Abstract

We introduce two distinct ways to relate WSML and RDF: The RDF representation of WSML describes the preferred way of representing WSML descriptions in RDF. Due to the context-free nature of RDF, not all of WSML is captured faithfully in this RDF representation. The RDF syntax is a serialization for WSML which syntactically uses RDF but diverges from the RDF semantics.

Table of Contents

1. Introduction
2. Relation between WSML and RDF
3. RDF representation of WSML
   3.1. General structure of WSML/RDF graph
   3.2. IRIs and data values
   3.3. Ontologies
   3.4. Goals and Web Services
   3.5. Mediators
1. Introduction

We introduce two distinct ways to relate WSML and RDF:

RDF Representation

The RDF representation for WSML is the preferred way of representing WSML goals, web services, mediators and ontologies in RDF.

Because of the nature of RDF, where every triple can be interpreted separately, it is hard to accurately capture WSML, because many parts of WSML descriptions are context-dependent example, non-functional properties. In WSML, URIs are interpreted depending on their context, because their context is always clear. In RDF, there is no such distinction of context and therefore it is very hard to capture WSML completely in RDF.

The RDF representation of WSML is therefore not completely accurate with respect to standard WSML. For example, in the RDF representation, one cannot use the same identifier both for an attribute and a non-functional property. We believe, however, that such cases will occur seldom and that engineering tools should actually discourage such duplicate use of identifiers.

Below, we explain the relationship between WSML specifications and RDF graphs in Chapter 2. We present the RDF representation for WSML in Chapter 3. In Chapter 4 we introduce the formal mapping between WSML surface syntax and WSML/RDF. This mapping specifies the RDF graphs which can be derived from WSML specifications.

RDF Syntax

As pointed out above, the RDF representation of WSML does not accurately capture WSML. Therefore, we present the WSML RDF syntax in Chapter 5. This RDF syntax is a faithful representation of WSML using RDF triples. This representation, however, does not completely adhere to the RDF semantics, but can be used for exchanging WSML specification over the Semantic Web and using RDF tools for storing and retrieving WSML specifications.

Both the RDF representation and the RDF syntax only treat the conceptual syntax of WSML. Logical expressions in WSML are not translated to RDF, because RDF is not suitable for representing complex logical formulas. Logical expressions can either be included as XML literals in WSML/XML format, as defined in [WSML] (preferred), or as plain literals using the WSML logical expression syntax.

We conclude with a summary in Chapter 6.

2. Relation between WSML and RDF

The Web Service Modeling Language (WSML) is a language for specifying ontologies (including their instances) and for describing various aspects of Web Services. In order to allow WSML data to become an integral part of the Semantic Web, we need an RDF representation, as it is assumed that the Semantic Web will most likely represent data in this format.

WSML data can be translated to RDF in a straightforward way, making WSML an ontology and the WSML data an instance of that ontology. The following listing shows a simple WSML document and how it could be translated into RDF (following D16.1_v0.21):

```xml
// trivial WSML ontology and Web service
namespace (ns _"http://example.com/")
ontology ns:Transportation
    concept ns:Vehicle
```
Such direct translation is a useful solution for most parts of WSML data, but it is suboptimal for WSML ontologies and their instances, because the direct translation of WSML to RDF does not indicate that WSML concepts are similar to RDF classes. In fact, the semantics of WSML concepts and RDF classes differ slightly (for example, rdfs:Resource is a class that contains a resource, i.e. everything, but there is no such concept in WSML, and rdfs:Class contains itself, which cannot be modeled in WSML).

The direct translation approach (taken originally by D16.1) is based on the assumption that WSML must be represented completely and accurately in RDF. This document relaxes this assumption, recognizing that the result harms reusability, and reusability is the main point of WSML/RDF. Therefore in this document we attempt to transform from WSML to RDF only those parts that are expressible in RDFS, trying to capture the intention of a WSML ontology, not all its properties. The main differences between WSML data and the same data after translation to RDF are listed below:

1. WSML allows putting different and independent non-functional properties on entities with the same identifier but different type (concept, instance, attribute etc.), but in RDF non-functional properties are attached to the identifier, so in effect all the entities with the same name share the same non-functional properties. However, when multiple entities have the same IRI identifier, they must be considered the same in some sense already, so this limitation should not be a problem.

2. In the RDF representation we model attribute values directly as triples, and we do the same for non-functional properties. To differentiate between non-functional properties and (functional) attribute values, we mark non-functional properties as members of the class owl:AnnotationProperty. The following listing illustrates the mapping:

   // wsml source data
   instance instA
   nonFunctionalProperties
dc:title hasValue "Instance A"
endNonFunctionalProperties
attr hasValue "value"

   // resulting RDF triples
   instA dc:title "Instance A".
   instA attr "value".
   dc:title rdf:type owl:AnnotationProperty .

   This approach has the limitation that one identifier cannot be used as an attribute and as a non-functional property in the same ontology, but we do not expect this to be a problem.

In effect the RDF representation of WSML cannot reliably serve as syntax for WSML, but it is not intended to do so.

3. RDF representation for WSML

Please find the complete RDF Schema for WSML in Appendix A.

Non-functional properties in WSML are translated to ordinary triples in RDF. Additionally, for each non-functional property \( nfp \), the following triple is added (with rdf and owl denoting the RDF and OWL namespaces, respectively):

\[ nfp \text{ rdf:type owl:AnnotationProperty } . \]
In the remainder of the chapter, wsml stands for the namespace http://www.wsmo.org/wsml/wsml-syntax#, rdf stands for the namespace http://www.w3.org/1999/02/22-rdf-syntax-ns#, rdfs stands for the namespace http://www.w3.org/2000/01/rdf-schema#, owl stands for the namespace http://www.w3.org/2002/07/owl#, and xsd stands for the namespace http://www.w3.org/2001/XMLSchema#. Finally, prefix:localname stands for the concatenation of the namespace and the local name.

