



D16.7v0.1. WSML-Core

WSML Working Draft 13 August 2004

This version:

<http://www.wsmo.org/2004/d16/d16.7/v0.1/20040813/>

Latest version:

<http://www.wsmo.org/2004/d16/d16.7/v0.1/>

Previous version:

<http://www.wsmo.org/2004/d16/d16.7/v0.1/20040802/>

Editors:

Jos de Bruijn

Authors:

Jos de Bruijn
Douglas Foxvog
Eyal Oren
Dieter Fensel

Reviewer:

Ian Horrocks

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.

Abstract

We introduce WSML-Core, an ontology modeling language, based on the semantics of OWL Lite⁻ (a restricted subset of OWL Lite with a datatype extension) and the conceptual model of ontologies presented in WSMO.

We describe each modeling element in the syntax of WSML-Core and describe the semantics through a mapping to (the OWL Lite⁻ subset of) the OWL abstract syntax, which has a direct model-theoretic semantics associated with it. When analyzing the applicability of WSML-Core to the current version of the WSMO use cases we see some limitations of WSML-Core.

Table of contents

- [1. Introduction](#)
- [2. WSML-Core Elements](#)
 - [2.1 Basic WSML-Core Syntax](#)
 - [2.2 Ontologies](#)
 - [2.3 Namespace Definitions](#)

- [2.4 Non-functional properties](#)
 - [2.5 Import Ontology Definitions](#)
 - [2.6 Mediators](#)
 - [2.7 Concept Definitions](#)
 - [2.8 Relation Definitions](#)
 - [2.9 Instance Definitions](#)
 - [2.10 Axiom Definitions](#)
 - [2.11 Datatype Definitions](#)
 - [3. WSMML-Core Logical Expressions](#)
 - [3.1 Simple Logical Expressions](#)
 - [3.1.1 Relation Expressions](#)
 - [3.1.2 Molecules](#)
 - [3.1.3 Datatype predicates](#)
 - [3.2 Complex Logical Expressions](#)
 - [3.2.1 Compound Logical Expressions](#)
 - [3.2.2 Formulas](#)
 - [4. Mapping WSMML-Core to OWL Lite⁻](#)
 - [5. Mapping OWL Lite⁻ to WSMML-Core](#)
 - [6. WSMML-Core in the WSMO Use Cases](#)
 - [7. Conclusions](#)
 - [Changelog](#)
 - [References](#)
 - [Acknowledgement](#)
 - [Appendix A. Reviewer Comments](#)
-

1. Introduction

The major goal of the [WSMML](#) 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 deliverable [D16.0 \[Oren, 2004\]](#), there are several WSMML languages with different underlying logical formalisms. The two main logical formalisms exploited in different WSMML languages are Description Logics [\[Baader et al., 2003\]](#) (exploited in WSMML-DL) and Rule Languages [\[Lloyd, 1987\]](#) (exploited in WSMML-RL). WSMML-Core (described in this document) exploits the intersection of both formalisms.

We introduce WSMML-Core, an ontology modeling language, based on the semantics of OWL Lite⁻ [\[de Bruijn et al., 2004\]](#), which is based on [\[Grosz et al., 2003\]](#) with a restricted form of the OWL-E datatype extension [\[Pan and Horrocks, 2004\]](#), and the conceptual model of ontologies presented in WSMO [\[Roman et al., 2004\]](#). Because WSMML-Core is based on OWL Lite⁻ and there exists a translation from OWL Lite⁻ to plain (function- and negation-free) Datalog, the decidability and complexity results of Datalog apply to WSMML-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.

[Chapter 2](#) describes all the elements of the WSMML-Core language. [Chapter 3](#) specifies the mapping from WSMML-Core to the OWL Lite⁻ abstract syntax, thereby actually defining the semantics of WSMML-Core. [Chapter 4](#) provides a mapping from the OWL Lite⁻ abstract syntax to WSMML-Core. [Chapter 5](#) summarizes the applicability of WSMML-Core to the use cases of WSMO D3.2 [\[Stollberg et al., 2004\]](#). Finally, we present some conclusions in [Chapter 6](#).

2. WSML-Core Elements

In this chapter 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/d16.7/v0.1/>.

