D16v0.2 The WSML Family of Representation Languages

WSML Working Draft 21 September 2004

This version
http://www.wsmo.org/2004/d16/v0.2/20040921/

Latest version
http://www.wsmo.org/2004/d16/v0.2/

Editor:
Jos de Bruijn

Authors:
Jos de Bruijn
Douglas Foxvog
Holger Lausen
Eyal Oren
Dumitru Roman
Dieter Fensel

Reviewer:
Ian Horrocks

For printing and off-line reading, 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.

---

Namespaces

For referring to WSML variants of this version of WSML the following URIs should be used for identifying this particular version (v0.2) of each WSML variant:

<table>
<thead>
<tr>
<th>WSML Variant</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSML-Full</td>
<td><a href="http://www.wsmo.org/2004/d16/v0.2/#wsml-full">http://www.wsmo.org/2004/d16/v0.2/#wsml-full</a></td>
</tr>
<tr>
<td>WSML-Core</td>
<td><a href="http://www.wsmo.org/2004/d16/v0.2/#wsml-core">http://www.wsmo.org/2004/d16/v0.2/#wsml-core</a></td>
</tr>
<tr>
<td>WSML-Flight</td>
<td><a href="http://www.wsmo.org/2004/d16/v0.2/#wsml-flight">http://www.wsmo.org/2004/d16/v0.2/#wsml-flight</a></td>
</tr>
<tr>
<td>WSML-DL</td>
<td><a href="http://www.wsmo.org/2004/d16/v0.2/#wsml-dl">http://www.wsmo.org/2004/d16/v0.2/#wsml-dl</a></td>
</tr>
<tr>
<td>WSML-Rule</td>
<td><a href="http://www.wsmo.org/2004/d16/v0.2/#wsml-rule">http://www.wsmo.org/2004/d16/v0.2/#wsml-rule</a></td>
</tr>
</tbody>
</table>
Notice that this document subsumes all previous WSML specification deliverables (D16.x). More specifically, this document supersedes the following deliverables:

- D16.0: Languages for WSMO, http://www.wsmo.org/2004/d16/d16.0/v0.2/20040803/

Furthermore, this deliverable will (in future versions) implement WSML/RDF, WSML/OWL, WSML-Full, WSML-Rule, WSML-DL, and WSML-Flight, which were previously foreseen as separate deliverables.

This deliverable has been given the initial version 0.2 to stay compliant with the numbering of the deliverable D16.0, which had version number v0.2 at the time of creating D16.

With respect to previous versions of d16.0, d16.3 and d16.7, the following has changed:

- An appendix on datatype built-ins in WSML has been added.
- A chapter on implementations has been added.
- Changes to WSML-Core can be found in the WSML-Core changelog.
- A section explaining the concept and attribute definitions in WSML-Flight has been added, to be used for discussion.

To be discussed are the discussion issues raised in the Section 2.2.9, WSML-Core changelog and the proposed features for WSML-Flight in Section 2.4 WSML-Flight.

Abstract

We introduce WSML, a family of formal representation languages with its roots in Description Logics and Logic Programming. The conceptual modeling elements of WSML are based on the meta-model of WSMO.

The WSML variants have increasing expressiveness, starting with the intersection of Description Logic and Horn Logic and ending with full First-Order Logic with non-monotonic extensions.

- WSML-Core semantically corresponds with the intersection of Description Logic and Horn Logic, extended with extensive datatype support in order to be useful in practical applications. WSML-Core is fully compliant with a subset of OWL, albeit that the datatype support in WSML-Core is already beyond OWL, because the datatype support in OWL is very limited.
- WSML-Flight extends WSML-Core with more intuitive value restrictions and cardinality constraints. WSML-Flight is the preferred ontology modeling language.
- WSML-DL extends WSML-Core to a fully-fledged Description Logic.
- WSML-Rule extends WSML-Core to a fully-fledged Logic Programming language, including default negation and higher-order syntactical feature of F-Logic and HiLog.
- WSML-Full unifies all WSML variants under a common First-Order umbrella.
with non-monotonic extensions.

All WSML variants are described in terms of a normative human-readable syntax. Besides the human-readable syntax we provide an XML and an RDF syntax for exchange between machines. Furthermore, we provide a mapping to OWL for basic inter-operation with OWL ontologies.

Table of Contents

- 1. Introduction
- 2. The WSML Variants
  - 2.1 WSML-Full
  - 2.2 WSML-Core
    - 2.2.1 Basic WSML-Core Syntax
      - 2.2.1.1 Namespaces in WSML-Core
      - 2.2.1.2 Identifiers in WSML-Core
      - 2.2.1.3 Datatypes in WSML-Core
    - 2.2.2 WSML-Core Elements
      - 2.2.2.1 WSML Variant
      - 2.2.2.2 Namespace Definitions
      - 2.2.2.3 Ontologies
      - 2.2.2.4 Non-functional properties
      - 2.2.2.5 Import Ontology Definitions
      - 2.2.2.6 Mediators
      - 2.2.2.7 Concept Definitions
        - 2.2.2.7.1 Attributes
        - 2.2.2.8 Relation Definitions
        - 2.2.2.9 Functions
        - 2.2.2.10 Instance Definitions
        - 2.2.2.11 Axiom Definitions
        - 2.2.2.12 Datatype Definitions
    - 2.2.3 WSML-Core Logical Expressions
      - 2.2.3.1 Simple Logical Expressions
        - 2.2.3.1.1 Relation Expressions
        - 2.2.3.1.2 Molecules
        - 2.2.3.1.3 Datatype predicates
      - 2.2.3.2 Complex Logical Expressions
        - 2.2.3.2.1 Compound Logical Expressions
        - 2.2.3.2.2 Formulas
    - 2.2.4 Web Services in WSML-Core
      - 2.2.4.1 Goals
      - 2.2.4.2 Mediators
      - 2.2.4.3 Web Services
    - 2.2.5 Mapping WSML-Core to OWL DL
    - 2.2.6 Mapping OWL DL to WSML-Core
    - 2.2.7 WSML-Core in the WSMO Use Cases
    - 2.2.8 Conclusions
    - 2.2.9 WSML-Core Changelog
  - 2.3 WSML-Rule
  - 2.4 WSML-Flight
  - 2.5 WSML-DL
1. Introduction

The conceptual model and language for WSMO is described in [Roman et al. 2004]. However, different applications need different logical expressivity. Therefore, the WSML working group will provide several variants of WSML with different logical expressivity. In Chapter 2 we introduce these different variants and indicate in which deliverables they will be defined. In addition, different applications need different syntaxes. We introduce these various syntaxes in Chapter 3. We describe the implementations for WSML in Chapter 4. Finally, we present conclusions and future work in Chapter 4.

Table 1 provides a short overview of the different languages, distinguishing between the syntaxes for WSML and the WSML variants. The different WSML variants and the different syntaxes are described in more detail in the following chapters.

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Syntaxes for WSML</strong></td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Human-readable syntax for WSML</td>
</tr>
<tr>
<td>Section 3.1</td>
<td>XML syntax for WSML</td>
</tr>
<tr>
<td>Section 3.2</td>
<td>RDF syntax for WSML</td>
</tr>
<tr>
<td>Section 3.3</td>
<td>Mapping to OWL</td>
</tr>
<tr>
<td><strong>WSML variants</strong></td>
<td></td>
</tr>
<tr>
<td>Section 2.1</td>
<td>WSML-Full</td>
</tr>
</tbody>
</table>
2. The WSML Variants

In Figure 1 the different variants of WSML and the relation between them are shown. These variants differ in the logical expressivity they offer, and thus in the computational complexity they imply. By offering these variants, we allow users to make the trade-off between the provided expressivity and the implied complexity on a per-application basis.

**WSML-Core**
This language is defined by the intersection of Description Logic and Horn Logic, based on [Grosos et al., 2003]. It has the least expressive power of all the languages of the WSML family and therefore the most preferable computational characteristics. The main features of the language are the support for modeling classes, attributes, binary relations and instances.
Furthermore, the language supports class hierarchies, as well as relation hierarchies. WSML-Core provides extensive support for datatypes and datatype predicates, as well as user-defined datatypes. WSML-Core is based on a subset of OWL, called OWL’ [de Bruijn et al., 2004], with a datatype extension based on OWL-E [Pan and Horrocks, 2004], which adds richer datatype support to OWL.

**WSML-DL**

This language is an extension of WSML-Core which fully captures the Description Logic SHOIN(D), which underlies the (DL species of the) Web Ontology Language OWL [Dean et al., 2004]. The language can be seen as an alternate syntax for OWL DL, based on the WSMO conceptual model.

**WSML-Flight**

This language is an extension of WSML-Core with several features from OWL Full, such as meta-classes, and several other features, such as constraints. WSML-Flight is based on OWL Flight [de Bruijn et al., 2004a], which adds features such as constraints and meta-classes to a subset of OWL DL.

**WSML-Rule**

This language is an extension of WSML-Core that supports Horn Logic based on minimal model semantics. Furthermore, the language also captures several extensions developed in the area of Logic Programming, such as default negation, F-Logic [Kifer et al., 1995] and HiLog [Chen et al., 1993].

**WSML-Full**

WSML-Full unifies WSML-DL and WSML-Rule under a First-Order umbrella with extensions to support specific features of WSML-Rule, such as minimal model semantics and default negation. WSML-Full is of all WSML variants the closest to the WSMO conceptual model. The syntax for WSML-Full is in fact defined as the basic syntax for WSMO [Roman et al., 2004].

The only language currently specified in this document is WSML-Core. The expressiveness of WSML-Full corresponds with the intuitive semantics of the basic logical language for WSMO and the grammar is specified in [Roman et al., 2004].

### 2.1. WSML-Full

WSML-Full corresponds in expressiveness with the logical language defined in Chapter 7 of [Roman et al., 2004]. Furthermore, the syntax of WSML-Full captures all aspects of the WSMO meta-model.

The grammar for WSML-Full can be found in Appendix B of [Roman et al., 2004].

A future version of this deliverable will contain a complete language specification for WSML-Full.

### 2.2. WSML-Core

The major goal of the WSML working group is to develop a formal language for the description of Semantic Web Services based on the WSMO conceptual model. As is described in the introduction to this Chapter, there are several WSML languages with different underlying logical formalisms. The two main logical formalisms exploited in different WSML languages are Description Logics [Baader et al., 2003] (exploited in WSML-DL) and Rule Languages [Lloyd, 1987] (exploited in WSML-Rule). WSML-Core (described in this section) exploits the intersection of both formalisms.
WSML-Core is an ontology modeling language, based on the semantics of OWL DL\cite{deBruijn2004}, which is based on \cite{Grosof2003}. Furthermore, WSML-Core uses a restricted form of the OWL-E datatype extension \cite{Pan2004}. This extension to OWL DL was described in \cite{deBruijn2004a}. The modeling constructs of WSML-Core are based on the conceptual model of ontologies presented in WSMO \cite{Roman2004}. Because WSML-Core is based on OWL DL and there exists a translation from OWL DL to plain (function- and negation-free) Datalog, the decidability and complexity results of Datalog apply to WSML-Core as well. The most important result is that Datalog is data complete for P, which means that query answering can be done in polynomial time.

