D36v1.0 WSML/XML

WSML Working Draft 28 July 2008

This version
http://www.wsmo.org/TR/d36/v1.0/20080728/

Latest version
http://www.wsmo.org/TR/d36/v1.0/

Previous version
http://www.wsmo.org/TR/d36/v0.1/20080715/

Editor:
Ioan Toma

Authors:
The WSML working group members

Copyright 2008 STI Innsbruck. All Rights Reserved. STI Innsbruck liability, trademark, document use, and software licensing rules apply.

This document is part of the specification of the Web Service Modeling Language (WSML), version 1.0.

The specification of WSML 1.0 consists of the following four documents.

- WSML Language Reference
- WSML Abstract Syntax and Semantics
- WSML/XML
- WSML/RDF
Abstract

This document presents the XML representation of WSML, called WSML/XML. A particular WSML description is an XML document which conforms to the WSML/XML schema.
Table of Contents

1. Introduction
2. XML Syntax for WSML
References
Acknowledgements
1. Introduction

This document presents the XML syntax for WSML. The XML exchange syntax is introduced to overcome the limitations with respect to the exchange of the human-readable syntax over the Web. This syntax is the preferred syntax for the exchange of WSML specifications between machines. The XML syntax for WSML is based on abstract syntax presented in [WSML-Semantics] and is compliant with the human-readable syntax presented in [WSML].

The Rule Interchange Format (RIF) Working Group is in the process of specifying the RIF Basic Logic Dialect (RIF-BLD) [RIF-BLD], which is an XML-based language for exchanging rules over the Web. RIF-BLD captures many of the features of the logical expression syntax of WSML-Rule. The RIF Framework for Logic Dialects (RIF-FLD) [RIF-FLD] may be used for defining extensions (or restrictions) of RIF-BLD; it captures all of the features of the WSML logical expression syntax. One may thus imagine a dialect of RIF that captures all features of the WSML logical expression syntax, and consequently the conceptual syntax for ontologies (e.g., through annotations). In addition, an RIF dialect may be defined that can capture WSML Web service, goal, and mediator descriptions. It is expected that future versions of WSML will specify how RIF may be used for the XML-based exchange of WSML descriptions over the Web.
2. XML Syntax for WSML

In this chapter, we explain the XML syntax for WSML. The XML syntax for WSML captures all WSML variants. The user can specify the variant of a WSML/XML document through the 'variant' attribute of the <wsml> root element.

The complete XML Schema, including documentation, for the WSML/XML syntax can be found is available online at http://www.wsmo.org/TR/d36/v1.0/20080728/xml-syntax/wsml_xml_syntax.xsd.

This schema includes two module schemas:

- WSML identifiers (http://www.wsmo.org/TR/d36/v1.0/20080728/xml-syntax/wsml_identifiers.xsd)
- Logical expressions of WSML (http://www.wsmo.org/TR/d36/v1.0/20080728/xml-syntax/wsml_expr.xsd).

The WSML identifiers schema is a XML schema defining the WSML identifiers, such as IRIs, anonymous identifiers, variables, etc. Elements of this schema are used to define elements of WSML/XML schema document and WSML Logical expression schema. The WSML Logical expressions schema elements are used to define the elements of WSML/XML schema document.

Furthermore, the schema imports an additional schema for the basic Dublin Core elements (http://dublincore.org/schemas/xmls/qdc/2003/04/02/dc.xsd).

Userfriendly documentation for the schemas is available from the following locations:

- XML syntax for WSML logical expressions: http://www.wsmo.org/TR/d36/v1.0/xml-syntax/documentation/wsml_expr.xsd.html

In the rest of the section we introduce the transformation from WSML abstract and surface syntax to WSML XML syntax. Table 2.1 provides the transformations of the surface and abstract syntax, Table 2.2 presents the transformation of logical expressions, while in Table 2.3 a simple mapping example is given.

The basic namespace for WSML/XML is http://www.wsmo.org/wsml/wsml-syntax#. This is the namespace for all elements in WSML.

Before beginning the transformation process the following pre-processing steps need to be performed:

- sQNames need to be resolved to full IRIs, i.e., there will not be any namespace definitions in the XML syntax.
- Datatype identifiers are resolved to full IRIs by replacing the leading underscore '_' with the WSML namespace, according to [WSML], Appendix B, Table B.1. For example, the datatype identifier '_date' is resolved to http://www.wsmo.org/wsml/wsml-syntax#date.
- Built-in functions and predicates are resolved to the corresponding full IRIs, as defined in [WSML] Appendix B.
- Inside logical expressions, compound molecules are split into a conjunction of simple molecules.

