



D36v1.0 WSML/XML

WSML Final Draft 08 August 2008

This version

<http://www.wsmo.org/TR/d36/v1.0/20080808/>

Latest version

<http://www.wsmo.org/TR/d36/v1.0/>

Previous version

<http://www.wsmo.org/TR/d36/v1.0/20080728/>

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/v1.0/20080808/xml-syntax/wsml_xml_syntax.xsd.

This schema includes two module schemas:

- WSML identifiers (http://www.wsmo.org/TR/d36/v1.0/20080808/xml-syntax/wsml_identifiers.xsd)
- Logical expressions of WSML (http://www.wsmo.org/TR/d36/v1.0/20080808/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:*
http://www.wsmo.org/TR/d36/v1.0/xml-syntax/documentation/wsml_xml_syntax.xsd.html
- *XML syntax for WSML identifiers:*
http://www.wsmo.org/TR/d36/v1.0/xml-syntax/documentation/wsml_identifiers.xsd.html
- *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 WSMML/XML is <http://www.wsmo.org/wsmml/wsmml-syntax#>. This is the namespace for all elements in WSMML.

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

- Compact URIs need to be resolved to full IRIs, i.e., there will not be any namespace definitions in the XML syntax.
- in the conceptual syntax universal truth, universal falsehood and unnumbered anonymous IDs are resolved to: <http://www.wsmo.org/wsmml/wsmml-syntax#true>, <http://www.wsmo.org/wsmml/wsmml-syntax#false>, and <http://www.wsmo.org/wsmml/wsmml-syntax#anonymousID>, respectively. Numbered anonymous IDs are resolved as such: [_#1](http://www.wsmo.org/wsmml/wsmml-syntax#anonymousID1) to <http://www.wsmo.org/wsmml/wsmml-syntax#anonymousID1>, [_#2](http://www.wsmo.org/wsmml/wsmml-syntax#anonymousID2) to <http://www.wsmo.org/wsmml/wsmml-syntax#anonymousID2>, ..., [_#n](http://www.wsmo.org/wsmml/wsmml-syntax#anonymousIDn) to <http://www.wsmo.org/wsmml/wsmml-syntax#anonymousIDn>.
- Datatype identifiers are resolved to full IRIs by replacing the leading underscore '_' with the WSMML namespace, according to [WSMML], Appendix B, Table B.1. For example, the datatype identifier '_date' is resolved to <http://www.wsmo.org/wsmml/wsmml-syntax#date>.
- Built-in functions and predicates are resolved to the corresponding full IRIs, as defined in [WSMML] 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 WSMML abstract syntax or surface syntax to the XML syntax. An WSMML abstract syntax fragment or WSMML surface syntax fragment is given as parameter to function T.

In Table 2.1, all WSMML keywords are marked bold. *A,B,C* stand for identifiers, *D* stands for a datatype identifier, DV_i stands for an integer value, DV_d stands for a decimal, and DV_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 [WSMML], Appendix A. Note that in the table we use the familiar sQName macro mechanism to abbreviate the IRIs of resources.