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

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 [Fensel et al., 2001], which also offers the possibility of either grouping descriptions together in frames or spreading the descriptions throughout the document.

Identifiers in WSML-Core follow the conventions of WSMO [Roman et al., 2004]. 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 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>
```

2.3 Namespace Definitions

Below the **ontology** URI declaration, 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.wsmo.org/ontologies/purchase#>
  dc: <http://purl.org/dc/elements/1.1#>
  wsmo: <http://www.wsmo.org/2004/d2/v1.0#>

  targetNamespace <http://www.wsmo.org/ontologies/purchase#>
```

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:

```
nonFunctionalProperties
  dc:title hasValues "Purchase ontology"
  dc:creator hasValues <http://www.example.org/deri/foaf.agent>
  dc:subject hasValues "Buyer", "Seller", "Product", "Price",
    "Payment method", "Delivery"
  dc:description hasValues "General purchase order request ontology"
  dc:publisher hasValues <http://www.example.org/deri/foaf.agent>
  dc:contributor hasValues <http://www.example.org/deri/armin.haller/foaf.agent>
  dc:date hasValues "2004-07-19"
  dc:type hasValues <http://www.wsmo.org/2004/d2/v0.3/20040329/#ontos>
  dc:format hasValues "text/plain"
  dc:language hasValues "en-US"
  dc:relation hasValues
    <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040607/resources/loc.wsmo>,
    <http://www.wsmo.org/ontologies/dateTime#>
  dc:coverage hasValues "ID:7029392 Name:World"
  dc:rights hasValues <http://www.deri.org/privacy.html>
  version hasValues "Revision: 1.12"
endNonFunctionalProperties
```

2.5 Import Ontology Definitions

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:

importOntologies

```
<http://www.wsmo.org/ontologies/dateTime#>  
<http://www.wsmo.org/ontologies/currency#>
```

2.6 Mediators

Mediators are used to import other ontologies. This concept of mediation between ontologies is more flexible than the **importOntologies** statement, which is used to import an WSMML ontology into another WSMML 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.

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

```
<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/owlFactbookMediator.wsmml>  
<http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/resources/owlAddressMediator.wsmml>
```

2.7 Concept Definitions

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 a relation, the **ofType** keyword and the identifier of a concept of which the attribute values must be instances.

A concept can only be a **subConceptOf** named concepts. WSMML-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 (in which case the attribute corresponds to an *ObjectProperty*) or a datatype (in which case the attribute corresponds to a *DatatypeProperty*). In addition to the usual built-in datatypes, user-defined datatypes, as in OWL Lite⁻ are permitted. Note however that we do not allow datatype restrictions of higher arity, as in OWL Lite⁻. 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, asserts that for each instance of concept *C*, each value for the attribute *A* is of the type *D*. This corresponds with universal value restrictions 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 2.11 for the definition). The calculation of the Body Mass Index is part of the necessary concept definition.

An example:

```
concept Human subConceptOf Primate, LegalAgent
  nonFunctionalProperties
    dc:description "Members of the species Homo sapiens"
  endNonFunctionalProperties
  parentOf ofType Human
  length ofType xsd:integer
  weight ofType xsd:integer
  bmi ofType xsd:float
  definedBy
    bodyMassIndex(?b, ?l, ?w) <- ?x memberOf Human and
      ?x[length hasValues ?l,
        weight hasValues ?w,
        bmi hasValues ?b].
```

2.8 Relation Definitions

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 Lite⁷. Parameters can be used to restrict the domain and range of the property (denoted by the **domain** and **range** keywords, respectively). In 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 [Chapter 4](#).