Section 2.2.2 describes all the elements of the WSML-Core language. Section 2.2.3 describes the relation between WSML-Core and Web Service-related specifications, such as goal, mediator and web service descriptions. Section 2.2.4 specifies the mapping from WSML-Core to the OWL DL abstract syntax, thereby actually defining the semantics of WSML-Core. Section 2.2.5 provides a mapping from the OWL DL abstract syntax to WSML-Core, thereby specifying the inter-operability between OWL and WSML-Core. Section 2.2.6 summarizes the applicability of WSML-Core to the use cases of WSMO D3.2 \cite{Stollberg2004}. Finally, we present some conclusions in Section 2.2.7.

### 2.2.1 Basic WSML-Core Syntax

The syntax for WSML-Core has a frame-like style, where a collection of information about a class or property is given in one large syntactic construct, instead of being divided into a number of atomic chunks. It is possible to spread the information about a particular class, relation, instance or axiom over several constructs, but we do not recommend this. In fact, in this respect, WSML-Core is similar to OIL \cite{Fensel2001}, which also offers the possibility of either grouping descriptions together in frames or spreading the descriptions throughout the document. One important difference with OIL (and OWL) is that attributes defined locally for a class are truly local and do not have a meaning outside of the context of a class definition.

Identifiers in WSML-Core follow the conventions of WSMO \cite{Roman2004}. An identifier is either a URI or a Qualified Name (QName). QNames without a prefix are resolved with the default namespace. Conventionally, internal words within QNames are designated by initial capital letters, not by hyphens or underscores. Variable names start with an initial question mark, "?". Depending upon context, a variable can stand in for a concept, attribute, datatype, instance, attribute value, or literal; it may not stand in for a language keyword or special symbol.

Argument lists are separated by commas. Statements in WSML-Core start with a keyword and can be multi-lined. There is no specific end-of-statement syntax -- a keyword for a new statement is sufficient.

#### 2.2.1.1 Namespaces in WSML-Core

WSML-Core inherits the namespace mechanism of WSMO, which is inherited from XML. Each element in a WSML-Core ontology is created in the target namespace of the ontology. The target namespace of an ontology is by default the identifier of the ontology, which is typically a URI.
2.2.1.2 Identifiers in WSML-Core

WSML-Core inherits the part of the use of identifiers from WSMO. In WSML-Core, an identifier is either a QName (Qualified Name), a URI or a Literal. WSML-Core does not support anonymous IDs. As in RDF, a QName is equivalent to the URI that is obtained by concatenating the namespace (to which the prefix refers) with the local part of the QName. Therefore, a QName can be seen as an abbreviation for a URI which enhances the legibility of the specification.

A URI in WSML-Core is enclosed in double angle brackets ‘<<’ and ‘>>’, a literal is enclosed in double quotes ‘”’ and a variable start with a question mark ‘?’. The type of a literal is indicated with the double caret ‘^^’, e.g. “4^^xsd:integer stands for the integer 4.

Notice that the vocabulary of WSML-Core is separated similarly to OWL DL. More precisely, the following sets of identifiers are pairwise disjoint:

- The WSML-Core keywords
- The set of datatype identifiers
- The set of attribute identifiers
- The set of data-values relation identifiers
- The set of general relation identifiers
- The set of concept identifiers
- The set of relation identifiers

2.2.1.3 Datatypes in WSML-Core

The treatment of datatypes in WSML-Core are treated inherited from WSMO [Roman et al., 2004]. The recommended datatypes in WSML-Core are the XML Schema datatypes. In fact, any implementation of WSML-Core is required to support the xsd:string and the xsd:integer datatypes. Furthermore, the following built-in predicates are to be supported:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
<th>Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Integer addition</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Integer subtraction</td>
<td>xsd:integer</td>
</tr>
<tr>
<td>*</td>
<td>Integer multiplication</td>
<td>xsd:integer</td>
</tr>
<tr>
<td>/</td>
<td>Integer division</td>
<td>xsd:integer</td>
</tr>
<tr>
<td>=</td>
<td>Equality</td>
<td>xsd:integer, xsd:string</td>
</tr>
<tr>
<td>~=</td>
<td>Inequality</td>
<td>xsd:integer, xsd:string</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater-than comparison</td>
<td>xsd:integer</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less-than comparison</td>
<td>xsd:integer</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less-than or equal comparison</td>
<td>xsd:integer</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater-than or equal comparison</td>
<td>xsd:integer</td>
</tr>
</tbody>
</table>

2.2.2. WSML-Core Elements
In this section we explain the ontology modeling elements in the WSML-Core language. The modeling elements are based on the WSMO conceptual model of ontologies [Roman et al., 2004]. Each description is accompanied by an example. Most of the examples were taken from [Stollberg et al., 2004].

The recommended namespace to be used to reference this version of WSML-Core is http://www.wsmo.org/2004/d16/v0.2/20040921#wsml-core.

Please note that WSML-Core definitions should follow the ordering as presented in the sections below.

2.2.2.1 WSML Variant

Any WSML specification document needs to start with the wsmlVariant keyword, followed by an identifier for the WSML variant which is used for the WSML specification. The proper identifier for the version of WSML-Core specified in this document is http://www.wsmo.org/2004/d16/v0.2/#wsml-core. The specification of the wsmlVariant is required for each WSML specification. The following illustrate the WSML variant reference for a WSML-Core specification:

wsmlVariant <<http://www.wsmo.org/2004/d16/v0.2/#wsml-core>>

2.2.2.2 Namespace Definitions

Before the actual ontology definition, there is an optional block of namespace definitions, which is preceded by the namespace keyword. The namespace keyword is optionally followed by the default namespace and a number of namespace definitions. Each namespace definition, except for the default namespace, consists of the chosen prefix, a `:` and the URI, which identifies the namespace. Finally, the (optional) target namespace is specified with the use of the targetNamespace keyword. If the target namespace is designated by the URI which is specified by the ontology declaration, the targetNamespace line is redundant and may be omitted.

An example:

namespace <<http://www.example.org/example/>>
dc: <<http://purl.org/dc/elements/1.1#>>
xsd: <<http://www.w3.org/2001/XMLSchema#>>
targetNamespace: <<http://www.example.org/example/>>

2.2.2.3 Ontologies

An ontology definition in WSML-Core starts with the ontology keyword followed by an identifier, which serves as the identifier of the ontology. When a full URI is used for this identifier and no explicit target namespace definition exists, this identifier is used as the target namespace of the definitions in the ontology specification document.

An example:

ontology <<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/po.wsml>>
2.2.2.4 Non-functional properties

Following the namespace definition block is an optional non-functional properties definition block, identified by the keyword nonFunctionalProperties. Following the keyword is a list of properties and property values. The recommended properties are the properties of the Dublin Core [Weibel et al. 1998], but the list of properties is extensible and thus the user can choose to use properties coming from different sources. WSMO defines one property, absent in the Dublin Core, namely version. The value of each of the properties is either a URI or a literal. If a property has multiple values, these are separated by commas.

An example:

```xml
<nonFunctionalProperties>
  <dc:title hasValues> "WSML example collection"</dc:title>
  <dc:subject hasValues> "family"</dc:subject>
  <dc:description hasValues> "fragments of a family ontology to provide WSML examples"</dc:description>
  <dc:contributor hasValues> <![CDATA[<<http://homepage.uibk.ac.at/~c703240/foaf.rdf>>,
                                <<http://homepage.uibk.ac.at/~csaa5569>>,
                                <<http://homepage.uibk.ac.at/~c703239/foaf.rdf>>]]></dc:contributor>
  <dc:date hasValues> "2004-09-20"</dc:date>
  <dc:type hasValues> "<http://www.wsmo.org/2004/d2/v1.0/20040827/#ontologies>"</dc:type>
  <dc:format hasValues> "text/plain"</dc:format>
  <dc:language hasValues> "en-US"</dc:language>
  <dc:rights hasValues> "<http://www.deri.org/privacy.html>"</dc:rights>
  <version hasValues> "$Revision: 1.7 $"</version>
</endNonFunctionalProperties>
```

2.2.2.5 Importing Ontologies

Following the nonFunctionalProperties definition block is an optional imported ontologies definition block, identified by the keyword importOntologies. Following the keyword is a list of URIs identifying the ontologies being imported. An importOntologies definition serves to merge ontologies, just as an OWL import statement does. This means the resulting ontology is the union of all axioms in the importing and imported ontologies. Please note that recursive importation of ontologies is also supported. This means that if an imported ontology specifies any importation of ontologies, also the axioms in these ontologies are taken in the union.

An example:

```xml
<importOntologies>
  <!-- http://www.wsmo.org/ontologies/datetime# -->
  <!-- http://www.wsmo.org/ontologies/currency# -->
</importOntologies>
```

2.2.2.6 Mediators

Mediators are used to import other ontologies and resolve heterogeneity (typically through ontology mapping). This concept of mediation between ontologies is more flexible than the importOntologies statement, which is used to import an WSML ontology into another WSML ontology. The ontology import mechanism simply appends the definitions in the imported ontology to the importing ontology. The importing ontology should contain the axioms to relate the definitions in the imported ontology to the local definitions. This mechanism enforces a strong coupling between
the ontologies, which is undesirable in the general case and it does not allow importing parts of ontologies.

In fact, importing ontologies can be seen as a simple form of mediation, in which no heterogeneity is resolved. However, in the general case there is overlap between the different ontologies, which requires mediation.

By externalizing the mediation from the ontology, WSMO allows loose coupling of ontologies; the mediator is responsible for relating the different ontologies to each other. A mediator can provide, for example, translation services between ontology languages or adjust for argument-order differences. This notion of mediators corresponds with ooMediators in WSMO [Roman et al., 2004].

The (optional) mediators definition block is identified by the usedMediators, followed by one or more identifiers of WSMO ooMediators.

An example:

```
usedMediators ooMediator
    { <<http://www.wsmo.org/ontologies/dateTime>>, 
      <<http://www.wsmo.org/ontologies/trainConnection>>, 
      <<http://www.wsmo.org/ontologies/purchase>>, 
      <<http://www.wsmo.org/ontologies/location>> }
```

### 2.2.2.7 Concepts

A concept definition starts with the concept keyword, which is followed by the identifier of the concept. This is followed by zero or more implicitly conjoined direct superconcept definitions (using the subConceptOf keyword followed by the identifier for a named concept), an optional nonFunctionalProperties block, and zero or more attribute specifications. An attribute specification consists of the identifier of an attribute, the ofType keyword and the identifier of a concept of which the attribute values must an instance.

A concept can only be a subConceptOf named concepts. WSML-Core does not allow complex concepts definitions as superconcepts, as is allowed in OWL.