T(...) denotes a function which is recursively called to transform the a fragment of WSML abstract syntax or surface synatx to the XML syntax. An WSML abstract syntax
fragment or WSML surface syntax fragment is given as parameter to function $T$.

In Table 2.1, all WSML keywords are marked bold. $A, B, C$ stand for identifiers, $D$ stands for a datatype identifier, $D_V$ stands for an integer value, $D_V^d$ stands for a decimal, and $D_V^s$ stands for a string data value. $Name$ stand for identifier or symbol nil, $k, m, n$ are integer numbers. Productions in the grammar are underlined and linked to the production rules in [WSML]. Appendix A. Note that in the table we use the familiar sQName macro mechanism to abbreviate the IRIs of resources.

In the table, &wsml; stands for the WSML namespace http://www.wsmo.org/wsml/wsml-syntax# and &xs; stands for the XML schema namespace http://www.w3.org/2001/XMLSchema#.

### Table 2.1: Mapping of WSML abstract syntax and WSML surface syntax to WSML/XML syntax

<table>
<thead>
<tr>
<th>WSML abstract syntax</th>
<th>WSML surface syntax</th>
<th>XML Tree</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(varID, O.G.W.S.M.C)_{description}$</td>
<td>$T(wsmlVariant varID$</td>
<td>&lt;wsml xmlns=&quot;http://www.wsmo.org/wsml/wsml-syntax#&quot; variant=&quot;varID&quot;&gt;</td>
<td>definition, ... definition, are Ontology, Goal, WebService, Mediators, Capability and Interface: O.G.W.S.M.C are the set of previous mentioned entities; varID is the WSML variant</td>
</tr>
<tr>
<td>$T(ontology Name$</td>
<td>$&lt;ontology name=&quot;Name&quot;&gt;</td>
<td>$T(header_1$</td>
<td>An ontology _element represents all possible content of an ontology definition, i.e., concepts, relations, instances, ...; header,...,header, are header elements that contain elements from annotations, ontology identifiers (imported ontologies) and ooMediators identifiers sets denoted by: ann,ontology.medID; ontology_element,...,ontology_element, are ontology elements from the sets: concept, relation.instance.relinstance.axiom</td>
</tr>
<tr>
<td>$T(concept Name$</td>
<td>$&lt;concept name=&quot;Name&quot;&gt;</td>
<td>$T(annotations$</td>
<td>ann is a set of WSML annotations represented using surface syntax as annotations; conceptID is the set of super-concepts identifiers represented using surface syntax as $(B_1,...,B_n)$ and attribute is the set of attributes represented using surface syntax as attribute,...,attribute,</td>
</tr>
<tr>
<td>$T(Name,ann,conceptID,attribute)_{concept}$</td>
<td>$T(annotations)$$&lt;superConcept&gt;<em>{B_1},&lt;superConcept&gt;</em>{B_2}$</td>
<td>$T(attribute...)$</td>
<td></td>
</tr>
<tr>
<td>$T(Name,defType,feature,...)$</td>
<td>$&lt;range&gt;C&lt;range&gt;$</td>
<td>$T(cardinality)$</td>
<td></td>
</tr>
<tr>
<td>$T(Name,defType,feature,...)$</td>
<td>$&lt;range&gt;C&lt;range&gt;$</td>
<td>$T(cardinality)$</td>
<td></td>
</tr>
</tbody>
</table>

An ontology _element represents all possible content of an ontology definition, i.e., concepts, relations, instances, ...; header, ... header, are header elements that contain elements from annotations, ontology identifiers (imported ontologies) and ooMediators identifiers sets denoted by: ann,ontology.medID; ontology_element,...,ontology_element, are ontology elements from the sets: concept, relation.instance.relinstance.axiom.
T (annotations)  
<attribute name="Name" type="inferring">  
<range>C</range>  
T(attributefeature)  
...  
T(attributefeature)  
T(cardinality)  
T(annotations)  
</attribute>

ann is a set of WSML annotations represented using surface syntax as annotations; defType is impliesType, feature is the set of attribute features represented using surface syntax as attributefeature₁,...,attributefeatureₙ; minCard and maxCard correspond to cardinality and rangeID is a set of concept identifiers.