3.1. General structure of WSML/RDF graph

WSML descriptions group all data related to a particular ontology, Web service, goal or mediator in one description. This is achieved in the WSML surface syntax by grouping all descriptions under the ontology, webService, goal, ggMediator, wgMediator, wwMediator, and ooMediator keywords. The same holds for lower-level entities: concepts in an ontology, for example, group a number of attribute definitions and Web service capabilities group preconditions, postconditions, assumptions and effects. Conceptually, a WSML description can be seen as a part-whole hierarchy. An ontology has as parts the concept, relation, axiom, and instance definitions; in turn, these definitions are part of the ontology. Similarly it works for Web services, goals and mediators.

In WSML/RDF, WSML descriptions are part-whole hierarchies. For this purpose we use a part-whole ontology inspired by the work of the by the Semantic Web Best Practices Working Group: http://www.wsmo.org/TR/d32/v0.1/part.owl.

Each ontology, Web service, goal, and mediator is a node in the RDF graph, which is connected to all its parts using the relationship hasPart_directly. This is illustrated in Figure 1.

![Figure 1. WSML Descriptions as part-whole hierarchy](image)

In some cases it is necessary to disambiguate between different parts of a whole using the hasPart property. This is for example the case for the preconditions, postconditions, assumptions and effects of a capability. For example, one axiom may be a precondition of one capability a postcondition of another capability. For this reason, we define sub-properties of hasPart_directly, namely hasPrecondition, hasPostcondition, hasAssumption, and hasEffect.

Logical expressions defined in axioms are converted to:

- XML literals, according to [WSML], Chapter 9, or
- plain literals, using the WSML logical expression syntax, defined in [WSML], Section 2.8.

The WSML top-level entities ontology, goal, webService, and mediator correspond to RDFS classes. These RDFS classes, together with the properties which can be used in combination with these classes, are listed in Tables 1 and 2. These classes share the same superclass wsml:topLevelEntity.
3.2. IRI s and data values

As can be seen from [WSML], Section 2.1.2, there are three kinds of identifiers in WSML:

- IRIs can be used directly in the RDF representation
- Data values are translated to typed XML literals in RDF, according to [WSML], Appendix C.1. For example, `_date(2005,12,2)` is translated to "2005-12-02"^^xs:date, assuming xs stands for the XML Schema datatype namespace.
- Unnumbered anonymous identifiers are translated to new unique identifiers, following the UUID URI scheme, as specified in the IETF RFC 4122 [RFC4122], for example: `urn:uuid:989C6E5C-2CC1-11CA-9044-08002B1BB4F5`.

Anonymous identifiers cannot be translated to blank nodes in RDF, because there is a difference in the way these are treated. Blank nodes in RDF can be seen as existentially quantified variables, whereas anonymous identifiers in WSML can be seen as fresh constant symbols; similar to the notion of skolem constants. Anonymous identifiers in WSML correspond to what are sometimes called `rigid` bNodes.

3.3. Ontologies

Table 3 describes the different classes used for representing entities in WSML ontologies. The section on the WSML specification [WSML] which describes the particular WSML entity is mentioned in parenthesis. The parts comprising the whole are listed in the 3rd column.

Table 4 lists the properties one can use in ontology definitions. In case there are several classes mentioned in the domain or range of a property, separated with a comma ',', these should be interpreted disjunctively, i.e., the domain (range) is a member of one of these classes, but not necessarily all. Notice that it is not possible to specify disjunctive domains and ranges in R. Therefore, disjunctive domains and ranges are not explicitly specified in the RDFS ontology in Appendix A.

![Table 1. Classes for top-level elements of WSML/RDF](image)

<table>
<thead>
<tr>
<th>Class</th>
<th>WSML entity</th>
<th>Has parts</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsml:TopLevelElement</td>
<td></td>
<td>wsml:Concept, wsml:Relation, wsml:RelationInstance, wsml:Axiom</td>
<td>Superclass for all WSML top-level elements.</td>
</tr>
<tr>
<td>wsml:Ontology</td>
<td>ontology (2.3)</td>
<td></td>
<td>The WSML variant is indicated using the property wsml:variant.</td>
</tr>
<tr>
<td>wsml:Goal</td>
<td>Goal (2.5)</td>
<td>wsml:Capability, wsml:Interface</td>
<td>The WSML variant is indicated using the property wsml:variant.</td>
</tr>
<tr>
<td>wsml:WebService</td>
<td>WebService (2.7)</td>
<td></td>
<td>The WSML variant is indicated using the property wsml:variant.</td>
</tr>
<tr>
<td>wsml:Mediator</td>
<td>(2.6)</td>
<td></td>
<td>The WSML variant is indicated using the property wsml:variant. Source, target and used service are indicated using the respective properties.</td>
</tr>
</tbody>
</table>

![Table 2. Properties for top-level elements of WSML/RDF](image)

<table>
<thead>
<tr>
<th>Property</th>
<th>WSML entity</th>
<th>Range</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsml:variant</td>
<td>wsmiVariant (2.2.1)</td>
<td></td>
<td>Use one of the standard URIs for WSML variants.</td>
</tr>
<tr>
<td>wsml:importsOntology</td>
<td>importsOntology (2.2.3)</td>
<td>wsml:Ontology</td>
<td></td>
</tr>
<tr>
<td>wsml:usesMediator</td>
<td>usesMediator (2.2.3)</td>
<td>wsml:Mediator</td>
<td></td>
</tr>
</tbody>
</table>

