of concepts, their instances and the relations between them. Inputs and outputs of the Web service are represented as instances of certain concepts that can be read or written by the client communicating with the service. Each concept in the choreography state ontology can be assigned to a role which determines whether the clients may read or update (or both) the values of instances of that concept. We will call these concepts and instances *accessible*.

The accessible concepts must be available to the client using some underlying messaging protocol. Therefore, a choreography definition includes *grounding*, which defines the protocol for reading and writing of the accessible concepts by the clients; in other words, the grounding specifies how the client may communicate with the service. The basic grounding is specified in [WSMOD24.2].

2.2 WSDL

The Web Services Description Language [WSDL] describes Web services in two levels – an XML-based reusable abstract interface and the concrete details regarding how and where this interface can be accessed. All descriptions in WSDL are centered on the Web service and all terminology follows the service's point of view, for example input messages are messages coming into the service from the network and output messages are messages

generated by the service and sent to the network. The first three subsection below talk about various aspects of WSDL descriptions, based on WSDL version 2.0, namely about abstract Web service interfaces (Section 2.1.1), binding them to concrete wire protocols and endpoints (Section 2.1.2) and finally about the overall organization of WSDL documents (Section 2.1.3). The fourth subsection details the relevant differences to the older version, WSDL 1.1.

2.2.1 Web Service Interface

On the abstract level, a Web service interface is described in terms of data schemas and simple message exchanges. In particular, WSDL models interfaces as sets of related operations, each consisting of one or more messages. For example an interface of a ticket booking Web service can have operations for querying for a trip price and for the actual ticket booking:

Listing 1. Illustrative example of a WSDL interface

```
01 <interface name="BookTicketInterface">
02   <operation name="queryPrice" pattern="http://www.w3.org/ns/wsdl/in-out">
03     <input element="tns:TripSpecification"/>
04     <output element="tns:PriceQuote"/>
05     <outfault ref="tns:TripNotPossible"/>
06   </operation>
07   <operation name="bookTicket" pattern="http://www.w3.org/ns/wsdl/in-out">
08     <input element="tns:BookingRequest"/>
09     <output element="tns:Reservation"/>
10     <outfault ref="tns:CreditCardNotValid"/>
11     <outfault ref="tns:TripNotPossible"/>
12   </operation>
13   <fault name="TripNotPossible" element="tns:TripFailureDetail" />
14   <fault name="CreditCardNotValid" element="tns:CreditCardInvalidityDetail" />
15 </interface>
```

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

To describe their content, particular messages (incoming, outgoing) in an operation reference XML Schema element declarations. Fault messages, however, reference faults defined on the interface level (see above the <outfault> element), with the intention that semantically equivalent faults can be shared by different operations. Additionally, there may be multiple fault references for the same MEP fault message – in effect WSDL faults are typed and one operation can declare that it can result in any number of alternative faults (apart from the single success message).

2.2.2 Web Service Endpoints, Bindings

In order to communicate with a Web service described by an abstract interface, a client must know how the XML messages are serialized on the network and where exactly they should be sent. In WSDL, on-the-wire message serialization is described in a binding and then a service construct enumerates a number of concrete endpoint addresses.

A binding generally follows the structure of an interface and specifies the necessary serialization details. The WSDL specification contains two predefined binding specifications, one for SOAP (over HTTP) and one for plain HTTP. These bindings specify how an abstract XML message is embedded inside a SOAP message envelope or in an HTTP message, and how the message exchange patterns are realized in SOAP or HTTP. Due to extensive use of defaults, simple bindings only need to specify very few parameters, as in the example below. A notable exception to defaulting in binding are faults, as in SOAP every fault must have a so called fault code with two main options, Sender or Receiver, indicating who seems to have a problem. There is no reasonable default possible for the fault code.

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

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

Listing 2. Example of a WSDL binding and service

```
<binding
      name="SOAPTicketBooking"
      interface="tns:BookTicketInterface"
      type="http://www.w3.org/ns/wsdl/soap"
      wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/" >
  <fault ref="TripNotPossible" wsoap:code="soap:Receiver"/>
  <fault ref="CreditCardNotValid" wsoap:code="soap:Sender"/>
</binding>

<service
      name="DERITicketBooking"
      interface="tns:BookTicketInterface">
  <endpoint
        name="normal"
        binding="tns:SOAPTicketBooking"
        address="http://deri.example.org/tickets" />
</service>
```

2.2.3 WSDL Documents

Apart from the interfaces, bindings and services described above, WSDL documents can contain further elements, enclosed in the root <description> element.

In order to facilitate true reuse of interfaces or bindings, WSDL documents can be modularized by using include and import mechanisms. When a WSDL document is parsed,

imports and includes are resolved so the resulting model is not aware that some pieces may have come from different actual files.

