



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 WSMML

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

The complete XML Schema, including documentation, for the WSMML/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:

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

The WSMML identifiers schema is a XML schema defining the WSMML identifiers, such as IRIs, anonymous identifiers, variables, etc. Elements of this schema are used to define elements of WSMML/XML schema document and WSMML Logical expression schema. The WSMML Logical expressions schema elements are used to define the elements of WSMML/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 WSMML:*
http://www.wsmo.org/TR/d36/v1.0/xml-syntax/documentation/wsml_xml_syntax.xsd.html
- *XML syntax for WSMML identifiers:*
http://www.wsmo.org/TR/d36/v1.0/xml-syntax/documentation/wsml_identifiers.xsd.html
- *XML syntax for WSMML 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 WSMML abstract and surface syntax to WSMML 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/wsml/wsml-syntax#>. This is the namespace for all elements in WSMML.

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.
- in the conceptual syntax universal truth, universal falsehood and unnumbered anonymous IDs are resolved to: <http://www.wsmo.org/wsml/wsml-syntax#true>, <http://www.wsmo.org/wsml/wsml-syntax#false>, and <http://www.wsmo.org/wsml/wsml-syntax#anonymousID>, respectively. Numbered anonymous IDs are resolved as such: [_#1](http://www.wsmo.org/wsml/wsml-syntax#anonymousID1) to <http://www.wsmo.org/wsml/wsml-syntax#anonymousID1>, [_#2](http://www.wsmo.org/wsml/wsml-syntax#anonymousID2) to <http://www.wsmo.org/wsml/wsml-syntax#anonymousID2>, ..., [_#n](http://www.wsmo.org/wsml/wsml-syntax#anonymousIDn) to <http://www.wsmo.org/wsml/wsml-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/wsml/wsml-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 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, *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 [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

	WSML abstract syntax	WSML surface syntax	XML Tree	Remarks
WSML Description	T((<i>varID</i> , O,G,WS,M,C,I) _{description})	T (wsmlVariant <i>varID</i> <u>definition₁</u> ... <u>definition_n</u>)	<wsml xmlns="http://www.wsmo.org/wsml/wsml-syntax#" variant=" <i>varID</i> "> T(<u>definition₁</u>) ... T(<u>definition_n</u>) </wsml>	<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,ann,ontID,medID,concept,relation,instance,relInstance,axiom</i>) _{ontology})	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 <u>ontology_element</u> 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 oolMediators 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,relInstance,axiom
WSML Concept	T ((<i>Name,ann,conceptID,attribute</i>) _{concept})	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,ann,defType,feature,minCard,maxCard,rangeID</i>) _{ad})	T (<i>Name</i> <u>attributefeature₁</u> ... <u>attributefeature_n</u> ofType <u>cardinality</u> <i>C</i> <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>)	ann is a set of WSML annotations represented using surface syntax as <u>annotations</u> ; <i>defType</i> is <i>ofType</i> , feature is the set of attribute features represented using surface syntax as <u>attributefeature₁</u> ... <u>attributefeature_n</u> , <i>minCard</i> and <i>maxCard</i> correspond to <i>cardinality</i> and rangeID is a set of

			T(annotations) </attribute>	concept identifiers
WSML Attribute Definition	T ((Name,ann,defType,feature,minCard,maxCard,rangeID) _{ad})	T (Name attributefeature ₁ ... attributefeature _n impliesType cardinality C annotations)	<attribute name="Name" type="inferring"> <range>C</range> T(attributefeature ₁) ... T(attributefeature _n) T(cardinality) T(annotations) </attribute>	ann is a set of WSML annotations represented using surface syntax as annotations ; defType is <i>impliesType</i> , feature is the set of attribute features represented using surface syntax as attributefeature₁,...,attributefeature_n , minCard and maxCard correspond to cardinality and rangeID is a set of concept identifiers
	T (transitive)	T (transitive)	<transitive/>	
	T (symmetric)	T (symmetric)	<symmetric/>	
	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 ((minCard maxCard))	<minCardinality>minCard</minCardinality> <maxCardinality>maxCard</maxCardinality>	
	T ((minCard,maxCard))	T ((card))	<minCardinality>card</minCardinality> <maxCardinality>card</maxCardinality>	
WSML Instance	T ((Name,ann,conceptID,attVal) _{instance})	T (instance Name memberOf C ₁ ,...,C _n annotations attributevalue ₁ ... attributevalue _n)	<instance name="Name"> <memberOf>C ₁ </memberOf> ... <memberOf>C _n </memberOf> T(annotations) T(attributevalue ₁) ... T(attributevalue _n) </instance>	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 C ₁ ,...,C _n , attVal is a set of attribute value pairs represented using the surface syntax as attributevalue₁,...,attributevalue_n