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

	WSML abstract syntax	WSML surface syntax	XML Tree	Remarks
WSML Description	T((<i>varID</i> , O,G,WS,M,C,I) <i>description</i>)	T(wsmIVariant <i>varID</i> <u>definition₁</u> ... <u>definition_n</u>)	< wsmI xmlns="http://www.wsmo.org/wsmI/wsmI-syntax#" variant=" <i>varID</i> "> T(<u>definition₁</u>) ... T(<u>definition_n</u>) </ wsmI >	<u>definition₁</u> ... <u>definition_n</u> are Ontology, Goal, WebService, Mediators, Capability and Interface; O,G,WS,M,C,I are the set of previous mentioned entities; <i>varID</i> is the WSML variant
WSML Ontology	T((<i>Name</i> , ann,ontID,medID,concept,relation,instance,rellInstance,axiom) <i>ontology</i>)	T ₂ (ontology <i>Name</i> <u>header₁</u> ... <u>header_n</u> <u>ontology_element₁</u> ... <u>ontology_element_n</u>)	< ontology name=" <i>Name</i> "> T(<u>header₁</u>) ... T(<u>header_n</u>) T(<u>ontology_element₁</u>) ... T(<u>ontology_element_n</u>) </ ontology >	An ontology_element represents all possible content of an ontology definition, i.e., concepts, relations, instances, ...; <u>header₁</u> , ..., <u>header_n</u> are header elements that contain elements from annotations, ontology identifiers (imported ontologies) and ooMediators identifiers sets denoted by: ann,ontID,medID ; <u>ontology_element₁</u> , ..., <u>ontology_element_n</u> are ontology elements from the sets: concept, relation, instance, rellInstance, axiom
WSML Concept	T((<i>Name</i> , ann,conceptID,attribute) <i>concept</i>)	T(concept <i>Name</i> subConceptOf { <i>B₁</i> , ..., <i>B_n</i> } <u>annotations</u> <u>attribute₁</u> ... <u>attribute_n</u>)	< concept name=" <i>Name</i> "> T(<u>annotations</u>) < superConcept > <i>B₁</i> < superConcept > ... < superConcept > <i>B_n</i> < superConcept > T(<u>attribute₁</u>) ... T(<u>attribute_n</u>) </ concept >	ann is a set of WSML annotations represented using surface syntax as <u>annotations</u> ; conceptID is the set of super-concepts identifiers represented using surface syntax as { <i>B₁</i> , ..., <i>B_n</i> } and attribute is the set of attributes represented using surface syntax as <u>attribute₁</u> , ..., <u>attribute_n</u>
WSML Attribute Definition	T((<i>Name</i> , ann,defType,feature,minCard,maxCard,rangeID) <i>ad</i>)	T(<i>Name</i> <u>attributefeature₁</u> ... <u>attributefeature_n</u> ofType <u>cardinality C</u> <u>annotations</u>)	< attribute name=" <i>Name</i> " type="constraining"> < range > <i>C</i> </ range > T(<u>attributefeature₁</u>) ... T(<u>attributefeature_n</u>) T(<u>cardinality</u>) T(<u>annotations</u>) </ attribute >	ann is a set of WSML annotations represented using surface syntax as <u>annotations</u> ; defType is ofType, feature is the set of attribute features represented using surface syntax as <u>attributefeature₁</u> ... <u>attributefeature_n</u> , minCard and maxCard correspond to cardinality and rangeID is a set of concept identifiers
WSML Attribute Definition	T((<i>Name</i> , ann,defType,feature,minCard,maxCard,rangeID) <i>ad</i>)	T(<i>Name</i> <u>attributefeature₁</u> ... <u>attributefeature_n</u> impliesType <u>cardinality C</u> <u>annotations</u>)	< attribute name=" <i>Name</i> " type="inferring"> < range > <i>C</i> </ range > T(<u>attributefeature₁</u>) ... T(<u>attributefeature_n</u>) T(<u>cardinality</u>) T(<u>annotations</u>) </ attribute >	ann is a set of WSML annotations represented using surface syntax as <u>annotations</u> ; defType is impliesType, feature is the set of attribute features represented using surface syntax as <u>attributefeature₁</u> , ..., <u>attributefeature_n</u> , minCard and maxCard correspond to cardinality and rangeID is a set of concept identifiers
	T(transitive)	T(transitive)	< transitive />	
	T(symmetric)	T(symmetric)	< symmetric />	

	WSML abstract syntax	WSML surface syntax	XML Tree	Remarks
))		
	T(reflexive)	T(reflexive)	<reflexive/>	
	T(inverseOf)	T(inverseOf (A))	<inverseOf type="A"/>	
	T(subAttributeOf)	T(subAttributeOf (A))	<subAttributeOf type="A"/>	
	T((minCard,maxCard))	T((<i>minCard maxCard</i>))	<minCardinality> <i>minCard</i> </minCardinality> <maxCardinality> <i>maxCard</i> </maxCardinality>	
	T((minCard,maxCard))	T((<i>card</i>))	<minCardinality> <i>card</i> </minCardinality> <maxCardinality> <i>card</i> </maxCardinality>	
WSML Instance	T((<i>Name,ann,conceptID,attVal</i>) <i>instance</i>)	T(instance <i>Name</i> memberOf <i>C₁,...,C_n</i> <u>annotations</u> <u>attributevalue₁</u> ... <u>attributevalue_n</u>)	<instance name=" <i>Name</i> "> <memberOf> <i>C₁</i> </memberOf> ... <memberOf> <i>C_n</i> </memberOf> T(<u>annotations</u>) T(<u>attributevalue₁</u>) ... T(<u>attributevalue_n</u>) </instance>	ann is a set of WSML annotations represented using surface syntax as <u>annotations</u> , conceptID is a set of concept identifiers represented using the surface syntax as <i>C₁,...,C_n</i> , attVal is a set of attribute value pairs represented using the surface syntax as <u>attributevalue₁,...,attributevalue_n</u>
WSML Attribute Value Pair	T((<i>Name,valueID</i>) <i>av</i>)	T(<i>Name hasValue</i> { <u>value₁</u> , ..., <u>value_n</u> })	<attributeValue name=" <i>Name</i> "> T(<u>value₁</u>) ... T(<u>value_n</u>) </attributeValue>	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 <u>value₁,...,value_n</u> . Please note that attribute values pairs used in a nonfunctional property definition might include also variables.
WSML IRI	T(<i>IRI</i>)	T(<i>IRI</i>)	<value type="&wsm;iri"> <i>IRI</i> </value>	
String	T(<i>DV_s</i>)	T(<i>DV_s</i>)	<value type="&xsd:string"> <i>DV_s</i> </value>	
Integer	T(<i>DV_i</i>)	T(<i>DV_i</i>)	<value type="&xsd:integer"> <i>DV_i</i> </value>	