As a container for data type information, WSDL documents have a section called <types>. Actual schemas can either be embedded directly in this section or referred to using the appropriate import statements, for example external XML Schema documents can be imported by putting the <xs:import> element directly in the <types> section. By default, WSDL uses XML Schema to describe data, but WSDL extensibility allows other data type systems to be used instead.

Finally, every element in a WSDL document can be annotated with documentation elements or it can contain extensibility elements or attributes.

2.2.4 Note on the differences between WSDL 2.0 and WSDL 1.1

This note details the differences between WSDL version 1.1 [WSDL11], a specification authored by several companies and submitted to the W3C as the basis for standardization work, and WSDL version 2, the resulting draft standard. While this document uses the cleaner version 2 of WSDL, actual deployment prefers WSDL 1.1 because WSDL 2 is not yet widely implemented. This note aims to limit any confusion stemming from the situation that some readers may only be familiar with WSDL 1.1.

The first notable difference is that several constructs from WSDL 1.1 were renamed in WSDL 2. In particular, portType in WSDL 1.1 is known as interface in WSDL 2 and port in WSDL 1.1 (occurring within a service) is now known as endpoint. Also, the WSDL document root element is called definitions in WSDL 1.1 and description in WSDL 2. Importantly, the intention of all these renamed constructs is unchanged between the two WSDL versions.

A larger difference is that while WSDL 2 uses XML Schema element declarations to describe messages, WSDL 1.1 had a special construct, message, that contained potentially several parts, each referencing a single XML Schema element or type declaration. However, the use of multiple parts in a single message is usually translatable to a single element containing a sequence of elements (one for each part), making the different approaches in WSDL 1.1 and in WSDL 2 equivalent for all practical purposes.

2.3 SAWSDL

Semantic Annotations in WSDL and XML Schema [SAWSDL] is a W3C specification that defines how to add semantic annotations to Web Service Description Language and to XML Schema [XMLSchema]. It defines extension attributes that can be applied to elements in both WSDL and XML Schema in order to annotate WSDL interfaces, operations and their input and output messages. SAWSDL is the first step towards standardization in the area of Semantic Web Services.

Semantic annotations in WSDL and XML Schema are used for these purposes:

- associating WSDL interfaces with some taxonomical categories to help semantic Web service discovery,
- describing the purpose or applicability of WSDL operations to help discovery or composition,
- linking and mapping inputs, outputs and faults of WSDL operations to semantic concepts to help facilitate mediation and service discovery and composition.

While the semantic annotations are used to point to taxonomies, ontologies or mappings, SAWSDL is independent of any particular ontology language or mapping language. The mechanism only requires that the concepts in the semantic models can be identified with

URIs.

SAWSDL can be split in two parts: semantic model references from elements in WSDL or XML Schema to concepts in a semantic model (usually an ontology or taxonomy), and data mappings between XML and semantic models. These two parts are described in the following two subsections.

2.3.1 Model references

The first major part of SAWSDL is an attribute called modelReference. The value of the attribute is a list of URIs that reference concepts in a semantic model. SAWSDL defines how model references can be used on WSDL interfaces, operations, faults, and on XML Schema element declarations or type definitions.

On a WSDL interface, a model reference can provide a classification of the interface, for example by pointing into a products and services taxonomy like eCl@ss (eCl@ss Standardized Material and Service Classification, http://eclass-online.com/).

Model references on a WSDL operation define what the operation does. This can be done with a direct reference to a verb concept or to a logical axiom or by specifying the operation's preconditions and effects. Known techniques like planning (automatic composition) can then use this information. On a WSDL fault, model references define what kind of failure the fault means, so that the fault can be handled more appropriately by the client.

Model references on XML Schema element declarations and type definitions define the semantics of the inputs or outputs of WSDL operations. These annotations can, for example, complement the preconditions and effects from the operation for the purpose of planning, or the types can be used to verify type correctness of compositions.

In general, model references can have many uses, and indeed, SAWSDL does not limit the applicability of the attribute.

2.3.2 Schema mappings

Schema mappings transform between XML data described with XML Schema and semantic data described by a semantic model. Mappings can be used for example to support invocation of a Web service from a client that works natively with semantic data.

SAWSDL defines two extension attributes – liftingSchemaMapping and loweringSchemaMapping. These attributes are used to point from a schema element declaration or type definition to a mapping that specifies (in any suitable mapping language, e.g. XSPARQL) how data is transformed from XML to the semantic level (*lifting*) or back (*lowering*).

A lifting schema mapping defines how an XML instance document conforming to the element declaration or type definition specified in a schema is transformed to data that conforms to a semantic model. The input to the transformation is an XML element that represents a Web service message and the output will be semantic data (for example an RDF graph).