WSML Attribute Value Pair	T ((Name,valueID) _{av})	T (Name hasValue {value ₁ , ..., value _n })	<attributeValue name="Name"> T(value ₁) ... T(value _n) </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 value ₁ ,...,value _n . Please note that attribute values pairs used in a nonfunctional property definition might include also variables.
WSML IRI	T (IRI)	T (IRI)	<value type="&wsmli;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.arg ₁ </argument> ... <argument>DV.arg _n </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 (V)	T (V)	<value> <variable name="V"/> </value>	
WSML Relation	T ((Name,ann,relationID,parameter) _{relation})	T (relation Name / n (paramtype ₁ ,...,paramtype _n) subRelationOf {C ₁ ,...,C _k })	<relation name="Name" arity="n"> <parameters> T(paramtype ₁) ... T(paramtype _n) </parameters> <superRelation>C ₁ </superRelation>	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

		<pre> annotations) </pre>	<pre> ... <superRelation>C_k</superRelation> T(annotations) </relation> </pre>	<p>C_1, \dots, C_n, <u>parameter</u> is a vector of WSML parameter definitions represented using the surface syntax as <u>paramtype₁ ... paramtype_n</u></p>
WSML Parameter Definition	<pre> T ((ann, defType, domainID)_{pd}) </pre>	<pre> T (ofType {C₁, ..., C_n} annotations) </pre>	<pre> <parameter type="constraining"> <range>C₁</range> ... <range>C_n</range> T(annotations) </parameter> </pre>	<p>ann is a set of WSML annotations represented using surface syntax as <u>annotations</u>, defType is <i>ofType</i>, domainID is a set of concept identifiers represented using the surface syntax as C_1, \dots, C_n</p>
WSML Parameter Definition	<pre> T ((ann, defType, domainID)_{pd}) </pre>	<pre> T (impliesType {C₁, ..., C_n} annotations) </pre>	<pre> <parameter type="inferring"> <range>C₁</range> ... <range>C_n</range> T(annotations) </parameter> </pre>	<p>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 C_1, \dots, C_n</p>
WSML Relation Instance	<pre> T ((Name, ann, relationID, parVal)_{ri}) </pre>	<pre> T (relationInstance Name B (value₁, ..., value_n) annotations) </pre>	<pre> <relationInstance name="Name"> <memberOf>B</memberOf> T(value₁) ... T(value_n) T(annotations) </relationInstance> </pre>	<p>ann is a set of WSML annotations represented using surface syntax as <u>annotations</u>, relationID is a set of <i>impliesType</i>, domainID is a set of concept identifiers represented using the surface syntax as C_1, \dots, C_n</p>
WSML Parameter Values	<pre> T ((ann, valueID)_{pv}) </pre>	<pre> T ({value₁, ..., value_n}) </pre>	<pre> <parameterValue> T(value₁) ... T(value_n) </parameterValue> </pre>	<p>Note that the parameters of a relationInstance are ordered and thus the order of the value elements is important. 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></p>
WSML Axiom	<pre> T ((Name, ann, logExp)_{axiom}) </pre>	<pre> T (axiom Name axiomdefinition) </pre>	<pre> <axiom name="Name"> T(axiomdefinition) T(annotations) </axiom> </pre>	<p>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></p>
WSML Logical Expression	<pre> T ((logExp)) </pre>	<pre> T (annotations definedBy log_expr) </pre>	<pre> T(annotations) <definedBy> T(log_expr) </definedBy> </pre>	<p>The mapping of logical expressions is defined in Table 2.2.</p>