An example:

```
relation hasAncestor subRelationOf hasRelative
  transitive
  nonFunctionalProperties
    dc:description "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 Lite⁷. 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:

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

2.9 Instance Definitions

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 Lite⁷. 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. In order to distinguish mediators used for linking ontologies and mediators used for linking instance stores, we introduce the keyword **instance-stores** followed by one or more identifiers to indicate the instance stores.

An example:

```
instanceStores
  <http://www.example.org/instance-stores/personnel-store>
  <http://www.example.org/instance-stores/client-store>
```

2.10 Axiom Definitions

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 [Chapter 3](#) and the allowed expressions are detailed in Table 2 in [Chapter 4](#).

An example:

```
axiom humanDefinition
  definedBy
    ?x memberOf Human <->
    ?x memberOf Animal and
    ?x memberOf LegalAgent.
```

2.11 Datatype Definitions

Analogously to OWL Lite⁻, we allow the user to create user-defined datatype predicates and user-defined datatypes. A datatype expressions 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 and optional set of parameters and a datatype expression preceded by the **definedBy** keyword. The syntax for the datatype expressions is explained in Section [insert cross-reference].

The example below defines a new predicate for calculating the Body Mass Index. The built-in predicates used in the example are borrowed from SWRL [[Horrocks et al., 2004](#)]. [TODO: think about infix notation for predicates]

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

```
function bodyMassIndex
nonFunctionalProperties
  dc:description "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
range ofType xsd:float
length ofType xsd:integer
weight ofType xsd:integer
definedBy
  bodyMassIndex(range:?bmi, length:?length, weight:?weight) <-
    swrlb:divide(?bmi, ?weight, ?sqLength) and swrlb:multiply(?sqLength,?length,?length)
```

An example of a user-defined datatype:

```
datatype positiveInt
nonFunctionalProperties
  dc:description "Positive integers."
endNonFunctionalProperties
?value ofType xsd:integer
definedBy
  swrlb:greaterThan(?value, "0"^^xsd:integer).
```

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 [Chapter 4](#) in order to allow for a direct translation into OWL Lite⁷.

3.1 Simple Logical Expressions

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

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)
```

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* **memberOf** *C*, where *I* is an instance identifier and *C* is a concept identifier.
- An attribute value list of the form [*A*₁ **hasValues** *v*₁, ..., *A*_{*n*} **hasValues** *v*_{*n*}], where *I* is an instance identifier, *A*₁, ..., *A*_{*n*} are attribute identifiers and *v*₁, ..., *v*_{*n*} are either instance identifiers or data values.

A concept membership assertion of the form *I* **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* **subConceptOf** *D*, where *C* and *C* are a concept identifiers.
- An attribute definition list of the form *I* [*A*₁ **ofType** *D*₁, ..., *A*_{*n*} **ofType** *v*₁], where *I* is an

instance identifier, A_1, \dots, A_n are attribute identifiers and D_1, \dots, 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]
```

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:

```
owlx:integerEqual(3,4)
```

Currently, we do not allow infix notation for certain predicates (such as '=' denoting equality of data values). This will be part of future work, which will be based on the mapping of XPath infix operators to XQuery and XPath functions and operators [Malhotra et al., 2004], to be found in [Appendix B.2](#) of [Berglund et al., 2004].

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.

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

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 (\leftarrow), right implication (\rightarrow) or dual implication (\leftrightarrow). In formulas, variables are allowed to occur in the place of identifiers.

- Left implication: $E_1 \leftarrow E_2$, 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: $E_1 \rightarrow E_2$, 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:

```
?x memberOf Human  $\leftrightarrow$  ?x memberOf Animal and ?x memberOf LegalAgent.
```

4. Mapping WSML-Core to OWL Lite⁻

In this chapter we define the semantics of WSML-Core through a mapping to the OWL Lite⁻ 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 Lite⁻. 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 Lite⁻. 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 Lite⁻. 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^[2] 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.

WSML-Core conceptual syntax	OWL Lite ⁻ Abstract syntax	Remarks
<i>Logical Declarations</i>		
ontology <i>O</i>	Ontology(<i>O</i>)	All definitions and axioms in the ontology are nested inside the Ontology statement, according to the other mappings in this table.
concept <i>C</i> subConceptOf <i>D</i> ₁ , ..., <i>D</i> _{<i>n</i>} <i>A</i> ₁ ofType <i>C</i> ₁ . <i>A</i> _{<i>i</i>} ofType <i>C</i> _{<i>i</i>} <i>A</i> _{<i>i</i>+1} ofType <i>T</i> _{<i>i</i>+1} . <i>A</i> _{<i>n</i>} ofType <i>T</i> _{<i>n</i>}	Class(<i>C</i> partial <i>D</i> ₁ ... <i>D</i> _{<i>n</i>} restriction(<i>A</i> ₁ allValuesFrom <i>C</i> ₁) ... restriction(<i>A</i> _{<i>i</i>} allValuesFrom <i>C</i> _{<i>i</i>}) restriction(<i>A</i> _{<i>i</i>+1} allValuesFrom <i>T</i> _{<i>i</i>+1}) ... restriction(<i>A</i> _{<i>n</i>} allValuesFrom <i>T</i> _{<i>n</i>}))	
relation <i>R</i> [subRelationOf <i>R</i> ₁ , ..., <i>R</i> _{<i>n</i>}] [transitive] [symmetric] [inverseOf(<i>R</i> ₀)] [domain ofType <i>C</i> ₁ , ..., <i>C</i> _{<i>n</i>}] [range ofType <i>D</i> ₁ , ..., <i>D</i> _{<i>n</i>}]	ObjectProperty(<i>R</i> super(<i>R</i> ₁) ... super(<i>R</i> _{<i>n</i>}) [inverseOf(<i>R</i> ₀)] [Symmetric] [Transitive] domain(<i>C</i> ₁) ... domain(<i>C</i> _{<i>n</i>}) range(<i>D</i> ₁) ... range(<i>D</i> _{<i>n</i>}))	
relation <i>U</i> [subRelationOf <i>U</i> ₁ , ..., <i>U</i> _{<i>n</i>}] dataRelation [domain ofType <i>C</i> ₁ , ..., <i>C</i> _{<i>n</i>}] [range ofType <i>T</i> ₁ , ..., <i>T</i> _{<i>n</i>}]	DatatypeProperty(<i>U</i> super(<i>U</i> ₁) ... super(<i>U</i> _{<i>n</i>}) domain(<i>C</i> ₁) ... domain(<i>C</i> _{<i>n</i>}) range(<i>T</i> ₁) ... range(<i>T</i> _{<i>n</i>}))	

datatype T [y ofType T_1] definedBy $deCom$	$DatatypeExpression(T$ $and(domain(T_1), deCom))$	
function F [range ofType T_1] [y ₂ ofType T_2] . . [y _k ofType T_k] definedBy $deCom$	$DatatypeExpression(F$ $and(domain(T_1, \dots, T_n),$ $deCom))$	
instance o memberOf C_1, \dots, C_n A_1 hasValues o_1 . . A_i hasValues o_i A_{i+1} hasValues v_{i+1} . . A_n hasValues v_n	$Individual(o$ type(C_1) ... type(C_n) value(A_1 o_1) ... value(A_i o_i) value(A_{i+1} v_{i+1}) ... value(A_n v_n))	
<i>Extra-Logical Declarations</i>		
nonFunctionalProperties P_1 v_1 . . P_n v_n [version v_0]	$annotation(P_1$ $v_1)$. . $annotation(P_n$ $v_n)$ [annotation(owl:versionInfo v_0)]	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: 1.81 \$")).
importOntologies $O_1,$. . O_n		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 $M_1,$. . M_n		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 Lite ⁻ abstract syntax		

WSML-Core Logical Expression	OWL Lite ⁻ Abstract Syntax	Remarks