WSML Attribute Definition

<table>
<thead>
<tr>
<th>T (Name, ann, defType, feature, minCard, maxCard, rangeID)</th>
<th>T (Name, ann, defType, feature, minCard, maxCard, rangeID)</th>
<th>T (feature)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T (Name, ann, defType, feature, minCard, maxCard, rangeID)</td>
<td>T (feature)</td>
<td>T (feature)</td>
</tr>
<tr>
<td>T (transitive)</td>
<td>T (symmetric)</td>
<td>T (reflexive)</td>
</tr>
<tr>
<td>T (inverseOf)</td>
<td>T (subAttributeOf)</td>
<td>T (minCard, maxCard)</td>
</tr>
<tr>
<td>T (minCard, maxCard)</td>
<td>T (card)</td>
<td>T (instance Name, memberOf C₁,...,Cₙ)</td>
</tr>
</tbody>
</table>

ann is a set of WSML annotations represented using surface syntax as annotations; defType is impliesType, feature is the set of attribute features represented using surface syntax as attributefeature₁,...,attributefeatureₙ; minCard and maxCard correspond to cardinality and rangeID is a set of concept identifiers.

WSML Instance

<table>
<thead>
<tr>
<th>T (Name, ann, conceptID, attVal)</th>
<th>T (instance Name, memberOf C₁,...,Cₙ)</th>
<th>T (attVal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T (instance Name, memberOf C₁,...,Cₙ)</td>
<td>T (attVal)</td>
<td>T (attVal)</td>
</tr>
<tr>
<td>T (attVal)</td>
<td>T (attVal)</td>
<td>T (attVal)</td>
</tr>
</tbody>
</table>

ann is a set of WSML annotations represented using surface syntax as annotations; conceptID is a set of concept identifiers represented using the surface syntax as C₁,...,Cₙ; attVal is a set of attribute value pairs represented using the surface syntax as attributevalue₁,...,attributevalueₙ.
<table>
<thead>
<tr>
<th>WSML Attribute Value Pair</th>
<th>T ( (Name, valueID)_{av}</th>
<th>T ( Name hasValue (value_1, ..., value_n) )</th>
<th>&lt;attributeValue name=&quot;Name&quot;&gt; T(value_1) ... T(value_n) &lt;/attributeValue&gt;</th>
</tr>
</thead>
</table>

A value has always a datatype. There are four built-in datatypes: IRI, string, integer, decimal. In addition any arbitrary datatype can be defined by use of datatype wrappers. The next four transformations show how to handle these five cases; valueID is a set of instance and data value identifiers represented using surface syntax as value_1,...,value_n. Please note that attribute value pairs used in a nonfunctional property definition might include also variables.

| WSML IRI | T ( IRI ) | T ( IRI ) | <value type="&wsml;iri"> IRI </value> |

| String | T ( DV_s ) | T ( DV_s ) | <value type="&xsd;string"> DV_s </value> |

| Integer | T ( DV_i ) | T ( DV_i ) | <value type="&xsd;integer"> DV_i </value> |

| Decimal | T ( DV_d ) | T ( DV_d ) | <value type="&xsd;decimal"> DV_d </value> |

| WSML Data Value | T ( DV ) | T ( DV ) | <value type="http://www.example.org/datatype#any"> <argument>DV.arg1</argument> ... <argument>DV.arg_n</argument> </value> |

| WSML Variable | T ( V ) | T ( V ) | <value> <variable name="V"/> </value> |

| WSML Relation | T ( (Name, ann, relationID, parameter)_{relation} ) | T ( relation Name / n (paramtype_1,...,paramtype_m) ) | <relation name="Name" arity="n"> <parameters> T(paramtype_1)> ... T(paramtype_m) </parameters> <superRelation>C_1</superRelation> </relation> |

Note that the parameters of a relation are ordered and thus the order of the parameters elements in the XML representation is important. ann is a set of WSML annotations represented using surface syntax as annotations, relationID is a set of (super-)relation identifiers represented using the surface syntax as...
WSML Parameter Definition

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{ann,defType,domId} )</td>
<td>[ T( \text{ann,defType,domId} ) ] [ T( \cdot\cdot\cdot ) ] [ T( \cdot\cdot\cdot ) ]</td>
</tr>
</tbody>
</table>

WSML Relation Instance

<table>
<thead>
<tr>
<th>Relation Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Name,ann,relationID,parVal} )</td>
<td>[ T( \text{Name,ann,relationID,parVal} ) ] [ T( \cdot\cdot\cdot ) ] [ T( \cdot\cdot\cdot ) ]</td>
</tr>
</tbody>
</table>