An attribute value can only be restricted to a datatype. In addition to the usual built-in datatypes, user-defined datatypes are permitted. Note however that we do not allow datatype restrictions of higher arity, as in OWL Flight. Instead, such a restriction should be introduced in the logical definition.

An attribute definition of the form A ofType D, where A is an attribute identifier and D is a datatype identifier, is a constraint on the values for attribute A. If the value for the attribute A is not of type D, the constraint is violated and the attribute value is inconsistent with respect to the ontology. This notion of constraints corresponds the usual database-style constraints and also the the universal values restrictions for DatatypeProperties in OWL.

In the example below, the attributes length and weight have as their type the datatype xsd:integer. bmi has datatype xsd:float. BMI, in this case, stands for Body Mass Index, which is a certain ratio between the length, the weight and the age of a person. The Body Mass Index is calculated through the user-defined datatype predicate bodyMassIndex (see Section 2.2.2.9 for the definition). The calculation of the Body
Mass Index is part of the necessary concept definition.

An example:

```xml
<concept Human subConceptOf Primate, LegalAgent
nonFunctionalProperties
dc:description hasValues "Members of the species Homo sapiens"
endNonFunctionalProperties
length ofType xsd:integer
weight ofType xsd:integer
bmi ofType xsd:float
definedBy
  constraint wsml:not(bodyMassIndex [range hasValue ?b, length hasValue ?l, weight hasValue ?w])
  and ?x memberOf Human and
  ?x[length hasValues ?l, weight hasValues ?w, bmi hasValues ?b].
```

2.2.2.7.1 ATTRIBUTES

In WSML-Core the keyword ofDataType is used for the datatype attribute definitions. No general attribute definitions are allowed, as in WSML-Full.

OWL allows local universal value restrictions for properties. However, the semantics for these restrictions is not intuitive and does not correspond to the semantics of attribute type definitions in WSML-Full. Therefore, WSML-Core does not allow for the definition of general attributes.

2.2.2.8 Relations

A relation definition starts with the relation keyword, which is followed by the identifier of the relation and an optional specification of direct super-relations through the subRelationOf keyword. This is followed by an optional nonFunctionalProperties block. In WSML-Core, relations are restricted to binary predicates, which correspond with ObjectProperties in OWL DL. Parameters can be used to restrict the domain and range of the property (denoted by the domain and range keywords, respectively). It fact, no other parameters are allowed.

Particular aspects of the relation are denoted by particular keywords. The relation can be a specialization of a different named relation, denoted with the subRelationOf keyword. Transitive, symmetric, and inverse properties are denoted with the transitive, symmetric, and inverseOf keywords, respectively. For a complete account, see Table 1 in Section 2.2.4.

An example:

```xml
relation hasAncestor subRelationOf hasRelative
transitive
nonFunctionalProperties
dc:description hasValues "Relation between ancestors"
endNonFunctionalProperties
domain ofType Person
range ofType Person
```

Besides defining a relation over two concepts, it is also possible to define a relation
over a concept and a datatype, corresponding to the DatatypeProperties in OWL. If a certain relation is a Datatype relation, this has to be indicated through the `dataRelation` keyword. Note that a `dataRelation` can not be transitive, symmetric or inverse. Therefore, the `transitive`, `symmetric`, and `inverseOf` keywords are not allowed to occur if the `dataRelation` keyword occurs. A dataRelation can have as its range a datatype, but has as its domain a concept. Thus, a dataRelation can only be a subrelation of another dataRelation.

An example:

```xml
relation length
  dataRelation
  nonFunctionalProperties
dc:description hasValues "Length indicator"
  endNonFunctionalProperties
  range ofType xsd:integer
```

### 2.2.2.9 Functions

WSML-Core allows the user to create user-defined datatype predicates and user-defined datatypes just as OWL DL does. A datatype expression starts with the `function` keyword, followed by the name (identifier) of the function (datatype predicate). This is followed by an optional `nonFunctionalProperties` block and then by an optional range and optional set of parameters and a datatype expression preceded by the `definedBy` keyword. The syntax for the datatype expressions is explained in Section 3.

The example below defines a new predicate for calculating the Body Mass Index. The symbols used for the datatype functions are '/', '=' and '*'. These symbols stand for numerical division, equality and multiplication, respectively. Notice that these symbols are shortcuts for datatype predicates. The translation of symbols to datatype predicates is given in Appendix D.1.

An example of a user-defined datatype predicate (function):

```xml
function bodyMassIndex
  nonFunctionalProperties
dc:description hasValues "Calculates the Body Mass Index. This version is not really accurate, but hopefully shows how a datatype predicate is created. bmi = kg/m^2. Notice that the link between the parameters and the variables in the logical expression are a bit strange. This is inherited from d2. We need some clarification here."
  endNonFunctionalProperties
  length ofType xsd:integer
  weight ofType xsd:integer
  range ofType xsd:float
  definedBy
    bodyMassIndex
```

### 2.2.2.10 Instances

An instance definition starts with the `instance` keyword, followed by the identifier of
the instance, the `memberOf` keyword and the name of the concept to which the instance belongs. An instance corresponds with an individual in OWL DL. The `memberOf` keyword identifies the concept to which the instance belongs. This definition is followed by the attribute values associated with the instance. Each property filler consists of the property identifier, the keyword `hasValues`[1] and the value for the attribute. If an attribute has a datatype as its range, the attribute value should be a (possibly typed) literal. Note that both literals in the example are typed. They are both of type `xsd:string`. Notice that a literal written between single quotes (‘…’) is interpreted as a typed literal of type `xsd:string`.

An example:

```
instance innsbruckHbf memberOf station
  name hasValues 'Innsbruck Hauptbahnhof'
  code hasValues "INN"^^xsd:string
  locatedIn hasValues loc:innsbruck
```

The second way of accessing instances, as described in [Roman et al., 2004], is by providing a link to an instance store. Instance stores contain large numbers of instances and they are linked to the ontology with the use of a mediator. We do not restrict the user in the way an instance store is linked to a WSML-Core ontology. This would often be done outside of the ontology, since an ontology a shared and reusable for different instance stores.

### 2.2.2.11 Axioms

An axiom definition starts with the `axiom` keyword, followed by the name (identifier) of the axiom. This is followed by an optional `nonFunctionalProperties` block and then by a logical expression preceded by the `definedBy` keyword. The language allowed for the logical expression is explained in Section 2.2.3 and the allowed expressions are detailed in Table 2 in Section 2.2.4.

An example:

```
axiom humanDefinition
  definedBy
    ?x memberOf Human equivalent
    ?x memberOf Animal and
    ?x memberOf LegalAgent.
```

### 2.2.2.12 Datatype definitions

WSML-Core allows the user to create and user-defined datatypes.

An example of a user-defined datatype:

```
datatype positiveInt
  nonFunctionalProperties
    dc:description hasValues "Positive integers."
endNonFunctionalProperties
  ?value ofType xsd:integer
  definedBy
    ?value > "0"^^xsd:integer.
```
2.2.3. WSML-Core Logical Expression Syntax

In this chapter we explain the syntax of logical expressions used in the WSML-Core language. Each description is accompanied by an example.

Logical expressions may be simple or complex. A logical expression is terminated by a period. Simple logical expressions can be combined to form complex expressions.

WSML-Core has the following simple logical expressions:

- Relation Expressions
- Molecules
- Datatype predicates

WSML-Core has the following complex logical expressions:

- Compound Logical Expressions
- Rules

Notice that the use of the logical expression syntax is restricted by the allowed logical expressions of Table 2 in Section 2.2.4 in order to allow for a direct translation into OWL DL.

2.2.3.1 Simple Logical Expressions

The three basic types of simple logical expressions are relation expressions, molecules and datatype predicates.

2.2.3.1.1 Relation Expressions

A relation logical expression consists of a predicate identifier followed by the comma-separated arguments of the predicate, enclosed by parentheses. The predicate corresponds to a regular relation or a dataRelation. A relation value logical expression is one that can be expressed by a single binary relation (whether dataRelation or not) relating the two arguments. The first argument is an instance of a concept. The second argument is an instance of a concept if the relation is a regular relation and a data value if the relation is a dataRelation.

An example:

```
ageInYears(doug, 99)
```

2.2.3.1.2 Molecules

A molecule in WSML-Core is either a concept molecule or an instance molecule. An instance molecule is one of the following statements:
- A concept membership assertion of the form $I \text{memberOf} C$, where $I$ is an instance identifier and $C$ is a concept identifier.
- An attribute value list of the form $[A_1 \text{hasValues} v_1, \ldots, A_n \text{hasValues} v_n]$, where $I$ is an instance identifier, $A_1, \ldots, A_n$ are attribute identifiers and $v_1, \ldots, v_n$ are either instance identifiers or data values.

A concept membership assertion of the form $I \text{memberOf} C$ states that $I$ is an instance of concept $C$. An attribute value list specifies the values for certain attributes for this particular instance.

Two examples:

```
myCC memberOf creditCard
myCC[number hasValues '12345', owner hasValues jos]
```

A concept molecule is one of the following statements:

- A subconcept assertion of the form $C \text{subConceptOf} D$, where $C$ and $D$ are concept identifiers.
- An attribute definition list of the form $C[A_1 \text{ofType} D_1, \ldots, A_n \text{ofType} D_n]$, where $I$ is an instance identifier, $A_1, \ldots, A_n$ are attribute identifiers and $D_1, \ldots, D_n$ are either concept identifiers or datatype identifier.

A subconcept assertion of the form $C \text{subConceptOf} D$ states that $C$ is a subconcept of $D$, which means that each instance of $C$ is also an instance of $D$. An attribute definition of the form $C[A \text{ofType} D]$ asserts that for each instance of concept $C$, each value for the attribute $A$ is of the type $D$.

Two examples:

```
creditCard subConceptOf paymentMethod
creditCard[number ofType xsd:string, owner ofType person]
```

### 2.2.3.1.3 DATATYPE PREDICATES

A datatype predicate consists of a predicate identifier and parenthesis-delimited, comma-separated list of arguments. The number of arguments depends on the arity of the predicate. Each argument must be a data value. The satisfiability of the datatype predicate is determined by an external datatype oracle.

An example:

```
wsml:numeric-equals(3,4)
```

The user is free to choose the built-in predicates as long as the implementation knows how to handle the built-ins (similar to the allowed datatypes). However, we recommend a minimal list of supported datatype predicates, which are listed in Appendix D.1. These predicates operate on the domains of xsd:string and xsd:integer, which are the basic datatypes, which should be supported by any WSML-Core implementation.
For certain datatype predicates we allow infix notation for certain functions and certain relations. More specifically, we allow infix notation for the following built-in functions: numeric addition ("+") subtraction ("-"), multiplication ("*") and division ("/"). We allow infix notation for the following built-in relations: numeric and string equality ("="), numeric and string inequality ("\neq"), and the following numeric comparisons: greater than (">"), less than (<), greater or equal (">=) and less or equal ('<'). See appendix D.2 for a list of syntactic shortcuts and a translation to datatype predicates.