WSML Goal	<pre>T ((Name,ann,ontID,medID,nfp,ontID,capability,interface) goal)</pre>	<pre>T (goal A header₁ ... header_n nfp₁ ... nfp_n capability interface₁ ... interface_n)</pre>	<pre><goal name="A"> T(header₁) ... T(header_n) T(nfp₁) ... T(nfp_n) T(capability) T(interface₁) ... T(interface_n) </goal></pre>	<p><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; nfp is a set of nonfunctional properties represented using the surface syntax as <u>nfp₁</u>,...,<u>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</u>,...,<u>interface_n</u></p>
WSML ooMediator	<pre>T ((Name,ann,ontID,nfp,sourceID,targetID,serviceID) goal)</pre>	<pre>T (ooMediator A annotations importsontology nfp₁ ... nfp_n source target use_service)</pre>	<pre><ooMediator name="A"> T(annotations) T(importsontology) T(nfp₁) ... T(nfp_n) T(source) T(target) T(use_service) </ooMediator></pre>	<p>annotations and importsontology correspond to sets ann respectively ontID; <u>ontology_element₁</u>,...,<u>ontology_element_n</u> are ontology elements from the sets: concept, relation,instance,relInstance,axiom</p>
WSML ggMediator	<pre>T ((Name,ann,ontID,medID,nfp,sourceID,targetID,serviceID) ggm)</pre>	<pre>T (ggMediator A header₁ ... header_n nfp₁ ... nfp_n source target use_service)</pre>	<pre><ggMediator name="A"> T(header₁) ... T(header_n) T(nfp₁) ... T(nfp_n) T(source) T(target) T(use_service) </ggMediator></pre>	<p><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>nfp₁</u>,...,<u>nfp_n</u> are nonfunctional properties from the set nfp, source and target are goal and ggMediator identifiers from the sets sourceID and targetID, use_service are goal, Web service and wwMediator identifiers from the set serviceID</p>
WSML wgMediator	<pre>T ((Name,ann,ontID,medID,nfp,sourceID,targetID,serviceID) wgm)</pre>	<pre>T (wgMediator A header₁ ... header_n nfp₁ ... nfp_n source target use_service)</pre>	<pre><wgMediator name="A"> T(header₁) ... T(header_n) T(nfp₁) ... T(nfp_n) T(source) T(target) T(use_service) </wgMediator></pre>	

)		
WSML wwMediator	T ((Name,ann,ontID,medID,nfp,sourceID,targetID,serviceID) wwm)	T (wwMediator A header ₁ ... header _n nfp ₁ ... nfp _n source target use_service)	<wwMediator name="A"> T(header ₁) ... T(header _n) T(nfp ₁) ... T(nfp _n) T(source) T(target) T(use_service) </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 header ₁ ... header _n nfp ₁ ... nfp _n capability interface ₁ ... interface _n)	<webService name="A"> T(header ₁) ... T(header _n) T(nfp ₁) ... T(nfp _n) T(capability) T(interface ₁) ... T(interface _n) </webService>	
WSML Capability	T ((Name,ann,ontID,medID,nfp,sharedVar,pre,post,ass,eff) capability)	T (capability C header ₁ ... header _n nfp ₁ ... nfp _n)	<capability name="C"> T(header ₁) ... T(header _n) T(nfp ₁) ... T(nfp _n)	pre_post_ass_or_eff unites the axiom definitions for precondition, assumption, postcondition and effect.