WSML Parameter Values

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{ann,defType,domId} )</td>
<td>[ T( \text{ann,defType,domId} ) ] [ T( \cdot\cdot\cdot ) ] [ T( \cdot\cdot\cdot ) ]</td>
</tr>
</tbody>
</table>

WSML Axiom

<table>
<thead>
<tr>
<th>Axiom Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Name,ann,logExp} )</td>
<td>[ T( \text{Name,ann,logExp} ) ] [ T( \cdot\cdot\cdot ) ] [ T( \cdot\cdot\cdot ) ]</td>
</tr>
</tbody>
</table>

WSML Logical Expression

<table>
<thead>
<tr>
<th>Expression Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{logExp} )</td>
<td>[ T( \text{logExp} ) ] [ T( \cdot\cdot\cdot ) ] [ T( \cdot\cdot\cdot ) ]</td>
</tr>
</tbody>
</table>

Note that the parameters of a relation instance are ordered and thus the order of the value elements is important. Ann is a set of WSML annotations represented using surface syntax as \( \cdot\cdot\cdot \), defType is of type, domId is a set of concept identifiers represented using the surface syntax as \( \cdot\cdot\cdot \).

The mapping of logical expressions is defined in Table 2.2.
WSML Goal

\( T \left( \begin{array}{l}
\text{Name}, \text{ann}, \text{ontID}, \text{medID}, \text{nfp}, \text{ontID}, \text{capability}, \text{interface}
\end{array} \right) \)

\( <\text{goal name} = \text{A} > \\
\ T(\text{header}_1) \\
\ T(\text{nfp}_1) \\
\ ... \\
\ T(\text{nfp}_n) \\
\ T(\text{capability}) \\
\ T(\text{interface}_1) \\
\ ... \\
\ T(\text{interface}_n) \\
\</goal> 

header\(_1, \ldots, \text{header}_n\), are header elements that contain elements from annotations, ontology identifiers (imported ontologies) and ooMediators identifiers sets denoted by: \text{ann}, \text{ontID}, \text{medID}; \text{nfp}\(_1, \ldots, \text{nfp}_n\) are nonfunctional properties represented using the surface syntax as \text{nfp}_1, \ldots, \text{nfp}_n; \text{capability} is a WSML capability represented using the surface syntax as \text{capability}; \text{interface}\(_1, \ldots, \text{interface}_n\) are sets of WSML interfaces represented using the surface syntax as \text{interface}_1, \ldots, \text{interface}_n.

WSML ooMediator

\( T \left( \begin{array}{l}
\text{Name}, \text{ann}, \text{ontID}, \text{nfp}_1, \ldots, \text{nfp}_n, \text{sourceID}, \text{targetID}, \text{serviceID}
\end{array} \right) \)

\( <\text{ooMediator name} = \text{A} > \\
\ T(\text{annotations}) \\
\ T(\text{importsontology}) \\
\ T(\text{nfp}_1) \\
\ ... \\
\ T(\text{nfp}_n) \\
\ T(\text{source}) \\
\ T(\text{target}) \\
\ T(\text{use} \_ \text{service}) \\
\</ooMediator> 

\text{annotations} and \text{importsontology} correspond to sets \text{ann} respectively \text{ontID}; \text{ontology}-element\(_1, \ldots, \text{ontology}-element\(_n\) are ontology elements from the sets: concept, relation, instance, relInstance, axiom

WSML ggMediator

\( T \left( \begin{array}{l}
\text{Name}, \text{ann}, \text{ontID}, \text{nfp}_1, \ldots, \text{nfp}_n, \text{sourceID}, \text{targetID}, \text{serviceID}
\end{array} \right) \)

\( <\text{ggMediator name} = \text{A} > \\
\ T(\text{header}_1) \\
\ T(\text{nfp}_1) \\
\ ... \\
\ T(\text{nfp}_n) \\
\ T(\text{source}) \\
\ T(\text{target}) \\
\ T(\text{use} \_ \text{service}) \\
\</ggMediator> 

header\(_1, \ldots, \text{header}_n\), are header elements that contain elements from annotations, ontology identifiers (imported ontologies) and ooMediators identifiers sets denoted by: \text{ann}, \text{ontID}, \text{medID}; \text{nfp}\(_1, \ldots, \text{nfp}_n\) are nonfunctional properties from the set \text{nfp}; \text{source} and \text{target} are goal and ggMediator identifiers from the sets \text{sourceID} and \text{targetID}; \text{use} \_ \text{service} are goal, Web service and wwMediator identifiers from the set \text{serviceID}