Similarly, a lowering schema mapping defines how data in a semantic model is transformed to XML instance data. The input is some semantic data and the output will be an XML element that forms a Web service message.

Lifting and lowering schema mapping annotations can only be specified on global (top-level) XML Schema element declarations and type definitions. This is because the input of a lifting mapping and the output of a lowering mapping always form the whole Web service message, therefore only global (top-level) XML Schema can be used to validate it.

2.4 XSPARQL

There are two machine-oriented data representation technologies standardized specifically

for the Web: the Extensible Markup Language XML and the Resource Description Framework RDF. XML is very popular as a data exchange format, because its hierarchical structure maps very well into programming language structures and database records; on the other hand, RDF is a less constrained graph data model designed for combining and querying freely data from diverse sources.

XML and RDF have coexisted for nearly a decade now, and until recently, there has been a gap between XML and RDF because there were no tools that could gracefully handle transformations between the two technologies. The standardization of the RDF query language SPARQL and the XML query language XQuery has spurred the development of XSPARQL [XSPARQL], a combination of these two query languages that natively supports both XML and RDF, and thus enables relatively easy transformations between the two data formats.

XQuery is a functional programming language for querying (through XPath expressions) and creating XML documents. A single query can reach into multiple documents and produce a single XML document as its output. Due to the important role of literal data (numbers, strings etc.) in XML, XQuery has a powerful operator and function library for manipulating such data.

SPARQL is a declarative query language for RDF data; a single query can combine multiple RDF data sources and produce either a single yes/no answer, a list of variable bindings, or a new RDF graph. SPARQL provides only limited means for manipulating literal data, which is often seen as a drawback.

XSPARQL can process inputs in XML (using XPath expressions) and in RDF (using SPARQL graph patterns). Literal data from RDF is converted into the XQuery/XPath data model, which allows it to be subjected to the full power of the XQuery operator and function library. As the output, an XSPARQL query can produce either an XML document or an RDF graph.

In summary, XSPARQL is especially suitable for transforming between XML and RDF (as discussed in Section 4) and for combining XML and RDF inputs. More detail on the XSPARQL language can be found in the technical report at http://xsparql.deri.ie, and this deliverable contains example XSPARQL queries for transformations between XML and RDF in Section 4.

3. SAWSDL Grounding of WSMO Choreographies

For WSMO grounding, we need to connect the accessible concepts from the Web service choreography description with the appropriate WSDL messages so that we know how to transfer instances of the accessible concepts. When a client system follows the choreography of a Web service, the choreography dictates when certain data can be sent or received, and the grounding specifies how exactly the it can be sent or received. In grounding to WSDL, sending means that the client will form an input message of a Web service operation, and receiving means the client will receive an output message from an operation.

SAWSDL supports, among others, model references to semantic concepts from XML Schema element declarations and type definitions. To ground a WSMO choreography, we put model references on the element declarations that are inputs or outputs of WSDL operations. In particular, an element that is an input message to a WSDL operation should contain a model reference to an "in" or "shared" concept in a WSMO choreography, and an output message element should have a model reference to an "out" or "shared" concept. Listing 3 below contains the heading of a WSMO choreography description, adapted from an example in [WSMOD24.2], indicating the "in" and "out" concepts on lines 25 through 30. The service has reservationRequest as its input, and reservation or negativeAcknowledgement as output.

Listing 3. Example header of a choreography description, without grounding.

```
01  namespace {_"http://example.org/bookTicket#",
02     tr     _"http://example.org/tripReservationOntology#",
03     wsml   _"http://www.wsmo.org/wsml/wsml-syntax#",
04     po     _"http:/example.org/purchaseOntology#"
05    }
06
07  ontology _"http://example.org/BookTicketInterfaceOntology#"
08
09    importsOntology { _"http://www.example.org/tripReservationOntology",
10                      _"http://www.wsmo.org/ontologies/purchaseOntology"
11     }
12
13    concept reservationRequest subConceptOf tr#reservationRequest
14    concept reservation subConceptOf tr#reservation
15    concept temporaryReservation subConceptOf tr#reservation
16    concept creditCard subConceptOf po#creditCard
17    concept negativeAcknowledgement
18
19  webService _"http://example.org/BookTicketService#"
20
21    interface BookTicketInterface
22      choreography BookTicketChoreography
23        stateSignature
24          importsOntology _"http://example.org/BookTicketInterfaceOntology#"
25          in
26            reservationRequest
28          out
29            reservation
30            negativeAcknowledgement
31          controlled
32            temporaryReservation
33        transitionRules
```

```
34        [...]
```

The following Listing 4 shows the WSDL description with the SAWSDL grounding links (lines 10, 15, 20 and 25), equivalent to the grounding information in the example in [WSMOD24.2]. In particular, the element BookingRequest represents the reservationRequest input message, the element Reservation represents the successful reservation output message, and the two fault elements both represent a negativeAcknowledgement output message. Additionally, note the liftingSchemaMapping and loweringSchemaMapping attributes on lines 11, 16, 21 and 26, indicating pointers to the data grounding transformations (further described in Section 4).

