



# D2v01. Web Service Modeling Ontology (WSMO)

DERI Working Draft 14 February 2004

**Final version:**

<http://www.nextwebgeneration.org/projects/wsmo/2004/d2/v01/20040214>

**Latest version:**

<http://www.nextwebgeneration.org/projects/wsmo/2004/d2/v01>

**Previous version:**

<http://www.nextwebgeneration.org/projects/wsmo/2004/d2/v01/20040213>

**Editors:**

Dumitru Roman  
Uwe Keller  
Holger Lausen

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. Principles and Approach**

[2.1 Principles](#)

[2.2 Approach](#)

**3. Non functional properties - core properties**

**4. Ontologies**

[4.1 Concepts](#)

[4.2 Relations](#)

[4.3 Axioms](#)

[4.4 Instances](#)

**5. Goals**

**6. Mediators**

**7. Web Services**

[7.1 Non functional properties](#)

[7.2 Capability](#)

[7.3 Interfaces](#)

[7.4 Groundings](#)

**8. Formal Language Specification for WSMO**

**9. Conclusions and further directions**

**References**

**Acknowledgement**

**Appendix**

---

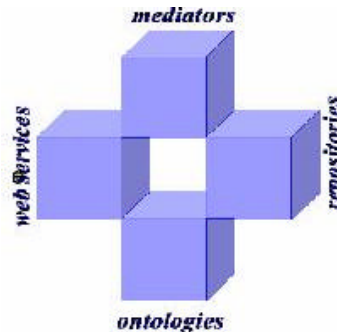
## 1. Introduction

This document presents an ontology called Web Service Modeling Ontology (WSMO) for describing various aspects related to Semantic Web Service. Having the Web Service Modeling Framework (WSMF) [[Fensel & Bussler, 2002](#)] as a starting point, we refine this framework and develop a formal ontology and

a formal language.

The WSMF [Fensel & Bussler, 2002] consists of four different main elements: ontologies that provide the terminology used by other elements, goal repositories that define the problems that should be solved by web services, web services descriptions that define various aspects of a web service and mediators which bypass interoperability problem.

Figure 1. The main elements of WSMF



[Section 2](#) presents the principles and the approach for WSMO. [Section 3](#) presents the non functional properties (core properties) of each modelling element of WSMO. Following the philosophy of WSMF, we further refine in the next sections the ontologies ([Section 4](#)), goals ([Section 5](#)), mediators ([Section 6](#)) and web service ([Section 7](#)). [Section 8](#) presents the formal language we use for specifying WSMO and [Section 9](#) presents our conclusions and further intentions.

## 2. Principles and Approach

While there are many existing efforts trying to define Web Services as well as Semantic Web Services like OWL-S, BPEL4WS and so on, none of them explicitly states its underlying principles and approaches. Furthermore, none of the approaches follows through from the conceptualization all the way to a formal execution model backed up by an open source implementation.

The WSMO addresses all these deficiencies and is therefore a solution of a different quality, representing the first effort ever addressing the domain of Semantic Web Services completely.

### 2.1 Principles

Every ontology design is based on principles of the domain that is going to be formally represented. In the following the principles of the WSMO are outlined that guide the ontology design decisions.

- **Interface vs. Implementation.** When formally describing interacting entities then it is important to recognize the difference between the interface of an interacting entity, its internal implementation as well as its externally visible and internal behavior. The WSMO shall make this distinction to be always very specific on when describing external interfaces and external behavior versus internal implementations and internal behavior.
- **Peer-to-peer vs. Client/Server.** Interacting entities, sometimes called agents, are characterized in general by one entity interacting with one or more other entities. In the simplest case, two entities are interacting. In the context of Semantic Web Services, the interaction is between equal partners, i.e., one entity sending a message and the other one receiving the message are equal in their level of control over the other entity. This is called a peer-to-peer approach in contrast to a client/server approach where the client drives the interaction as the controlling entity. The WSMO shall recognize the fact that all entities are equal in their control structure leading in the general case to conversations amongst equal partners.
- **Type and instance.** Semantic Web Services are defined at design time and executed at runtime. This implies a distinction between the general description of a Semantic Web Service and its instantiation. One defined Semantic Web Service can be executed many times, concurrently as well as sequentially. In general, not every concept in the domain follows that general principle. For example, agents are not typed in general. For example, there is only one instance of a specific

person, not a type. Hence, the instance of a person has to be described with WSMO, not its type (while of course a concept of person is an agent). The WSMO shall be diligent in making this distinction in order to allow a real representation of the domain of Semantic Web Services.