2.2.3.2 Complex Logical Expressions

WSML-Core has the following complex logical expressions:

- Compound Logical Expressions, which consist of several molecules and/or (datatype) predicates, separated by the and keyword.
- Formulas, which consist of a logical expression, an implication symbol, and a logical expression. Both logical expressions can be either a simple or a compound logical expression.

2.2.3.2.1 Compound Logical Expressions

A compound logical expression consists of a number of simple logical expressions connected with the keyword and. The compound logical expression is satisfied if each of the simple logical expressions is satisfied.

An example:

```
myCC memberOf creditCard and myCC[number hasValues '12345', owner hasValues jos]
```

Molecules involving the same instance or concept, occurring in a compound logical expression, can be collapsed into one compound molecule to allow for more concise syntax. The example above can be thus rewritten:

```
myCC[number hasValues '12345', owner hasValues jos] memberOf creditCard
```

2.2.3.2.2 Formulas

A formula in WSML-Core consists of two (simple or compound) logical expressions, separated by an implication symbol. This implication symbol can be left implication ("<-"), right implication ("->") or dual implication ("<->"). In formulas, variables are allowed to occur in the place of identifiers.

- Left implication: $E_1 <- E_2$, where $E_1$ and $E_2$ are (simple or compound) logical expressions. $E_1$ is true wrt. a certain variable binding, if $E_2$ is true wrt. the same variable binding. $E_2$ is called the antecedent and $E_1$ is called the consequent of the formula.
- Right implication: $E_1 -> E_2$, where $E_1$ and $E_2$ are (simple or compound) logical expressions. $E_2$ is true wrt. a certain variable binding, if $E_1$ is true wrt. the same variable binding. $E_1$ is called the antecedent and $E_2$ is called the consequent of the formula.
- Dual implication: $E_1 \leftrightarrow E_2$, where $E_1$ and $E_2$ are (simple or compound) logical expressions. $E_1$ is true wrt. a certain variable binding if and only if $E_2$ is true wrt. the same variable binding. A dual implication $E_1 \leftrightarrow E_2$ is actually equivalent to the two implications: $E_1 \leftarrow E_2$ and $E_1 \rightarrow E_2$. Therefore, both $E_1$ and $E_2$ are both the antecedent and the consequent of the formula.

Note that variables occurring in the consequent of a formula must also occur in the antecedent of the same formula. Note also that all variables are implicitly universally quantified outside of the formula.

An example:

```xml
?x memberOf Human <-> ?x memberOf Animal and ?x memberOf LegalAgent.
```

## 2.2.4. Web Services in WSML-Core

Readers not interested in Web Service specification can skip this section, since WSML-Core is principally an ontology specification language.

This section describes Web Service-specific modeling features in WSML-Core. Notice that Web Service description is limited in WSML-Core, because WSML-Core has limited expressiveness and is intended as light-weight ontology language. For fully-fledged Web Service modeling we refer the reader to WSML-Rule (Section 2.3) and WSML-Full (Section 2.1).

### 2.2.4.1 Goals

Goal specification is inherited from WSML-Full with the exception that logical expressions are limited to that part of WSML-Full logical expressions that fall inside WSML-Core. See Section 2.2.3 for an elaborate description of logical expressions in WSML-Core.

Notice that in goals, logical expressions only occur in postconditions and effects.

Besides the logical expressions in postconditions and effects, none of the elements of a goal definition have a meaning in a logical languages. WSML-Core also does not prescribe a relationship between postconditions and effects. It is up to the user of the definitions to use them appropriately. WSML deliverable D5.1 [insert ref] contains suggestions on how to use these elements for Web Service discovery.

Ontology imported by the goal, either through an `importOntologies` or a `usedMediators` statement, are logically appended to both the postcondition and the effect. The resulting postcondition is the union of the axiom on the postcondition and the set of axioms in the ontolog(y)(ies). Similar for the effect. Notice that a mediator used to import an ontology typically contains axioms which resolve heterogeneity between ontologies. From the point-of-view of the goal, these logical axioms are part of the imported ontology and thus also of the logical union with the postcondition/effect.

### 2.2.4.2 Mediators

The specification of mediators in WSML-Core is completely inherited from
WSML-Full. None of the elements in a mediator has any meaning in the logical language.

2.2.4.3 Web Services

Web Services in WSML-Core are similar to goals in the sense that they are inherited from WSML-Full and that only the logical expressions in the capability of the Web Service have a meaning in the logical language. Also, as in goals, there is no formal relationship between the different elements of a Web Service capability. As in goals, axioms of all imported ontologies are logically appended to each precondition, assumption, postcondition and effect. The resulting precondition/assumption/postcondition/effect is the union of the original axiom and the axioms of the imported ontologies.

In the remainder of this section, we are only concerned with ontology modeling in WSML-Core.

2.2.5. Mapping WSML-Core to OWL DL∗

In this chapter we define the semantics of WSML-Core through a mapping to the OWL DL∗ abstract syntax [de Bruijn et al., 2004], which is a subset of the OWL abstract syntax. The syntax and semantics of the OWL abstract syntax can be found in [Patel-Schneider et al., 2004].

Table 1 shows the mapping between the WSML-Core conceptual syntax and OWL DL∗. In the table, C and D refer to named concepts (classes in OWL), T refers to a datatype, R refers to an ObjectProperty, U refers to a DatatypeProperty, A refers to an attribute (this can be either an ObjectProperty or a DatatypeProperty in OWL), F refers to a datatype predicate, O refers to an ontology and M refers to a mediator.

Through the mapping to the OWL abstract syntax, the precise semantics of the WSML-Core primitives is defined. Please note in WSML-Core, each construct can have non-functional properties associated with it. This is not reflected in the table. Rather, there is a separate row in the table, which addresses the non-functional properties and the way they are reflected in the OWL abstract syntax. The annotation properties occur inside each class, property or instance definition.

We need to make two notes here about the WSML-Core syntax compared to the syntax presented in WSMO-Standard [Roman et al., 2004]. First of all, the attribute definitions in the concept signature (see the second row of Table 1) are interpreted as universal value restrictions in OWL DL∗. This does not correspond with the intuition behind these attribute definitions, as was pointed out in [de Bruijn et al., 2004]. Second, we on the one hand restrict the WSMO-Standard syntax for relations by allowing only two parameters (domain and range) in order to restrict the relations to those of binary arity. On the other hand, we extend the WSMO-Standard syntax with the keywords subRelationOf, transitive, symmetric, and inverseOf in order to capture the epistemology of the ObjectProperties in OWL.

In Table 2, we have identified exactly which logical expressions can be translated to OWL DL∗. In the table, C and D refer to named concepts (classes in OWL), T refers to a datatype, R and S refer to ObjectProperties, U refers to a DatatypeProperty, A
refers to an attribute (this can be either an ObjectProperty or a DatatypeProperty in OWL), and $F$ refers to a datatype predicate. These logical expressions add little\cite{footnote} in expressivity over the conceptual syntax, but sometimes the user prefers a different way of modeling axioms. Furthermore, this syntax for logical expressions can be directly used in goal descriptions and capability descriptions of web services for the modeling of assumptions, pre-conditions, effects and post-conditions. All and only those logical expressions that are either in the first column of Table 2 or can be reduced to logical expressions in Table 2 are valid in WSML-Core.

<table>
<thead>
<tr>
<th>WSML-Core conceptual syntax</th>
<th>OWL DL’ Abstract syntax</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Declarations</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ontology</strong> $O$</td>
<td>Ontology($O$)</td>
<td>All definitions and axioms in the ontology are nested inside the Ontology statement, according to the other mappings in this table.</td>
</tr>
<tr>
<td><strong>concept</strong> $C$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subConceptOf $D_1$, ..., $D_n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_{i+1}$ ofType $T_i$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_n$ ofType $T_n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>relation</strong> $R$ [subRelationOf $R_1$, ..., $R_n$]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[transitive]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[symmetric]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[inverseOf($R_0$)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[domain ofType $C_1$, ..., $C_n$]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[range ofType $D_1$, ..., $D_n$]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>relation</strong> $U$ [subRelationOf $U_1$, ..., $U_n$] dataRelation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[domain ofType $C_1$, ..., $C_n$]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[range ofType $T_1$, ..., $T_n$]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>datatype</strong> $T$</td>
<td>DatatypeExpression($T$ and(domain($T_1$), deCom)))</td>
<td></td>
</tr>
<tr>
<td>$y$ ofType $T_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>definedBy deCom</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>function</strong> $F$</td>
<td>DatatypeExpression($F$ and(domain($T_1$, ..., $T_n$), deCom)))</td>
<td></td>
</tr>
<tr>
<td>$[\text{range ofType } T_1]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$[y_2$ ofType $T_2]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$[y_k$ ofType $T_k]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>definedBy deCom</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
instance o memberOf C₁, ..., Cₙ
  A₁ hasValues o₁
  ...
  Aᵢ hasValues oᵢ
  Aᵢ₊₁ hasValues vᵢ₊₁
  ...
  Aₙ hasValues vₙ

Individual(ο type(C₁) ...
  type(Cₙ)
  value(A₁ o₁) ...
  value(Aᵢ oᵢ)
  value(Aᵢ₊₁ vᵢ₊₁) ...
  value(Aₙ vₙ))

Extra-Logical Declarations

nonFunctionalProperties
  P₁ v₁
  ...
  Pᵢ vᵢ
  [version v₀]

annotation(P₁ v₁)
  ...
  annotation(Pᵢ vᵢ)
  [annotation(owl:versionInfo v₀)]

Annotation properties are nested inside other definitions, such as ontology, individual, class and property definitions. For some strange reason, if annotations occur on the ontology level, they should be written with a capital 'A' (e.g. Annotation(owl:versionInfo "$Revision: 1.7 $").)

importOntologies
  O₁
  ...
  Oₙ

The OWL abstract syntax does not provide a construct for the import of ontologies, although the RDF/XML serialization does provide this facility through the import statement.

usedMediators
  M₁
  ...
  Mₙ

OWL does not have the concept of a mediator. Therefore, this construct cannot be translated to OWL.