	WSML abstract syntax	WSML surface syntax	XML Tree	Remarks
Decimal	T(<i>DV_d</i>)	T(<i>DV_d</i>)	<value type="xsd:decimal"> <i>DV_d</i> </value>	
WSML Data Value	T(<i>DV</i>)	T(<i>DV</i>)	<value type="http://www.example.org/datatype#any"> <argument> <i>DV.arg₁</i> </argument> ... <argument> <i>DV.arg_n</i> </argument> </value>	A datatype wrapper is defined by the type and a number of arguments that define the value, e.g., xsd#date(2005,12,12) for December 12, 2005 (and thus the value of type date has three arguments). In case there is just one argument, the element <argument> is optional and the value can be given as above for IRI, string, integer, decimal.
WSML Variable	T(<i>V</i>)	T(<i>V</i>)	<value> <variable name="V"/> </value>	
WSML Relation	T((<i>Name</i> , ann , <i>relationID</i> , <u>parameter</u>) <i>relation</i>)	T(relation <i>Name</i> / <i>n</i> (<i>paramtype₁</i> ,..., <i>paramtype_n</i>) subRelationOf { <i>C₁</i> ,..., <i>C_k</i> } <u>annotations</u>)	<relation name="Name" arity="n"> <parameters> T(<i>paramtype₁</i>) ... T(<i>paramtype_n</i>) </parameters> <superRelation> <i>C₁</i> </superRelation> ... <superRelation> <i>C_k</i> </superRelation> T(<u>annotations</u>) </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 <u>annotations</u> , relationID is a set of (super-)relation identifiers represented using the surface syntax as <i>C₁</i> ,..., <i>C_n</i> , <u>parameter</u> is a vector of WSML parameter definitions represented using the surface syntax as <i>paramtype₁</i> ... <i>paramtype_n</i>
WSML Parameter Definition	T((ann , <i>defType</i> , <i>domainID</i>) <i>pd</i>)	T(offType { <i>C₁</i> ,..., <i>C_n</i> } <u>annotations</u>)	<parameter type="constraining"> <range> <i>C₁</i> </range> ... <range> <i>C_n</i> </range> T(<u>annotations</u>) </parameter>	ann is a set of WSML annotations represented using surface syntax as <u>annotations</u> , <i>defType</i> is <i>offType</i> , <i>domainID</i> is a set of concept identifiers represented using the surface syntax as <i>C₁</i> ,..., <i>C_n</i>
WSML Parameter Definition	T((ann , <i>defType</i> , <i>domainID</i>) <i>pd</i>)	T(impliesType { <i>C₁</i> ,..., <i>C_n</i> } <u>annotations</u>)	<parameter type="inferring"> <range> <i>C₁</i> </range> ... <range> <i>C_n</i> </range> T(<u>annotations</u>) </parameter>	ann is a set of WSML annotations represented using surface syntax as <u>annotations</u> , <i>defType</i> is <i>impliesType</i> , <i>domainID</i> is a set of concept identifiers represented using the surface syntax as <i>C₁</i> ,..., <i>C_n</i>
WSML Relation Instance	T((<i>Name</i> , ann , <i>relationID</i> , <u>parVal</u>) <i>ri</i>)	T(relationInstance <i>Name B</i> (<i>value₁</i> ,..., <i>value_n</i>) <u>annotations</u>)	<relationInstance name="Name"> <memberOf> <i>B</i> </memberOf> T(<i>value₁</i>) ... T(<i>value_n</i>) T(<u>annotations</u>) </relationInstance>	ann is a set of WSML annotations represented using surface syntax as <u>annotations</u> , <i>relationID</i> is a set of <i>impliesType</i> , <i>domainID</i> is a set of concept identifiers represented using the surface syntax as <i>C₁</i> ,..., <i>C_n</i>
WSML Parameter Values	T((ann , <i>valueID</i>) <i>p_v</i>)	T({ <i>value₁</i> ,..., <i>value_n</i> })	<parameterValue> T(<i>value₁</i>) ...	Note that the parameters of a relationInstance are ordered and thus the order of the value elements is important.