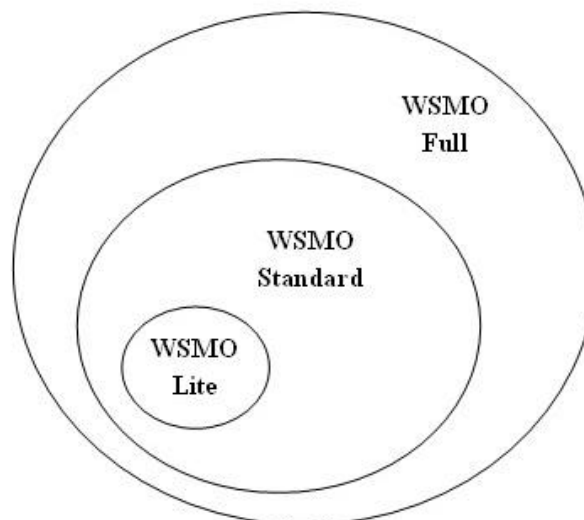
In addition to principles that are underlying the domain of representation, ontologies have a purpose and are used in a specific context. The WSMO is not only describing the domain of Semantic Web Services, but also serves as the conceptual model for a formal language design as well as an execution environment for the actual execution of Semantic Web Service interactions.

- **Execution completeness.** The WSMO is the underlying conceptual model for defining and executing Semantic Web Services. Therefore, the WSMO shall be complete so that actual Semantic Web Services can not only be defined but the definition is complete for execution purposes.
- **Execution property completeness.** Semantic Web Services are not a passive data representation but the description of the interaction or conversation patterns of interacting agents. Properties have to be ensured to support the flawless execution of conversations. This includes deadlock freedom, livelock freedom, reachability of permitted and consistent end states, interoperability as well as correct compensation amongst others. The WSMO shall be designed in such a way that the mentioned properties can be ensured statically before runtime.
- **Execution semantics.** The WSMO is a representation of the domain of Semantic Web Services. Since it formally describes the interactions of agents a formal execution semantics has to be defined in order to uniquely specify the execution behavior at runtime. The concepts of WSMO shall have a formal execution semantics to ensure a consistent execution model.

## 2.2 Approach

While it is possible and tempting to design one big, all encompassing WSMO, this is not followed due to its inherent complexity and difficulty to comprehend. Instead, a layered approach is followed that starts with a basic WSMO, called WSMO-Lite, continues with a mature set of concepts called WSMO-Standard and ends with a full ontology called WSMO-Full.

Figure 2. WSMO-Lite, WSMO-Standard, WSMO-Full



As Figure 2 shows, the different variants of WSMO include each other, one building on top of the other.

## 3. Non-functional properties - core properties

Non functional properties are defined as a set of tuples, where each tuple consists of a property and its value constraint. The core properties are defined globally, meaning that they can be used for all the modelling elements of WSMO. They consist of the Dublin Core Metadata Element Set plus the `version` element:

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

```

nonFunctionalProperties[
title => title
creator => creator
subject => subject
description => description
publisher => publisher
contributor => contributor
date => date
type => type
format => format
identifier => identifier
source => source
language => language
relation => relation
coverage => coverage
rights => rights
version => version
]

```

### **Title**

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

### **Creator**

An entity primarily responsible for making the content of the element. Examples of `creator` include a person, an organization, or a service. Typically, the name of a `creator` should be used to indicate the entity.

### **Subject**

A topic of the content of the element. Typically, `subject` will be expressed as keywords, key phrases or classification codes that describe a topic of the element. Recommended best practice is to select a value from a controlled vocabulary or formal classification scheme.

### **Description**

An account of the content of the element. Examples of `description` include, but is not limited to: an abstract, table of contents, reference to a graphical representation of content or a free-text account of the content.

### **Publisher**

An entity responsible for making the element available. Examples of `publisher` include a person, an organization, or a service. Typically, the name of a `publisher` should be used to indicate the entity.

### **Contributor**

An entity responsible for making contributions to the content of the element. Examples of `contributor` include a person, an organization, or a service. Typically, the name of a `contributor` should be used to indicate the entity.

### **Date**

A date of an event in the lifecycle of the element. Typically, `date` will be associated with the creation or availability of the element.

### **Type**

The nature or genre of the content of the element. `Type` includes terms describing general categories, functions, genres, or aggregation levels for content.

### **Format**

The physical or digital manifestation of the element. Typically, `format` may include the media-type or dimensions of the element. Format may be used to identify the software, hardware, or other equipment needed to display or operate the element. Examples of dimensions include size and duration.

### **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 element Identifier (URI) (including the Uniform element Locator (URL)), the Digital Object Identifier (DOI) and the International Standard Book Number (ISBN).

### **Source**

A reference to a element from which the present element is derived. The present element may be derived from the `source` element in whole or in part. Recommended best practice is to identify the

referenced element by means of a string or number conforming to a formal identification system.

**Language**

A language of the intellectual content of the element.

**Relation**

A reference to a related element. Recommended best practice is to identify the referenced element by means of a string or number conforming to a formal identification system.

**Coverage**

The extent or scope of the content of the element. Typically, `coverage` will include spatial location (a place name or geographic coordinates), temporal period (a period label, date, or date range) or jurisdiction (such as a named administrative entity).

**Rights**

Information about rights held in and over the element. Typically, `rights` will contain a rights management statement for the element, or reference a service providing such information. Rights information often encompasses Intellectual Property Rights (IPR), Copyright, and various Property Rights. If the Rights element is absent, no assumptions may be made about any rights held in or over the element.