		<pre> sharedvardef pre_post_ass_or_eff₁ ... pre_post_ass_or_eff_n) </pre>	<pre> T(sharedvardef) T(pre_post_ass_or_eff₁) ... T(pre_post_ass_or_eff_n) </capability> </pre>	
	<pre> T (sharedVar) </pre>	<pre> T (sharedVariables {?v₁, ..., ? v_n}) </pre>	<pre> <sharedVariables> <variable name=" ?v₁" /> ... <variable name=" ?v_n" /> </sharedVariables> </pre>	
	<pre> T (pre) </pre>	<pre> T (precondition B axiomdefinition) </pre>	<pre> <precondition name="B"> T(axiomdefinition) </precondition> </pre>	
	<pre> T (ass) </pre>	<pre> T (assumption B axiomdefinition) </pre>	<pre> <assumption name="B"> T(axiomdefinition) </assumption> </pre>	
	<pre> T (post) </pre>	<pre> T (postcondition B axiomdefinition) </pre>	<pre> <postcondition name="B"> T(axiomdefinition) </postcondition> </pre>	
	<pre> T (eff) </pre>	<pre> T (effect B axiomdefinition) </pre>	<pre> <effect name="B"> T(axiomdefinition) </effect> </pre>	
WSML Interface	<pre> T ((Name,ann,ontID,medID,nfp,chor,orchID)_{interface}) </pre>	<pre> T (interface A header₁ ... header_n nfp₁ ... nfp_n choreography orchestration) </pre>	<pre> <interface name="A"> T(header₁) ... T(header_n) T(nfp₁) ... T(nfp_n) T(choreography) T(orchestration) </interface> </pre>	
WSML Choreography	<pre> T ((Name,ann,ontID,medID,nfp,signature,rule)_{choreography}) </pre>	<pre> T (choreography C header₁ ... header_n statesignature) </pre>	<pre> <choreography name="C"> T(header₁) ... T(header_n) T(statesignature) T(transitions) </pre>	<p>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></p>

		transitions)	</choreography>	
WSML State Signature	T ((Name,ann,ontID,medID,nfp,mode) _{ss})	T (statesignature C header ₁ ... header _n mode ₁ ... mode _n)	<statesignature name="C"> T(header ₁) ... T(header _n) T(mode ₁) ... T(mode _n) </statesignature>	
WSML Mode	T ((type,conceptID,groundingID) _{cm})	T (type concept A grounding)	<mode> T(type) <concept/> <iri>A</iri> T(grounding) </mode>	
WSML Mode	T ((type,conceptID,groundingID) _{rm})	T (type relation A grounding)	<mode> T(type) <relation/> <iri>A</iri> T(grounding) </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 (if condition then rule ₁ ... rule _n endif)	<rule> <if/> T(condition) <then/> T(rule ₁) ... T(rule _n) <endif/> </rule>	rule is a set of WSML transition rules expressed using the surface syntax as: rule ₁ ,...,rule _n and logExp is a WSML logical expression expressed using the surface syntax as <u>condition</u>
WSML Transition Rule	T ((varID,logExp,rule) _{forall})	T (forall variable ₁ ... variable _n with condition do rule ₁ ... rule _n endforall)	<rule> <forall/> T(variable ₁) ... T(variable _n) <with/> T(condition) <do/> T(rule ₁) ... T(rule _n) <endforall/> </rule>	varID is a set of variable identifiers expressed using the surface syntax as: variable ₁ ,...,variable _n , rule is a set of WSML transition rules expressed using the surface syntax as: rule ₁ ,...,rule _n and logExp is a WSML logical expression expressed using the surface syntax as: <u>condition</u>
WSML Transition Rule	T ((varID,logExp,rule) _{choose})	T (choose variable ₁ ... variable _n with condition do rule ₁ ... rule _n endchoose)	<rule> <choose/> T(variable ₁) ... T(variable _n) <with/> T(condition) <do/> T(rule ₁) ... T(rule _n) <endchoose/> </rule>	varID is a set of variable identifiers expressed using the surface syntax as: variable ₁ ,...,variable _n , rule is a set of WSML transition rules expressed using the surface syntax as: rule ₁ ,...,rule _n and logExp is a WSML logical expression expressed using the surface syntax as <u>condition</u>
WSML Transition Rule	T ((rule) _{pipel})	T (rule ₁ ... rule _n)	<rule> T(rule ₁) <pipe/> ... <pipe/> T(rule _n) </rule>	
WSML Transition Rule	T ((fact) _{add})	T (add fact)	<rule> <add/> T(fact) </rule>	fact is a ground atomic fomula
WSML Transition Rule	T ((fact) _{delete})	T (delete fact)	<rule> <delete/> T(fact) </rule>	fact is a ground atomic fomula