------------------------------	---------------------------------------	---------

<i>TODO: find a better way to characterize the allowed logical expressions</i>		
$C \text{ subConceptOf } D_1, \dots, D_n$	Class (C partial $D_1 \dots D_n$)	
$?x \text{ memberOf } D \leftarrow$ $?x \text{ memberOf } C_1$ and ... and $?x \text{ memberOf } C_n$	Class (D partial $C_1 \dots C_n$)	
$?x \text{ memberOf } D \leftrightarrow$ $?x \text{ memberOf } C_1$ and ... and $?x \text{ memberOf } C_n$	Class (D complete $C_1 \dots C_n$)	
$C[R \text{ ofType } D]$	Class (C partial restriction(R allValuesFrom D))	
$C[U \text{ ofType } T]$	Class (C partial restriction(U allValuesFrom T))	
$S(?x, ?y) < R(?x, ?y)$	ObjectProperty (R super(S))	
$U_2(?x, ?y) < U_1(?x, ?y)$	DatatypeProperty (U_1 super(U_2))	
$?x \text{ memberOf } C \leftarrow$ $R(?x, ?y)$	ObjectProperty (R domain(C))	
$?y \text{ memberOf } C \leftarrow$ $R(?x, ?y)$	ObjectProperty (R range(S))	
$S(?y, ?x) \leftarrow$ $R(?x, ?y)$ $R(?y, ?x) \leftarrow$ $S(?x, ?y)$	ObjectProperty (R inverseOf(S))	
$R(?y, ?x) \leftarrow$ $R(?x, ?y)$	ObjectProperty (R Symmetric)	
$R(?x, ?y) \leftarrow$ $R(?x, ?y)$ and $R(?y, ?z)$	ObjectProperty (R Transitive)	
$o \text{ memberOf } C$ [$A_1 \text{ hasValues } o_1,$. $A_n \text{ hasValues } o_n$]	Individual(o type(C) value($A_1 o_1$) ... value($A_n o_n$))	Both the memberOf C and all the property values are optional.
<i>Datatype expressions</i>		

$F_1(y_1, \dots, y_k)$ and ... and $F_n(y_1, \dots, y_k)$	$\text{and}(F_1, \dots, F_n)$	F_i stands for a datatype expression component, which in this case amounts to a datatype predicate of arity k .
$\text{deCom}(y_1, \dots, y_k)$	deCom	deCom stands for a datatype predicate of arity k .
Table 2: Mapping between WSMML-Core logical expressions and OWL Lite ⁻ abstract syntax		

5. Mapping OWL Lite⁻ to WSMML-Core

Table 3 shows the mapping between OWL Lite⁻ (abstract syntax) and WSMML-Core. It contains the conceptual syntax as well as any additional logical expression that might be necessary to capture the semantics of the OWL Lite⁻ statement. The table shows for each construct supported by OWL Lite⁻ the WSMML-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 Lite⁻ can be captured in WSMML-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.

OWL Lite ⁻ Abstract syntax	WSMML-Core conceptual syntax	WSMML-Core logical expression
<i>Logical Definitions</i>		
Class(C partial $D_1 \dots D_n$)	concept C subConceptOf D_j (in case D_j is a named class) R_j ofType C_j (in case D_j is a value restriction of the form R_j allValuesFrom C_j) U_j ofType P_j (in case D_j is a value restriction of the form U_j allValuesFrom P_j)	$P_i(?y_1, \dots, y_n) \leftarrow ?x \text{ memberOf } C$ and $?x[U_{j_1} \text{ hasValues } ?y_1,$ \dots $U_{j_n} \text{ hasValues } ?y_n]$ (in case D_j is a value restriction of the form $U_{j_1} \dots U_{j_n}$ allValuesFrom P_j)
Class(C complete $D_1 \dots D_n$)	concept C subConceptOf D_1, \dots, D_n	$?x \text{ memberOf } C \leftarrow$ $?x \text{ memberOf } D_1$ and \dots and $?x \text{ memberOf } D_n$
EquivalentClasses($C_1 \dots C_n$)		$?x \text{ memberOf } C_i \leftarrow$ $?x \text{ memberOf } C_j$