![Table 3. Classes for ontologies in WSML/RDF](image)

<table>
<thead>
<tr>
<th>Class</th>
<th>WSML entity</th>
<th>Has parts</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsml:Concept</td>
<td>concept (2.3.1)</td>
<td>wsml:attributeDefinition</td>
<td>is a subclass of rdfs:Class</td>
</tr>
</tbody>
</table>
### Table 4. Properties for ontologies in WSML/RDF

<table>
<thead>
<tr>
<th>Property</th>
<th>WSML entity</th>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsml:hasAttributeDefinition</td>
<td>wsml:Concept</td>
<td>wsml:AttributeDefinition</td>
<td></td>
</tr>
<tr>
<td>wsml:forAttribute</td>
<td>wsml:AttributeDefinition</td>
<td>wsml:AttributeDefinition</td>
<td></td>
</tr>
<tr>
<td>wsml:ofType</td>
<td>ofType (2.3.1)</td>
<td>wsml:AttributeDefinition, wsml:ParameterDefinition</td>
<td>wsml:Concept</td>
</tr>
<tr>
<td>wsml:impliesType</td>
<td>impliesType (2.3.1)</td>
<td>wsml:AttributeDefinition, wsml:ParameterDefinition</td>
<td>wsml:Concept</td>
</tr>
<tr>
<td>wsml:maxCardinality</td>
<td>(2.3.1)</td>
<td>wsml:AttributeDefinition</td>
<td>xsd:nonNegativeInteger</td>
</tr>
<tr>
<td>wsml:minCardinality</td>
<td>(2.3.1)</td>
<td>wsml:AttributeDefinition</td>
<td>xsd:nonNegativeInteger</td>
</tr>
<tr>
<td>wsml:inverseOf</td>
<td>inverseOf (2.3.1)</td>
<td>wsml:AttributeDefinition</td>
<td></td>
</tr>
<tr>
<td>wsml:arity</td>
<td>(2.3.2)</td>
<td>wsml:Relation</td>
<td>xsd:nonNegativeInteger</td>
</tr>
<tr>
<td>wsml:subRelationOf</td>
<td>subRelationOf (2.3.2)</td>
<td>wsml:Relation</td>
<td></td>
</tr>
</tbody>
</table>

Example 2.1 Given the following WSML ontology:

```xml
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
onontology _"http://example.org/amazonOntology"
nonFunctionalProperties
dc#title hasValue "Example Book ontology"
dc#description hasValue "Example ontology about books and shopping carts"
```
endNonFunctionalProperties
concept book
title ofType _string
hasAuthor ofType author
concept author subConceptOf person
    authorOf inverseOf(hasAuthor) ofType book
concept cart
nonFunctionalProperties
dc#description hasValue "A shopping cart has exactly one id and zero or more items, which are books."
endNonFunctionalProperties
id ofType (1) _string
items ofType book
instance crimeAndPunishment memberOf book
title hasValue "Crime and Punishment"
hasAuthor hasValue dostoyevsky
relation authorship(impliesType author, impliesType document)
nonFunctionalProperties
dc#relation hasValue authorshipFromAuthor
endNonFunctionalProperties
axiom authorshipFromAuthor
    definedBy
    authorship(?x,?y) :- ?x[authorOf hasValue ?y] memberOf author.

The WSML/RDF representation is the following (using N3 notation):

@prefix :<http://example.org/bookOntology#>.
@prefix dc:<http://purl.org/dc/elements/1.1/>.
@prefix wsml:<http://www.wsmo.org/wsml/wsml-syntax#>.
@prefix part-whole:<http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/part.owl#>.
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd:<http://www.w3.org/2001/XMLSchema#>.

<http://example.org/amazonOntology>
    rdf:type wsml:Ontology;
    dc:description "Example ontology about books and shopping carts";
    dc:title rdf:type owl:AnnotationProperty.
    dc:description rdf:type owl:AnnotationProperty.

<http://example.org/bookOntology>
    part-whole:hasPart_directly :book;
    part-whole:hasPart_directly :author;
    part-whole:hasPart_directly :cart;
    part-whole:hasPart_directly :crimeAndPunishment;
    part-whole:hasPart_directly :authorship;
    part-whole:hasPart_directly :authorshipFromAuthor.

:book
    rdf:type wsml:Concept;
    part-whole:hasPart_directly _:title;
part-whole:hasPart_directly _hasAuthor.

_:title
  rdf:type wsml:AttributeDefinition;
  wsml:forAttribute :title;
  wsml:ofType xsd:string.

_:hasAuthor
  rdf:type wsml:AttributeDefinition;
  wsml:forAttribute :hasAuthor;
  wsml:ofType :author.

:author
  rdf:type wsml:Concept;
  rdfs:subClassOf :person;
  part-whole:hasPart_directly _:authorOf.

_:authorOf
  rdf:type wsml:AttributeDefinition;
  wsml:forAttribute :authorOf;
  wsml:ofType :book;
  wsml:inverseOf :hasAuthor.

:cart
  rdf:type wsml:Concept;
  dc:description "A shopping cart has exactly one id and zero or more items, which are books."
  part-whole:hasPart_directly _:id;
  part-whole:hasPart_directly _:items.

_:id
  rdf:type wsml:AttributeDefinition;
  wsml:forAttribute :id;
  wsml:ofType xsd:string;
  wsml:minCardinality "1"^^xsd:integer;
  wsml:maxCardinality "1"^^xsd:integer.

_:items
  rdf:type wsml:AttributeDefinition;
  wsml:forAttribute :items;
  wsml:ofType :book.

:crimeAndPunishment
  rdf:type :book;
  :title "Crime and Punishment";
  :hasAuthor :dostoyevsky.