WSML Orchestration	T (orchestration)	T (orchestration A)	<orchestration>A</orchestration>	
WSML Nonfunctional Property	T ((name, ann, value, logExp) _{nip})	T (nonFunctionalProperty annotations name hasValue {value ₁ , ..., value _n } log_definition)	<nonFunctionalProperty name="name"> T(annotations) T(value ₁) ... T(value _n) T(log_definition) </nonFunctionalProperty>	
WSML Logical Definition	T (logDefinition)	T (definedBy log_expr ₁ , ..., log_expr _n)	<logicalDefinition> <definedBy> T(log_expr ₁) ... T(log_expr _n) </definedBy> </logicalDefinition>	
WSML Annotations	T ((name, value) _{ann})	T (annotations attributevalue ₁ ... attributevalue _n endAnnotations)	<annotations> T(attributevalue ₁) ... T(attributevalue _n) </annotations>	
	T (ontID)	T (importsOntology {A ₁ , ..., A _n })	<importsOntology>A ₁ </importsOntology> ... <importsOntology>A _n </importsOntology>	
	T (medID)	T (usesMediator {B ₁ , ..., B _n })	<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.

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

	WSML Abstract syntax	WSML Surface syntax	XML Tree	Remarks
WSML Predicate	T' (A(term ₁ , ..., term _n))	T' (A(term ₁ , ..., term _n))	<term name="A"> T' (term ₁ , arg) ... T' (term _n , arg) </term>	

WSML Term	$T(V)$	$T(V)$	<code><term name="V"/></code>
WSML Atom	$T(A(\text{term}_1, \dots, \text{term}_n))$	$T(A(\underline{\text{term}}_1, \dots, \underline{\text{term}}_n))$	<code><atom name="A"> T(<u>term</u>₁, arg) ... T(<u>term</u>_n, arg) </atom></code>
true	$T(T)$	$T(\text{true})$	<code><true/></code>
false	$T()$	$T(\text{false})$	<code><false/></code>
equal	$T(T_1 = T_2)$	$T(T_1 \text{ equal } T_2)$	<code><equal> T(<u>T</u>₁) T(<u>T</u>₂) </equal></code>
subConceptOf	$T(T_1 :: T_2, \dots, T_n)$	$T(T_1 \text{ subConceptOf } T_2, \dots, T_n)$	<code><molecule> T(<u>T</u>₁) <isa type="subConceptOf"> T(<u>T</u>₂) ... T(<u>T</u>_n) </isa> </molecule></code>
memberOf	$T(T_1 : T_2, \dots, T_n)$	$T(T_1 \text{ memberOf } T_2, \dots, T_n)$	<code><molecule> T(<u>T</u>₁) <isa type="memberOf"> T(<u>T</u>₂) ... T(<u>T</u>_n) </isa> </molecule></code>
	$T(T[\text{attr_relation}_1, \dots, \text{attr_relation}_n])$	$T(T[\underline{\text{attr_relation}}_1, \dots, \underline{\text{attr_relation}}_n])$	<code><molecule> T(<u>T</u>) T(<u>attr_relation</u>₁) ... T(<u>attr_relation</u>_n) </molecule></code>
			<code><attributeDefinition type="constraining"></code>