WSML wgMediator

\( T \left( \begin{array}{l}
\text{Name}, \text{ann}, \text{ontID}, \text{nfp}_1, \ldots, \text{nfp}_n, \text{sourceID}, \text{targetID}, \text{serviceID}
\end{array} \right) \)

\( <\text{wgMediator name} = \text{A} > \\
\ T(\text{header}_1) \\
\ T(\text{nfp}_1) \\
\ ... \\
\ T(\text{nfp}_n) \\
\ T(\text{source}) \\
\ T(\text{target}) \\
\ T(\text{use} \_ \text{service}) \\
\</wgMediator> 

\text{header}_1, \ldots, \text{header}_n\), are header elements that contain elements from annotations, ontology identifiers (imported ontologies) and ooMediators identifiers sets denoted by: \text{ann}, \text{ontID}, \text{medID}; \text{nfp}_1, \ldots, \text{nfp}_n\) are nonfunctional properties from the set \text{nfp}; \text{source} and \text{target} are goal and ggMediator identifiers from the sets \text{sourceID} and \text{targetID}; \text{use} \_ \text{service} are goal, Web service and wwMediator identifiers from the set \text{serviceID}
pre_post_ass_or_eff unites the axiom definitions for precondition, assumption, postcondition and effect.
| WSML Interface | T (interface A (Name, ann, ontID, mediID, nfp, choreID)) | <interface name="A"> T(header, ...) T(header, ...) T(nfp, ...) T(choreography) T(orchestration) </interface> |
| WSML Choreography | T (choreography C (Name, ann, ontID, mediID, nfp, signature, rule)) | <choreography name="C"> T(header, ...) T(header, ...) T(statesignature) T(transitions) </choreography> |
| | | rule is a set of WSML rules represented using surface syntax as transitions, where a WSML rule can be an if rule, forall rule, choose rule, piped rule, add rule, delete rule |

| T (sharedVar) | T(sharedVar) T(pre_post_ass_or_eff, ...) T(pre_post_ass_or_eff, ...) |
| T(pre_post_ass_or_eff) | T(pre_post_ass_or_eff) T(pre_post_ass_or_eff) |
| <sharedVariables> | <variable name="?v1"/> <variable name="?v2"/> ... <variable name="?vn"/> </sharedVariables> |
| T(pre) | T(precondition B axiomdefinition ) <precondition name="B"> T(axiomdefinition) </precondition> |
| T(ass) | T(assumption B axiomdefinition ) <assumption name="B"> T(axiomdefinition) </assumption> |
| T(post) | T(postcondition B axiomdefinition ) <postcondition name="B"> T(axiomdefinition) </postcondition> |
| T(ef) | T(effect B axiomdefinition ) <effect name="B"> T(axiomdefinition) </effect> |

WSML Interface

WSML Choreography
transitions

</choreography>

WSML State Signature

T ( Name, ann, ontID, medID, nfp, mode )

T ( statesignature C
header1
...
header1
mode1
...
mode n )

<statesignature name="C">
T(header1)
...
T(header1)
T(mode1)
...
T(mode n )
</statesignature>

WSML Mode

T ( type, conceptID, groundingID )

T ( type concept A
grounding )

<mode>
T(type)
<concept>
<iri>A</iri>
T(grounding)
</mode>

WSML Mode

T ( type, conceptID, groundingID )

T ( type relation A
grounding )

<mode>
T(type)
<relation>
<iri>A</iri>
T(grounding)
</mode>

T ( type, grounding )

T ( static )

<static/>

T ( type )

T ( in )

<in/>

T ( type )

T ( out )

<out/>

T ( type )

T ( shared )

<shared/>

T ( type )

T ( controlled )

<controlled/>

T ( grounding )

T ( grounding {A1,...,An} )