:authorship
  rdf:type wsml:Relation;
  dc:relation authorshipFromAuthor;
  part-whole:hasPart_fromAuthor _list1.

_list1
  rdf:first _par1;
  rdf:rest _list2.

_list2
  rdf:first _par2;
  rdf:rest rdf:nil.

_:par1
  rdf:type wsml:ParameterDefinition;
  wsml:impliesType :author.
3.4. Goals and Web Services

Tables 5 and 6, respectively, describe the classes and properties used for goals and web services in WSML/RDF.

<table>
<thead>
<tr>
<th>Class</th>
<th>WSML entity</th>
<th>Has parts</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsml:Capability</td>
<td>capability (2.4.1)</td>
<td>wsml:Axiom</td>
<td>The parts of the capability, namely the axioms, are subdivided into preconditions, postconditions, assumptions and effects by the use of the properties wsml:hasPrecondition, wsml:hasPostcondition, wsml:hasAssumption, and wsml:hasEffect, respectively.</td>
</tr>
<tr>
<td>wsml:interface</td>
<td>interface (2.4.2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wsml:Choreography</td>
<td>choreography (2.4.2)</td>
<td></td>
<td>Choreographies are external to WSML.</td>
</tr>
<tr>
<td>wsml:Orchestration</td>
<td>orchestration (2.4.2)</td>
<td></td>
<td>Orchestrations are external to WSML.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>WSML entity</th>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsml:hasPrecondition</td>
<td>precondition (2.4.1)</td>
<td>wsml:Capability</td>
<td>wsml:Axiom</td>
</tr>
<tr>
<td>wsml:hasPostcondition</td>
<td>postcondition (2.4.1)</td>
<td>wsml:Capability</td>
<td>wsml:Axiom</td>
</tr>
<tr>
<td>wsml:hasAssumption</td>
<td>assumption (2.4.1)</td>
<td>wsml:Capability</td>
<td>wsml:Axiom</td>
</tr>
<tr>
<td>wsml:hasEffect</td>
<td>effect (2.4.1)</td>
<td>wsml:Capability</td>
<td>wsml:Axiom</td>
</tr>
<tr>
<td>wsml:sharedVariable</td>
<td>sharedVariables (2.4.1)</td>
<td></td>
<td>xsd:string</td>
</tr>
</tbody>
</table>

3.5. Mediators

Tables 7 and 6, respectively, describe the classes and properties used for mediators in WSML/RDF.
4. Translating WSML to the RDF Representation

This section will contain a mapping from the WSML surface syntax to RDF. The mapping will be created as soon at there is agreement on the RDF representation of WSML. See Appendix A for the RDF Schema.

In this chapter an RDF [RDF] representation for WSML is introduced. The vocabulary used is an extension of the RDF Schema vocabulary defined in [Brickley & Guha, 2004]. The extension consists of WSML language components, as given in the previous chapters and the class owl:AnnotationProperty used to model the Non-Functional Properties available in WSML.

The big advantage of having an RDF/WSML representation encoded in <subject><predicate><object>-triples and of reusing the RDFS-vocabulary as much as possible, is the fact that there are many existing RDF(S)-based tools available. These tools are optimized to handle triples and are thus able to process the semantics of our specification and in a more general sense we can guarantee inter-operability with those.

Table 4.1 defines the mapping function T from WSML entities to RDF triples. Logical expressions are not translated to RDF. Instead, they are either translated to the WSML/XML syntax and are specified as literals of type rdf:XMLLiteral or WSML logical expression syntax, in which case they are specified as plain literals. The transformation creates an RDF-graph based on above introduced RDF-triples. As definitions, i.e., top-level entities like ontologies, Web services, goals and mediators are disjoint constructs, their graphs are not inter-related. In other won transformation defines one graph per definition. Note that in the table we use the N3 notation's prefix mechanism to abbreviate IRIs. The following prefix are applied:

- 'wsml' stands for 'http://www.wsmo.org/wsml/wsml-syntax#',
- 'rdf' stands for 'http://www.w3.org/1999/02/22-rdf-syntax-ns#',
- 'rdfs' stands for 'http://www.w3.org/2000/01/rdf-schema#',
- 'dc' stands for 'http://purl.org/dc/elements/1.1#',
- 'xsd' stands for 'http://www.w3.org/2001/XMLSchema#', and
- 'part-whole' stands for 'http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/part.owl#'.

In Table 4.1, A,B,C,Z stand for identifiers, D stands for a datatype identifier, DVl stands for an integer value, DVd stands for a decimal, and DVs stands for a string data value, and k,m,n are integer numbers.

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

Before transforming WSML into the RDF/WSML representation the following pre-processing steps need to be performed:

- If multiple top-level specifications (i.e., ontology, goal, web service, mediator) occur in the same document, the document must be split into multiple WSML documents. Each WSML document is then translated to a separate RDF graph.
sQNames need to be resolved to full IRIs, i.e., there will not be any namespace definitions in the RDF syntax.

In the conceptual syntax universal truth and universal falsehood are resolved to: http://www.wsmo.org/wsml/wsml-syntax#true, and http://www.wsmo.org/wsml/wsml-syntax#false respectively.

Datatype identifiers are translated to XML datatypes. For example, the datatype identifier '_date' is resolved to http://www.w3.org/2001/XMLSchema#date.

Unnumbered anonymous identifiers are translated to new universally unique identifiers (UUID), for example: urn:uuid:989C6E5C-2CC1-11CA-9044-08002B1BB4F5.