Table 1: Mapping between WSML-Core and OWL DL abstract syntax

<table>
<thead>
<tr>
<th>WSML-Core Logical Expression</th>
<th>OWL DL Abstract Syntax</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TODO: find a better way to characterize the allowed logical expressions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C subConceptOf D₁, ..., Dₙ</td>
<td>Class (C partial D₁ ... Dₙ)</td>
<td></td>
</tr>
<tr>
<td>?x memberOf D -&gt; ?x memberOf C</td>
<td>Class (D partial C)</td>
<td></td>
</tr>
<tr>
<td>?x memberOf D &lt;-&gt; ?x memberOf C₁ and ... and ?x memberOf Cₙ</td>
<td>Class (D complete C₁ ... Cₙ)</td>
<td></td>
</tr>
<tr>
<td>C[OfType T]</td>
<td>Class (C partial restriction(U allValuesFrom T))</td>
<td></td>
</tr>
<tr>
<td>Logical Expression</td>
<td>WSML-Core Syntax</td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>S(?x,?y) &lt; R(?x,?y)</td>
<td>ObjectProperty (R super(S))</td>
<td></td>
</tr>
<tr>
<td>U_2(?x,?y) &lt; U_1(?x,?y)</td>
<td>DatatypeProperty (U_1 super(U_2))</td>
<td></td>
</tr>
<tr>
<td>?x memberOf C &lt;- R(?x,?y)</td>
<td>ObjectProperty (R domain(C))</td>
<td></td>
</tr>
<tr>
<td>?y memberOf C &lt;- R(?x,?y)</td>
<td>ObjectProperty (R range(S))</td>
<td></td>
</tr>
<tr>
<td>S(?y,?x) &lt;- R(?x,?y)</td>
<td>ObjectProperty (R inverseOf(S))</td>
<td></td>
</tr>
<tr>
<td>R(?y,?x) &lt;- R(?x,?y)</td>
<td>ObjectProperty (R Symmetric)</td>
<td></td>
</tr>
<tr>
<td>R(?x,?y) &lt;- R(?x,?y) and R(?y,?z)</td>
<td>ObjectProperty (R Transitive)</td>
<td></td>
</tr>
<tr>
<td>o memberOf C [ A_1 hasValues o_1, ... A_n hasValues o_n ]</td>
<td>Individual(o type(C) value(A_1 o_1) ... value(A_n o_n))</td>
<td>Both the memberOf C and all the property values are optional.</td>
</tr>
</tbody>
</table>

Datatype expressions

- F_1(y_1,..,y_k) and ... and F_n(y_1,..,y_k) and(F_1,...,F_n) | F_i stands for a datatype expression component, which in this case amounts to a datatype predicate of arity k. |
- deCom(y_1,..,y_k) | deCom |
- deCom stands for a datatype predicate of arity k. |

Table 2: Mapping between WSML-Core logical expressions and OWL DL abstract syntax

### 2.2.6. Mapping OWL DL to WSML-Core

Table 3 shows the mapping between OWL DL (abstract syntax) and WSML-Core. It contains the conceptual syntax as well as any additional logical expression that might be necessary to capture the semantics of the OWL DL statement. The table shows for each construct supported by OWL DL the WSML-Core syntax in terms of the conceptual model and additional logical expressions necessary to capture the construct.
As we can see from Table 3, all of OWL DL can be captured in WSML-Core. Most of
the axioms can be captured with the conceptual model. However, some axioms rely
on additional logical expressions. Notice that if both conceptual syntax and a logical
expression is given in the table, this means that they are both introduced in the
translation and should be taken in conjunction.

<table>
<thead>
<tr>
<th>OWL DL(^{-}) Abstract syntax</th>
<th>WSML-Core conceptual syntax</th>
<th>WSML-Core logical expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logical Definitions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class((C\text{ partial } D_1 ... D_n))</td>
<td>concept (C)</td>
<td>(P(x, y_1, ..., y_n) \leftarrow \text{x memberOf C and } \text{y memberOf C})</td>
</tr>
<tr>
<td></td>
<td>subConceptOf (D_i)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(in case (D_i) is a named class)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(U_i\text{ ofType } P_i)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(in case (D_i) is a value restriction of the form (U_i\text{ allValuesFrom } P_i))</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class((C\text{ complete } D_1 ... D_n))</td>
<td>concept (C)</td>
<td>(\text{x memberOf C} \leftarrow \text{x memberOf } D_i \text{ and } ... \text{ and } \text{x memberOf } D_n)</td>
</tr>
<tr>
<td></td>
<td>subConceptOf (D_1, ..., D_n)</td>
<td></td>
</tr>
<tr>
<td>EquivalentClasses((C_1 ... C_n))</td>
<td></td>
<td>(\text{x memberOf } C_i \leftarrow \text{x memberOf } C_j)</td>
</tr>
<tr>
<td>ObjectProperty((R\text{ super}(R_1) ... super(R_n)))</td>
<td>relation (R)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>subRelationOf (R_1, ..., R_n)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>domain (C_1) ...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>domain (D_1) ...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>range (D_1) ... range (D_n)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[inverseOf((R_0))]</td>
<td>relation (R)</td>
<td>(R_0(x, y) \leftarrow R_0(y, x))</td>
</tr>
<tr>
<td></td>
<td>inverseOf (R_0)</td>
<td></td>
</tr>
<tr>
<td>[Symmetric]</td>
<td>relation (R)</td>
<td>(R\text{ symmetric})</td>
</tr>
<tr>
<td>[Transitive]</td>
<td>relation (R)</td>
<td>(R\text{ transitive})</td>
</tr>
<tr>
<td>SubPropertyOf((R_1 R_2))</td>
<td></td>
<td>(R_2(x, y) \leftarrow R_1(x, y))</td>
</tr>
<tr>
<td>EquivalentProperties($R_1$ ... $R_n$)</td>
<td>$R_i(?x,?y) \leftarrow R_j(?x,?y)$</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>----------------------------------</td>
<td></td>
</tr>
<tr>
<td>DatatypeProperty($U$ super($U_1$) ... super($U_n$) domain($C_1$) ... domain($C_n$) range($T_1$) ... range($T_n$))</td>
<td>relation $U$ subRelationOf $U_1$, ..., $U_n$ dataRelation domain ofType $C_1$, ..., $C_n$ range ofType $T_1$, ..., $T_n$</td>
<td></td>
</tr>
<tr>
<td>SubPropertyOf($U_1$ $U_2$)</td>
<td>$U_2(?x,?y) \leftarrow U_1(?x,?y)$</td>
<td></td>
</tr>
<tr>
<td>EquivalentProperties($U_1$ ... $U_n$)</td>
<td>$U_i(?x,?y) \leftarrow U_j(?x,?y)$</td>
<td></td>
</tr>
<tr>
<td>Individual($o$ type($C_1$) ... type($C_n$) value($R_1$ $o_1$) ... value($R_n$ $o_n$) value($U_1$ $v_1$) ... value($U_n$ $v_n$))</td>
<td>instance $o$ memberOf $C_1$, ..., $C_n$ $R_1$ hasValues $o_1$ ... $R_n$ hasValues $o_n$ $U_1$ hasValues $v_1$ ... $U_n$ hasValues $v_n$</td>
<td></td>
</tr>
<tr>
<td>DatatypeExpression($P$ deCom)</td>
<td>datatype $P$ definedBy deCom (in case deCom is a unary predicate or a conjunction of unary predicates)</td>
<td></td>
</tr>
<tr>
<td>domain($T_1$ ... $T_k$)</td>
<td>?value ofType $T_1$ (in case $k = 1$) ?$y_1$ ofType $T_1$ ... ?$y_k$ ofType $T_k$ (in case $k &gt; 1$)</td>
<td></td>
</tr>
<tr>
<td>and($F_1$ ... $F_n$)</td>
<td>$F_1(?y_1,\ldots,?y_k)$ and ... and $F_n(?y_1,\ldots,?y_k)$</td>
<td></td>
</tr>
<tr>
<td>Extra-Logical Definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>annotation($P$ $v$)</td>
<td>nonFunctionalProperties $P$ $v$</td>
<td></td>
</tr>
</tbody>
</table>

| Table 3: Mapping between OWL DL abstract syntax and WSML-Core |

### 2.2.7. WSML-Core in the WSMO Use Cases

This chapter enumerates the listings in the WSMO Use Cases document [Stollberg](#)
et al., 2004] and identifies whether the listing is in WSML-Core and if not, why it is not. Furthermore, we describe reductions, which would be necessary to make the listing fall inside WSML-Core. Note that WSML-Core does not distinguish between single-valued and set-valued attributes. Note also that in the evaluation we do not take into account inter-dependencies between the ontologies.

When evaluating the goal and web service descriptions, we only focus on the logical expressions, which are used for the assumptions, pre-conditions, effects and post-conditions.

Note that we have not evaluated the mediator descriptions, because they were not available in enough detail.

In the reason why a particular listing is not in WSML-Core we distinguish:

**mixing abstract and concrete domains**
A concrete domain corresponds with the domain of a datatype. The abstract domain corresponds with the elements defined in the ontology. The separation between abstract and concrete domains stems from the separation of these domain in Description Logic languages and the Description Logic-based ontology language OWL.

**integrity constraints**
Integrity constraints can be used in axiom definitions to express arbitrary restrictions.

**equality**
Equality here means that the equality symbol is used in the logical expressions. When equality is used in the head, this is mentioned explicitly, because the use of equality in the head of a rule entails complex term unification when reasoning with the ontology.

<table>
<thead>
<tr>
<th>Listing</th>
<th>In WSML-Core?</th>
<th>Why not?</th>
<th>Steps to make it fall inside WSML-Core</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ontologies</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1: Domain Ontology &quot;International Train Ticket&quot;</td>
<td>no</td>
<td>- integrity constraints</td>
<td>- Remove all (3) axioms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- equality</td>
<td></td>
</tr>
<tr>
<td>2: Domain Ontology &quot;Date and Time&quot;</td>
<td>no</td>
<td>- mixing abstract and concrete domains</td>
<td>- Complete remodeling of ontology</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- integrity constraints</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- equality</td>
<td></td>
</tr>
<tr>
<td>3: Domain Ontology &quot;Purchase&quot;</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4: Domain Ontology &quot;Locations&quot;</td>
<td>no</td>
<td>- mixing abstract and concrete domains</td>
<td>- Remove all axioms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- integrity constraints</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- equality (and also '&lt;' and '&gt;' in the head!)</td>
<td></td>
</tr>
<tr>
<td><strong>Goals and Web Services</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5: Goal - buying a train ticket online</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In many of the listings in the use cases document, abstract concepts are mixed with datatypes, for example, many concepts are defined as subconcepts of a datatype. This seems to be done mostly because it was necessary to create user-defined data types, which restrict a particular datatype. Because this version of WSML-Core introduces user-defined datatypes, we believe that many of the ontologies can be remodeled using user-defined datatypes.

2.2.8. Conclusions

In this section we have introduced WSML-Core, which is based on the WSMO conceptual model for ontologies [Roman et al., 2004]. We have given a precise characterization of the semantics through a mapping to the OWL abstract syntax [Patel-Schneider et al., 2004]. WSML-Flight, to be specified in Section 2.4, overcomes many of the limitations of WSML-Core.

From the evaluation of the ontologies developed in the WSMO use cases [Stollberg et al., 2004] we have seen that many ontologies fall inside WSML-Core. However, many of the ontologies use integrity constraints and equality in their definitions. An extension of WSML-Core with integrity constraints would solve many of the problems.

