



D11v0.2. Web Service Modeling Ontology - Lite (WSMO-Lite)

WSMO Working Draft 07 April 2004

Final version:

<http://www.wsmo.org/2004/d11/v0.2/20040307>

Latest version:

<http://www.wsmo.org/2004/d11/v0.2>

Previous version:

<http://www.wsmo.org/2004/d11/v02/20040405>

Editors:

Dumitru Roman
Holger Lausen
Eyal Oren
Ruben Lara

This document is also available in non-normative [PDF](#) version.

Copyright © 2004 [DERI](#)®, All Rights Reserved. [DERI](#) liability, trademark, document use, and software licensing rules apply.

Table of contents

[1. Introduction](#)

[2. Non functional properties - core properties](#)

[3. Ontologies](#)

[3.1 Concepts](#)

[3.2 Relations](#)

[3.3 Instances](#)

[3.4 OO-Mediators](#)

[4. Web Services](#)

[4.1 Capability](#)

[4.2 Interfaces](#)

[5. Conclusions and further directions](#)

[References](#)

[Acknowledgement](#)

1. Introduction

This document contains the WSMO-Lite ontology, a minimal yet meaningful subset

of WSMO-Standard v.01 [[Roman et al., 2004](#)]. WSMO-Standard is a meta-ontology for describing aspects of semantic web services, so that these web services can be automatically discovered and composed to satisfy some user's goal, mediating between all kinds of possible mismatches. WSMO-Standard provides modelling constructs to describe web services, the ontologies they use, goals a user may have and different mediators that can solve integration problems.

WSMO-Standard allows different levels of modelling: firstly, it is possible to extend WSMO-Standard by adding one's own constructs; secondly, one can decide to model elements with more, or less, detail. WSMO-Lite limits the elements that can be described, and does so for a very simple reason. The goal of WSMO-Lite is to provide a subset of WSMO-Standard that is meaningful (in the context of web service integration), but, for which, an execution environment is easily implementable. This execution environment would satisfy the goals of users by invoking web services; it would use WSMO-compliant definitions of these goals, these web services and the ontologies they are using. Having WSMO-Lite is very important, as it allows WSMO to 'bootstrap' itself into world-wide use: it allows easy implementation of a simple subset of WSMO, showing the viability and potential of WSMO, and thus attracting interested parties. Once such execution environments have been shown to work, the full potential of WSMO can be unleashed, and the execution environments could be extended to support WSMO-Standard.

If one were to try to immediately build such an execution environment based on WSMO-Standard, the environment would easily get quite complex, since WSMO-Standard allows modelling of very complex situations. Of course, since these complex situations are real-world situations, one must provide an execution environment that deals with these situations at some point in time. However, from an implementation point of view, it is not practical to try and build an environment that can completely handle these complex cases from scratch. It is easier (and it will most likely result in better software) to start with a smaller set, making very clear directions for future extensions. Since we expect (and want to stimulate) several implementations of a WSMO execution environment, it is useful to specifically distinguish a consistent subset of WSMO-Standard. By offering this subset, we can make sure that all these different implementations of the WSMO execution environment are based on the same conceptual model.

Therefore, defining WSMO-Lite serves two purposes: to attract people into building a WSMO execution environment, by deliberately keeping simple the WSMO-Lite that they would need to implement; and to ensure that future implementations do not define their own subset, building an environment that is incompatible with that of others.

Let us illustrate this with an example: in WSMO-Standard the communication with a web service follows a certain choreography pattern, defined by the web service. In order to communicate with this service, the requester must comply with the service's choreography. This choreography pattern can be a simple stateless conversation (e.g. request-response), or a more complex stateful conversation, in which the requester possibly has to supply more information. If an execution environment wants to implement WSMO-Standard, it needs to support all possible choreography patterns. Otherwise, it cannot communicate with every WSMO-Standard compliant web service. However, implementing every possible choreography pattern is technically more difficult than only supporting stateless conversations. To enable easy implementation, we restrict these communication patterns in WSMO-Lite, so

that the implementation only needs to deal with simple, stateless, conversations.