	WSML abstract syntax	WSML surface syntax	XML Tree	Remarks
			T(<u>value_n</u>) </ parameterValue >	ann is a set of WSML annotations represented using surface syntax as <u>annotations</u> , defType is <i>impliesType</i> , domainID is a set of concept identifiers represented using the surface syntax as <u>value₁,...,value_n</u>
WSML Axiom	T((<u>Name</u> , ann , logExp) <u>axiom</u>)	T(axiom <u>Name</u> <u>axiomdefinition</u>)	< axiom name=" <u>Name</u> "> T(<u>axiomdefinition</u>) T(<u>annotations</u>) </ axiom >	ann is a set of WSML annotations represented using surface syntax as <u>annotations</u> , defType is <i>impliesType</i> , domainID is a set of concept identifiers represented using the surface syntax as <u>value₁,...,value_n</u>
WSML Logical Expression	T((logExp))	T(<u>annotations</u> definedBy <u>log_expr</u>)	T(<u>annotations</u>) < definedBy > T(<u>log_expr</u>) </ definedBy >	The mapping of logical expressions is defined in Table 2.2.
WSML Goal	T((<u>Name</u> , ann , ontID , medID , nfp , ontID , capability , interface) <u>goal</u>)	T(goal <u>A</u> <u>header₁</u> ... <u>header_n</u> <u>nfp₁</u> ... <u>nfp_n</u> <u>capability</u> <u>interface₁</u> ... <u>interface_n</u>)	< goal name=" <u>A</u> "> T(<u>header₁</u>) ... T(<u>header_n</u>) T(<u>nfp₁</u>) ... T(<u>nfp_n</u>) T(<u>capability</u>) T(<u>interface₁</u>) ... T(<u>interface_n</u>) </ goal >	<u>header₁,...,header_n</u> are header elements that contain elements from annotations, ontology identifiers (imported ontologies) and ooMediators identifiers sets denoted by: ann , ontID , medID ; nfp is a set of nonfunctional properties represented using the surface syntax as <u>nfp₁,...,nfp_n</u> , capability is a WSML capability represented using the surface syntax as <u>capability</u> , interface is a set of WSML interfaces represented using the surface syntax as <u>interface₁,...,interface_n</u>
WSML ooMediator	T((<u>Name</u> , ann , ontID , nfp , sourceID , targetID , serviceID) <u>goal</u>)	T(ooMediator <u>A</u> <u>annotations</u> <u>importsontology</u> <u>nfp₁</u> ... <u>nfp_n</u> <u>source</u> <u>target</u> <u>use_service</u>)	< ooMediator name=" <u>A</u> "> T(<u>annotations</u>) T(<u>importsontology</u>) T(<u>nfp₁</u>) ... T(<u>nfp_n</u>) T(<u>source</u>) T(<u>target</u>) T(<u>use_service</u>) </ ooMediator >	<u>annotations</u> and <u>importsontology</u> correspond to sets ann respectively ontID ; <u>ontology_element₁,...,ontology_element_n</u> are ontology elements from the sets: concept , relation , instance , reInstance , axiom
WSML ggMediator	T((<u>Name</u> , ann , ontID , medID , nfp , sourceID , targetID , serviceID) <u>ggm</u>)	T(ggMediator <u>A</u> <u>header₁</u> ... <u>header_n</u> <u>nfp₁</u> ... <u>nfp_n</u> <u>source</u> <u>target</u>)	< ggMediator name=" <u>A</u> "> T(<u>header₁</u>) ... T(<u>header_n</u>) T(<u>nfp₁</u>) ... T(<u>nfp_n</u>) T(<u>source</u>) T(<u>target</u>) T(<u>use_service</u>)	<u>header₁,...,header_n</u> are header elements that contain elements from annotations, ontology identifiers (imported ontologies) and ooMediators identifiers sets denoted by: ann , ontID , medID ; <u>nfp₁,...,nfp_n</u> are nonfunctional properties from the set nfp , <u>source</u> and <u>target</u> are goal and ggMediator identifiers from the sets sourceID and targetID , <u>use_service</u> are goal, Web service and wwMediator

	WSML abstract syntax	WSML surface syntax	XML Tree	Remarks
		<u>use_service</u>)	</ggMediator>	identifiers from the set serviceID
WSML wgMediator	T((Name,ann,ontID,medID,nfp,sourceID,targetID,serviceID) _{wgm})	T(wgMediator A <u>header₁</u> ... <u>header_n</u> <u>nfp₁</u> ... <u>nfp_n</u> <u>source</u> <u>target</u> <u>use_service</u>)	< wgMediator name="A"> T(<u>header₁</u>) ... T(<u>header_n</u>) T(<u>nfp₁</u>) ... T(<u>nfp_n</u>) T(<u>source</u>) T(<u>target</u>) T(<u>use_service</u>) </ wgMediator >	
WSML wwMediator	T((Name,ann,ontID,medID,nfp,sourceID,targetID,serviceID) _{wwm})	T(wwMediator A <u>header₁</u> ... <u>header_n</u> <u>nfp₁</u> ... <u>nfp_n</u> <u>source</u> <u>target</u> <u>use_service</u>)	< wwMediator name="A"> T(<u>header₁</u>) ... T(<u>header_n</u>) T(<u>nfp₁</u>) ... T(<u>nfp_n</u>) T(<u>source</u>) T(<u>target</u>) T(<u>use_service</u>) </ wwMediator >	
	T(sourceID)	T(source { A ₁ ,...,A _n })	< source >A ₁ </ source > ... < source >A _n </ source >	
	T(targetID)	T(target A)	< target >A</ target >	
	T(serviceID)	T(usesService A)	< usesService >A</ usesService >	
WSML Web Service	T((Name,ann,ontID,medID,nfp,capability,interface) _{ws})	T(webService A <u>header₁</u> ... <u>header_n</u> <u>nfp₁</u> ... <u>nfp_n</u> <u>capability</u> <u>interface₁</u> ... <u>interface_n</u>)	< webService name="A"> T(<u>header₁</u>) ... T(<u>header_n</u>) T(<u>nfp₁</u>) ... T(<u>nfp_n</u>) T(<u>capability</u>) T(<u>interface₁</u>) ... T(<u>interface_n</u>) </ webService >	
WSML	T(T(< capability name="C">	pre_post_ass_or_eff unites the axiom