**Version**

As many properties of an element might change in time, an identifier of the element at a certain moment in time is needed.

## 4. Ontologies

In WSMO Ontologies are the key to link consensual real world semantics intended by humans with computers. Originating from field of philosophy, in AI the term “ontology” refers to a description of a part of the world in a program – a domain of discourse. An important definition of ontology, used by many researchers in the field of ontologies was introduced by Gruber [Gruber, 1993]: *An ontology is a formal explicit specification of a shared conceptualization*. Respecting this definition, WSMF defines the two essential aspects of ontologies as follows:

- Ontologies define formal semantics for information, consequently allowing information processing by a computer.
- Ontologies define real-world semantics which make it possible to link machine-processable content with meaning for humans based on consensual terminologies.

From this rather conceptual definition we want to extract the essential components which define an ontology. Eventually, we aim at representing ontologies no matter what ontology language is being used for describing them. Ontologies define a consensual terminology by providing concepts and relationships among the set of concepts. In order to capture semantic properties of relations and concepts, an ontology generally also provides a set of axioms, that means expressions in some logical framework, for instance First-Order Logic.

Each element that belongs to the established terminology, i.e. concepts and relations, can be further constrained semantically by means of a logical constraint that expresses some sort of real-world semantics related to this element.

Eventually, there might also be individuals included within an ontology that represent instances of concepts<sup>[1]</sup>.

There are several ways to describe the contents of ontologies by an ontology <sup>[2]</sup> itself. For now, we decide to keep the model as simple as possible and represent each element of an ontology as simple as possible.

In principle, an ontology constitutes of four main building blocks: concepts, relations, axioms and instances. An ontology is defined as follows <sup>[3]</sup>:

Listing 2. Ontology definition

```
ontology[
nonFunctionalProperties => nonFunctionalProperties
usedMediators =>> ooMediator
conceptDefinitions =>> conceptDefinition
relationDefinitions =>> relationDefinition
axioms =>> axiom
instances =>> instance
]
```

### Non functional properties

The non functional properties of an ontology consist of the core properties described in [Section 3](#).

### 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 (ooMediators) for importing ontologies.

### Concept definitions

The set of concepts that belong to the represented ontology.

### Relation definitions

The set of relations that belong to the represented ontology.

### Axioms

The set of axioms that belong to the represented ontology.

### Instances

The set of instances that belong to the represented ontology.

## 4.1 Concepts

Concepts constitute the basic elements of the consensual terminology for some problem domain. They provide an abstract view on real-existing and artificial artifacts within the addressed domain of discourse.

From a high-level perspective a concept – described by a concept definition – provides attributes with names and types. It has a name, can be textually described in natural language and might change over time and thus has a version (they are part of the non functional properties of the `concept`).

Furthermore, a concept can have several (possibly none) direct superconcepts as specified by the so-called "is\_a"-relation.

When describing the semantics of concepts within some ontology, we favor an uniform and rather general approach: we consider the semantics to be captured by means of a logical expression.

For instance, this allows us to state that some concept represents the union or intersection of two or more other concepts. Consider an ontology on social structures within a human society, then we can defined concepts like "Human-being" or "Female" and accurately describe the semantics of the concept "Granny" as precisely the intersection of the concepts "Human-being", "Female" and "Parent of some parent".

Such modeling styles are commonly used in many *Description Logics* [Baader et al., 2003] and can be found in widely-used ontology languages like *OWL* [McGuinness & van Harmelen, 2003] as well.

Hence, we extract the following abstract description for concepts:

Listing 3. Concept definition

```
conceptDefinition[
nonFunctionalProperties => nonFunctionalProperties
superConcepts =>> axiom
attributes =>> attributeDefinition
methods =>> methodDefinition
]
```

**Non functional properties**

The `non functional properties` of an ontology consist of the core properties described in [Section 3](#).

**Superconcepts**

There can be a finite number of concepts that serve as `direct superconcepts` for some concept. In particular, being a subconcept of some other concept means that a concept inherits the signature of this superconcept and the corresponding constraints.

**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 logical expression can be interpreted as a typing constraint.

Listing 4. Attribute definition

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

**Non functional properties**

The `non functional properties` of an attribute definition consist of the core properties described in [Section 3](#).

**Range definition**

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

**Methods**

Besides attributes we also allow a concept to have `methods` [4] that can be invoked on each instance of a concept and in response return some result value.

A method definition specifies a function that can be invoked on a specific instance of a concept. When invoking the function, one has to specify the values of the parameters, for which the function has to be computed. The specific instance for which the method is invoked, can be seen as an implicit input parameter of the function, that is not explicitly contained in the set of input parameters. The computed value will then be returned to the invoker.

Listing 5. Method definition

```
methodDefinition[
nonFunctionalProperties => nonFunctionalProperties
rangeDefinition => axiom
parameters => LIST(parameter)
]
```

**Non functional properties**

The `non functional properties` of a method definition consist of the core properties described in [Section 3](#).