<grounding>A1</grounding>
...
<grounding>Am</grounding>
<table>
<thead>
<tr>
<th>WSML Transition Rule</th>
<th>T (logExp, rule)</th>
<th>T (condition, rule, ...)</th>
<th>rule is a set of WSML transition rules expressed using the surface syntax as: rule, ..., rule, and logExp is a WSML logical expression expressed using the surface syntax as condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSML Transition Rule</td>
<td>T (forall variable, condition, rule, ...)</td>
<td>T (forall variable, condition, rule, ...)</td>
<td>varID is a set of variable identifiers expressed using the surface syntax as: variable, ..., variable, rule is a set of WSML transition rules expressed using the surface syntax as: rule, ..., rule, and logExp is a WSML logical expression expressed using the surface syntax as condition</td>
</tr>
<tr>
<td>WSML Transition Rule</td>
<td>T (forall variable, condition, rule, ...)</td>
<td>T (forall variable, condition, rule, ...)</td>
<td>varID is a set of variable identifiers expressed using the surface syntax as: variable, ..., variable, rule is a set of WSML transition rules expressed using the surface syntax as: rule, ..., rule, and logExp is a WSML logical expression expressed using the surface syntax as condition</td>
</tr>
<tr>
<td>WSML Transition Rule</td>
<td>T (choose variable, condition, rule, ...)</td>
<td>T (choose variable, condition, rule, ...)</td>
<td>varID is a set of variable identifiers expressed using the surface syntax as: variable, ..., variable, rule is a set of WSML transition rules expressed using the surface syntax as: rule, ..., rule, and logExp is a WSML logical expression expressed using the surface syntax as condition</td>
</tr>
<tr>
<td>WSML Transition Rule</td>
<td>T (rule)</td>
<td>T (rule)</td>
<td>rule is a set of WSML transition rules expressed using the surface syntax as: rule, ..., rule, and logExp is a WSML logical expression expressed using the surface syntax as condition</td>
</tr>
<tr>
<td>WSML Transition Rule</td>
<td>T (rule)</td>
<td>T (rule)</td>
<td>rule is a set of WSML transition rules expressed using the surface syntax as: rule, ..., rule, and logExp is a WSML logical expression expressed using the surface syntax as condition</td>
</tr>
<tr>
<td>WSML Transition Rule</td>
<td>T (fact)</td>
<td>T (fact)</td>
<td>fact is a ground atomic formula</td>
</tr>
<tr>
<td>WSML Transition Rule</td>
<td>T (fact)</td>
<td>T (fact)</td>
<td>fact is a ground atomic formula</td>
</tr>
</tbody>
</table>
The logical expression syntax is explained in Table 2.2. In the table, A stands for identifiers and $T_1, ..., T_n$ stand for terms. V stands for a variable.

<table>
<thead>
<tr>
<th>WSML Orchestration</th>
<th>T (orchestration)</th>
<th>&lt;orchestration&gt;A&lt;/orchestration&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSML Nonfunctional Property</td>
<td>T (name, ann, value, logExp)$_{np}$</td>
<td>&lt;nonFunctionalProperty name=&quot;name&quot;&gt; T(annotations) T(value) T(log_definition) &lt;/nonFunctionalProperty&gt;</td>
</tr>
<tr>
<td>WSML Logical Definition</td>
<td>T (logDefinition)</td>
<td>&lt;logicalDefinition&gt; &lt;definedBy&gt; T(log_expr) ... T(log_expr) &lt;/definedBy&gt; &lt;/logicalDefinition&gt;</td>
</tr>
<tr>
<td>WSML Annotations</td>
<td>T (name, value)$_{ann}$</td>
<td>&lt;annotations&gt; T(attributevalue) ... T(attributevalue) &lt;/annotations&gt;</td>
</tr>
<tr>
<td></td>
<td>T (ontID)</td>
<td>&lt;importsOntology&gt; A$_1$, &lt;importsOntology&gt; A$_2$, ... &lt;importsOntology&gt; A$_n$ &lt;/importsOntology&gt;</td>
</tr>
<tr>
<td></td>
<td>T (medID)</td>
<td>&lt;usesMediator&gt; B$_1$, &lt;usesMediator&gt; B$_2$, ... &lt;usesMediator&gt; B$_n$ &lt;/usesMediator&gt;</td>
</tr>
</tbody>
</table>