On top of linking the inputs and outputs to the accessible ("in", "out" and "shared") concepts in the choreography description, we also need to link the WSDL service to a WSMO webService, because the concept links may not be sufficient to identify the webService in case multiple service choreography descriptions use the same accessible concepts. To link the WSDL service to the WSMO webService (Listing 3, line 19), we attach a SAWSDL model reference to the WSDL service component, as also shown in Listing 4 on line 55.

Listing 4. WSDL description of the service, with SAWSDL grounding information

```
01 <description xmlns="http://www.w3.org/ns/wsdl"
02           targetNamespace="http://example.com/"
03           xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
04           xmlns:tns="http://example.com/">
05
06  <types>
07    <xs:schema targetNamespace="http://example.com/"
08          xmlns:xs="http://www.w3.org/2001/XMLSchema" >
09      <xs:element name="BookingRequest"
10            sawsdl:modelReference="http://example.org/bookTicket#reservationRequest"
11            sawsdl:loweringSchemaMapping="http://example.org/BookingRequestLowering.xsp" >
12        [...]
13      </xs:element>
14      <xs:element name="Reservation"
15            sawsdl:modelReference="http://example.org/bookTicket#reservation"
16            sawsdl:liftingSchemaMapping="http://example.org/ReservationLifting.xsp" >
17        [...]
18      </xs:element>
19      <xs:element name="TripFailureDetail"
20            sawsdl:modelReference="http://example.org/bookTicket#negativeAcknowledgement"
21            sawsdl:liftingSchemaMapping="http://example.org/BookingFailureLifting.xsp" >
22        [...]
23      </xs:element>
24      <xs:element name="CreditCardInvalidityDetail"
25            sawsdl:modelReference="http://example.org/bookTicket#negativeAcknowledgement"
26            sawsdl:liftingSchemaMapping="http://example.org/BookingFailureLifting.xsp" >
27        [...]
28      </xs:element>
29    </xs:schema>
30  </types>
31
32  <interface name="BookTicketInterface">
33    <operation name="bookTicket" pattern="http://www.w3.org/ns/wsdl/in-out">
34      <input element="tns:BookingRequest"/>
```

```
35      <output element="tns:Reservation"/>
36      <outfault ref="tns:CreditCardNotValid"/>
37      <outfault ref="tns:TripNotPossible"/>
38    </operation>
39    <fault name="TripNotPossible" element="tns:TripFailureDetail" />
40    <fault name="CreditCardNotValid" element="tns:CreditCardInvalidityDetail" />
41  </interface>
42
43  <binding
44          name="SOAPTicketBooking"
45          interface="tns:BookTicketInterface"
46          type="http://www.w3.org/ns/wsdl/soap"
47          wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/" >
48      <fault ref="TripNotPossible" wsoap:code="soap:Receiver"/>
49      <fault ref="CreditCardNotValid" wsoap:code="soap:Sender"/>
50  </binding>
51
52  <service
53          name="DERITicketBooking"
54          interface="tns:BookTicketInterface">
55          sawsdl:modelReference="http://example.org/BookTicketService#" >
56      <endpoint
57              name="normal"
58              binding="tns:SOAPTicketBooking"
59              address="http://deri.example.org/tickets" />
60    </service>
61  </description>
```

Note: the SAWSDL specification [SAWSDL] does not describe the use of model references on WSDL service components, as shown in the listing above on line 52. However, the specification allows semantic annotations to be used on all WSDL components; it is simply out of scope of SAWSDL to define what model references on WSDL services mean. Since WSMO has the top-level concept of webService, it needs to be grounded in a WSDL service and a SAWSDL model reference is a mechanism for doing it. Nevertheless, we plan to refactor WSMO according to the structure of SAWSDL and WSDL, and we expect that such refactoring will only need to use SAWSDL to the extent to which it is described in the specification.

The following list specifies a set of rules how a Web service designer can create a SAWSDL-based WSMO grounding between already existing WSMO and WSDL descriptions.

- For every operation input message (or input fault) in the WSDL service description, its element declaration or type definition in the XML Schema must be annotated with a modelReference to the appropriate “in” or “shared” concept(s) in the WSMO choreography, and with a loweringSchemaMapping pointing to the appropriate data lowering transformation.
- For every operation output message (or output fault) in the WSDL, its element declaration or type definition must be annotated with a modelReference to the appropriate “out” or “shared” concept(s) in the WSMO choreography, and with a liftingSchemaMapping pointing to the appropriate data lifting transformation.
- The WSDL service must be annotated with a pointer to the appropriate WSMO webService.