**Range definition**

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

**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 => axiom
]
```

**Non functional properties**

The non functional properties of a parameter consist of the core properties described in [Section 3](#).

**Domain definition**

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

## 4.2 Relations

The second main building block for the specification of an ontology are relations. Relations are used in order to model interdependencies between several concepts (respectively instances of these concepts) with respect to the problem domain.

Relations between concepts are more general than simple attributes or properties as for instance in OWL. Mathematically, relationships are simply sets of n-tuples, over the domain of instances of concepts. In popular and commonly used system modeling languages like UML [\[Fowler, 2003\]](#) such concrete tuples are often called links. The underlying semantics of cardinalities in the case of n-ary relations follows the definition in the UML framework [\[Rumbaugh et al., 1998\]](#).

Relations can be very specific in nature and only applicable in the context of a particular problem domain, but there are also relations that occur frequently when modeling ontologies for different application areas. There are several common properties that modeled relations can provide, e.g. symmetry, transitivity, reflexivity. Again, for the sake of simplicity we decide not to represent these common properties of relationships currently within the metaontology explicitly but implicitly by means of axioms.

Other dependencies between relationships (for instance subset, intersection, union, difference, inverse relationship between two or more relations) will be dealt with in the same way.

Listing 7. Relation definition

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

**Non functional properties**

The non functional properties of a relation definition consist of the core properties described in [Section 3](#).

**Parameters**

A list of [parameter](#) descriptions specifying each of the concepts that are interrelated.

## 4.3 Axioms

An axiom is considered to be a logical expression enriched by some extra-logical information.

Listing 8. Axiom definition

```
axiomDefinition[
nonFunctionalProperties => nonFunctionalProperties
defined_by => logicalExpression
]
```

**Non functional properties**

The non functional properties of an axiom definition consist of the core properties described in [Section 3](#).

**Defined by**

The logical constraint expressed in the formal language underlying the WSMO that represents the actual statement of the axiom.

## 4.4 Instances

Eventually, within an ontology there might be instances defined for some concept. Therefore we have to reflect the “instance\_of”-relation that can be given within an ontology specification.

Listing 9. Instance definition

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

### Non functional properties

The `non functional properties` of an instance consist of the core properties described in [Section 3](#).

### 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 logical expression.

### Attribute values

A list of attribute values for the instance.

Listing 10. Attribute value definition

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

### Non functional properties

The `non functional properties` of an attribute value definition consist of the core properties described in [Section 3](#).

### Value definition

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

## 5. Goals

In this section, we introduce the notion of `goals` and define the elements that are used in the description of a goal. Our definition of a goal is the one given in [\[Fensel & Bussler, 2002\]](#): A goal specifies the objectives that a client may have when he consults a web service.

In [\[Fensel & Bussler, 2002\]](#), a goal specification consists of two elements, namely:

- Pre-conditions
- Post-conditions

WSMO restricts the definition of goals to Post-conditions. In addition, the Web Services Modeling Ontology introduces `non-functional properties`, `used mediators`, and `effects`.

Listing 11. Goal definition

```
goal[
nonFunctionalProperties => nonFunctionalProperties
usedMediators =>> {ooMediator or ggMediator}
postConditions =>> axiom
effects =>> axiom
]
```

### Non functional properties

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

### Used mediators

By importing ontologies, a goal can make use of concepts and relations defined elsewhere. A goal can import ontologies using ontology mediators (`ooMediators`). A goal may be defined by reusing an already existing goal (e.g. for defining the goal “buy a book for children” one can reuse the

already existing goal “buy a book”) or by combining existing goals (e.g. for defining the goal “buy a book from Amazon written by Hemingway” one can reuse and combine the existing goals “buy a book from Amazon” and “buy a book written by Hemingway”). This is achieved by using goal mediators (ggMediators).

### Post-conditions

`Post-conditions` in WSMO describe the state of the information space that is desired.

### Effects

`Effects` describe the state of the world that is desired.

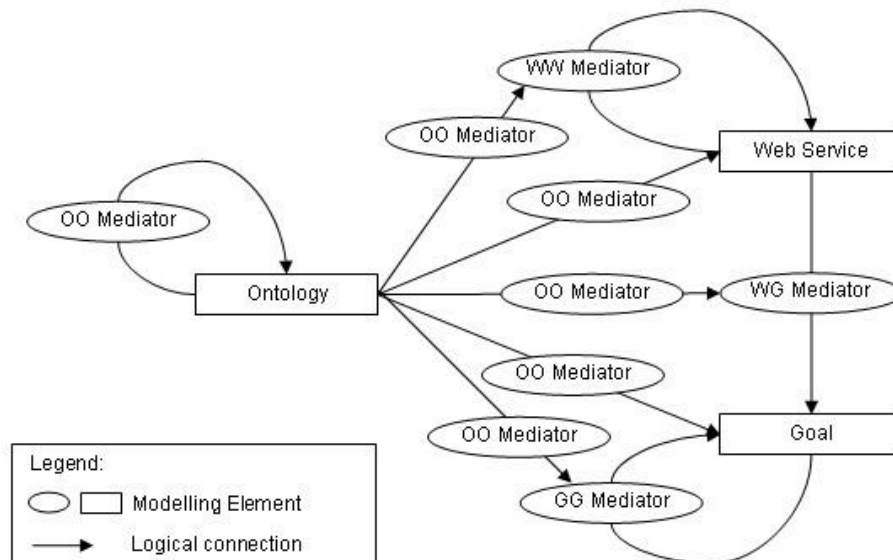
## 6. Mediators

In this section, we introduce the notion of `mediators` and define the elements that are used in the description of a mediator.

We distinguish between four conceptually different mediators:

- `ggMediators`: mediators that link two goals.
- `ooMediators`: mediators that import ontologies and resolve possible representation mismatches among all imported ontologies.
- `wgMediators`: mediators that link web service to goals. They explicitly may state the difference (reduction) between the two components and map different vocabularies.
- `wwMediators`: mediators linking two Web Services.

Figure 3. Illustration of mediators in WSMF



The `mediator` is defined as follows:

Listing 12. Mediators definition