<table>
<thead>
<tr>
<th>Name</th>
<th>Bits</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_low</td>
<td>0xFFFFFFFF00000000</td>
<td>989C6E5C</td>
</tr>
<tr>
<td>time_mid</td>
<td>0x00000000FFFFFF0000</td>
<td>2CC1</td>
</tr>
<tr>
<td>version</td>
<td>0x0000000000000000F0001</td>
<td>11CA</td>
</tr>
<tr>
<td>time_hi</td>
<td>0x00000000000000000FFF</td>
<td>11CA</td>
</tr>
</tbody>
</table>

The most significant 64 bits:

<table>
<thead>
<tr>
<th>Name</th>
<th>Bits</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>variant</td>
<td>0xE00000000000000000000</td>
<td>9044</td>
</tr>
<tr>
<td>clock_seq</td>
<td>0x1FFF000000000000000000</td>
<td>9044</td>
</tr>
<tr>
<td>node</td>
<td>0x0000000000000000000000</td>
<td>08002B1BB4F5</td>
</tr>
</tbody>
</table>

A Universally Unique Identifier (UUID) is a 128 bits long identifier and is guaranteed to be unique across space and time and requires no central registration.

The 128 bits are put together as listed above (from most to least significant), more details about the different elements of an UUID can be found in [RFC4122].
<table>
<thead>
<tr>
<th>WSML syntax</th>
<th>RDF Triples</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T (wsmlVariant A</code></td>
<td><code>T(def, A)</code></td>
<td>A definition is one (1) Ontology, Goal, Web service or Mediator description</td>
</tr>
<tr>
<td><code>definition)</code></td>
<td><code>T(def, A)</code></td>
<td></td>
</tr>
<tr>
<td><strong>namespace</strong> <code>{ N, P1 N1</code></td>
<td><code>T(def, A)</code></td>
<td>Because sQnames were resolved to full IRIs during pre-processing, there is no translation to RDF necessary for namespace definitions.</td>
</tr>
<tr>
<td><code>... Pn Nn }</code></td>
<td><code>T(def, A)</code></td>
<td></td>
</tr>
<tr>
<td><strong>T (ontology A</strong></td>
<td><code>A rdf:type wsml:Ontology</code></td>
<td>An ontology_element represents possible content of an ontology definition, i.e., concepts, relations, instances, ...</td>
</tr>
<tr>
<td><code>header1</code></td>
<td><code>A wsml:variant Z</code></td>
<td></td>
</tr>
<tr>
<td><code>... headern</code></td>
<td><code>T(header, A)</code></td>
<td></td>
</tr>
<tr>
<td><code>ontology_element1</code></td>
<td><code>T(ontology_element, A)</code></td>
<td></td>
</tr>
<tr>
<td><code>... ontology_elementn</code></td>
<td><code>T(ontology_element, A)</code></td>
<td></td>
</tr>
<tr>
<td><code>Z )</code></td>
<td><code>T(ontology, A)</code></td>
<td></td>
</tr>
<tr>
<td><strong>T (concept A</strong></td>
<td><code>Z part-whole:hasPart_directly A</code></td>
<td></td>
</tr>
<tr>
<td><code>subConceptOf { B1,...,Bn }</code></td>
<td><code>A rdf:type wsml:Concept</code></td>
<td></td>
</tr>
<tr>
<td><code>nfp attribute1</code></td>
<td><code>A rdfs:subClassOf B1</code></td>
<td></td>
</tr>
<tr>
<td><code>... attributeN</code></td>
<td><code>A rdfs:subClassOf Bn</code></td>
<td></td>
</tr>
<tr>
<td><code>Z )</code></td>
<td><code>T(attribute, A)</code></td>
<td></td>
</tr>
<tr>
<td>**T (A attributefeature1`</td>
<td><code>Z part-whole:hasPart_directly _X</code></td>
<td></td>
</tr>
<tr>
<td><code>... attributefeatureN ofType</code></td>
<td><code>_X rdf:type wsml:AttributeDefinition</code></td>
<td></td>
</tr>
<tr>
<td><code>cardinality C</code></td>
<td><code>_X wsml:forAttribute A</code></td>
<td></td>
</tr>
<tr>
<td><code>nfp Z</code></td>
<td><code>_X wsml:ofType C</code></td>
<td></td>
</tr>
<tr>
<td>**T (A attributefeature1`</td>
<td><code>Z part-whole:hasPart_directly _X</code></td>
<td></td>
</tr>
<tr>
<td><code>... attributefeatureN impliesType</code></td>
<td><code>A attributefeature1 _X rdf:type wsml:AttributeDefinition</code></td>
<td></td>
</tr>
<tr>
<td><code>cardinality C</code></td>
<td><code>A rdfs:subClassOf A</code></td>
<td></td>
</tr>
<tr>
<td><code>nfp Z</code></td>
<td><code>T(cardinality _X)</code></td>
<td></td>
</tr>
<tr>
<td>**T (A attributefeature1`</td>
<td><code>Z part-whole:hasPart_directly _X</code></td>
<td></td>
</tr>
<tr>
<td><code>... attributefeatureN impliesType</code></td>
<td><code>A attributefeature1 _X rdf:type wsml:AttributeDefinition</code></td>
<td></td>
</tr>
<tr>
<td><code>cardinality C</code></td>
<td><code>A rdfs:subClassOf A</code></td>
<td></td>
</tr>
<tr>
<td><code>nfp Z</code></td>
<td><code>T(cardinality _X)</code></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Mapping to the RDF representation
5. RDF Syntax for WSML

In this chapter the mapping to the fully round-trip supporting WSML/RDF syntax is presented (Table 5.1). In contrast to the RDF representation introduced previously, this syntax allows translation of the whole information content of a WSML document. It however still contains the known flaws mentioned in the Introduction.

For the translation of WSML surface syntax to the WSML/RDF syntax presented in this chapter depends on the same pre-processing steps as given in the previous chapter.