The logical expression syntax is explained in Table 2.2. In the table, A stands for identifiers and $T_1, ..., T_n$ stand for terms. V stands for a variable.
<table>
<thead>
<tr>
<th>WSML Term</th>
<th>T' (V)</th>
<th>T' (V)</th>
<th>&lt;term name=&quot;V&quot;/&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSML Atom</td>
<td>T (A(term₁, ..., termₙ))</td>
<td>T (A(term₁, ..., termₙ))</td>
<td>&lt;atom name=&quot;A&quot;&gt; T'(termᵢ, arg) ... T'(termₙ, arg) &lt;/atom&gt;</td>
</tr>
<tr>
<td>true</td>
<td>T (true)</td>
<td>T (true)</td>
<td>&lt;true/&gt;</td>
</tr>
<tr>
<td>false</td>
<td>T (false)</td>
<td>T (false)</td>
<td>&lt;false/&gt;</td>
</tr>
<tr>
<td>equal</td>
<td>T (T₁ = T₂)</td>
<td>T (T₁ equal T₂)</td>
<td>&lt;equal&gt; T(T₁) T(T₂) &lt;/equal&gt;</td>
</tr>
<tr>
<td>subConceptOf</td>
<td>T (T₁ :: T₁,..., Tₙ)</td>
<td>T (T₁ subConceptOf T₁,..., Tₙ)</td>
<td>&lt;molecule&gt; T(T₁) &lt;isa type=&quot;subConceptOf&quot;&gt; T(T₂) ... T(Tₙ) &lt;/isa&gt; &lt;/molecule&gt;</td>
</tr>
<tr>
<td>memberOf</td>
<td>T (T₁ : T₂,..., Tₙ)</td>
<td>T (T₁ memberOf T₂,..., Tₙ)</td>
<td>&lt;molecule&gt; T(T₁) &lt;isa type=&quot;memberOf&quot;&gt; T(T₂) ... T(Tₙ) &lt;/isa&gt; &lt;/molecule&gt;</td>
</tr>
<tr>
<td></td>
<td>T (T [attr_relation₁, ..., attr_relationₙ])</td>
<td>T (T [attr_relation₁, ..., attr_relationₙ])</td>
<td>&lt;molecule&gt; T(T) T(attr_relation₁) ... T(attr_relationₙ) &lt;/molecule&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;attributeDefinition type=&quot;constraining&quot;&gt;</td>
</tr>
</tbody>
</table>

Page 16 of 22
<table>
<thead>
<tr>
<th>ofType</th>
<th>T (T₀ ofType {T₁, ..., Tₙ})</th>
<th>T (T₀ ofType {T₁, ..., Tₙ})</th>
<th>T(T₀ name) T(T₀ type) ... T(Tₙ type) &lt;attributeDefinition&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>impliesType</td>
<td>T (T₀ impliesType {T₁, ..., Tₙ})</td>
<td>T (T₀ impliesType {T₁, ..., Tₙ})</td>
<td>T(T₀ name) T(T₀ type) ... T(Tₙ type) &lt;attributeDefinition&gt;</td>
</tr>
<tr>
<td>hasValue</td>
<td>T (T₀ hasValue {T₁, ..., Tₙ})</td>
<td>T (T₀ hasValue {T₁, ..., Tₙ})</td>
<td>T(T₀ name) T(T₀ value) ... T(Tₙ value) &lt;attributeValue&gt;</td>
</tr>
<tr>
<td>and</td>
<td>T (expr₁ ∧ expr₂)</td>
<td>T (expr₁ and expr₂)</td>
<td>T(expr₁) T(expr₂) &lt;/and&gt;</td>
</tr>
<tr>
<td>or</td>
<td>T (expr₁ ∨ expr₂)</td>
<td>T (expr₁ or expr₂)</td>
<td>T(expr₁) T(expr₂) &lt;/or&gt;</td>
</tr>
<tr>
<td>neg</td>
<td>T (¬expr)</td>
<td>T (neg expr)</td>
<td>T(expr) &lt;/neg&gt;</td>
</tr>
<tr>
<td>naf</td>
<td>T (not expr)</td>
<td>T (naf expr)</td>
<td>T(expr) &lt;/naf&gt;</td>
</tr>
<tr>
<td>implies</td>
<td>T (expr₁ implies expr₂)</td>
<td>T (expr₁ implies expr₂)</td>
<td>T(expr₁) T(expr₂) &lt;/implies&gt;</td>
</tr>
<tr>
<td>impliedBy</td>
<td>T (expr₁ impliedBy expr₂)</td>
<td>T (expr₁ impliedBy expr₂)</td>
<td>T(expr₁) T(expr₂) &lt;/impliedBy&gt;</td>
</tr>
</tbody>
</table>
Table 2.3 provides a simple translation example.