	WSML abstract syntax	WSML surface syntax	XML Tree	Remarks
Capability	$(Name, \text{ann}, \text{ontID}, \text{medID}, \text{nfp}, \text{sharedVar}, \text{pre}, \text{post}, \text{ass}, \text{eff})_{\text{capability}}$	capability C <u>header</u> ₁ ... <u>header</u> _{n} <u>nfp</u> ₁ ... <u>nfp</u> _{n} <u>sharedvardef</u> <u>pre_post_ass_or_eff</u> ₁ ... <u>pre_post_ass_or_eff</u> _{n})	$T(\text{header}_1)$... $T(\text{header}_n)$ $T(\text{nfp}_1)$... $T(\text{nfp}_n)$ $T(\text{sharedvardef})$ $T(\text{pre_post_ass_or_eff}_1)$... $T(\text{pre_post_ass_or_eff}_n)$ </ capability >	definitions for precondition, assumption, postcondition and effect.
	$T(\text{sharedVar})$	$T(\text{sharedVariables } \{?V_1, \dots, ?V_n\})$	< sharedVariables > < variable name="?" V_1 "/> ... < variable name="?" V_n "/> </ sharedVariables >	
	$T(\text{pre})$	$T(\text{precondition } B \text{ axiomdefinition})$	< precondition name="B"> $T(\text{axiomdefinition})$ </ precondition >	
	$T(\text{ass})$	$T(\text{assumption } B \text{ axiomdefinition})$	< assumption name="B"> $T(\text{axiomdefinition})$ </ assumption >	
	$T(\text{post})$	$T(\text{postcondition } B \text{ axiomdefinition})$	< postcondition name="B"> $T(\text{axiomdefinition})$ </ postcondition >	
	$T(\text{eff})$	$T(\text{effect } B \text{ axiomdefinition})$	< effect name="B"> $T(\text{axiomdefinition})$ </ effect >	
WSML Interface	$T((Name, \text{ann}, \text{ontID}, \text{medID}, \text{nfp}, \text{chor}, \text{orchID})_{\text{interface}})$	interface A <u>header</u> ₁ ... <u>header</u> _{n} <u>nfp</u> ₁ ... <u>nfp</u> _{n} <u>choreography</u> <u>orchestration</u>)	< interface name="A"> $T(\text{header}_1)$... $T(\text{header}_n)$ $T(\text{nfp}_1)$... $T(\text{nfp}_n)$ $T(\text{choreography})$ $T(\text{orchestration})$ </ interface >	
WSML Choreography	$T((Name, \text{ann}, \text{ontID}, \text{medID}, \text{nfp}, \text{signature}, \text{rule})_{\text{choreography}})$	choreography C <u>header</u> ₁ ... <u>header</u> _{n})	< choreography name="C"> $T(\text{header}_1)$... $T(\text{header}_n)$ $T(\text{statesignature})$	rule is a set of WSML rules represented using surface syntax as <u>transitions</u> , where a WSML rule can be an <i>if rule</i> , <i>forall rule</i> , <i>choose rule</i> , <i>piped rule</i> , <i>add rule</i> , <i>delete rule</i>

	WSML abstract syntax	WSML surface syntax	XML Tree	Remarks
		<u>statesignature</u> <u>transitions</u>)	T(transitions) </choreography>	
WSML State Signature	T((Name,ann,ontID,medID,nfp,mode) _{ss})	T(statesignature C <u>header</u> ₁ ... <u>header</u> _n <u>mode</u> ₁ ... <u>mode</u> _n)	< statesignature name="C"> T(<u>header</u> ₁) ... T(<u>header</u> _n) T(<u>mode</u> ₁) ... T(<u>mode</u> _n) </statesignature>	
WSML Mode	T((type,conceptID,groundingID) _{cm})	T(type concept A <u>grounding</u>)	< mode > T(type) < concept /> <iri>A</iri> T(<u>grounding</u>) </mode>	
WSML Mode	T((type,conceptID,groundingID) _{rm})	T(type relation A <u>grounding</u>)	< mode > T(type) < relation /> <iri>A</iri> T(<u>grounding</u>) </mode>	
	T(type)	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 {A ₁ ,...,A _n })	< grounding >A ₁ </grounding> ... < grounding >A _n </grounding>	
WSML Transition Rule	T((logExp, rule) _{if})	T(<u>if</u> <u>condition</u> <u>then</u> <u>rule</u> ₁)	< rule > < if /> T(<u>condition</u>) < then /> T(<u>rule</u> ₁)	rule is a set of WSML transition rules expressed using the surface syntax as: <u>rule</u> ₁ ,..., <u>rule</u> _n and logExp is a WSML logical expression expressed using the surface syntax as <u>condition</u>