- Separation of graphs
- Resolution of sQNames to full IRIs
- Resolution of universal truth and falsehood
- Resolution of anonymous identifiers to UUID
- Resolution of datatype identifiers to full IRIs

Furthermore the readers are asked to consider the same namespace-prefix bindings as given in Chapter 4. There is however one exception: to distinguish between the vocabularies used for the RDF representation and the WSML/RDF syntax the prefix ‘wsmlrdf’ is introduced to represent the namespace ‘http://www.wsmo.org/wsml/wsml-rdf-syntax#’.
<table>
<thead>
<tr>
<th>WSML syntax</th>
<th>RDF Triples</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>T (wsmlVariant A definition1 ... definitionn)</code></td>
<td><code>T(definition1, A)</code></td>
<td>definitions are Ontology, Goal, WebService and Mediators</td>
</tr>
<tr>
<td><code>namespace { N. P1 N1 ... Pn Nn }</code></td>
<td><code>A rdf:type wsmlrdf:Ontology</code></td>
<td>Because sQnames were resolved to full IRIs during pre-processing, there is no translation to RDF necessary for namespace definitions</td>
</tr>
<tr>
<td><code>T (ontology A header1 ... headern, A ontology_element1 ... ontology_elementn, Z)</code></td>
<td><code>Z wsmlrdf:hasConceptDescription _:X _:X wsmlrdf:hasConcept A T(nfp, _:X) _:X rdfs:subClassOf B1 ... _:X rdfs:subClassOf Bn T(attribute1, _:X) ... T(attribute_n, _:X)</code></td>
<td>An ontology_element represents possible content of an ontology definition, i.e., concepts, relations, instances, ...</td>
</tr>
<tr>
<td><code>T (concept A subConceptOf {B1,...,Bn} nfp attribute1 ... attribute_n, Z)</code></td>
<td><code>_:X wsmlrdf:attribute A T(attributefeature1, _:X) ... T(attributefeature_n, _:X)</code></td>
<td>The blank identifier _:X denotes a helper node to bind the attribute A to a defined owner: attributes are locally defined!</td>
</tr>
<tr>
<td><code>T (attributefeature1 ... attributefeature_n ofType cardinality C nfp, Z)</code></td>
<td><code>_:X wsmlrdf:attribute A T(attributefeature1, _:X) ... T(attributefeature_n, _:X)</code></td>
<td></td>
</tr>
<tr>
<td><code>T (attributefeature1 ... attributefeature_n impliesType cardinality C nfp, Z)</code></td>
<td><code>_:X wsmlrdf:attribute A T(attributefeature1, _:X) ... T(attributefeature_n, _:X)</code></td>
<td></td>
</tr>
<tr>
<td><code>T (transitive , Z)</code></td>
<td><code>Z rdf:type wsmlrdf:TransitiveAttribute</code></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Mapping to the WSML/RDF syntax
6. Summary

In this deliverable we have presented the RDF representation as well as the RDF syntax of WSML. The RDF representation comes with an RDF Schema ontology which describes its classes and properties. Both the RDF representation and the RDF syntax come with a formal mapping from the surface syntax of WSML to the RDF representation and syntax, respectively.

The RDF representation captures that part of WSML which can be captured in the spirit of RDF. This means that in the RDF representation one cannot, in general, use the same identifier in different contexts (e.g., as both non-functional property and attribute). We believe, however, that this is not a significant limitation.

The RDF syntax is a faithful representation of WSML using RDF triples. This means that this syntax is an alternative syntax for WSML, just like the surface syntax and XML syntax presented in the WSML specification. The advantage of using this RDF syntax over the surface syntax and XML syntax is that one can use RDF tools for storing, retrieving and manipulating WS specifications.

Appendix A. The RDFS ontology for WSML/RDF

The complete RDF Schema for WSML can also be found here.

```xml
<?xml version='1.0' encoding='UTF-8'?>

<!-- In the definition of the WSML vocabulary, we reuse the following vocabularies: -->
RDF: http://www.w3.org/1999/02/22-rdf-syntax-ns#
RDFS: http://www.w3.org/2000/01/rdf-schema#
XSD: http://www.w3.org/2001/XMLSchema#
OWL: http://www.w3.org/2002/07/owl#
DC (Dublin Core): http://purl.org/dc/elements/1.1/
Part-Whole relations (from the SWBP WG): http://www.wsmo.org/TR/d32/v0.1/part.owl#

Non-functional properties are captured using the owl:AnnotationProperty. Each NFP is made 
rdf:type owl:AnnotationProperty.

Note that this ontology was written in the spirit of RDF: any RDF processor may use the statements which it can work with and will ignore all statements it cannot work with. For example, in case the RDF processor cannot deal with owl:sameAs, it will not. This ontology was explicitly not constructed to fall inside any of the species of OWL, although it necessarily falls inside OWL Full, since every RDF graph is a valid OWL Full ontology.
```
An ontology can be seen as a part-whole hierarchy. An ontology (whole) has parts concepts, relations, instances, axioms and relation instances. A concept (whole) has parts attribute definitions.

It is related to owl:ontology, but not the same, nor equivalent, because there are many ontologies which are wsml ontologies, but not OWL ontologies (e.g. nonmonotonic ontologies).

A web service can be seen as a part-whole hierarchy. A web service (whole) has parts (at most one) capability and (possibly multiple) interfaces.

A goal can be seen as a part-whole hierarchy. A goal (whole) has parts (at most one) capability and (possibly multiple) interfaces.

A mediator may have a source and multiple targets and may use a service for its implementation.

Any WSML entity (goal, webService, ontology, mediator, interface, capability, choreography) may import ontologies in order to reuse vocabularies.