3.1 Comparison with WSMO-based grounding

The SAWSDL-based grounding for WSMO, as presented in this section, is a standards-compliant alternative to the grounding from [WSMOD24.2], which we refer to as "WSMO-based grounding". The table below shows the correspondences between the two grounding approaches. It is only intended for readers who are already familiar with [WSMOD24.2].

Table 1. Correspondences between the two alternative grounding approaches.

Grounding link	WSMO-based grounding	SAWSDL grounding
Linking accessible choreography concepts to messages that transfer their instances	*withGrounding* property in WSMO choreography state signature, the value is the WSDL operation message reference component identifier	SAWSDL model reference on the XML Schema element declaration which represents the accessible concept, the value of the reference is the concept identifier
Providing networking and addressing details for message communication	*endpointDescription* non-functional property pointing from a WSMO webService to the appropriate WSDL service, the value is the WSDL service component identifier	SAWSDL model reference on the WSDL service, the value is the identifier of the WSMO webService
Data grounding – links to lifting and lowering transformations of any kind	unspecified	SAWSDL schema mapping annotations

The link between a WSMO webService and a WSDL service is simply reversed between the two grounding forms.

The links grounding accessible choreography concepts are slightly changed, however. The purpose of grounding an accessible choreography concept is to provide information on how the client may submit the service's "in" concepts, and how the client may read the service's "out" concepts. In the WSMO-based grounding, it is sufficient to point from the accessible concept to a WSDL message that carries instances of the concept, because data grounding should take care of the transformations between the message contents and the semantic data. When using SAWSDL, we could also annotate a WSDL message with pointers to the concepts that it carries, but we increase the reusability of the semantic annotations by putting them directly on the data that corresponds to the accessible concepts.

Using SAWSDL for grounding, as presented in this deliverable, brings both benefits and drawbacks over the WSMO-based grounding, therefore we specify both approaches as alternatives. The following list describes the benefits and drawbacks that we are aware of:

- (-) SAWSDL-based grounding links are in a WSDL description, however a WSMO semantic execution environment (SEE, e.g. the WSMX) is primarily based on WSMO. With the grounding in WSMO, the links are readily available whenever the SEE needs them. With SAWSDL-based grounding, the grounding information needs to be looked

up by looking through all the known WSDL descriptions. This drawback can be mitigated heavily with optimizations.

- (-) SAWSDL annotations are in the WSDL document, which in some environments may be generated automatically, e.g. from the service implementation. Extending the WSDL document with SAWSDL annotations may not be a viable option in such settings.
- (+) The SAWSDL-based grounding highlights the relation of WSMO to Web services standards, making it easier for Web service users to grasp and make use of semantic descriptions.
- (+) The SAWSDL-based grounding allows for partial understanding of the semantic description. For instance, the links from XML Schema element declarations to choreography concepts are not only useful when executing choreography, but can also be used by human-oriented tools to enhance the manipulation of the schema with semantic information available from the ontology, even if the tool only understands WSMO ontologies (and it does not understand WSMO choreography).
- (+) SAWSDL provides schema mapping annotations to attach lifting and lowering transformations; this is as yet unspecified in the WSMO-based grounding.
- (+) Our experience with the direct SAWSDL-based grounding in this deliverable helps us refactor WSMO to a lighter-weight annotation mechanism based on WSDL and SAWSDL. This refactoring should simplify the management of semantic Web service descriptions (currently split in WSML and WSDL documents), and it should further simplify the interactions between various task-oriented components of the semantic execution environment.

The SAWSDL grounding, while useful on its own, is also the first step towards WSMO-Lite: a lightweight view on WSMO, completely in terms of SAWSDL.

4. Data Grounding: Lifting and Lowering

A semantic execution environment SEE, such as WSMX, works on the semantic level, with its data represented in WSML or RDF. In contrast, Web services and their clients usually exchange messages in XML or in a similar non-semantic structured data format. In order to enable the SEE to communicate with actual Web services, its semantic data must be *lowered* into the expected input messages, and the data coming from the service in its output messages must be *lifted* back up to the semantic level.

While the WSMX uses data in WSML, there is a direct mapping between WSML and RDF, therefore, we further only discuss RDF data as this is more applicable, for instance in the future context of WSMO-Lite; the reader can substitute WSML for RDF and keep in mind the necessary mapping.

If there were Web services that would accept and produce RDF data in their messages, lifting and lowering would be identity mappings; the SEE would only need to serialize and parse the data in on-the-wire messages. However, RDF-driven Web services are extremely rare. In fact, most Web services use XML-based messages, therefore we must support lowering from RDF to XML, and lifting back. In this section, we first describe the use of XSPARQL, described in Section 2.4, for implementing both transformation directions, and then we discuss a planned GUI tool that will simplify the task of specifying lifting and lowering mappings.

