



D18v0.1. A Language Neutral API for Ontology Interchange

WSML Working Draft 26 April 2004

This version:

<http://www.wsmo.org/2004/d18/v0.1/20040426/>

Latest version:

<http://www.wsmo.org/2004/d18/v0.1/>

Previous version:

<http://www.wsmo.org/2004/d18/v0.1/20040409/>

Editors:

Dieter Fensel
Jos de Bruijn

Authors:

Dieter Fensel
Jos de Bruijn
Dumitru Roman
Uwe Keller

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. Ontologies](#)
 - [2.1 Non functional properties](#)
 - [2.2 Mediators](#)
 - [2.3 Concepts](#)
 - [2.4 Relations](#)
 - [2.5 Instances](#)
 - [2.6 Axioms](#)
 - [3. Use of a Language Neutral Ontology API](#)
 - [4. Conclusions and further directions](#)
 - [References](#)
 - [Acknowledgement](#)
-

1. Introduction

Ontologies are the key to link consensual real world semantics intended by humans with computers. They conceptualize a certain domain and add formal semantics to its definition. Defining an ontology language or an API for Ontology interchange can be viewed as an ontological problem, too. The domain is here the proper modeling primitives for describing ontologies and the ontology that consists of these modeling primitives is a Meta-Ontology, that defines how to define ontologies. An API can be language neutral or language dependent. The latter case assumes one language that is used on a world-wide scale to define ontologies. However, this is a very strong assumption and we will have to see whether this ideal state can be achieved after the days of Babel. With XML(S), RDF(S), and OWL we already have three W3C recommendations in this area and further may follow around the standardization of rule languages for the web. Notice, that OWL alone comprises three different languages and similar may happen for the rule language (and quadratic for the combination of rules and OWL). A language-neutral API introduces weaker assumptions. It fixes a meta model (i.e., it standardizes at the epistemological level and not at the logical level underneath [Brachman, 1983]). This deliverable defines this language-neutral meta model for defining ontologies, based on the ontology meta model presented in [Roman et al., 2004, chapter 3]. The actual language for defining the semantics of the elementary slots of this model are kept open.

This document is organized as follows: [Chapter 2](#) presents the meta model for ontologies to be used for a language-neutral ontology API. [Chapter 3](#) clarifies the use of the presented meta model in the development of applications, which use ontologies. Finally, we present conclusions and future directions in [Chapter 4](#).

2. Ontologies

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, which 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. In principle, an ontology constitutes of four main building blocks: concepts, relations, instances and axioms. In addition, we need non-functional description of ontologies and mediators to enable distributed ontology specification. An ontology is defined as follows [1]:

```
ontology[
nonFunctionalProperties => nonFunctionalProperties
usedMediators =>> mediatorDefinition
concepts =>> conceptDefinition
relations =>> relationDefinition
instances =>> instanceDefinition
axioms =>> axiomDefinition
]
```

Listing 1. Ontology definition

Non functional properties

The non functional properties of an ontology consist of the properties described in [Section 2.1](#).

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 we use ontology mediators for importing ontologies ([Section 2.2](#)).

Concepts

The set of concepts that belong to the represented ontology ([Section 2.3](#)).

Relations

The set of relations that belong to the represented ontology ([Section 2.4](#)).

Instances

The set of instances that belong to the represented ontology ([Section 2.5](#)).

Axioms

The set of axioms that belong to the represented ontology ([Section 2.6](#)).

2.1 Non functional properties

Non functional properties are defined as a set of tuples, where each tuple consists of a property and its value constraint. They consist of the [Dublin Core Metadata ElementSet \[Weibel et al.\]](#) plus the `version` element:

```
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
]
```

Listing 2. Non functional properties definition

Title

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

Creator

An entity primarily responsible for creating 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 are 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. The `Type` includes terms describing general categories, functions, genres, or aggregation levels for content.

Format

A 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 an 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.

2.2 Mediators

In this section, we introduce the notion of `mediators` and define the elements that are used in the description of a mediator. Mediators import ontologies and resolve possible representation mismatches between ontologies. The `mediator` is defined as follows:

```
mediatorDefinition[
  nonFunctionalProperties => nonFunctionalProperties
  sourceComponent =>> ontology
  mediationService => implementation
]
```

Listing 3. Mediator definition

Non functional properties

The non functional properties of a mediator consist of the properties described in the [Section 2.1](#).

Source component

The source component defines the ontologies which are imported.

Mediation Service

The `mediation service` points to a declarative description of the way in which ontologies are imported and the way in which possible representation mismatches between ontologies involved in the mediator are resolved and/or a web service that actually implements the way in which ontologies are imported and the way in which possible representation mismatches are resolved.

2.3 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 a 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, and then we can define 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 [Dean et al., 2004] as well.