A WSML entity (goal, webService, mediator, ontology) may have a WSML variant associated with it. Defined values:
http://www.wsmo.org/wsml/wsml-syntax/wsml-full
http://www.wsmo.org/wsml/wsml-syntax/wsml-rule
http://www.wsmo.org/wsml/wsml-syntax/wsml-flight
http://www.wsmo.org/wsml/wsml-syntax/wsml-dl
http://www.wsmo.org/wsml/wsml-syntax/wsml-core
In case the same ontology, goal, etc. has different variants associated with it, the highest variant is chosen.
</rdfs:comment>
<rdfs:domain rdf:resource="#TopLevelElement"/>
</rdf:Property>

<rdfs:Property rdf:ID="usesMediator"
              rdfs:label="usesMediator">
  <rdfs:comment>Any WSML entity may use a mediator.</rdfs:comment>
  <rdfs:range rdf:resource="#Mediator"/>
</rdf:Property>

<!-- Ontologies -->
<rdfs:Class rdf:ID="Attribute"
          rdfs:label="Attribute">
  <rdfs:comment>An attribute is a specific type of rdf:Property.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>

<rdfs:Class rdf:ID="AttributeDefinition"
           rdfs:label="AttributeDefinition">
  <rdfs:comment>
    A concept may have a number of attribute definitions. An attribute definition consists of an attribute, a possible inverse attribute, and maximal and minimal cardinality definitions.
  </rdfs:comment>
  <dc:relation>&owl;Restriction</dc:relation>
</rdfs:Class>

<rdfs:Class rdf:ID="ReflexiveAttributeDefinition"
           rdfs:label="ReflexiveAttributeDefinition">
  <rdfs:comment>Reflexive attribute definition</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#AttributeDefinition"/>
</rdfs:Class>

<rdfs:Class rdf:ID="SymmetricAttributeDefinition"
           rdfs:label="SymmetricAttributeDefinition">
  <rdfs:comment>Symmetric attribute definition</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#AttributeDefinition"/>
</rdfs:Class>

<rdfs:Class rdf:ID="TransitiveAttributeDefinition"
           rdfs:label="TransitiveAttributeDefinition">
  <rdfs:comment>Transitive attribute definition</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#AttributeDefinition"/>
</rdfs:Class>

<rdfs:Property rdf:ID="hasAttributeDefinition"
              rdfs:label="hasAttributeDefinition">
  <rdfs:comment>A concept has zero or more attribute definitions.</rdfs:comment>
  <rdfs:range rdf:resource="#AttributeDefinition"/>
  <rdfs:domain rdf:resource="#Concept">
    <rdfs:subPropertyOf rdf:resource="&part-whole;hasPart_directly"/>
    <rdfs:subPropertyOf rdf:resource="#superClassOf"/>
  </rdfs:domain>
</rdf:Property>

<rdfs:Property rdf:ID="superClassOf"
              rdfs:label="superClassOf">
  <rdfs:comment>Inverse of the rdf subclass relation.</rdfs:comment>
  <owl:inverseOf rdf:resource="&rdfs;subClassOf"/>
</rdf:Property>
An attribute definition is associated with one attribute.

The attribute value or relation parameter definition is checked to be of the specific type.

Attribute values and relation parameter definitions are inferred to have a particular type.

An attribute definition may have a maximal cardinality. If no maximal cardinality is described, there is not constraint on the maximal cardinality.

An attribute definition may have a minimal cardinality. If no minimal cardinality is described, the attribute is optional.

An attribute definition may have an inverse attribute associated with it.

An axiom is an arbitrary logical specification which can be used for ontology, but also, for example, for the specification of the functionality of web services through capabilities.
The logical expression itself is either expressed using the WSML/XML syntax for logical expressions as an XML literal, or using the WSML logical expression syntax, as a plain literal. In either case, the logical expression is linked to the axiom using rdfs:isDefinedBy.

</rdfs:comment></rdfs:Class>

<rdfs:Class rdf:ID="Concept"
rdfs:label="Concept">    <rdfs:comment>        A type of rdfs:Class        A concept may have a number of attribute definitions associated with it through the hasPart relationship.    </rdfs:comment>    <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
</rdfs:Class>

<rdfs:Class rdf:ID="ParameterDefinition"
rdfs:label="ParameterDefinition">    <rdfs:comment>        A parameter definition consists of a type (ofType/impliesType) for the parameter, either via the property impliesType or ofType. A relation contains a single rdf:list of parameter definitions.    </rdfs:comment></rdfs:Class>

<rdfs:Class rdf:ID="Relation"
rdfs:label="Relation">    <rdfs:comment>        An RDF property can be seen as a specific kind of WSML relation, namely a binary relation.        A relation has either an arity or a parameter definition associated with it; a relation definition can be compared with the signature specification of a predicate in predicate calculus.    </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="RelationInstance"
rdfs:label="RelationInstance">    <rdfs:comment>        A relation instance is an actual ground fact which corresponds
to a particular relation; it can be seen as a ground atomic formula in predicate calculus.
A relation instance has a list of parameter values.
</rdfs:comment>
</rdfs:Class>

<!- Capabilities -->

<rdfs:Class rdf:ID="Capability"
    rdfs:label="Capability">
    <rdfs:comment>
        A web service capability has a number of postconditions, preconditions, effects, and assumptions, as well as a list of shared variables.
    </rdfs:comment>
    </rdfs:Class>

<rdfs:Property rdf:ID="hasPrecondition"
    rdf:label="hasPrecondition">
    <rdfs:comment>
        A pre-condition is an axiom which expresses conditions over the input of the service.
    </rdfs:comment>
    <rdfs:domain rdf:resource="#Capability"/>
    <rdfs:range rdf:resource="#Axiom"/>
</rdfs:Property>