4.1 Lifting and Lowering with XSPARQL

Both lifting and lowering transformations are attached to message descriptions in WSDL, using the SAWSDL attributes liftingSchemaMapping and loweringSchemaMapping respectively. A message in WSDL is described with an XML Schema element declaration. A lifting transformation should accept documents valid according to the schema of the element, and produce the equivalent RDF data. A lowering transformation takes RDF data as its input, and should produce an XML document that is valid according to the schema of the message element. Alas, verifying that a transformation would accept all valid documents, or that all its possible results are going to be valid, is a known hard (if not generally impossible) problem of program correctness proof. Transformation authors must rely on testing.

To illustrate XSPARQL lifting and lowering transformations, we assume a travel (train ticket) reservation service with its message schemas shown in Listing 5 and the respective data ontology shown in Listing 6. All these examples tie to the WSDL description in Listing 3.

We need a lowering transformation for the booking request element so that a SEE client can transform the data of the user goal (booking a train trip) into the appropriate XML/SOAP message; this transformation is shown in Listing 7. Further, we need a lifting transformation for the reservation response element; we show such a transformation in Listing 8.

The lowering transformation takes parts of the request data (starting on line 24) and mostly puts it into the resulting XML structure in a straightforward way. For the start and destination locations, which may be a train station or a city, the function *locationLowering* puts the respective place names in the corresponding XML structure.

The lifting transformation is even simpler, and it uses the input data together with the incoming message and constructs the reservation graph in RDF.

Listing 5: XML Schema for the example service messages

```
<xs:schema targetNamespace="http://example.com/bookTicket.xsd"
    xmlns="http://example.com/bookTicket.xsd"
    xmlns:sawsdl="http://www.w3.org/ns/sawsdl#"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" >
 <xs:element name="BookingRequest"
    sawsdl:modelReference="http://example.org/bookTicket#ReservationRequest"
    sawsdl:loweringSchemaMapping="http://example.org/BookingRequestLowering.xsp" >
  <xs:complexType>
   <xs:all>
    <xs:element name="start" type="location"/>
    <xs:element name="destination" type="location"/>
    <xs:element name="dateTime" type="xs:dateTime"/>
    <xs:element name="passengerCount" type="xs:short"/>
   </xs:all>
  </xs:complexType>
 </xs:element>

 <xs:complexType name="location">
  <xs:all>
   <xs:element name="country" type="xs:string"/>
   <xs:element name="city" type="xs:string"/>
   <xs:element name="station" type="xs:string" minOccurs="0"/>
  </xs:all>
 </xs:complexType>

 <xs:element name="Reservation"
    sawsdl:modelReference="http://example.org/bookTicket#Reservation"
    sawsdl:liftingSchemaMapping="http://example.org/ReservationLifting.xsp" >
  <xs:all>
   <xs:element name="confirmationID" type="xs:string"/>
   <xs:element name="description" type="xs:string"/>
  </xs:all>
 </xs:element>
</xs:schema>
```

Listing 6: Ontology for the example service message data

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
@prefix bkrdf: <http://example.org/bookTicket#> .

bkrdf:Reservation a rdfs:Class .
bkrdf:ReservationRequest a rdfs:Class .

bkrdf:time a rdf:Property .
 # domain: either a reservation request or reservation
 # range: xs:dateTime
bkrdf:from a rdf:Property .
```

```
  # domain: either a reservation request or reservation
  # range: either city or a train station
bkrdf:to a rdf:Property .
  # domain: either a reservation request or reservation
  # range: either city or a train station
bkrdf:passengerCount a rdf:Property .
  # domain: either a reservation request or reservation
  # range: xs:short

bkrdf:description a rdf:Property ;
  rdfs:subPropertyOf rdfs:comment .
bkrdf:confirmationID a rdf:Property .
  # range: xs:string

bkrdf:TrainStation a rdfs:Class .
bkrdf:isInCity a rdf:Property ;
  rdfs:domain bkrdf:TrainStation ;
  rdfs:range bkrdf:City .
bkrdf:City a rdfs:Class .
bkrdf:isInCountry a rdf:Property ;
  rdfs:domain bkrdf:City ;
  rdfs:range bkrdf:Country .
bkrdf:Country a rdfs:Class .
bkrdf:name a rdf:Property .
  # domain: train station, city, country
  # range: xs:string
```

Listing 7: XSPARQL example: lowering transformation