Another example where WSMO-Lite deliberately simplifies the meta-ontology to ensure swift implementation, is the relation between a goal that a user wants to satisfy, and a capability that a web service offers. To find out which capability (or which composition of web services) could satisfy a user goal, advanced logic reasoning techniques must be employed. While we certainly acknowledge that this is a very important aspect of WSMO and that it is a very promising area of semantic web services, it makes sense to simplify this framework for initial implementation. By hard-wiring a goal to a capability (so a user's goal exactly matches one capability) we deliberately limit the conceptual model for easy implementation. Once WSMO-Lite has shown the viability and possibility of this approach, people could then switch to WSMO-Standard, and employ its more powerful possibilities.

To summarize, WSMO-Lite is a specific, consistent, subset of WSMO-Standard, that is both meaningful and easily implementable. The reasons for introducing WSMO-Lite are:

- to allow WSMO to 'bootstrap' itself into the world
- attracting interest in implementations of an execution environment (by keeping it simple)
- ensuring a consistency across different implementations (as every implementations would otherwise define its own subset)

WSMO-Lite follows the same structure as WSMO-Standard, with the difference being that WSMO-Lite simplifies things by omitting specific concepts from WSMO-Standard. The remaining set of concepts are minimal in the context of web service integration. The expressiveness of WSMO-Lite is defined by exactly that which is necessary to enable sending a single message from a web service requester to a web service provider. This includes establishing the ontologies of the communicating web services, establishing the message formats and providing mediation between the ontologies of the web service requester and the web service provider. This minimal subset of WSMO-Standard allows the formulation of minimal, yet useful, integration scenarios in the context of web services.

Section 2 presents the non functional properties (core properties) of each modeling element of WSMO-Lite. Section 3 presents ontologies in WSMO-Lite. Section 4 presents a minimal set of elements for describing web services and Section 5 presents our conclusions and further intentions.

2. Non-functional properties - core properties

Non functional properties in WSMO-Lite are defined as in WSMO, except that the only non functional properties kept from WSMO are `title` and `identifier`:

Listing 1. Non-functional properties (core properties) definition

```
nonFunctionalProperties[
title => title
identifier => identifier
]
```

Title

A name given to the element. Typically, `title` will be a name by which the element is formally known.

Identifier

An unambiguous reference to the element within a given context.

Recommended best practice is to identify the element by means of a string or number conforming to a formal identification system. Formal identification systems include but are not limited to the Uniform Resource Identifier (URI) (including the Uniform Resource Locator (URL)), the Digital Object Identifier (DOI) and the International Standard Book Number (ISBN).

3. Ontologies

In order to keep WSMO-Lite as simple as possible, we eliminate the axioms from the definition of the ontology in WSMO-Standard (WSMO hereafter). Thus, an ontology in WSMO-Lite is defined as follows:

Listing 2. Ontology definition

```
ontology[
nonFunctionalProperties => nonFunctionalProperties
usedMediators =>> ooMediators
conceptDefinitions =>> conceptDefinition
relationDefinitions =>> relationDefinition
instances =>> instance
]
```

Non functional properties

The `non functional properties` of an ontology consist of the core properties described in [Section 2](#).

Used Mediators

Building an ontology for some particular problem domain can be a rather cumbersome and complex task. One standard way to deal with the complexity is modularization. Imported ontologies allow a modular approach for ontology design. By importing other ontologies, one can make use of concepts and relations defined elsewhere. Nevertheless, when importing an arbitrary ontology, most likely some steps for aligning, merging and transforming imported ontologies have to be performed. For this reason and in line with the basic design principles underlying the WSMF, we use ontology mediators for importing ontologies.

Concept definitions

The set of concepts that belong to the represented ontology ([Section 3.1](#)).

Relation definitions

The set of relations that belong to the represented ontology ([Section 3.2](#)).

Instances

The set of instances that belong to the represented ontology ([Section 3.3](#)).