<table>
<thead>
<tr>
<th>Translation</th>
<th>WSML Syntax</th>
<th>XML Tree</th>
</tr>
</thead>
</table>
| equivalent    | $T (\text{expr}_1 \equiv \text{expr}_2)$ | `<equivalent>`
|               | $T (\text{expr}_1 \text{ equivalent} \text{expr}_2)$ | $T(\text{expr}_1)$
|               | $T(\text{expr}_2)$ | `$\langle\text{equivalent}\rangle$` |
|forall         | $T (\forall \varphi_f, \ldots, \forall \varphi_n (\text{expr}))$ | `<forall>`
|               | $T (\forall \varphi_f, \ldots, \forall \varphi_n (\text{expr}))$ | $\langle\text{forall}\rangle$
|               | $T(\varphi_f)$ | `$\langle\text{forall}\rangle$`
|exists         | $T (\exists \varphi_f, \ldots, \exists \varphi_n (\text{expr}))$ | `<exists>`
|               | $T (\exists \varphi_f, \ldots, \exists \varphi_n (\text{expr}))$ | $\langle\text{exists}\rangle$
|               | $T(\varphi_f)$ | `$\langle\text{exists}\rangle$`
|constraint     | $T (\text{expr})$ | `<constraint>`
|               | $T (\text{expr})$ | `$\langle\text{constraint}\rangle$` |
|impliedByLP    | $T (\text{expr}_2 \text{ expr}_1)$ | `<impliedByLP>`
|               | $T (\text{expr}_1 \text{ :-} \text{expr}_2)$ | $T(\text{expr}_1)$
|               | $T(\text{expr}_2)$ | `$\langle\text{impliedByLP}\rangle$`

Table 2.3: A Mapping Example

<table>
<thead>
<tr>
<th>WSML Syntax</th>
<th>XML Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ontology _http://www.example.org/ex1&quot;</code></td>
<td><code>&lt;ontology name=&quot;http://www.example.org/ex1&quot;&gt;</code></td>
</tr>
<tr>
<td><code>annotations</code> dc#title hasValue &quot;WSML to RDF&quot; dc#date hasValue _date(2005,12,12)`</td>
<td><code>&lt;annotations&gt;</code></td>
</tr>
<tr>
<td><code>endAnnotations</code></td>
<td><code>&lt;attributeValue name=&quot;http://purl.org/dc/elements/1.1#title&quot;&gt; WSML to RDF </code>&lt;/value&gt;`</td>
</tr>
<tr>
<td><code>importsOntology _http://www.example.net/ex2&quot;</code></td>
<td><code>&lt;attributeValue&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;attributeValue name=&quot;http://purl.org/dc/elements/1.1#date&quot;&gt; 2005&lt;/argument&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;argument&gt;12&lt;/argument&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/attributeValue&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/attributeValue&gt;</code></td>
</tr>
</tbody>
</table>
concept Woman subConceptOf (ex#Human, ex#LivingBeing)
  name ofType _string
  ancestorOf transitive impliesType ex#Human
  age ofType (1) _integer

axiom GenderConstraint
  definedBy
  !- ?x memberOf ex#Man and
  ?x memberOf Woman.

instance Mary memberOf Woman
  name hasValue "Mary Jones"
  age hasValue 23

relation childOf
  (ofType ex#Human, impliesType ex#Parent)

webService ws
  capability itineraryInfo
  interface (_"http://example.org/i1", _"http://example.org/i2")
<relation name="http://www.example.org/ex1#childOf">
  <parameters>
    <parameter type="constraining">
      <range>http://www.example.org/ex2#Human</range>
    </parameter>
    <parameter type="inferring">
      <range>http://www.example.org/ex2#Parent</range>
    </parameter>
  </parameters>
</relation>

<wsmi>
  <ontology>
    <webService name="http://www.example.org/ex1#ws">
      <capability name="http://www.example.org/ex1#itineraryInfo"/>
      <interface name="http://example.org/i1"/>
      <interface name="http://example.org/i2"/>
    </webService>
  </ontology>
</wsmi>
References


Acknowledgements

The work is partially funded by the European Commission under the projects ASG, EASAIER, enIRaF, Knowledge Web, Musing, Salero, Seemp, Semantic- GOV, Super, SWING and TripCom; by Science Foundation Ireland under the DERI-Lion Grant No.SFI/02/CE1/I13; by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects Grisino, RW², Sem-Biz, SeNSE and TSC.

The authors would like to thank to all the members of the WSML working group for their advice and input into this document.