ObjectProperty(R super(R_1) ... super(R_n) domain(C_1) ... domain(C_n))	relation R subRelationOf R_1, \dots, R_n domain ofType C_1, \dots, C_n range ofType D_1, \dots, D_n	

range(D_1) ... range(D_n)		
[inverseOf(R_0)]	relation R inverseOf(R_0)	
[Symmetric]	relation R symmetric	
[Transitive]	relation R transitive	
SubPropertyOf(R_1 R_x)		$R_2(?x,?y) \leftarrow R_1(?x,?y)$
EquivalentProperties(R_1 ... R_n)		$R_i(?x,?y) \leftarrow R_j(?x,?y)$
DatatypeProperty(U super(U_1) ... super(U_n) domain(C_1) ... domain(C_n) range(T_1) ... range(T_n))	relation U subRelationOf U_1, \dots, U_n dataRelation domain ofType C_1, \dots, C_n range ofType T_1, \dots, T_n	
SubPropertyOf(U_1 U_x)		$U_2(?x,?y) \leftarrow U_1(?x,?y)$
EquivalentProperties(U_1 ... U_n)		$U_i(?x,?y) \leftarrow U_j(?x,?y)$
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))	instance o memberOf C_1, \dots, C_n R_1 hasValues o_1 . . R_n hasValues o_n U_1 hasValues v_1 . . U_n hasValues v_n	
DatatypeExpression(P $deCom$)	datatype P definedBy $deCom$ (in case $deCom$ is a unary predicate or a conjunction of unary predicates)	
domain(T_1 ... T_k)	?value ofType T_1 (in case $k = 1$) ? y_1 ofType T_1 . . ? y_n ofType T_k (in case $k > 1$)	

$\text{and}(F_1 \dots F_n)$		$F_1(?y_1, \dots, ?y_k)$ and ... and $F_n(?y_1, \dots, ?y_k)$
<i>Extra-Logical Definitions</i>		
$\text{annotation}(P \ v)$	nonFunctionalProperties $P \ v$	
Table 3: Mapping between OWL Lite ⁻ abstract syntax and WSML-Core		

6. 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.

Listing	In WSML-Core?	Why not?	Steps to make it fall inside WSML-Core
<i>Ontologies</i>			
1: Domain Ontology "International Train Ticket"	no	- integrity constraints - equality	- Remove all (3) axioms
2: Domain Ontology "Date and Time"	no	- mixing abstract and concrete domains	- Complete remodeling of ontology

		- integrity constraints - equality	
3: Domain Ontology "Purchase"	yes		
4: Domain Ontology "Locations"	no	- mixing abstract and concrete domains - integrity constraints - equality (and also '<' and '>'; in the head!)	- Remove all axioms
<i>Goals and Web Services</i>			
5: Goal - buying a train ticket online	yes		
6: ÖBB Web Service for Booking Online Train Tickets for Austria and Germany	no	- equality	- Removing all restrictions written in the preCondition and the postCondition

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.

7. Conclusions

In this deliverable we have introduced the syntax of 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].

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.

Changelog

The following major updates have been done since the August 2nd version of the deliverable:

- The syntax in chapters 2, 4 and 5 have been brought into accordance with the new syntax of D2.
- Chapter 3: WSML-Core Logical Expressions has been added. The chapter explains the syntax for logical expressions in WSML-Core.
- Following the updated version of OWL Lite⁻ (to be published 2004-08-06), datatypes have been introduced. Chapters 2, 3, 4, 5, 6 and 7 have been updated

accordingly.

- Introduction and conclusions have been brought into accordance with the updates.