2.2.9. WSML-Core Changelog

The following major updates have been done since the August 23rd version of WSML-Core:

- The semantic basis of WSML-Core is now OWL DL instead of OWL Lite
- Datatypes have been re-introduced
- WSML-Core now uses an extension of OWL DL to enable datatype support (as described in [de Bruijn et al., 2004a])
- WSML-Core has been integrated in d16
- The instanceStore keyword has been removed; there is no longer an explicit way to link instance stores. It is assumed that an instance store is often linked outside of the ontology
- An appendix (D) has been added, detailing the minimally supported built-in predicates
- Semantic basis is no longer pure datalog, but we allow now the symbols ‘true’ and ‘false’, where ‘false’ can be used to create integrity constraints, which are required for the use of datatypes
- A section (2.2.4) has been added on web service specifications in WSML-Core.
  - the wsmlVariant (Section 2.2.2.1) has been introduced
  - A more elaborate description for the use of identifiers has been added (Section 2.2.1)

What is still to be done for this version of WSML-Core is the following:

- It has to be checked which additional logical expressions are allowed in OWL DL, which were not allowed in OWL Lite.
A better characterization needs to be found for the logical expressions which are allowed in WSML-Core. A mapping of WSML-Core to Datalog needs to be added. This mapping defines the semantics of WSML-Core.

What needs to be discussed is the following:

- The use of the `ofDataType` keyword. Should we just replace it with `ofType`?
- The use of angle brackets for sets of values; why not use a comma-separated list?
- A special keyword for OWL-style universal value restrictions? Currently, abstract attributes cannot be specified in WSML-Core, because the semantics of `ofType` differs from the semantics of the OWL universal value restriction. The question is: do we need to have a special modeling element for this kind of restrictions or do we want to rely on logical expressions?
- A special keyword for complete class definitions: do we want it? A complete class definition can not have any attributes.
- WSML-Core vs. Web Service specifications
- The use of the `wsmlVariant` keyword for the specification of the variant and the version of the WSML variant being used for the specification.

2.3. WSML-Rule

2.4. WSML-Flight

WSML-Flight is both syntactically and semantically completely layered on top of WSML-Core. This means that every valid WSML-Core specification is also a valid WSML-Flight specification. Furthermore, all consequences inferred from a WSML-Core specification are also valid consequences of the same specification in WSML-Flight. Finally, if a WSML-Flight specification falls inside the WSML-Core fragment then all consequences wrt. the WSML-Flight semantics also hold wrt. the WSML-Core semantics.

The features added by WSML-Flight are the following:

- Attribute definitions for the abstract domain
- Cardinality constraints
- More expressive logical expressions. Namely, we allow full Datalog with integrity constraints, default negation and the use of the equality and inequality symbols in the body of a rule. We no longer restrict the kind of logical expressions that can be written down. Therefore, WSML-Flight already contains a powerful rule language
- The use of `hasValue` vs `hasValues` and `ofType` vs. `ofTypeSet` need to be discussed, together with general cardinality constraints (see the WSML Flight Section 2.4.1.1 for a proposal for the specification of cardinality constraints.

2.4.1. WSML-Flight Elements

2.4.1.1 Concepts

A concept definition starts with the `concept` keyword, which is followed by the identifier of the concept. This is followed by zero or more implicitly conjoined direct
superconcept definitions (using the subConceptOf keyword followed by the identifier for a named concept), an optional nonFunctionalProperties block, and zero or more attribute specifications. An attribute specification consists of the identifier of an attribute, the ofType keyword and the identifier of a concept of which the attribute values must an instance.

A concept can only be a subConceptOf named concepts. WSML-Core does not allow complex concepts definitions as superconcepts, as is allowed in OWL.

An attribute value can not only be restricted to a concept, but also to a datatype. However, note that an attribute can only be restricted to either a concept or a datatype (in which case the attribute corresponds to a DatatypeProperty). In addition to the usual built-in datatypes, user-defined datatypes are permitted. Note however that we do not allow datatype restrictions of higher arity, as in OWL Flight. Instead, such a restriction should be introduced in the logical definition.

An attribute definition of the form A ofType D, where A is an attribute identifier and D is either a concept or a datatype identifier, is a constraint on the values for attribute A. If the value for the attribute A is not of type D, the constraint is violated and the attribute value is inconsistent with respect to the ontology. This notion of constraints corresponds the usual database-style constraints.

In the example below, the attributes length and weight have as their type the datatype xsd:integer. bmi has datatype xsd:float. BMI, in this case, stands for Body Mass Index, which is a certain ratio between the length, the weight and the age of a person. The Body Mass Index is calculated through the user-defined datatype predicate bodyMassIndex (see Section 2.2.2.9 for the definition). The calculation of the Body Mass Index is part of the necessary concept definition.

An example:

```
concept Human
  subConceptOf Primate
  subConceptOf LegalAgent
  nonFunctionalProperties
    dc:description hasValues "Members of the species Homo sapiens"
  endNonFunctionalProperties
  parentOf ofType {0 n} transitive inverseOf(hasParent) Human
  hasParent ofType {1} inverseOf(parentOf) Human
  authorOf ofType inverseOf(hasAuthor) Publication
  isRelated ofType symmetric Human
  length ofType {0 1} xsd:integer
  weight ofType {0 1} xsd:integer
  bmi ofType {0 1} xsd:float
  definedBy
    constraint wsml:not(bodyMassIndex(range hasValue ?b, length hasValue ?l, weight hasValue ?w) and ?x men
      ?x[length hasValues ?l, weight hasValues ?w, bmi hasValues ?b].
```

2.4.1.1.1 ATTRIBUTES

Because in WSML-Flight there is a strict separation between regular attributes and datatype attributes, different keywords are used for the respective attribute definitions. The keyword ofType is used for the regular attribute definitions and the
keyword **ofDataType** is used for the datatype attribute definitions.

Regular attributes (i.e. attributes that do not have a datatype as range) can be specified as being transitive, symmetric, or being the inverse of another attribute, using the **transitive**, **symmetric** and **inverseOf** keywords, respectively. Notice that these keywords do not enforce a constraint on the attribute, but are used to infer additional information about the attribute.

When an attribute is specified as being transitive, this means that if three individuals $a$, $b$ and $c$ are related via a transitive attribute $att$ in such a way: $a \rightarrow b \rightarrow c$ then $c$ is also a value for the attribute $att$ at $a$: $a \rightarrow c$.

When an attribute is specified as being symmetric, this means that if an individual $a$ has a symmetric attribute $att$ with value $b$, then $b$ also has attribute $att$ with value $a$.

When an attribute is specified as being the inverse of another attribute, this means that if an individual $a$ has an attribute $att$ with value $b$ and $att$ is the inverse of a certain attribute $attb$, then it is inferred that $b$ has an attribute $attb$ with value $a$.

Finally, it is possible to specify cardinality constraints for each attribute. The cardinality constraints for a single attribute are specified by including two numbers between curly brackets (`{ }`), indicating the minimal and maximal cardinality, after the **ofType** (or **ofDataType**) keyword. The first number indicates the minimal cardinality. The second number indicates the maximal cardinality, where ‘n’ stands for unlimited maximal cardinality (and is not allowed for minimal cardinality). It is possible to write down just one number instead of two, which is interpreted as both a minimal and a maximal cardinality constraint. When the cardinality is omitted, then it is assumed that there are no constraints on the cardinality, which is equivalent to `{0 n}`. Notice that a maximal cardinality of 1 makes an attribute functional.

### 2.5. WSML-DL

3. The WSML Exchange Syntaxes

The semantical variants of WSML all share the modeling elements of WSMO. They differ mainly in the kind of logical expressions one is allowed to use. Therefore the syntaxes for these variants will differ only slightly for the WSMO modeling elements. For usability reasons, a semantical variant may include language shortcuts for certain often-used logical expressions.

The three syntaxes for WSML are:

**Human-readable syntax:**
A human readable syntax for WSML. This syntax is defined in deliverable D2 [Roman et al., 2004; appendix B]. Examples of the use of this syntax can be found in [Stollberg et al., 2004]. This syntax is machine-readable with a specialized parser (which we provide as open-source software).

**XML syntax:**
A syntax specifically tailored for machine processability, instead of human-readability; it is easily parsable by standard XML parsers, but is quite unreadable for humans. This syntax is defined in Section 3.1.

**RDF syntax**
An alternate exchange syntax for WSML is WSML/RDF. WSML/RDF can be
used to leverage the currently existing RDF tools, such as a triple stores, and to syntactically combine WSML/RDF descriptions with other RDF descriptions. WSML/RDF is defined in Section 3.2.

Mapping to OWL
A bidirectional mapping between (a subset of) OWL and WSML is given in Section 3.4.

3.1 WSML/XML

3.1.1. Introduction

This document provides an XML syntax for the Web Service Modeling Language WSML, based on the WSMO conceptual model and the WSML BNF grammar provided by [Roman et al., 2004]. This syntax, henceforth referred to as WSML/XML, can be used to encode arbitrary WSML and can be used to encode anything, which can be encoded in the BNF syntax provided by [Roman et al., 2004].

Currently, WSML/XML only covers the "umbrella" syntax presented in D2 and does not yet support specific WSML variants, such as WSML-Flight, WSML-Core and WSML-Rule.

3.1.2. WSML/XML Syntax

The XML Schema (see Appendix B.1) captures the syntax of WSML/XML, as described earlier.

3.1.3. Example

Appendix B.2 Presents an examples of a WSML/XML document.

3.1.4. Future Work

The example in Appendix B is to be expanded to include goal, web service and mediator descriptions, as well as richer logical expressions.

Based on the XML syntax presented in this document, an XML syntax for WSML-Core is to be defined. The hard deadline for the syntax to be (pre-)final is end of September 2004.

The use of built-in predicate and function symbols is to be evaluated. The current strategy is to rely on identifiers for these built-ins, but it would be possible to include some of these built-ins as XML elements.

The use of path expressions is to be evaluated. Currently, a path expression is seen as an ID. We need to evaluate whether we need additional XML elements for path expressions.

After the XML Schema has been finalized, converters need to be built to convert from and to the human-readable syntax of WSML. The latter can be done using an XSLT script, as was done for previous versions (March 2004) of WSML/XML.

3.2 WSML/RDF
3.3 Mapping to OWL

[JB: We need to consider here on what syntactical level we do the mapping. I would suggest to do the mapping on the human-readable/abstract syntax level. Then, we need to see whether we want to provide translation services for other levels, such as the XML syntax or the RDF syntax.]

4. Implementations

4.1 WSML validator

A WSML validator is online at http://dev1.deri.at:8080/wsml/. Currently, this validator can only validate WSML-Full specifications in the human-readable syntax. Future versions will also validate other variants and syntaxes of WSML. The first variant to be added is WSML-Core. The first syntax to be added is the XML syntax.

4.2 WSML reasoner

4.3 WSML syntax converters

Converters will be developed to convert between the different syntaxes of WSML, namely the human-readable syntax, the XML syntax and the RDF syntax. Furthermore, an importer/exporter for OWL will be created. [JB: here, we need to decide what syntaxes of OWL we accept and which syntaxes we export. Considerations are the abstract syntax, the XML syntax and the RDF syntax. Perhaps we can use a third-party tool to convert between OWL syntaxes and just select the most suitable one for our purposes?]