```

mediator[
nonFunctionalProperties => nonFunctionalProperties
sourceComponent =>> (ontology or goal or webService or mediator)
targetComponent =>> (ontology or goal or webService or mediator)
mediationService => (goal or wwMediator)
]

ooMediator :: mediator[
sourceComponent =>> (ontology or ooMediator)
]

ggMediator :: mediator[
sourceComponent => (goal or ggMediator)
targetComponent => (goal or ggMediator)
usedMediators =>> ooMediator
reduction => axiom
]

wgMediator :: mediator[
sourceComponent => (webService or wgMediator)
targetComponent => (goal or wgMediator)
usedMediators =>> ooMediator
reduction => axiom
]

wwMediator :: mediator[
sourceComponent => (webService or wwMediator)
targetComponent => (webService or wwMediator)
usedMediators =>> ooMediator
]

```

**Non functional properties**

The non functional properties of a mediator consist of the core properties described in the [Section 3](#) (where, in this case, an element in the core properties is equivalent to a mediator). Besides these properties, and taking into account that a mediator uses a mediation service, the non functional properties of a mediator also include aspects related to the quality aspect of the mediation service (see next section for a description of these properties). The quality aspects of the mediator and the mediation service, taken together, might be enhanced (e.g. improving the robustness) or weaken (e.g. increasing the response time) by the mediator.

**Source component**

The source component defines one of the two logically connected entities.

**Target component**

The target component defines one of the two logically connected entities.

**Mediation Service**

The `mediation service` points to a goal that declarative describes the mapping or to a `wwMediator` that links to a web service[6] that actually implements the mapping.

**Reduction**

A reduction only exists in a `wwMediator` or a `ggMediator`. It describes in a logical formula the differences between the functionality described in the goal and the one of the web service (if any) or another goal.

For an example of how the mediators are used, refer to the [Appendix](#).

## 7. Web Services

In this section we identify the concepts needed for describing various aspects of a web service. From a complexity point of view of the description of a web service, the following properties of a web service are considered: non functional properties, used mediators, capability, interfaces and groundings.

Listing 13. Web service definition

```
webService[
nonFunctionalProperties => nonFunctionalProperties
usedMediators =>> ooMediator
capability => capability
interfaces =>> interface
groundings =>> grounding
]
```

### Non functional properties

The non functional properties of a web service are described in [Section 7.1](#).

### 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 (ooMediators).

### Capability

The capability of a web service is described in [Section 7.2](#).

### Interfaces

The interfaces of a web service are described in [Section 7.3](#).

### Groundings

The groundings of a web service are described in [Section 7.4](#).

## 7.1 Non functional properties

The non functional properties of a web service consist of the core properties described in the appendix A (where, in this case, an element in the core properties is equivalent to a web service).

Besides these properties, the non functional properties of a web service also include aspects related to the quality aspect of a web service (QoS):

Listing 14. Non functional properties definition for web services

```
nonFunctionalProperties[
performance => performance
reliability => reliability
security => security
scalability => scalability
robustness => robustness
accuracy => accuracy
transactional => transactional
trust => trust
financial => financial
networkRelatedQoS => networkRelatedQoS
]
```

### Performance

It represents how fast a service request can be completed. According to [\[Rajesh & Arulazi, 2003\]](#) performance can be measured in terms of throughput, latency, execution time, and transaction time. The response time of a service can also be a measure of the performance. High quality web services should provide higher throughput, lower latency, lower execution time, faster transaction time and faster response time.

### Reliability

It represents the ability of a web service to perform its functions (to maintain its service quality). It can be measured by the number of failures of the service in a certain time interval.

### Security

It represents the ability of a service to provide authentication (entities - users or other services - who can access service and data should be authenticated), authorization (entities should be authorized so that they only can access the protected services), confidentiality (data should be treated properly so that only authorized entities can access or modify the data), traceability/auditability (it should be possible to trace the history of a service when a request was serviced), data encryption (data should be encrypted), and non-repudiation (an entity cannot deny requesting a service or data after the fact).

### Scalability

It represents the ability of the service to process more requests in a certain time interval. It can be measured by the number of solved requests in a certain time interval.

### Robustness

It represents the ability of the service to function correctly in the presence of incomplete or invalid inputs. It can be measured by the number of incomplete or invalid inputs for which the service still function correctly.

### Accuracy

It represents the error rate generated by the web service. It can be measured by the numbers of errors generated in a certain time interval.

### Transactional

It represents the transactional properties of the web service.

### Trust

It represents the trust worthiness of the service.

### Financial

It represents the cost-related properties of a web service.

### Network-related QoS

They represent the QoS mechanisms operating in the transport network which are rather independent of the web services. They can be measured by network delay, delay variation and/or message loss.

## 7.2 Capability

A *capability* defines the web service by means of its functionality, given by the following properties:

Listing 15. Capability definition