3.1 Concepts

Concepts in WSMO-Lite are defined in a similar way to the concepts in WSMO-Standard:

Listing 3. Concept definition

```
conceptDefintion[
nonFunctionalProperties => nonFunctionalProperties
superConcepts =>> simpleLogicalExpression
attributes =>> attributeDefintion
]
```

Non functional properties

The `non functional properties` of a concept consist of the core properties described in [Section 2](#).

Superconcepts

There can be a finite number of concepts that serve as `direct super concepts` for some concept. The range of a super concept is a simple logical expression.

Attributes

Each concept provides a (possibly empty) set of `attributes` that represent named slots for data values and instances that have to be filled at the instance level.

An attribute definition specifies a slot of a concept by fixing the name of the slot as well as a logical constraint on the possible values filling that slot. Hence, this simple logical expression can be interpreted as a typing constraint, which specifies the extension of some concept in the ontology in the simplest case; more sophisticated constraints can be modeled easily.

Listing 4. Attribute definition

```
attributeDefinition[
nonFunctionalProperties => nonFunctionalProperties
rangeDefinition => simpleLogicalExpression
]
```

Non functional properties

The `non functional properties` of an attribute consist of the core properties described in [Section 2](#).

Range definition

A simple logical expression constraining the possible values for filling the slot of any instance of a particular concept.

3.2 Relations

Relations are defined as in WSMO:

Listing 5. Relation definition

```
relationDefinition[
nonFunctionalProperties => nonFunctionalProperties
parameters => LIST(parameter)
]
```

Non functional properties

The `non functional properties` of a relation consist of the core properties described in [Section 2](#).

Parameters

A list of the input parameters of the method. Concrete values for these parameters have to be specified when the method will be invoked.

A parameter is a named placeholder for some value. This concept is used in the definition of methods as well as in the definition of n-ary relations.

Listing 6. Parameter definition

```
parameter[
nonFunctionalProperties => nonFunctionalProperties
domainDefinition => simpleLogicalExpression
]
```

Non functional properties

The non functional properties of a parameter consist of the core properties described in [Section 2](#).

Domain definition

A simple logical expression constraining the possible values that the parameter can take.

3.3 Instances

Instances in WSMO-Lite are defined in a similar way to the instances in WSMO-Standard:

Listing 7. Instance definition

```
instance[
nonFunctionalProperties => nonFunctionalProperties
instanceOf =>> logicalExpression
attributeValues =>> attributeValueDefinition
]
```

Non functional properties

The non functional properties of an instance consist of the core properties described in [Section 2](#).

Instance of

We consider the general case, where an instance might be the instance of some (complex) concept which is defined in terms of a simple logical expression.

Attribute values

A list of attribute values for the instance.

Listing 8. Attribute value definition

```
attributeValueDefinition[
nonFunctionalProperties => nonFunctionalProperties
valueDefinition => simpleLogicalExpression
]
```

Non functional properties

The non functional properties of an attributeValueDefinition consist of the core properties described in [Section 2](#).

Value definition

A simple logical expression defining the values for filling the slot of the instance.

3.4 OO-Mediators

OO-Mediators in WSMO-Lite are defined in a similar as in WSMO-Standard:

Listing 9. ooMediator definition

```
ooMediator[
nonFunctionalProperties => nonFunctionalProperties
sourceComponent =>> (ontology or ooMediator)
targetComponent =>> (ontology or webService or ooMediator)
mediationService => webService
]
```

Non functional properties

The non functional properties of an ooMediator consist of the core properties described in [Section 2](#).

Source component

The source component defines the source ontology or ooMediator for this mediator.

Target component

The target component defines the target ontology, webService or ooMediator for this mediator.

Mediation Service

The mediation service points to a web service that actually implements this mediation.

4. Web Services

The elements for describing a web service in WSMO-Lite are basically the same as in WSMO, except that the non functional properties are reduced to the non functional properties presented in [Section 2](#), and the interface is simplified to not having a choreography but instead using simple message exchange patterns. Also, no orchestration can be specified.