	WSML abstract syntax	WSML surface syntax	XML Tree	Remarks
		<pre> ... rule_n endif) </pre>	<pre> ... T(rule_n) <endif/> </rule> </pre>	
WSML Transition Rule	<pre> T ((varID,logExp,rule)_{forall}) </pre>	<pre> T (forall variable₁ ... variable_n with condition do rule₁ ... rule_n endforall) </pre>	<pre> <rule> <forall/> T(variable₁) ... T(variable_n) <with/> T(condition) <do/> T(rule₁) ... T(rule_n) <endforall/> </rule> </pre>	<p>varID is a set of variable identifiers expressed using the surface syntax as: <u>variable₁</u>,...,<u>variable_n</u>, rule is a set of WSML transition rules expressed using the surface syntax as: <u>rule₁</u>,...,<u>rule_n</u> and logExp is a WSML logical expression expressed using the surface syntax as: <u>condition</u></p>
WSML Transition Rule	<pre> T ((varID,logExp,rule)_{choose}) </pre>	<pre> T (choose variable₁ ... variable_n with condition do rule₁ ... rule_n endchoose) </pre>	<pre> <rule> <choose/> T(variable₁) ... T(variable_n) <with/> T(condition) <do/> T(rule₁) ... T(rule_n) <endchoose/> </rule> </pre>	<p>varID is a set of variable identifiers expressed using the surface syntax as: <u>variable₁</u>,...,<u>variable_n</u>, rule is a set of WSML transition rules expressed using the surface syntax as: <u>rule₁</u>,...,<u>rule_n</u> and logExp is a WSML logical expression expressed using the surface syntax as <u>condition</u></p>
WSML Transition Rule	<pre> T ((rule)_{piped}) </pre>	<pre> T (rule₁ ... rule_n) </pre>	<pre> <rule> T(rule₁) <pipe/> ... <pipe/> T(rule_n) </rule> </pre>	
WSML Transition Rule	<pre> T ((fact)_{add}) </pre>	<pre> T (add fact) </pre>	<pre> <rule> <add/> T(fact) </rule> </pre>	fact is a ground atomic fomula
WSML Transition Rule	<pre> T ((fact)_{delete}) </pre>	<pre> T (delete fact) </pre>	<pre> <rule> <delete/> T(fact) </rule> </pre>	fact is a ground atomic fomula
WSML Orchestration	<pre> T (orchestration) </pre>	<pre> T (orchestration A) </pre>	<pre> <orchestration>A</orchestration> </pre>	
WSML Nonfunctional Property	<pre> T ((name, ann, value, logExp)_{nfp}) </pre>	<pre> T (nonFunctionalProperty annotations name hasValue {value₁, ..., value_n}) </pre>	<pre> <nonFunctionalProperty name="name"> T(annotations) T(value₁) ... T(value_n) </pre>	

	WSML abstract syntax	WSML surface syntax	XML Tree	Remarks
		<u>log_definition</u>)	T(log_definition) </nonFunctionalProperty>	
WSML Logical Definition	T(logDefinition)	T(definedBy <u>log_expr₁</u> , ..., <u>log_expr_n</u>)	<logicalDefinition> <definedBy> T(log_expr ₁) ... T(log_expr _n) </definedBy> </logicalDefinition>	
WSML Annotations	T((name, value) _{ann})	T(annotations <u>attributevalue₁</u> ... <u>attributevalue_n</u> endAnnotations)	<annotations> T(attributevalue ₁) ... T(attributevalue _n) </annotations>	
	T(ontID)	T(importsOntology { <u>A₁</u> , ..., <u>A_n</u>)	<importsOntology>A ₁ </importsOntology> ... <importsOntology>A _n </importsOntology>	
	T(medID)	T(usesMediator { <u>B₁</u> , ..., <u>B_n</u>)	<usesMediator>B ₁ </usesMediator> ... <usesMediator>B _n </usesMediator>	

The logical expression syntax is explained in Table 2.2. In the table, A stands for identifiers and T_1, \dots, T_n stand for terms. V stands for a variable.

	WSML Abstract syntax	WSML Surface syntax	XML Tree	Remarks
WSML Predicate	T(A(term ₁ , ..., term _n))	T(A(<u>term₁</u> , ..., <u>term_n</u>))	<term name="A"> T(<u>term₁</u> , arg) ... T(<u>term_n</u> , arg) </term>	
WSML Term	T(V)	T(V)	<term name="V"/>	
WSML Atom	T(A(term ₁ , ..., term _n))	T(A(<u>term₁</u> , ..., <u>term_n</u>))	<atom name="A"> T(<u>term₁</u> , arg) ... T(<u>term_n</u> , arg) </atom>	
true	T(T)	T(true)	<true/>	
false	T(⊥)	T(false)	<false/>	