ofType	$T (T_0 \text{ of } \{T_1, \dots, T_n\})$	$T (T_0 \text{ ofType } \{T_1, \dots, T_n\})$	$T(T_0, \text{name})$ $T(T_1, \text{type})$... $T(T_n, \text{type})$ </attributeDefinition>
impliesType	$T (T_0 \text{ it } \{T_1, \dots, T_n\})$	$T (T_0 \text{ impliesType } \{T_1, \dots, T_n\})$	<attributeDefinition type="inferring"> $T(T_0, \text{name})$ $T(T_1, \text{type})$... $T(T_n, \text{type})$ </attributeDefinition>
hasValue	$T (T_0 \text{ hv } \{T_1, \dots, T_n\})$	$T (T_0 \text{ hasValue } \{T_1, \dots, T_n\})$	<attributeValue> $T(T_0, \text{name})$ $T(T_1, \text{value})$... $T(T_n, \text{value})$ </attributeValue>
and	$T (\text{expr}_1 \wedge \text{expr}_2)$	$T (\text{expr}_1 \text{ and } \text{expr}_2)$	<and> $T(\text{expr}_1)$ $T(\text{expr}_2)$ </and>
or	$T (\text{expr}_1 \vee \text{expr}_2)$	$T (\text{expr}_1 \text{ or } \text{expr}_2)$	<or> $T(\text{expr}_1)$ $T(\text{expr}_2)$ </or>
neg	$T (\neg \text{expr})$	$T (\text{neg } \text{expr})$	<neg> $T(\text{expr})$ </neg>
naf	$T (\text{not } \text{expr})$	$T (\text{naf } \text{expr})$	<naf> $T(\text{expr})$ </naf>
implies	$T (\text{expr}_1 \square \text{expr}_2)$	$T (\text{expr}_1 \text{ implies } \text{expr}_2)$	<implies> $T(\text{expr}_1)$ $T(\text{expr}_2)$ </implies>
impliedBy	$T (\text{expr}_1 \square \text{expr}_2)$	$T (\text{expr}_1 \text{ impliedBy } \text{expr}_2)$	<impliedBy> $T(\text{expr}_1)$ $T(\text{expr}_2)$ </impliedBy>

equivalent	T (expr ₁ ≡ expr ₂)	T (<u>expr₁</u> equivalent <u>expr₂</u>)	<equivalent> T(<u>expr₁</u>) T(<u>expr₂</u>) </equivalent>
forall	T (□ ?v ₁ , ..., ?v _n (expr))	T (forall ?v ₁ , ..., ?v _n (<u>expr</u>))	<forall> <variable name=" ?v ₁ " /> ... <variable name=" ?v _n " /> T(<u>expr</u>) </forall>
exists	T (□ ?v ₁ , ..., ?v _n (expr))	T (exists ?v ₁ , ..., ?v _n (<u>expr</u>))	<exists> <variable name=" ?v ₁ " /> ... <variable name=" ?v _n " /> T(<u>expr</u>) </exists>
constraint	T (expr □ □)	T (!- <u>expr</u>)	<constraint> T(<u>expr</u>) </constraint>
impliedByLP	T (expr ₂ □ expr ₁)	T (<u>expr₁</u> :- <u>expr₂</u>)	<impliedByLP> T(<u>expr₁</u>) T(<u>expr₂</u>) </impliedByLP>

Table 2.3 provides a simple translation example.

Table 2.3: A Mapping Example

WSML syntax	XML Tree
<pre> wsmIVariant _ "http://www.wsmo.org/wsml/wsml-syntax/wsmI-flight" namespace { _ "http://www.example.org/ex1#", dc "http://purl.org/dc/elements/1.1#", wsml _ "http://www.wsmo.org/wsml/wsml-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" </pre>	<pre> <wsmI xmlns="http://www.wsmo.org/wsml/wsml-syntax#" variant="http://www.wsmo.org/wsml/wsml-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/wsml/wsml-syntax#string"> WSML to RDF </value> </attributeValue> <attributeValue name="http://purl.org/dc/elements/1.1#date"> <value type="http://www.wsmo.org/wsml/wsml-syntax#date"> <argument>2005</argument> <argument>12</argument> <argument>12</argument> </value> </attributeValue> </pre>

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"}

```
</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/wsm/wsm-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/wsm/wsm-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>
<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/wsm/wsm-syntax#string">
      Mary Jones
    </value>
  </attributeValue>
  <attributeValue name="http://www.example.org/ex1#age">
    <value type="http://www.wsmo.org/wsm/wsm-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>
</wsml>
```

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 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/CE1/113; 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.