5. Conclusions

References


Available at http://www.w3.org/TR/2004/REC-owl-ref-20040210/.


Acknowledgement

We would especially like to thank the reviewer of the deliverable, Ian Horrocks, for useful comments and discussions around this deliverable.
The work is funded by the European Commission under the projects DIP, Knowledge Web, SEKT, SWWS, Esperonto, 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.

---

Appendix A. BNF grammars for the human-readable syntax

This section will contain the BNF grammars for all the WSML variants [to be inserted], as well as an example document.

The example currently in this Appendix is based on the example provided with the WSML validator [http://138.232.65.151:8080/wsml/validator.html]. The example is to be updated to be a real-life example, which reflects most of the features of WSML. Furthermore, the example should be a web-accessible resource.

Appendix A.1. An example of WSML-Full in the human-readable syntax

```xml
namespace <<http://www.example.org/example/>>
dc: <<http://purl.org/dc/elements/1.1#>>
xsd: <<http://www.w3.org/2001/XMLSchema#>>
targetNamespace: <<http://www.example.org/example/>>

ontology <<http://www.example.org/example/test.wsml>>
nonFunctionalProperties
dc:title hasValues "WSML example collection"
dc:subject hasValues "family"
dc:description hasValues "fragments of a family ontology to provide WSML examples"
dc:date hasValues "2004-09-20"
dc:type hasValues <<http://www.wsmo.org/2004/d2/v1.0/20040827/ontologies>>
dc:format hasValues "text/plain"
dc:language hasValues "en-US"
dc:rights hasValues <<http://www.deri.org/privacy.html>>
version hasValues "$Revision: 1.3 "$
endNonFunctionalProperties

concept Human
nonFunctionalProperties
dc:description hasValue "concept of a human being"
endNonFunctionalProperties
name ofType xsd:string age

constraint ~ bodyMassIndex ( ?b, ?l, ?w) and ?xmemberOf Human and ?x[ length hasValues?l, weight hasValues?w

concept Man subConceptOf Human
nonFunctionalProperties
dc:description hasValue "concept of a man"
endNonFunctionalProperties

concept Woman subConceptOf Human
nonFunctionalProperties
dc:description hasValue "concept of a woman"
endNonFunctionalProperties

definedBy
```

forall ?x( ?xmemberOf Woman -> not ( ?xmemberOf Man ) ) .

concept Boy subConceptOf Man
nonFunctionalProperties
dc:description hasValue "human being not older than 14"
endNonFunctionalProperties
definedBy

concept Parent subConceptOf Human
nonFunctionalProperties
dc:description hasValue "human being with at least one child"
endNonFunctionalProperties
children ofType set Human definedBy
forall ?x( ?xmemberOf Parent implies ?xmemberOf Human and exists ?y( hasChild [ parent hasValue?x, child hasValue?y] )

instance Mary memberOf Parent
name hasValue "Maria Smith"
age hasValue "50" ^^ xsd:integer
children hasValues { Michael, Susan }

instance Paul memberOf Man
name hasValue "Paul Jones"
age hasValue "25" ^^ xsd:integer

relation hasAncestor subRelationOf hasRelative
nonFunctionalProperties
dc:description hasValue "Relation between ancestors"
endNonFunctionalProperties
domain ofType Person

relation length
nonFunctionalProperties
dc:description hasValue "Length indicator"
endNonFunctionalProperties

relation hasChild
parent ofType Human
child ofType Human

relation childOf
child ofType Human
parent ofType Human
definedBy
forall ?x, ?y( childOf [ child hasValue?x, parent hasValue?y] equivalent hasChild [ parent hasValue?y, child hasValue?x] )

relation hasSon subRelationOf hasChild
parent ofType Human
child ofType Man

function hasAge
person ofType Human
range ofType xsd:integer
definedBy
forall ?x, ?y( hasAge [ person hasValue?x, result hasValue?y] equivalent ?x. age = ?y) .

function bodyMassIndex
nonFunctionalProperties
dc:description hasValue "Calculates the Body Mass Index. This version is not really accurate, but hopefully shows how a datatype predicate is created. bmi = kg/m2. Notice that the link between the parameters and the variables in the logical expression are a bit strange. This is inherited from d2. We need some clarification here."
endNonFunctionalProperties

length ofType xsd:integer
weight ofType xsd:integer
range ofType xsd:float
definedBy


relationInstance hasChildMaryMichael memberOf hasChild
nfp
dc:description hasValue "Mary is a parent of Michael"
endnfp
parent hasValue Mary
child hasValue Michael

axiom disjointManWoman
nonFunctionalProperties
dc:description hasValue "Man and Woman are two disjoint concepts without constraint notation"
endNonFunctionalProperties
definedBy

axiom constraintManWoman
nfp
dc:description hasValue "no individual can be man and a woman"
endnfp
definedBy

webservice <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040719/resources/ws.wsml>
nonFunctionalProperties
dc:title hasValue "ÖBB Online Ticket Booking Web Service"
dc:creator hasValue "DERI International"
dc:description hasValue "web service for booking online train tickets for Austria and Germany"
dc:contributor hasValues [ "Michael Stollberg", "Ruben Lara", "Holger Lausen" ]
dc:publisher hasValue "DERI International"
dc:creator hasValue "DERI International"
dc:date hasValue "2004-07-19"
dc:format hasValue "text/plain"
dc:language hasValue "en-us"
dc:coverage hasValues [ tc:austria , tc:germany ]
dc:rights hasValue <http://deri.at/privacy.html>
version hasValue "$Revision: 1.3 $"
endNonFunctionalProperties

usedMediators ooMediator
capability _#
precondition axiom _#
nonFunctionalProperties
dc:description hasValue "the input has to be a buyer with a purchase intention for an itinerary wherefore the start- and endlocation have to be in Austria or in Germany, and the departure date has to be later than the current Date. A credit card as payment method is expected."
endNonFunctionalProperties
definedBy
?BuyermemberOf po:buyer and ?TripmemberOf tc:trainTrip [ tc:start hasValue?Start, tc:end hasValue?End, tc:departure hasValue?date ]
assumption axiom _#
nonFunctionalProperties
dc:description hasValue "the credit card has to be valid, i.e. not expired. The current date is provided as a built-in functionality (currently defined explicitly as built-in function is not available)."
endNonFunctionalProperties
definedBy
?CreditCard[ po:cardholdername hasValue_#1 , po:creditcardidentifier hasValue_#2 , po:expirydate hasValue_#3
The output of the service is a train trip where the start- and endlocation have to be in Austria or in Germany and the departure date has to be later than the current Date.

there shall be a trade for the train trip of the postcondition

describes the Interface of Web Service

Appendix B. XML Schemas for the XML exchange syntax

In the following sections we present the XML Schemas for the XML syntax of WSML-Full, as well as an example. This example is the XML version of the example of Section A.1 [Notice that at the moment these are not completely in-sync and there might be discrepancies between the two examples].

Appendix B.1. XML Schema for the XML syntax of WSML-Full

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- version: 2004-09-09 -->
<!-- author: Jos de Bruijn -->
<!-- this document provides an XML Schema for the XML version of the WSML syntax. -->
<!-- this document is based on the BNF syntax for WSML (WSMO D2, Appendix B) v1.0 dated 2004-08-27 -->
<!-- there is currently elementary support for logical expressions; path expressions are not supported and no built-ins are defined. It is assumed that built-ins are specific function or predicate symbols with a certain identifier. Path expressions are treated currently as a kind of ID. -->
<xs:element name="wsml">
  <xs:annotation>
    <xs:documentation>the root element, containing the WSML document</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="ontology" type="ontologyType"/>
      <xs:element name="webService" type="webServiceType"/>
      <xs:element name="goal" type="goalType"/>
      <xs:element name="mediator" type="mediatorType"/>
    </xs:choice>
    <xs:attribute name="id" type="wsmlID" use="required"/>
  </xs:complexType>
</xs:element>
<xs:complexType name="goalType">
  <xs:sequence>
    <xs:element name="nonFunctionalProperties" type="nonFunctionalPropertiesType" minOccurs="0",
      maxOccurs="unbounded"/>
    <xs:element name="importOntology" type="wsmlID" minOccurs="0",
      maxOccurs="unbounded"/>
    <xs:element name="ooMediator" type="wsmlID" minOccurs="0",
      maxOccurs="unbounded"/>
    <xs:element name="ggMediator" type="wsmlID" minOccurs="0",
      maxOccurs="unbounded"/>
    <xs:element name="postCondition" type="axiomType" minOccurs="0",
      maxOccurs="unbounded"/>
    <xs:element name="effect" type="axiomType" minOccurs="0",
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```
<xs:complexType name="nonFunctionalPropertiesType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="version"/>
        <xs:any/>
    </xs:choice>
</xs:complexType>

<xs:complexType name="nonFunctionalPropertiesWSType">
    <xs:complexContent>
        <xs:extension base="nonFunctionalPropertiesType">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="performance"/>
                <xs:element name="security"/>
                <xs:element name="scalability"/>
                <xs:element name="robustness"/>
                <xs:element name="accuracy"/>
                <xs:element name="transactional"/>
                <xs:element name="trust"/>
                <xs:element name="financial"/>
                <xs:element name="networkRelatedQoS"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="importOntologyType"/>

<!-- start of the non-functional properties part -->
<xs:complexType name="webServiceType">
    <xs:sequence>
        <xs:element name="nonFunctionalProperties" type="nonFunctionalPropertiesWSType" minOccurs="0"/>
        <xs:element name="importOntology" type="wsmlID" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="ooMediator" type="wsmlID" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="capability" type="capabilityType"/>
        <xs:element name="interface" type="interfaceType" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="wsmlID" use="required"/>
</xs:complexType>

<xs:complexType name="capabilityType">
    <xs:sequence>
        <xs:element name="nonFunctionalProperties" type="nonFunctionalPropertiesType" minOccurs="0"/>
        <xs:element name="importOntology" type="wsmlID" minOccurs="0" maxOccurs="unbounded"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="sourceOntology" type="wsmlID"/>
            <xs:element name="sourceooMediator" type="wsmlID"/>
        </xs:choice>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="targetOntology" type="wsmlID"/>
            <xs:element name="targetGoal" type="wsmlID"/>
            <xs:element name="targetWebService" type="wsmlID"/>
            <xs:element name="targetMediator" type="wsmlID"/>
        </xs:choice>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="goal" type="wsmlID"/>
            <xs:element name="wwMediator" type="wsmlID"/>
            <xs:element name="webService" type="wsmlID"/>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="id" type="wsmlID" use="required"/>
</xs:complexType>