```
capability[
nonFunctionalProperties => nonFunctionalProperties
usedMediators =>> (ooMediator or wgMediator)
preconditions =>> axiom
postconditions =>> axiom
assumptions =>> axiom
effects =>> axiom
]
```

### 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).

### Used mediators

By importing ontologies, a capability can make use of concepts and relations defined elsewhere. A capability can import ontologies using ontology mediators (*ooMediators*). A capability can be linked to a goal using a *wgMediator*.

### Pre-conditions

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

### Post-conditions

*Post-conditions* in WSMO 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.

The purpose of defining again pre-conditions, post-conditions, assumptions, effects, non-functional properties and used mediators in the service capability is to have self-contained web service descriptions, that can be referred or not to the defined goals. In this way, we provide greater flexibility for the use of goals and web services.

## 7.3 Interfaces

An *interface* extends the description of the capability to a higher level of complexity, by introducing the following properties:

Listing 16. Interface definition

```

interface[
nonFunctionalProperties => nonFunctionalProperties
errors =>> (errorCode or errorDomainConcept)
orchestration => orchestrationDescription
compensation => (wwMediator or Goal)
messageExchange =>> messageExchangePatterns
]

```

### Non functional parameters

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

### 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).

### Orchestration

`Orchestration` defines the sequence and conditions under which multiple cooperating `proxies` exchange information in order to achieve some useful function. A **proxy** is defined as being either a goal or a `wwMediator`. Proxies are used when a web service invokes other web services in order to provide its service. Each time a web service needs to be invoked, a proxy needs to be declared (by either declaring a goal or linking it to a `wwMediator`). This way both dynamic (on the fly) composition (by declaring proxies consisting of goals descriptions) and static composition (by linking proxies to `wwMediators`) are supported. In addition, `orchestration` describes an access to the intermediate states of the service. Very related to `orchestration` are data and control flow – both implementing the external accessible part of the business logic of the web service.

### Compensation

When an invoked service fails (i.e. an error code is returned), the service that invoked it may implement a strategy for compensation. The compensation in fact represents either a `wwMediator` or a goal.

### Message exchange

A `message exchange pattern` describes the temporal and causal relationships, if any, of multiple messages exchanged [\[5\]](#).

## 7.4 Groundings

A `grounding` specifies how to access the service. It has mainly to do with protocol and message formats, serialization, transport and addressing. The grounding is a mapping from the abstract specification to a concrete specification of the elements that describe a service, which are required for interacting with the service. The abstract elements for which we need a concrete specification are:

- inputs/outputs from the message exchange in the interface
- error messages generated by the service through it's interface

