WSML Deliverable

D37 v0.2
WSML/OWL Mapping

WSML Working Draft – May 13, 2008

Editor:
   Nathalie Steinmetz
Authors:
   Nathalie Steinmetz
   Jos de Bruijn
   Andreas Frankl

This version:
   `http://www.wsmo.org/TR/d37/v0.2/20080513/`
Latest version:
   `http://www.wsmo.org/TR/d37/v0.2/`
Previous version:
   `http://www.wsmo.org/TR/d37/v0.1/20080125/`

Abstract

This document presents the mapping from WSML-DL to OWL
DL, as well as the mapping from OWL DL to WSML-DL. The
mapping is applicable to ontologies and logical expressions only,
not to Web services, Goals or Mediators. Within this document
we currently map the WSML surface syntax (as presented in
D16.1, the WSML Language Reference, to OWL and not the
WSML abstract syntax (as presented in D16.3, the WSML Ab-
stract Syntax).

Contents

1   Introduction

Within the following chapters we present a WSML/OWL mapping in two diretions: from WSML-DL to OWL DL and from OWL DL to WSML-DL. The main purpose of this document is to help people who are common with OWL[2] to understand the relationship of OWL to WSML.

The mapping presented here is applicable to ontologies and logical expressions only, not to Web services, Goals or Mediators. Please note that logical expressions might occur in ontologies, as well as goal and web service capability descriptions. For a mapping of non-ontology constructs except logical expressions the RDF Syntax for WSML has to be used.

This deliverable contains a mapping from WSML-DL to OWL DL. Other WSML variants will not be mapped directly to OWL, since their semantics are not compatible. If a mapping is desired first such an ontology has to be reduced to either WSML-DL or WSML-Core.

Currently this document maps the WSML surface syntax from D16.1[5], WSML Language Reference to OWL abstract syntax [7]. We do not present a mapping from the WSML abstract syntax as presented in D16.3[3], WSML Abstract Syntax and Semantics.

2   Mapping WSML-DL to OWL DL

In the following we will provide both the preprocessing steps that we need to do before translating WSML-DL to OWL DL and the actual mapping table.

2.1   Pre-processing Steps

In order to simplify the translation from WSML-DL to OWL DL, we try to build simpler expressions. Therefore we perform the following pre-processing steps:

- Replace all compact URIs with full IRIs. For example, "`dc\#title`" is substituted by "`http://purl.org/dc/elements/1.1\#title`".

- Replace all unnumbered anonymous identifiers by `http://www.wsmo.org/reasoner/anonymous_<system_time_in_milliseconds>-<random_int>`.

- Rewrite all data term shortcuts ("`string`" := `xsd#string("string")`; `integer` := `xsd#integer(integer)`; `decimal` := `xsd#decimal(decimal)`).

- Replace idlists with single ids (in the case of **ofType**, **impliesType**, **hasValue**, **memberOf** and **subConceptOf**).

  For example, "$Mary[hasChild$ **hasValue** $\{Bob, Anna\}]$" is replaced by "$Mary[hasChild$ **hasValue** $\{Bob\}]$" and "$Mary[hasChild$ **hasValue** $\{Anna\}]$".

- Replace the remaining conceptual syntax, consisting of concepts and instances, by logical expressions. The transformation is done according to [4, Table 8.1].

- Within logical expressions, the following pre-processing steps are applied:

  - Equivalence and right implication is replaced by left implication.
    For example, "$lexpr$ **implies** $rexpr$." is replaced by "$rexpr$ **impliedBy** $lexpr$." and "$lexpr$ **equivalent** $rexpr$" is substituted by "$lexpr$ **impliedBy** $rexpr$ **and** $rexpr$ **impliedBy** $lexpr$.".

  - Within left implications, left-side conjunctions and right-side disjunctions are eliminated by splitting the expressions. This normalizing step is only applied if the operands of the respective conjunctions or disjunctions do not contain any dependencies from each other.
    For example, the left implication "A **and** B **impliedBy** C" is replaced by "(A **impliedBy** C) **and** (B **impliedBy** C) and the left implication "A **impliedBy** B **or** C" is replaced by "(A **impliedBy** B) **and** (A **impliedBy** C)".

  - If the expression contains unquantified variables, we explicitly universally quantify the variables outside the formula (the WSML language

reference [5, Section 2.8] says: "If a variable is not quantified inside a formula, the variable is implicitly universally quantified outside the formula"). This quantification is applied to all unquantified variables except the root variable. Next we check, step by step, in which molecule inside of the expression a specified variable is contained and push the quantifiers as far as possible inside of the expression.

For example, the expression "$?x$ **memberOf** A **impliedBy** $?x[att1$ **impliesType** $?y]$ **and** $?y[att2$ **impliesType** $?z]$ **and** $?z$ **memberOf** B" is replaced by "$?x$ **memberOf** A **impliedBy exists** $?y(?x[att1$ **impliesType** $?y]$ **and exists** $?z(?y[att2$ **impliesType** $?z]$ **and** $?z$ **memberOf** $B))$".