<xs:complexType name="interfaceType">
    <xs:sequence>
        <xs:element name="nonFunctionalProperties" type="nonFunctionalPropertiesType" minOccurs="0"/>
        <xs:element name="importOntology" type="wsmlID" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="ooMediator" type="wsmlID" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="choreography" type="wsmlID"/>
        <xs:element name="orchestration" type="wsmlID"/>
    </xs:sequence>
    <xs:attribute name="id" type="wsmlID" use="required"/>
</xs:complexType>

<!-- end of the mediator part -->

<xs:complexType name="mediatorType">
    <xs:choice>
        <xs:element name="ooMediator" type="ooMediatorType"/>
        <xs:element name="ggMediator" type="ggMediatorType"/>
        <xs:element name="wgMediator" type="wgMediatorType"/>
        <xs:element name="wwMediator" type="wwMediatorType"/>
    </xs:choice>
</xs:complexType>

<xs:complexType name="ooMediatorType">
    <xs:sequence>
        <xs:element name="nonFunctionalProperties" type="nonFunctionalPropertiesWSType" minOccurs="0"/>
        <xs:element name="importOntology" type="wsmlID" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="ooMediator" type="ooMediatorType" minOccurs="0" maxOccurs="unbounded"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="sourceOntology" type="wsmlID"/>
            <xs:element name="sourceooMediator" type="wsmlID"/>
        </xs:choice>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="targetOntology" type="wsmlID"/>
            <xs:element name="targetGoal" type="wsmlID"/>
            <xs:element name="targetWebService" type="wsmlID"/>
            <xs:element name="targetMediator" type="wsmlID"/>
        </xs:choice>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="goal" type="wsmlID"/>
            <xs:element name="wwMediator" type="wsmlID"/>
            <xs:element name="webService" type="wsmlID"/>
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="id" type="wsmlID" use="required"/>
</xs:complexType>

<xs:complexType name="ggMediatorType">
    <!-- start of the mediator part -->

<!-- end of the Web Services part -->

<!-- start of the mediator part -->

<!-- end of the Web Services part -->

<!-- start of the non-functional properties part -->

<!-- end of the mediator part -->

<!-- start of the mediator part -->

<!-- end of the non-functional properties part -->
Appendix B.2. An example of a WSML/XML ontology

<?xml version="1.0" encoding="UTF-8"?>
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
<wsml:ontology id="http://www.example.org/example.wsml">
<!-- this example WSML/XML ontology is based on the example given with the WSML validator -->
<!-- Notice that the ontology is rather complex when it comes to logical expressions; however, XML is just an exchange syntax -->
<wsml:nonFunctionalProperties>
<dc:title>WSML example collection</dc:title>
<dc:subject>family</dc:subject>
<dc:description>fragments of a family ontology to provide WSML examples</dc:description>
<dc:date>2004-06-28</dc:date>
<dc:rights>&lt;http://www.deri.org/privacy.html&gt;</dc:rights>
<wsml:version>$Revision: 1.2 $</wsml:version>
</wsml:nonFunctionalProperties>
<wsml:concept id="Human">
<wsml:nonFunctionalProperties>
<dc:description>concept of a human being</dc:description>
</wsml:nonFunctionalProperties>
<wsml:range>xsd:string</wsml:range>
<wsml:attribute>
<wsml:range>xsd:string</wsml:range>
<wsml:range>xsd:string</wsml:range>
</wsml:attribute>
<wsml:concept id="Man">
<wsml:nonFunctionalProperties>
<dc:description>concept of a man</dc:description>
</wsml:nonFunctionalProperties>
</wsml:concept>
<wsml:concept id="Woman">
</wsml:concept>
</wsml:ontology>
</wsml:wsml>
Appendix C. RDF Schemas for the RDF exchange syntax

This section will contain the RDF Schema definitions for the RDF syntax for all the WSM defense.

Appendix D. Built-ins in WSML

The appendix will contain an preliminary list of built-in functions and relations for datatypes in WSML. Furthermore, it will contain a translation of syntactic shortcuts to datatype predicates.

Appendix D.1. WSML Datatype predicates
This section contains a list of datatype predicates suggested for use in WSML. These predicates correspond to functions in XQuery/XPath [Malhotra et al., 2004]. Notice that SWRL [Horrocks et al., 2004] built-ins support is also based on XQuery/XPath.

The current list is only based on the built-in support in the WSML language through the use of special symbols. Find a translation of the built-in symbols to datatype predicates in the next section. The symbol ?range signifies the range of the function. Functions in XQuery have a defined range, whereas predicates only have a domain. Therefore, the first argument of a WSML datatype predicate which represents a function represents the range of the function. Comparators in XQuery are functions, which return a boolean value. These comparators are directly translated to predicates. If the XQuery function returns 'true', the arguments of the predicate are in the extension of the predicate.

<table>
<thead>
<tr>
<th>WSML datatype predicate</th>
<th>XQuery function</th>
<th>Datatype (A)</th>
<th>Datatype (B)</th>
<th>Return datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsml:numeric-equal(A,B)</td>
<td>op:numeric-equal(A,B)</td>
<td>numeric</td>
<td>numeric</td>
<td>numeric</td>
</tr>
<tr>
<td>wsml:numeric-greater-than(A,B)</td>
<td>op:numeric-greater-than(A,B)</td>
<td>numeric</td>
<td>numeric</td>
<td>numeric</td>
</tr>
<tr>
<td>wsml:numeric-less-than(A,B)</td>
<td>op:numeric-less-than(A,B)</td>
<td>numeric</td>
<td>numeric</td>
<td>numeric</td>
</tr>
<tr>
<td>wsml:string-equal(A,B)</td>
<td>op:numeric-equal(fn:compare(A,B), 1)</td>
<td>xsd:string</td>
<td>xsd:string</td>
<td>numeric</td>
</tr>
<tr>
<td>wsml:numeric-add(?range,A,B)</td>
<td>op:numeric-add(A,B)</td>
<td>numeric</td>
<td>numeric</td>
<td>numeric</td>
</tr>
<tr>
<td>wsml:numeric-subtract(?range,A,B)</td>
<td>op:numeric-subtract(A,B)</td>
<td>numeric</td>
<td>numeric</td>
<td>numeric</td>
</tr>
<tr>
<td>wsml:numeric-multiply(?range,A,B)</td>
<td>op:numeric-multiply(A,B)</td>
<td>numeric</td>
<td>numeric</td>
<td>numeric</td>
</tr>
<tr>
<td>wsml:numeric-divide(?range,A,B)</td>
<td>op:numeric-divide(A,B)</td>
<td>numeric</td>
<td>numeric</td>
<td>numeric</td>
</tr>
<tr>
<td>wsml:not(A)</td>
<td>fn:not(A)</td>
<td>boolean</td>
<td></td>
<td>boolean</td>
</tr>
</tbody>
</table>

Here, wsml:not signifies the complement of a datatype predicate. For example, by applying wsml:not to the predicate wsml:greater-than, a less-than-or-equal relation is obtained.

Each implementation is required to either implement the complement operator wsml:not or implement for each datatype predicate its compliment. The complement is required for constraint checking of attribute values.

**Appendix D.2. Translating built-in symbols to Datatype predicates**

In this section, we provide the translation of the built-in (function and predicate) symbols for datatype predicates to these datatype predicates.

We distinguish between built-in functions and built-in relations. Functions have a defined domain and range. Relations only have a domain and can in fact be seen as functions, which return a boolean, as in XPath/XQuery [ref.]. We first provide the translation of the built-in relations and then present the rewriting rules for the built-in functions.

The following table provides the translation of the built-in relations:
We now list the built-in functions and their translation to datatype predicates. In the table, ?x1 represents a unique newly introduced variable, which stands for the range of the function:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Datatype (A)</th>
<th>Datatype (B)</th>
<th>Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = B</td>
<td>xsd:string</td>
<td>xsd:string</td>
<td>wsml:string-equal(A,B)</td>
</tr>
<tr>
<td>A ~ B</td>
<td>xsd:string</td>
<td>xsd:string</td>
<td>wsml:not(wsml:string-equal(A,B))</td>
</tr>
<tr>
<td>A = B</td>
<td>xsd:integer</td>
<td>xsd:integer</td>
<td>wsml:numeric-equal(A,B)</td>
</tr>
<tr>
<td>A ~ B</td>
<td>xsd:integer</td>
<td>xsd:integer</td>
<td>wsml:not(wsml:numeric-equal(A,B))</td>
</tr>
<tr>
<td>A &lt; B</td>
<td>xsd:integer</td>
<td>xsd:integer</td>
<td>wsml:less-than(A,B)</td>
</tr>
<tr>
<td>A &lt;= B</td>
<td>xsd:integer</td>
<td>xsd:integer</td>
<td>wsml:less-equal(A,B)</td>
</tr>
<tr>
<td>A &gt; B</td>
<td>xsd:integer</td>
<td>xsd:integer</td>
<td>wsml:not(wsml:less-equal(A,B))</td>
</tr>
<tr>
<td>A &gt;= B</td>
<td>xsd:integer</td>
<td>xsd:integer</td>
<td>wsml:less-than(A,B)</td>
</tr>
</tbody>
</table>

Function symbols in WSML are not as straightforward to translate to datatype predicates as are relations. However, if we see the predicate as a function, which has the range as its first argument, we can introduce a new variable for the range and append the newly introduced predicate to the conjunction of which the top-level predicate is part.

Formulas containing built-in function symbols can be rewritten to datatype predicate conjunctions according to the following algorithm:

1. Select an atomic occurrence of a datatype function symbol. An atomic occurrence is an occurrence of the function symbol with only identifiers (which can be variables) as arguments.
2. Replace this occurrence with a newly introduced variable and append the newly introduced predicate to the conjunction of which the top-level predicate is part.
3. If there are still occurrences of function symbols in the formula, go back to step (1), otherwise, return the formula.

We present an example of the application of the algorithm to following expression:

\[ ?w = ?x + ?y + ?z \]

We first substitute the first occurrence of the function symbol `+' with a newly introduced variable ?x1 and append the predicate wsml:numeric-add(?x1, ?x, ?y) to
the conjunction:

\[ w = x_1 + z \text{ and } \text{wsml:numeric-add}(x_1, x, y) \]

Then, we substitute the remaining occurrence of '+' accordingly:

\[ w = x_2 \text{ and } \text{wsml:numeric-add}(x_1, x, y) \text{ and } \text{wsml:numeric-add}(x_2, x_1, z) \]

Now, we don't have any more built-in function symbols to substitute and we merely substitute the built-in relation '=' to obtain the final conjunction of datatype predicates:

\[ \text{wsml:numeric-equal}(w, x_2) \text{ and } \text{wsml:numeric-add}(x_1, x, y) \text{ and } \text{wsml:numeric-add}(x_2, x_1, z) \]

Footnotes

[1] Note that in WSML-Core there is no keyword hasValue for single-valued attributes, because there are not cardinality restrictions in WSML-Core. Only set-valued attributes are allowed through the hasValues keyword.

[2] The only expressivity added by the logical expressions over the conceptual syntax is the complete class definition, which is an equivalence relation between the defined class and the intersection (conjunction) of the defining classes.