Hence, we extract the following abstract description for concepts:

```
conceptDefinition :: axiomDefinition
conceptDefinition[
  superConcepts =>> conceptDefinition
  attributes =>> attributeDefintion
  methods =>> methodDefintion
]
```

Listing 4. Concept definition

Superconcepts

There can be a finite number of concepts that serve as `superconcepts` for some concept.

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 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.

```
attributeDefintion :: axiomDefinition
attributeDefintion[
  range => axiomDefinition
]
```

Listing 5. Attribute definition

Range

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` that can be invoked on each instance of a concept and in response return some result value.

A method 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, which is not explicitly contained in the set of input parameters. The computed value will then be returned to the invoker.

```
methodDefintion :: axiomDefinition
methodDefintion[
  range => axiomDefinition
  parameters => LIST(parameterDefinition)
]
```

Listing 6. Method definition

Range

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.

```
parameterDefinition :: axiomDefinition
parameterDefinition[
domain => axiomDefinition
]
```

Listing 7. Parameter definition

Domain

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

2.4 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, and reflexivity. Again, for the sake of simplicity we decide not to represent these common properties of relationships currently within the Meta-Ontology explicitly but implicitly by means of axioms.

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

```
relationDefinition :: axiomDefinition
relationDefinition[
parameters => LIST(parameterDefinition)
]
```

Listing 8. Relation definition

Parameters

A list of `parameter` descriptions specifying each of the concepts that are interrelated.

2.5 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.

```
instanceDefinition :: axiomDefinition
instanceDefinition[
instanceOf =>> conceptDefinition
attributeValues =>> attributeValueDefinition
]
```

Listing 9. Instance definition

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.

```
attributeValueDefinition :: axiomDefinition
attributeValueDefinition[
value => axiomDefinition
]
```

Listing 10. Attribute value definition

Value

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

2.6 Axioms

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

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

Listing 11. Axiom definition

Non functional properties

The `non functional properties` of an axiom consist of the properties described in [Section 2.1](#).

Defined by

The logical constraint expressed in the formal ontology language.

3. Use of the Language Neutral Ontology API

In the previous chapter, we have presented a meta model for a language-neutral ontology API. In this chapter we clarify the use of such an API. We clarify both the use of the meta model in order to achieve inter-operation between different ontology languages and the relation between the meta model and an API.

The meta model presented in this document can not straightforwardly be implemented as a concrete API for all possible ontology languages. In fact, it is not possible, even with this meta model, to use the same concrete API for different

ontology language, because of the particularities of the different ontology languages.

Different ontology languages have different syntax and different semantics. This makes it impossible to entirely use the same concrete API for these different languages. However, the different ontology languages all share the concepts in the meta model presented in the previous chapter. Each ontology language can be used to specify concepts, relations, axioms and instances. This shared meta model provides a basis for inter-operation between different ontology languages.

Each ontology language will require a different concrete API. However, the only difference between the different concrete APIs should be the logical expression, which is specified in the axiom definition (cf. [Section 2.6](#)). Because the larger part of the API is shared between ontology languages, it should be easy for a tool to switch between different implementations, which use different ontology languages, but which do commit to the same meta model.

The use of a common meta-model enables tools to abstract away (to some extent) from the particularities of ontology languages.

4. Conclusions and further directions

This document presents a meta model for a language-neutral API for Ontology interchange that conforms to the Web Service Modeling Ontology WSMO [\[Roman et al., 2004\]](#), which is currently under development in the context of the [SDK project cluster](#).

A concrete API, based on the meta model presented in this document, is currently under development in the context of the DIP (<http://dip.semanticweb.org/>) and SEKT (<http://sekt.semanticweb.org/>) projects.

References

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

[Brachman, 1983] R. J. Brachman: *What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks*. IEEE Computer 16(10): 30-36 (1983).

[Dean et al., 2004] M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein: *OWL Web Ontology Language Reference*, W3C Recommendation, 10 February 2004. Available at <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>.

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

[Roman et al., 2004] D. Roman, U. Keller, H. Lausen (eds.): *Web Service Modeling Ontology - Standard (WSMO - Standard)*, version 0.2 available at <http://www.wsmo.org/2004/d2/v02/>.

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

1998.

[Weibel et al. 1998] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf: *RFC 2413 - Dublin Core Metadata for Resource Discovery*, September 1998.

Acknowledgement

The work is funded by the European Commission under the projects [DIP](#), [Knowledge Web](#), [Ontoweb](#), [SEKT](#), [SWWS](#), [Esperanto](#), and [h-TechSight](#); by [Science Foundation Ireland](#) under the DERI-Lion project; and by the Vienna city government under the [CoOperate](#) program.

The editors would like to thank to all the [members of the WSML working group](#) for their advice and input into this document.

[1] The notation used for defining different concepts in this deliverable is based on F-Logic syntax.

\$Date: 2004/04/26 10:18:01 \$