Listing 9. Web service definition

```
webService[
nonFunctionalProperties => nonFunctionalProperties
usedMediator =>> ooMediator
capability => capability
interfaces =>> interface
]
```

Non functional properties

The non functional properties of a web service consists of the non functional properties described in [Section 2](#).

Used Mediators

By importing ontologies, a web service can make use of concepts and relations defined elsewhere. A web service can import ontologies using ontology mediators.

Capability

The capability of a web service is described in [Section 4.1](#).

Interfaces

The interfaces of a web service are described in [Section 4.2](#).

4.1 Capability

A capability is defined as in WSMO, except that the non functional properties are reduced to the non functional properties presented in [Section 2](#).

Listing 10. Capability definition

```
capability[
nonFunctionalProperties => nonFunctionalProperties
preconditions =>> simpleLogicalExpression
postconditions =>> simpleLogicalExpression
assumptions =>> simpleLogicalExpression
effects =>> simpleLogicalExpression
]
```

Non functional properties

The non functional properties of a capability consist of the core properties described in the [Section 3](#) (where, in this case, an element in the core properties is equivalent to a capability).

Pre-conditions

Pre-conditions describe what a web service expects for enabling it to provide its service. They define conditions over the input.

Post-conditions

Post-conditions describe what a web service returns in response to its input. They define the relation between the input and the output.

Assumptions

Assumptions are similar to pre-conditions, however, also reference aspects of the state of the world beyond the actual input.

Effects

Effects describe the state of the world after the execution of the service.

4.2 Interfaces

The properties of an interface in WSMO-Lite are non functional properties, errors and messages exchange and grounding. The idea is to allow only simple conversation (message passing without composition).

Listing 11. Interface definition

```
interface[
nonFunctionalProperties => nonFunctionalProperties
messageExchange => messageExchangePattern
]
```

Non functional parameters

The non functional properties of an interface consist of the core properties described in the [Section 2](#).

Message exchange

It consists of a Message Exchange Patterns (MEP). MEPs model stimuli-respons patterns; they define the sequence and the cardinality of the multiple messages exchanged.

Listing 12. Message Exchange Pattern

```
messageExchangePattern[
inMessages =>> message
outMessages =>> message
errorGenerationRule =>errorGenerationRule
]
```

Different instantiations of MEPs can be derived based on the cardinality of the messages:

- in-only: the MEP consists of only one message which has the direction "in".
- in-out: the MEP consists of one message which has the direction "in" followed by another message which has the direction "out".
- in-multi-out: the MEP consists of one message which has direction "in" followed by zero or more messages having the direction "out".
- out-only: the MEP consists of only one message which has the direction "out".
- out-in: the MEP consists of one message which has the direction

"out" followed by another message which has the direction "in".

As one can see the only elements needed for a direct invocation of the service are the message exchange patterns instances. Being defined in a very similar way to the [WSDL 2.0 message patterns](#), the mapping of MEPs to [WSDL 2.0 message patterns](#) is straight forward.

Error Generation Rule

MEPs also specify their error generation model by indicating where errors may occur:

- Errors Replaces Messages: any message after the first in the pattern may be replaced with an error message, which must have identical cardinality and direction.
- Message Triggers Error: any message, including the first, may trigger an error message in response.

5. Conclusions and further directions

This document presented WSMO-Lite, the minimal subset of concepts of [WSMO-Standard](#). WSMO-Lite is, as shown, a subset of WSMO-Standard established through elimination or simplification of concepts.

WSMO-Lite will be verified by a corresponding implementation supporting the execution of ontologies thereby achieving mediated web service invocation. Feedback that will emerge from the implementation will be fed back into WSMO-Lite and WSMO-Standard.

References

[Roman et al., 2004] D. Roman, U. Keller, H. Lausen (eds.): Web Service Modeling Ontology - Standard (WSMO - Standard), version 0.1 available at <http://www.wsmo.org/2004/d2/v01/>

Acknowledgement

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, SWWS, Esperonto, COG and h-TechSight; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate programm.

The editors would like to thank to all the [members of the WSMO working group](#) for their advises and inputs to this document.

[webmaster](#)