```
declare namespace bkrdf="http://example.org/bookTicket#";
declare namespace bkxml="http://example.com/bookTicket.xsd";

declare function locationLowering ($node, $name) {
  for $city $country $station from <input.rdf>
    where { optional { $node a bkrdf:TrainStation ;
                       name $station ;
                       isInCity $cityNode .
                     $cityNode name $city ;
                       isInCountry $countryNode .
                     $countryNode name $country . }
            optional { $node a bkrdf:City ;
                       name $city ;
                       isInCountry $countryNode .
                     $countryNode name $country . } }
    return
     element { $name } {
       <bkxml:country>{$country}</bkxml:country>
       <bkxml:city>{$city}</bkxml:city>
       { if ($station) then <bkxml:station>{$station}</bkxml:station> else () }
     }
}

for $date $count $from $to from <input.rdf>
```

```
25  where { $x a bkrdf:ReservationRequest ;
26          $x bkrdf:time $date ;
27          $x bkrdf:from $from ;
28          $x bkrdf:to $to ;
29          $x bkrdf:passengerCount $count . }
30  return
31   <bkxml:BookingRequest>
32     <bkxml:dateTime>{$date}</bkxml:dateTime>
33     <bkxml:passengerCount>{$count}</bkxml:passengerCount>
34     { locationLowering($from, bkxml:start) }
35     { locationLowering($to, bkxml:destination) }
36   </bkxml:BookingRequest>
```

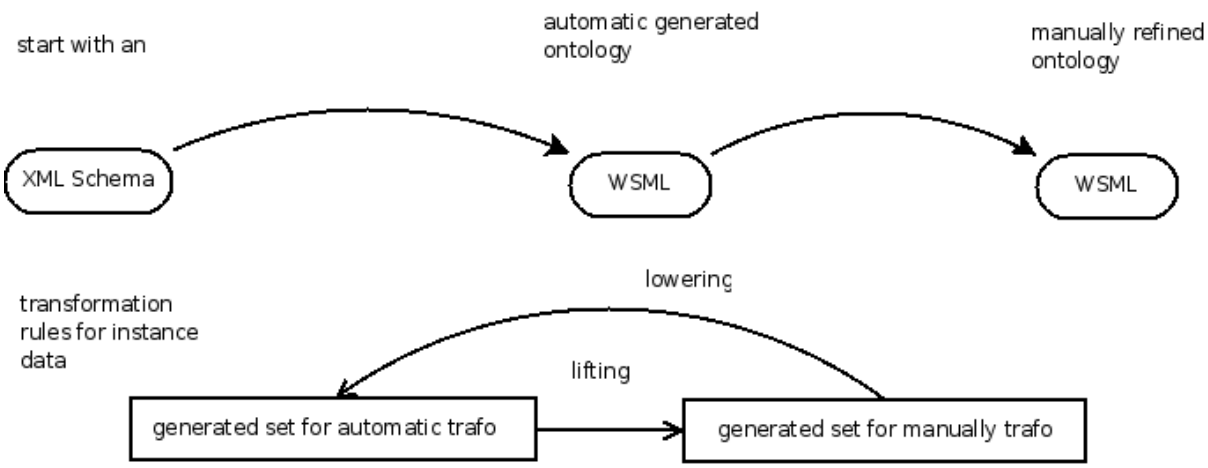
Listing 8: XSPARQL example: lifting transformation