2.2   Mapping Table

Table 2.1 and Table **??** contain the mapping between the WSML-DL surface syntax and the OWL DL abstract syntax. The mapping is described through the mapping function τ. In Table **??** we will introduce the functions ε and α, which are needed for the correct translation of WSML-DL descriptions.

Boldfaced words in the tables refer to keywords in the WSML language. X and Y are meta-variables and are replaced with actual identifiers and variables during the translation, while DES stands for WSML-DL descriptions. IRIs are abbreviated by compact URIs. The prefix 'wsml' stands for '`http://wsmo.org/wsml/wsml-syntax#`', 'xsd' stands for '`http://www.w3.org/2001/XMLSchema#`', 'owl' stands for '`http://www.w3.org/2002/07/owl#`' and 'wsml2owl' stands for '`http://wsml2owlTransformation#`'.

WSML-DL	OWL-DL	Remarks
Mapping for ontologies		
$\tau($**ontology** id $\quad header_1$ $\quad ...$ $\quad header_n$ $\quad ontology\_element_1$ $\quad ...$ $\quad ontology\_element_n$ $)$	**Ontology**(id $\quad \tau($header$_1)$ $\quad ...$ $\quad \tau($header$_n)$ $\quad \tau($ontology_element$_1)$ $\quad ...$ $\quad \tau($ontology_element$_n)$ $)$	A header can contain *annotations*, *usesMediator* and *importsOntology* statements. An *ontology_element* can be a *concept*, an *instance* or an *axiom*.
$\tau($**annotations** $\quad$ id$_1$ **hasValue** value$_1$ $\quad ...$ $\quad$ id$_n$ **hasValue** value$_n$ **endAnnotations**$)$	**Annotation**(id$_1$ $\tau($**value**$_1))$ $...$ **Annotation**(id$_n$ $\tau($**value**$_n))$	For annotations on the ontology level "Annotation" instead of "annotation" has to be written.
$\tau($**importsOntology** id$)$	**Annotation**(owl#import id)	"id" stands for the identifier of a WSML file.
$\tau($**usesMediator** id$)$	**Annotation**(wsml#usesMediator id)	As OWL doesn't have the concept of a mediator, a wsml#usesMediator annotation is used.

$\tau(\text{datatype\_id}(x_1,...,x_n))$	$\tau_{serializer}(\text{datatype\_id}(x_1,...,x_n))\^\^$ $\tau_{datatypes}(\text{datatype\_id})$	The function $\tau_{serializer}$, which serializes the WSML representation of a data value to a string representation that can be readiliy used in OWL, is defined in Table 2.3. $\tau_{datatypes}$ maps eventual WSML datatypes to XML Schema datatypes, according to [5, Table B.1].
$\tau(\text{id})$	id	In WSML an IRI is enclosed by _" and ", which are omitted in OWL abstract syntax.
Mapping for axioms		
$\tau(\textbf{axiom}\ \text{id log\_expr})$	$\tau(\text{log\_expr})$	A log_expr can be a logical expression like the following.
$\tau(\text{id}[\text{att\_id}\ \textbf{impliesType}$ $\text{range\_id}\,])$	**Class**(id **restriction** (att_id **allValuesFrom** range_id)) **ObjectProperty**(att_id)	
$\tau(\text{id}[\text{att\_id}\ \textbf{ofType}\ \text{range\_id}])$	**Class**(id **restriction** (att_id **allValuesFrom** range_id)) **DatatypeProperty**(att_id)	
$\tau(\text{id1}\ \textbf{subConceptOf}\ \text{id2})$	**Class**(id1 **partial** id2)	
$\tau(\text{id}[\text{att\_id}\ \textbf{hasValue}\ \text{value}])$	**Individual** (id **value**(att_id $\tau(\textbf{value})$))	
$\tau(\text{id1}\ \textbf{memberOf}\ \text{id2})$	**Individual** (id1 **type**(id2))	
$\tau(?x[\text{att\_id2}\ \textbf{hasValue}\ ?y]$ **impliedBy** ?x[att_id **hasValue** ?y])	SubProperty(att_id att_id2)	A left implication with attribute values as left-hand and right-hand sides is mapped to an OWL subProperty.
$\tau(?x[\text{att\_id}\ \textbf{hasValue}\ ?y]$ **impliedBy** ?x[att_id **hasValue** ?z] **and** ?z[att_id **hasValue** ?y])	**ObjectProperty**(att_id **Transitive**)	Transitive Property
$\tau(?x[\text{att\_id}\ \textbf{hasValue}\ ?y]$ **impliedBy** ?y[att_id **hasValue** ?x])	**ObjectProperty**(att_id **Symmetric**)	Symmetric Property
$\tau(?x[\text{att\_id}\ \textbf{hasValue}\ ?y]$ **impliedBy** ?y[att_id2 **hasValue** ?x])	**ObjectProperty**(att_id **inverseOf**(att_id2))	Inverse Property

τ(?x **memberOf** concept_id2 **impliedBy** ?x **memberOf** concept_id)	**Class**(concept_id **partial** concept_id2)	Equivalence of concepts can be expressed as follows, with A and B being membership molecules: "A equivalent B" :=: "A impliedBy B and B impliedBy A"
τ(?x **memberOf** concept_id **impliedBy** ?x[att_id **hasValue** ?y])	**ObjectProperty**(att_id **domain**(concept_id))	
τ(?y **memberOf** concept_id **impliedBy** ?x[att_id **hasValue** ?y])	**ObjectProperty**(att_id **range**(concept_id))	
τ(DES1 **impliedBy** DES2)	α(DES1) α(DES2) subClassOf(ε(DES2) ε(DES1))	"A impliedBy B" can be written as "subClassOf(B,A)".
τ()		If τ is applied for a non-occurring production no translation has to be made.

