



D11v02. Web Service Modeling Ontology - Lite (WSMO-Lite)

WSMO Working Draft 05 March 2004

This version:

<http://www.wsmo.org/2004/d11/v02/20040305>

Latest version:

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

Previous version:

<http://www.wsmo.org/2004/d11/v02/20040305>

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](#)

[4. Web Services](#)

[4.1 Capability](#)

[4.2 Interfaces](#)

[5. Conclusions and further directions](#)

[Acknowledgement](#)

1. Introduction

This document contains the WSMO-Lite ontology, a minimal subset of [WSMO-Standard v.01](#). The reason for introducing WSMO-Lite are manifold:

- definition of the minimal useful subset of WSMO-Standard
- establishment of the simplest subset of concepts that are useful for defining integration
- easing the implementation as a starting point for WSMO-Standard and WSMO-Full

WSMO-Lite follows the same structure as WSMO-Standard with the difference that WSMO-Lite simplifies by omitting specific concepts from WSMO-Standard. The remaining set of concepts are minimal in the context of web service integration.

Fundamentally WSMO-Lite's expressiveness is equivalent to sending a single message to a web service interface from a web service requester. This requires to establish the communication web services' ontologies, the message definition as well as mediation between the ontologies of the web service requester and web service provider.

This minimal subset of WSMO-Standard concepts allows to formulate the minimal possible useful integration scenario in 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). Also, WSMO-Lite does not contain any concept of mediators, so we replace the mediator-based

mechanism used in WSMO for importing ontologies by the simple property: `imported ontologies`. Thus, an ontology in WSMO-Lite is defined as follows:

Listing 2. Ontology definition

```
ontology[
nonFunctionalProperties => nonFunctionalProperties
importedOntologies =>> ontology
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](#).

Imported ontologies

`Imported ontologies` reference other ontologies containing definitions whose meaning is considered to be part of the meaning of the ontology which imports them. The problems to solve the mismatching, alignment, etc. between imported ontologies is left to the tools that implement WSMO-Lite, as opposed to the use of mediators in WSMO to solve these problems.

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 =>> logicaExpression
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.

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](#), the mechanism for importing ontologies is not by using mediators as in WSMO, but by using a simple property: `imported ontologies`, and a concrete grounding is given.

Listing 9. Web service definition

```
webService[
nonFunctionalProperties => nonFunctionalProperties
importedOntologies =>> ontology
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](#).

Imported ontologies

By importing ontologies, a web service can make use of concepts and relations defined elsewhere. The problems to solve the mismatching, alignment, etc between imported ontologies is left to the tools that implement WSMO-Lite, as opposed to the use of mediators in WSMO to solve these problems. The imported ontologies will provide concepts and relations needed also in the definitions of the capability and interfaces.

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
errors =>> (errorCode or errorDomainConcept)
messageExchange => messageExchangePattern
grounding => grounding
]
```

Non functional parameters

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

Errors

The service should be able to express `error data`: data indicating problems during the execution. This is represented in terms of error codes. We classify errors in domain-independent and domain dependent errors. The domain independent errors are similar to [HTTP errors](#). The domain dependent errors are considered to be defined in ontologies (for example a service may return an error Invalid account number – a domain dependent error, described in a domain ontology).

Message exchange

It consists of a Message Exchange Patterns (MEP). MEPs model stimuli-responses 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 types 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".

In Messages

It consists of the messages having the "in" direction.

Out Messages

It consists of the messages having the "out" direction.

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.

Grounding

It represents a mapping from the abstract specification to a concrete specification of the elements that describe an interface, which are required for interacting with the service. As one can see the only elements needed for a direct invocation of the service are the message exchange patterns. 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.

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.

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 programme.

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

[webmaster](#)