Table 2.2: Mapping the WSML logical expression syntax to the WSML/XML syntax

	WSML Abstract syntax	WSML Surface syntax	XML Tree	Remarks
equal	$\mathbb{T}(\ T_1 = T_2 \)$	$\mathbb{T}(\ T_1 \ \mathbf{equal} \ T_2 \)$	<code><equal></code> $\mathbb{T}(T_1)$ $\mathbb{T}(T_2)$ <code></equal></code>	
subConceptOf	$\mathbb{T}(\ T_1 :: T_2, \dots, T_n \)$	$\mathbb{T}(\ T_1 \ \mathbf{subConceptOf} \ T_2, \dots, T_n \)$	<code><molecule></code> $\mathbb{T}(T_1)$ <code><isa type="subConceptOf"></code> $\mathbb{T}(T_2)$... $\mathbb{T}(T_n)$ <code></isa></code> <code></molecule></code>	
memberOf	$\mathbb{T}(\ T_1 : T_2, \dots, T_n \)$	$\mathbb{T}(\ T_1 \ \mathbf{memberOf} \ T_2, \dots, T_n \)$	<code><molecule></code> $\mathbb{T}(T_1)$ <code><isa type="memberOf"></code> $\mathbb{T}(T_2)$... $\mathbb{T}(T_n)$ <code></isa></code> <code></molecule></code>	
	$\mathbb{T}(\ T \ [\ \mathit{attr_relation}_1, \dots, \mathit{attr_relation}_n \] \)$	$\mathbb{T}(\ T \ [\ \underline{\mathit{attr_relation}}_1, \dots, \underline{\mathit{attr_relation}}_n \] \)$	<code><molecule></code> $\mathbb{T}(T)$ $\mathbb{T}(\underline{\mathit{attr_relation}}_1)$... $\mathbb{T}(\underline{\mathit{attr_relation}}_n)$ <code></molecule></code>	
ofType	$\mathbb{T}(\ T_0 \ \mathbf{of} \ \{T_1, \dots, T_n\} \)$	$\mathbb{T}(\ T_0 \ \mathbf{ofType} \ \{T_1, \dots, T_n\} \)$	<code><attributeDefinition type="constraining"></code> $\mathbb{T}(T_0, \text{name})$ $\mathbb{T}(T_1, \text{type})$... $\mathbb{T}(T_n, \text{type})$ <code></attributeDefinition></code>	
impliesType	$\mathbb{T}(\ T_0 \ \mathbf{it} \ \{T_1, \dots, T_n\} \)$	$\mathbb{T}(\ T_0 \ \mathbf{impliesType} \ \{T_1, \dots, T_n\} \)$	<code><attributeDefinition type="inferring"></code> $\mathbb{T}(T_0, \text{name})$ $\mathbb{T}(T_1, \text{type})$... $\mathbb{T}(T_n, \text{type})$ <code></attributeDefinition></code>	
hasValue	$\mathbb{T}(\ T_0 \ \mathbf{hv} \ \{T_1, \dots, T_n\} \)$	$\mathbb{T}(\ T_0 \ \mathbf{hasValue} \ \{T_1, \dots, T_n\} \)$	<code><attributeValue></code> $\mathbb{T}(T_0, \text{name})$ $\mathbb{T}(T_1, \text{value})$... $\mathbb{T}(T_n, \text{value})$ <code></attributeValue></code>	
and	$\mathbb{T}(\ \text{expr}_1 \ \wedge \ \text{expr}_2 \)$	$\mathbb{T}(\ \underline{\text{expr}}_1 \ \mathbf{and} \ \underline{\text{expr}}_2 \)$	<code><and></code> $\mathbb{T}(\underline{\text{expr}}_1)$ $\mathbb{T}(\underline{\text{expr}}_2)$ <code></and></code>	

	WSML Abstract syntax	WSML Surface syntax	XML Tree	Remarks
or	$T(\text{expr}_1 \vee \text{expr}_2)$	$T(\underline{\text{expr}_1} \text{ or } \underline{\text{expr}_2})$	<code><or></code> <code>T(<u>expr₁</u>)</code> <code>T(<u>expr₂</u>)</code> <code></or></code>	
neg	$T(\neg \text{expr})$	$T(\text{neg } \underline{\text{expr}})$	<code><neg></code> <code>T(<u>expr</u>)</code> <code></neg></code>	
naf	$T(\text{not } \text{expr})$	$T(\text{naf } \underline{\text{expr}})$	<code><naf></code> <code>T(<u>expr</u>)</code> <code></naf></code>	
implies	$T(\text{expr}_1 \supset \text{expr}_2)$	$T(\underline{\text{expr}_1} \text{ implies } \underline{\text{expr}_2})$	<code><implies></code> <code>T(<u>expr₁</u>)</code> <code>T(<u>expr₂</u>)</code> <code></implies></code>	
impliedBy	$T(\text{expr}_1 \subset \text{expr}_2)$	$T(\underline{\text{expr}_1} \text{ impliedBy } \underline{\text{expr}_2})$	<code><impliedBy></code> <code>T(<u>expr₁</u>)</code> <code>T(<u>expr₂</u>)</code> <code></impliedBy></code>	
equivalent	$T(\text{expr}_1 \equiv \text{expr}_2)$	$T(\underline{\text{expr}_1} \text{ equivalent } \underline{\text{expr}_2})$	<code><equivalent></code> <code>T(<u>expr₁</u>)</code> <code>T(<u>expr₂</u>)</code> <code></equivalent></code>	
forall	$T(\forall ?v_1, \dots, ?v_n (\text{expr}))$	$T(\text{forall } ?v_1, \dots, ?v_n (\underline{\text{expr}}))$	<code><forall></code> <code><variable name="?"v₁"/></code> ... <code><variable name="?"v_n"/></code> <code>T(<u>expr</u>)</code> <code></forall></code>	
exists	$T(\exists ?v_1, \dots, ?v_n (\text{expr}))$	$T(\text{exists } ?v_1, \dots, ?v_n (\underline{\text{expr}}))$	<code><exists></code> <code><variable name="?"v₁"/></code> ... <code><variable name="?"v_n"/></code> <code>T(<u>expr</u>)</code> <code></exists></code>	
constraint	$T(\text{expr} \supset \perp)$	$T(\text{!- } \underline{\text{expr}})$	<code><constraint></code> <code>T(<u>expr</u>)</code> <code></constraint></code>	
impliedByLP	$T(\text{expr}_2 \supset \text{expr}_1)$	$T(\underline{\text{expr}_1} \text{ :- } \underline{\text{expr}_2})$	<code><impliedByLP></code> <code>T(<u>expr₁</u>)</code> <code>T(<u>expr₂</u>)</code> <code></impliedByLP></code>	

Table 2.3 provides a simple translation example.