<rdfs:Property rdf:ID="hasPostcondition"
    rdf:label="hasPostcondition">
    <rdfs:comment>
        A post-condition is an axiom which describes the relation between the input and the output of the service, as well as conditions which are guaranteed to hold over the output.
    </rdfs:comment>
    <rdfs:domain rdf:resource="#Capability"/>
    <rdfs:range rdf:resource="#Axiom"/>
</rdfs:Property>

<rdfs:Property rdf:ID="hasAssumption"
    rdf:label="hasAssumption">
    <rdfs:comment>
        An assumption is an axiom which describes conditions on the state of the world which must hold for the web service to be able to execute.
    </rdfs:comment>
    <rdfs:domain rdf:resource="#Capability"/>
    <rdfs:range rdf:resource="#Axiom"/>
</rdfs:Property>

<rdfs:Property rdf:ID="hasEffect"
    rdf:label="hasEffect">
    <rdfs:comment>
        An effect is an axiom which describes conditions which are guaranteed to hold over the state of the world after execution of the service.
    </rdfs:comment>
    <rdfs:domain rdf:resource="#Capability"/>
    <rdfs:range rdf:resource="#Axiom"/>
</rdfs:Property>

<rdfs:Property rdf:ID="sharedVariable"
    rdf:label="sharedVariable">
    <rdfs:comment>
        A capability has a number of variables which are shared across all definitions.
    </rdfs:comment>
    </rdfs:Property>
By default, all free variables in a logical expression are implicitly universally quantified over this logical expression. Shared variables are free in the logical expression and a universally quantified over the entire capability.

<!-- Interfaces -->

<rdfs:Class rdf:ID="Interface"
  rdfs:label="Interface">  
  A Web Service interface may have a number of choreographies and one orchestration associated with it.  
  </rdfs:Class>

<rdfs:Class rdf:ID="Choreography"
  rdfs:label="Choreography">  
  A Web Service interface may have a choreography associated with it.  
  </rdfs:Class>

<rdfs:Class rdf:ID="Orchestration"
  rdfs:label="Orchestration">  
  A Web Service interface may have an orchestration associated with it.  
  </rdfs:Class>

<!-- Mediators -->

<rdfs:Class rdf:ID="GGMediator"
  rdfs:label="GGMediator">  
  </rdfs:Class>

<rdfs:Class rdf:ID="OOMediator"
  rdfs:label="OOMediator">  
  </rdfs:Class>

<rdfs:Class rdf:ID="WGMediator"
  rdfs:label="WGMediator">  
  </rdfs:Class>

<rdfs:Class rdf:ID="WWMediator"
  rdfs:label="WWMediator">  
  </rdfs:Class>

<rdfs:Property rdf:ID="target"
  rdfs:label="target">  
  A mediator may have one or more targets.  
  </rdfs:Property>

<rdfs:Property rdf:ID="usesService"
  rdfs:label="usesService">  
  A mediator may use a service for its implementation.  
  </rdfs:Property>
Appendix B. Serving WSML specifications on the Web

When publishing ontologies on the Web, the namespace of the ontology need not be the same as the URI where the ontology file is located. A namespace URI should be a neutral URI instead, which allows changes in the definition or the language (HTML, WSML or even OWL) in which the ontology is defined, without changing the namespace and thus keeping compatibility. An example neutral URI is http://example.com/ontologies/e-shop which can be contrasted to the following implementation-specific URI http://dev.example.com/ontologies/e-shop/v3.14/e-shop.wsml. Redirection or internal URI rewriting can be used to serve the WSML document from the neutral URI.

The issue of what should be available at the namespace URI is not yet resolved in an agreed standard way, see W3C Technical Architecture Group issue namespaceDocument-8. However, it seems useful to make different content available for different agents, for example a Web browser going to an ontology URI might get the textual HTML page, whereas an automated agent retrieve the WSML document from the same URI. The separate documents (the HTML description and the WSML ontology) can still have their own separate URIs, but they should be redundantly available at the namespace URI as well.

In [HTTPExamples] the W3C Semantic Web Best Practices and Deployment Working Group drafts a number of techniques for configuring the popular Apache Web server for serving different content to different user agents, depending on the agents' capabilities represented with the set of accepted media types. These tips can also be followed when serving together the HTM description and the WSML ontology in any of the provided syntaxes (human-readable or WSML/XML) or in the RDF form presented in this document. The media types in the tips must be changed as follows: files in the human-readable syntax of WSML should be served under the media type application/x-wsml and WSML/XML files should have the media type application/x-wsml+xml, as specified in [WSML]. Finally files in the RDF form presented in this document should be served using the media type application/rdf+xml (provided RDF/XML is used to serialize the RDF graph).

The media type for the RDF form is the same as the one used by the direct WSML/RDF syntax from [WSML], therefore files in both forms cannot be served from the same URI, as there is no way to distinguish what the requesting agent wants. In fact, the use of WSML/RDF (the direct and complete translation of WSML into an RDF syntax) is discouraged — an agent who wants an RDF version will probably work better with the RDF form presented in this document.

References


Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperonto; by Science Foundation Ireland under the DERI-Lion project; by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects RW² and TSC.
The editors would like to thank all the members of the WSML working group for their advice and input into this document. We would especially like to thank Douglas Foxvog and Eyal Oren for their work on deliverables superseded by this deliverable.

Footnotes

[1] The part-whole ontology we use has a distinction between hasPart and hasPart_directly. hasPart is a transitive property, and thus we would not be able to infer which part is directly contained in which whole. hasPart_directly is a subproperty of hasPart; thus, for each pair in the hasPart_directly relation is also in the hasPart relation, but it allows us to distinguish direct part-whole containment.