- Reviewer comments have been implemented (see [Appendix A](#)).
- Changelog has been added

References

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

[Berglund et al., 2004] A. Berglund, S. Boag, D. Chamberlin, Mary F. Fernández, M. Kay, J. Robie, and Jérôme Siméon (eds.): *XML Path Language (XPath) 2.0*, W3C Working Draft, available from <http://www.w3.org/TR/xpath20/>.

[de Bruijn et al., 2004] J. de Bruijn, A. Polleres, R. Lara and D. Fensel. *OWL⁻*. Deliverable d20v0.2, WSMML, 2004. To be published at <http://www.wsmo.org/2004/d20/v0.2/> on 2004-08-16.

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

[Fensel et al., 2001] D. Fensel, F. van Harmelen, I. Horrocks, D.L. McGuinness, and P.F. Patel-Schneider: OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16:2, May 2001.

[Grosz et al., 2003] B. N. Grosz, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48-57. ACM, 2003.

[Horrocks et al., 2004] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosz and M. Dean: *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission 21 May 2004. Available from <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>.

[Lloyd, 1987] J. W. Lloyd. *Foundations of Logic Programming (2nd edition)*. Springer-Verlag, 1987.

[Malhotra et al., 2004] A. Malhotra, J. Melton, N. Walsh: *XQuery 1.0 and XPath 2.0 Functions and Operators*, W3C Working Draft, available at <http://www.w3.org/TR/xpath-functions/>.

[Oren, 2004] E. Oren (ed.): *Languages for WSMO*, WSMML deliverable D16.0 version 0.2 Working Draft. available from <http://www.wsmo.org/2004/d16/d16.0/v0.2/>.

[Pan and Horrocks, 2004] J. Z. Pan and I. Horrocks, I.: *OWL-E: Extending OWL with expressive datatype expressions*. IMG Technical Report IMG/2004/KR-SW-01/v1.0, Victoria University of Manchester, 2004. Available from <http://dl-web.man.ac.uk/Doc/IMGTR-OWL-E.pdf>.

[Patel-Schneider et al., 2004] P. F. Patel-Schneider, P. Hayes, and I. Horrocks: *OWL web ontology language semantics and abstract syntax*. Recommendation 10 February 2004, W3C, 2004. Available from <http://www.w3.org/TR/owl-semantics/>.

[Roman et al., 2004] D. Roman, H. Lausen, and U. Keller (eds.): *Web Service Modeling Ontology - Standard (WSMO - Standard)*, WSMO deliverable D2 version 1.0 Working Draft 9 August 2004. available from <http://www.wsmo.org/2004/d2/v1.0/20040809/>.

[Stollberg et al., 2004] M. Stollberg, H. Lausen, A. Polleres, and R. Lara (eds.): *WSMO Use Case Modeling and Testing*, WSMO d3.2v0.1, version 2004-06-28. available from <http://www.wsmo.org/2004/d3/d3.2/v0.1/20040628/>.

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

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](#), [Esperanto](#), and [h-TechSight](#); by [Science Foundation Ireland](#) under the [DERI-Lion](#) project; and by the Vienna city government under the [CoOperate](#) program.

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

[1]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.

Appendix A. Reviewer Comments

Here follow the reviewer comments from the reviewer Ian Horrocks on the August 2nd version of this deliverable. Find in-line the responses of the authors to the comments:

```
| The language seems to fall into the same trap as OWL Lite in that it
| tries to impose *syntactic* restrictions on what can be said without
| being clear about the objective in terms of a target logic. The result
| may be (as in OWL Lite) that the restrictions don't achieve their
| objective (the language is stronger than was intended), or that some
| things that are allowed can only be expressed in a very convoluted
| way.
```

```
JB: The objective of the language is to have an ontology modeling language
| based on the conceptual model of ontologies presented in WSMO, which can
| be translated both to a subset of OWL Lite (OWL Lite-) and to plain
| Datalog.
```

```
You are right that the conceptual syntax does not capture the full ex-
| pressiveness of the language, but we believe that these features will
| not be used very much in practice. If, however, it turns out that certain
| feature are used in practice, we can always extend the conceptual syntax.
```

I would strongly recommend that you specify what the intended language is in terms of a DL or some other subset of FOL (e.g., function free Horn). You will then have a much crisper definition, and a much better understanding of the characteristics of your language (decidability, complexity etc); the underlying motivation for the design can also be made much clearer (e.g., easier reasoning). Of course it may also be useful to specify translations to/from languages such as OWL Lite, but these don't tell you anything very useful about your language.

JB: This has now been stated more clearly in the introduction.

Choosing to only include value (allValuesFrom) restrictions seems rather strange, is (as you point out) counter-intuitive, and makes the language of very limited value in many applications. Why not support both some/all values as in OWL Lite? If the reason is a requirement to restrict expressive power, this would be made much clearer by using a translation into logic as suggested above (to make it clear if the restriction really has any justification/benefit). If you do need to avoid having both forms of restriction, why not use a combined semantics (attributes translate to a combination of some/all), which is what is often intended in any case?

JB: The translation to a logic is done in OWL Lite-, therefore the translation is done indirectly. I'm not sure whether it would be necessary to provide a direct translation, although I guess we will have to do this when building an implementation.

You say that the language is an intersection of a DL and rules language - it might be appropriate to cite our WWW paper at this point [1].

JB: Reference is added to the introduction.

Are you sure that the syntax is parsable? (i.e., did you use a parser generator to produce a parser?). I am a little sceptical about this. E.g., using key-words (such as "subconcept-of") as the names of non-functional properties looks as though it could be problematical. (You don't mention any restriction on the use of key-words.)

JB: Note that the syntax in this version has been slightly updated to reflect the syntax updates to WSMO D2. The syntax is in fact based on the syntax defined in D2 (<http://www.wsmo.org/2004/d2/>). In fact it is a restriction and a slight extension of it. The grammar in Appendix B of D2 is parsable. There is not yet a grammar for WSML-Core, but this will be produced as soon as the syntax is fixed.

Is order important anywhere/everywhere in the syntax? This isn't made very clear.

JB: Yes, the order is important. I added a sentence about this to the introduction of chapter 2.

"non-functional-properties" seems a rather strange and unintuitive term for annotations.

JB: You could also say the 'annotations' is an unintuitive term for non-functional properties ;-)
We have decided to use this term in WSMO and I actually like it, because it seems to give the non-functional properties more importance than the 'annotations' keyword.

The split of expressivity between frames and axiom definitions seems arbitrary and unintuitive. You say that the axiom definitions don't really add anything, but are just a different way of writing things down. It would be much better if this were true, but it isn't! In particular, frames provide no way of specifying complete definitions (which you do note later on). Why not add a "complete" key word to frames?

JB: You are right, this statement is not completely true. I actually considered

adding the 'complete' keyword to the frames, but for complete class definitions, allValuesFrom restrictions are not allowed in OWL Lite-, because universal value restriction on the left-hand side of the GCI can not be translated to Datalog. Therefore, I think it would be un-intuitive to have this keyword here.
I also think that people who use WSML-Core are not that interested in complete class definitions. People who want more expressiveness can use of the extensions of WSML-Core. These extensions are to be developed. An overview of WSML languages can be found at:
<http://www.wsmo.org/2004/d16/d16.0/>

| It may be rather difficult for users to understand the restrictions on
| the form of axiom definitions - the only specification is a list of
| allowable forms, which is rather hard to understand/remember. There
| must be some clearer way of specifying the restrictions than this. As
| far as I can understand it, the example you give in S2.10 doesn't fit
| into the forms given in Table 2!

JB: Indeed. We'll try to come up with a better way of describing these in the next version of the deliverable.

| You need to state explicitly what all of the notation (C, D, R, S etc)
| in the tables stands for.

JB: Will be done for the next version.

| S2: "In this chapter we explain ... Most of the examples were taken
| from [Stollberg et al., 2004]" is repeated twice.

JB: fixed

| S2.5 Your description of the meaning of import makes it sound as
| though some concatenation of ontologies is going on at the syntactic
| level, whereas OWL imports simply specifies that the *semantics* is
| equivalent to a union of the axioms in all relevant ontologies. This
| is important because of, e.g., cyclical imports statements. You also
| don't explicitly state if you support recursive importation.

JB: The semantics is the same as in OWL. I added a note about this.

| S2.6 The specification of mediators is rather vague.

JB: I added a reference to WSMO D2 and tried to make it a bit more clear.

| S2.7 "implicitly anded" - I would say "explicitly conjoined" (anded is
| rather ugly). Moreover, given that you don't have any "complete"
| definitions, conjoining the superconcepts is superfluous - a set of
| superconcept statements is completely equivalent to a single conjoined
| one.

JB: fixed.

| You should also say something here about the semantics of attribute
| specifications (even though we find out later).

JB: fixed.

| You say that "We do not allow complex concepts definitions as
| superconcepts". This isn't quite true - attribute specifications are
| complex concepts.

JB: indeed, attribute specifications are complex concepts in the translation to OWL, but not in WSML-Core. We see concepts and attributes as conceptually very different elements and therefore, I would not want to say that an attribute definition is a complex concept.

| S2.8 "specialization of a direct super-relation" - do you really mean
| specialisation or specification? Also, this makes it sound as though
| there can only be one such super-relation, which is not the case.

JB: fixed

| S2.10 Axiom definitions. See general comments above.

JB: fixed.

| S4 "The table shows..." is repeated.

JB: fixed

| S5 the list of reasons for use case problems ends with "equality",
| with no explanation as to what this is about.

JB: fixed

| [1] Benjamin N. Grosf, Ian Horrocks, Raphael Volz, and Stefan
| Decker. Description logic programs: Combining logic programs with
| description logic. In Proc. of the Twelfth International World Wide
| Web Conference (WWW 2003), pages 48-57. ACM, 2003.

\$Date: 2004/08/13 18:08:48 \$