WSML syntax

wsmIVariant

```
_"http://www.wsmo.org/wsmI/wsmI-syntax/wsmI-flight"
```

```
namespace {_"http://www.example.org/ex1#",
  dc_"http://purl.org/dc/elements/1.1#",
  wsmI_"http://www.wsmo.org/wsmI/wsmI-syntax#",
  ex_"http://www.example.org/ex2#"}
```

```
ontology _"http://www.example.org/ex1"
```

annotations

```
  dc#title hasValue "WSML to RDF"
  dc#date hasValue _date(2005, 12, 12)
```

endAnnotations

```
importsOntology _"http://www.example.net/ex2"
```

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"}
```

XML Tree

```
<wsmI xmlns="http://www.wsmo.org/wsmI/wsmI-syntax#"
  variant="http://www.wsmo.org/wsmI/wsmI-syntax/wsmI-flight">
  <ontology name="http://www.example.org/ex1">
    <annotations>
      <attributeValue name="http://purl.org/dc/elements/1.1#title">
        <value type="http://www.wsmo.org/wsmI/wsmI-syntax#string">
          WSML to RDF
        </value>
      </attributeValue>
      <attributeValue name="http://purl.org/dc/elements/1.1#date">
        <value type="http://www.wsmo.org/wsmI/wsmI-syntax#date">
          <argument>2005</argument>
          <argument>12</argument>
          <argument>12</argument>
        </value>
      </attributeValue>
    </annotations>
    <importsOntology>http://www.example.org/ex2</importsOntology>
    <concept name="http://www.example.org/ex1#Woman">
      <superConcept>
        http://www.example.org/ex2#Human
      </superConcept>
      <superConcept>
        http://www.example.org/ex2#LivingBeing
      </superConcept>
      <attribute name="http://www.example.org/ex1#name" type="constraining">
        <range>http://www.wsmo.org/wsmI/wsmI-syntax#string</range>
      </attribute>
      <attribute name="http://www.example.org/ex1#ancestor" type="inferring">
        <range>http://www.example.org/ex2#Human</range>
        <transitive/>
      </attribute>
      <attribute name="http://www.example.org/ex1#age" type="inferring">
        <range>http://www.wsmo.org/wsmI/wsmI-syntax#integer</range>
        <minCardinality>1</minCardinality>
        <maxCardinality>1</maxCardinality>
      </attribute>
    </concept>
    <axiom name="http://www.example.org/ex1#GenderConstraint">
      <definedBy>
        <constraint>
          <and>
            <molecule>
              <term name="?x"/>
              <isa type="memberOf">
                <term name="http://www.example.org/ex1#Woman"/>
              </isa>
            </molecule>
            <molecule>
              <term name="?x"/>
              <isa type="memberOf">
                <term name="http://www.example.org/ex2#Man"/>
              </isa>
            </molecule>
          </and>
        </constraint>
      </definedBy>
    </axiom>
  </ontology>
</wsmI>
```

WSML syntax**XML Tree**

```

    </and>
  </constraint>
</definedBy>
</axiom>
<instance name="http://www.example.org/ex1#Mary">
  <memberOf>http://www.example.org/ex1#Woman</memberOf>
  <attributeValue name="http://www.example.org/ex1#name">
    <value type="http://www.wsmo.org/wsm1/wsm1-syntax#string">
      Mary Jones
    </value>
  </attributeValue>
  <attributeValue name="http://www.example.org/ex1#age">
    <value type="http://www.wsmo.org/wsm1/wsm1-syntax#integer">
      23
    </value>
  </attributeValue>
</instance>
<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>
</ontology>
<webService name="http://www.example.org/ex1#ws">
  <capability name="http://www.example.org/ex1#itineraryInfo">
    <interface name="http://example.org/i1</interface">
    <interface name="http://example.org/i2</interface">
  </webService>
</wsm1>

```

References

[RIF-BLD] H. Boley & M. Kifer, (Eds.). *RIF Basic Logic Dialect*, W3C Working Draft. Available from <http://www.w3.org/TR/rif-blld/>.

[RIF-FLD] H. Boley & M. Kifer, (Eds.). *RIF Framework for Logic Dialects*, W3C Working Draft. Available from <http://www.w3.org/TR/rif-flld/>.

[WSML] N. Steinmetz and I. Toma , editors. *The Web Service Modeling Language WSML*. 2008. WSML Working Draft D16.1v1.0 <http://www.wsmo.org/TR/d16/d16.1/v1.0/>.

[WSML-Semantics] J. de Bruijn, editor. *WSML Abstract Syntax and Semantics*. 2008. WSML Working Draft D16.3v1.0 <http://www.wsmo.org/TR/d16/d16.3/v1.0/>.

Acknowledgements

The work is partially funded by the European Commission under the projects SOA4ALL and ServiceWeb 3.0.

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