```
01  declare namespace bkrdf="http://example.org/bookTicket#";
02  declare namespace bkxml="http://example.com/bookTicket.xsd";
03
04  let $reservation := /bkxml:Reservation
05  for $date $count $from $to from <input.rdf>
06  construct {
07     :a a bkrdf:Reservation ;
08        bkrdf:time $date ;
09        bkrdf:from $from ;
10        bkrdf:to $to;
11        bkrdf:passengerCount $count;
12        bkrdf:description { $reservation/description } ;
13        bkrdf:confirmationID { $reservation/confirmationID } .
14  }
```

4.2 Description of WSMO Data Grounding Tool

The aim for the Grounding Support Tool (SOA4ALL deliverable D1.2.2, due M18) will be to provide support for creating WSML ontologies from XML Schemas, using an automatic generation step followed by manual correction. In parallel, the tool will provide a transformation rule-set for instance data in XML and WSML; see Figure 4.

Figure 4: Rough work-flow of the Grounding Support Tool.



It will have to implement the following 5 main tasks:

1. An automatic transformation of an XML Schema to a WSMO Ontology. This includes: defining the conceptual mappings between XML Schema conceptual model and WSMO Ontology model and implementing an engine that uses these mappings to automatically produce a WSMO ontology out of an XML Schema.
2. Maintain a transformation of XML data to WSML ontology instances in parallel to the Schema to Ontology transformation. Implement an engine which is based on the first step and will take as input the XML data according to the XML Schema and it will produce as output the WSML ontology instances.
3. Also support the transformation of WSML ontology instances to XML data: this will be done with an engine which is also based on task one: it will take as input the WSML instances and the initial XML Schema and it will produce as output the XML data.
4. Adapting the Ontology Mapping Tool in WSMT to support bi-directional mappings between XML Schema and WSMO ontologies. The aim of this task is to provide a graphical tool for refining the machine-generated ontology from step 1 to a proper hand-crafted ontology, while in parallel maintaining the instance data transformation rules. This will be done by the creation of an abstract view on the XML Schema to map. The ontology generated by the implementation developed in the first task will not be visible to the user but will be internally used in order to reuse the existing tool support for ontology to ontology mapping – i.e., the ontology mapping language and the run-time instance transformation engine.
5. Integrate the results from the tasks above with the data mediation run-time engine. This task will assure that a complete round-trip for data lifting/lowering can be performed for a given XML Schema and the manually refined ontology.

To achieve this, the tool will consist mainly of the following parts:

- An automatic transformation engine that maintains the instance data mapping rules in parallel; it can be implemented in some XML stylesheet language.
- An engine for transforming between instance data according to the given XML Schema and instances of the generated ontology.
- A refinement procedure for the machine-generated ontology into a handcrafted nicer ontology while maintaining the mapping of the instance data between the original XML Schema and the final version of the ontology.

- An integrated GUI for WSMT to enable the manual refinement process in a convenient way for the users. There will be a presentation of the XML Schema on the left side and the automatically generated ontology on the right side, with the possibility to change the ontology while maintaining the mapping between the two representations of the same ontology in the background.

The focus of this tool is to provide on the one hand a simple usable GUI for converting a XML Schema to WSML Ontologies and on the other hand to produce transformation rules for the lifting and lowering transformation of the instance data. This generation of the instance transformation rules will be completely hidden from the user. The user will have the possibility to control the mappings of the schema to the ontology, but they do not have to think about the data transformations.

5. Conclusions

WSMO is a framework for semantic descriptions of Web services, independent from underlying communication technologies. WSDL is the standard language for describing the syntactical and networking aspects of Web services. Grounding is the mechanism that ties service descriptions in WSMO and WSDL together, so that a semantic tool that processes WSMO descriptions can also invoke the services, as prescribed in WSDL.

In its set of specifications, WSMO provides a direct grounding to WSDL, putting the grounding information in the description of a service choreography. In this deliverable, we provide an alternate grounding mechanism that puts the grounding information in the WSDL document, using SAWSDL, a standard for semantic annotations of WSDL. This way, the low-level infrastructure that deals with WSDL has a more direct link to the semantic description in WSMO.

Also, this deliverable describes a simple way to use a new proposed language, XSPARQL, to describe the data grounding, a problem which was up to now unresolved in WSMO. Beside this manual approach to data grounding, we also discuss a GUI tool which will be developed within Deliverable 1.2.2.

The SAWSDL-based grounding of WSMO highlights the relationship between WSMO and the underlying Web service technologies. Additionally, SAWSDL allows us to view WSMO from the point of view of those underlying technologies, and refactor it to produce a more lightweight mechanism for semantic annotations. Such refactoring work will be done in Work Package 3, Deliverable 3.1.1: WSMO-Lite.

Finally, the grounding specified in this deliverable will be used in the distributed service bus (Task 1.4) and in the test-beds (Task 1.5) to provide the links between low-level service descriptions in WSDL and the semantic descriptions in WSMO.

6. References

1. [SAWSDL] Semantic Annotations for WSDL and XML Schema. Recommendation, W3C, August 2007. Available at http://www.w3.org/TR/sawsdl/.
2. [WSDL] Web Services Description Language (WSDL) Version 2.0. Recommendation, W3C, June 2007. Available at http://www.w3.org/TR/wsdl20/.
3. [WSDL11] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. Technical note, March 2001. Available at http://www.w3.org/TR/wsdl.
4. [WSML] Jos de Bruijn (editor): The Web Service Modeling Language WSML, version 0.21 available at http://www.wsmo.org/TR/d16/d16.1/v0.21/
5. [WSMO] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. Applied Ontology, 1(1):77-106, 2005.
6. [WSMOD24.2] Jacek Kopecký, Matthew Moran, Tomas Vitvar, Dumitru Roman, and Adrian Mocan. WSMO Grounding. Available at http://www.wsmo.org/TR/d24/d24.2/v0.1/
7. [XMLSchema] XML Schema Part 1: Structures. Recommendation, W3C, October 2004. Available at http://www.w3.org/TR/xmlschema-1/.
8. [XSPARQL] Waseem Akhtar, Jacek Kopecký, Thomas Krennwallner, and Axel Polleres. XSPARQL: Traveling between the XML and RDF worlds - and avoiding the XSLT pilgrimage. In Sean Bechhofer and Manolis Koubarakis, editors, The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, volume 5021 of Lecture Notes in Computer Science, LNCS, pages 674-689, Tenerife, Spain, June 2008. Springer. http://xspargl.deri.ie/