From the conceptual point of view the grounding can link to various sorts of interface implementations, e.g. WSDL, CORBA, ebXML and others. The W3C defines web services as a software system offering and interface accessible over SOAP with machine processable interface description and especially mentions WSDL. Since the presented ontology is aiming at describing web services it is reasonable to choose the current standard for such a description which is WSDL 2.0, a specification language proposal, with strong industry backing. One error message generated by the service through it's interface is translated to a `fault` message in WSDL and a message exchange in the interface corresponds to a combination of WSDL message exchange pattern.

## 8. Formal Language Specification for WSMO

As shown in the previous sections, many properties of goals, mediators, ontologies and web services are considered to be axioms, defined by logical expressions. In order to be as specific as possible, we consider these logical expressions as being described in F-Logic [\[Kifer et al., 1995\]](#), by complex formulas.

F-Logic combines the advantages of conceptual high-level approaches typical for frame-based language and the expressiveness, the compact syntax, and the well defined semantics from logics. Features of F-Logic include: object identity, methods, classes, signatures, inheritance and rules.

The reasons for which we choose F-Logic to represent logical expressions are:

- it provides a standard model theory
- it is a full first order logic language
- it provides second order syntax while staying in the first order logic semantics
- it has a minimal model semantics
- implemented inference engines are already available

For a detailed description of F-Logic refer to [\[Kifer et al., 1995\]](#). Furthermore, an evaluation of F-Logic and comparison to other languages is given in [\[Keller et al., to appear\]](#).

## 9. Conclusions and further directions

This document presented the Web Service Modeling Ontology (WSMO) for describing several aspects related to web services, by refining the Web Service Modeling Framework (WSMF). Further versions of this document will contain more detailed descriptions of some concepts presented here as well as refinements that add concepts to the current ontology.

WSMO is centered around the underlying technology to be described. The next step is to define a layer on top in terms of the application that is supported by it: fully flexible eCommerce and eWork.

## References

**[Baader et al., 2003]** F. Baader, D. Calvanese, and D. McGuinness: *The Description Logic Handbook*, Cambridge University Press, 2003.

**[Fensel & Bussler, 2002]** D. Fensel and C. Bussler: *The Web Service Modeling Framework WSMF*, Electronic Commerce Research and Applications, 1(2), 2002.

**[Fowler, 2003]** M. Fowler: *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley, 3rd edition, 2003.

**[Gruber, 1993]** T. Gruber: A translation approach to portable ontology specifications, *Knowledge Acquisition*, 5:199-220, 1993.

**[Keller et al., to appear]** U. Keller, A. Polleres, R. Lara, and Y. Ding: *Language Evaluation and Comparison*, to appear.

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

**[McGuinness & van Harmelen, 2003]** D. L. McGuinness and F. van Harmelen: OWL Web Ontology Language Overview, W3C Proposed Recommendation, 15 December 2003. Latest version is available at <http://www.w3.org/TR/owl-features/>.

**[Rajesh & Arulazi, 2003]** S. Rajesh and D. Arulazi: *Quality of Service for Web Services-Demystification, Limitations, and Best Practices*, March 2003. (See <http://www.developer.com/services/article.php/2027911>.)

**[Rumbaugh et al., 1998]** J. Rumbaugh, I. Jacobson, and G. Booch: *The Unified Modeling Language Reference Manual*, Object Technology Series, Addison-Wesley, 1998.

## 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 project.

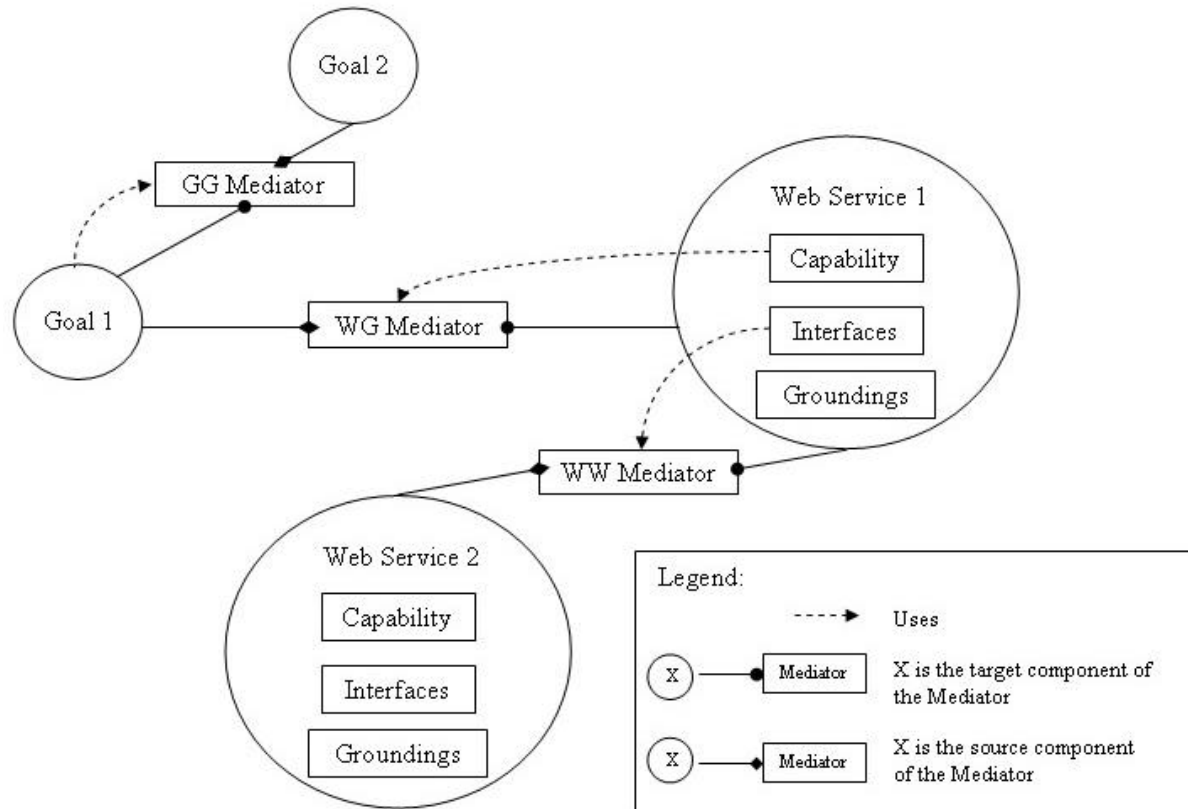
## Appendix

In the figure 4 below an example of how ggMediators, wgMediators and wwMediators are used is given.

Goal1 uses a ggMediator to connect to (e.g. import) Goal2. The source component of the ggMediator becomes Goal2 and the target component becomes Goal1.

The capability of the Web Service 1 uses a wgMediator to connect to Goal1. The source component of the wgMediator becomes Goal1 and the target component becomes Web Service 1.

Figure 4. An example of how ggMediator, wgMediator and wwMediator are used.



An interface of the Web Service 1 uses a wwMediator to connect to (e.g. invoke) Web Service 2. The source component of the wwMediator becomes Web Service 2 and the target component becomes Web Service 1.

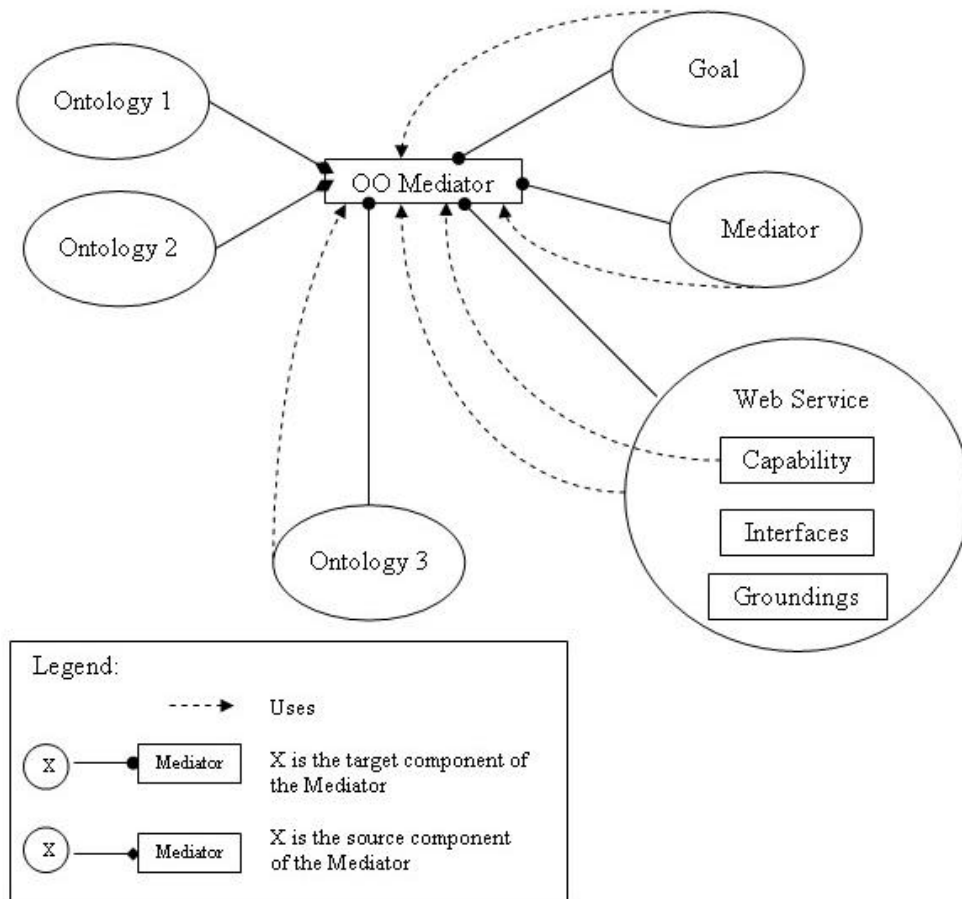
(Note: for the simplicity of the figure we did not consider chains of mediators of the same type).

In the figure 5 below an example of how ooMediators are used is given.

Ontology 3 imports Ontology 1 and Ontology 2 by using the ooMediator. The source components of the ooMediator become Ontology 1 and Ontology 2 and the target component Ontology 3.

A Goal imports Ontology 1 and Ontology 2 by using the ooMediator. The source components of the ooMediator become Ontology 1 and Ontology 2 and the target component the Goal.

Figure 5. An example of how ooMediator is used.



A Mediator (ggMediator, wgMediator, wwMediator or ooMediator) imports Ontology 1 and Ontology 2 by using the ooMediator. The source components of the ooMediator become Ontology 1 and Ontology 2 and the target component the Mediator.

A Web Service imports Ontology 1 and Ontology 2 by using the ooMediator. The source components of the ooMediator become Ontology 1 and Ontology 2 and the target component the Web Service.

The capability of a Web Service imports Ontology 1 and Ontology 2 by using the ooMediator. The source components of the ooMediator become Ontology 1 and Ontology 2 and the target component the Web Service.

---

[1] Notice that WMSO is based on F-Logic, therefore, not only instances but also concepts and variables can be bound to variables. Therefore it does not suffer from artificial modelling constraints of other languages that often require to model concepts by instances in order to refer to them via variables.

[2] Such an ontology actually represents a metaontology.

[3] The notation used for defining the elements of WSMO is based on F-Logic syntax.

[4] It's obvious, that this additional modeling element for describing ontologies only makes sense, if the logical framework underlying WSMO allows one to define and use methods within axioms. Since WSMO is principally based on F-Logic, this is clearly the case.

[5] Quoted from Web Service Architecture, [W3C Working Draft 8 August 2003](#).

[6] Notice that the capability of this web service will also provide a declarative description of this mapping, however, in addition a link to an implementation of the mapping is provided, too.