Table 2.1: Mapping WSML-DL ontologies and axioms to OWL DL

Table 2.2 introduces us to the mapping of WSML-DL descriptions. Those are used inside of axioms, as you can see in Table **??**. We need to translate the descriptions to concept expressions and to axioms. Concept expressions are used within other expressions, while the axioms are added as such to the OWL ontology. The mapping τ is translated into a tuple of concept expressions and axioms as follows: $\tau(DES) = (\varepsilon(DES), \alpha(DES))$.

The table also indicates a mapping for Qualified Cardinality Restrictions (QCRs). In WSML-DL the QCRs are represented by a combination of WSML-DL descriptions. The mapping to OWL DL is done according the workaround with OWL subproperties, described in Section **??**.

Example of usage of $\varepsilon(DES)$ and $\alpha(DES)$.

α is used to translate WSML-DL descriptions to axioms that are directly inserted into OWL DL, while ε translates the descriptions to concept expressions that are nested into OWL expressions (as e.g. restrictions).

The following very easy example illustrates this:

DES1 impliedBy DES2 is translated to:
α(DES1),
α(DES2),
subClassOf(ε(DES2) ε(DES1))

In a concrete example this can look as following:
DES1 = ?x memberOf Human
DES2 = ?x memberOf Woman

Thus we have the following logical expression:
?x memberOf Human impliedBy ?x memberOf Woman

α translates to the OWL Class description:
Class(Woman)
Class(Human)

ε translates to the expressions that are used within the OWL subClassOf expression:
subClassOf(Woman Human)

Instead of Woman we could also have a more complex expression as DES2, as e.g. an intersection of two classes:
DES2 = ?x memberOf Woman and ?x memberOf Child

α for DES2:
Class(Woman)
Class(Child)

ε for DES2:
intersectionOf(Woman Child)

ε translates to the whole expression now to:
subClassOf(intersectionOf(Woman Child) Human)

WSML-DL	OWL-DL - concept expression ε	OWL-DL - axiom α	Remarks
Mapping for descriptions (DES)			
$\tau($?x **memberOf** id$)$	id	**Class**(id)	Description. Membership molecule
$\tau($?x[att_id **hasValue** ?y]$)$	**restriction** (att_id **allValuesFrom**(owl:Thing))	**ObjectProperty**(att_id)	Description. Attribute value molecule with ?y being an unbound variable withing the outer logical expression.
$\tau($?x[att_id **hasValue** ?y] **and** ?y **memberOf** id$)$	**restriction** (att_id **someValuesFrom**(id))	**Class**(id) **ObjectProperty**(att_id)	Description. Attribute value molecule with ?y being a bound variable.
$\tau($DES$_1$ **and** ... **and** DES$_n$$)$	**intersectionOf**($\varepsilon($DES$_1$),...,$\varepsilon($DES$_n$))	$\alpha($DES$_1$) ... $\alpha($DES$_n$)	Description. Conjunction
$\tau($DES$_1$ **or** ... **or** DES$_n$$)$	**unionOf**($\varepsilon($DES$_1$),...,$\varepsilon($DES$_n$))	$\alpha($DES$_1$) ... $\alpha($DES$_n$)	Description. Disjunction

$\tau(\textbf{neg}\ \text{DES})$	**complementOf**$(\varepsilon(\text{DES}))$	$\alpha(\text{DES})$	Description. Negation
$\tau(\textbf{exists}\ ?x\ (?y[\ \text{att\_id}$ **hasValue** $?x]$ **and** DES$))$	**restriction** (att_id **someValuesFrom**($\varepsilon(\text{DES})))$	$\alpha(\text{DES})$ **ObjectProperty**(att_id)	Description. Existential quantification
$\tau(\textbf{exists}\ ?x\ (?x[\ \text{att\_id}$ **hasValue** $?y]$ **and** DES$))$	**restriction** (inverseOf(att_id) **someValuesFrom**($\varepsilon(\text{DES})))$	$\alpha(\text{DES})$ **ObjectProperty**(att_id)	Description. Existential quantification with inverse role.
$\tau(\textbf{forall}\ ?x\ (\text{DES}$ **impliedBy** $?y[$ att_id **hasValue** $?x]))$	**restriction** (att_id **allValuesFrom** ($\varepsilon(\text{DES})))$	$\alpha(\text{DES})$ **ObjectProperty**(att_id)	Description. Universal quantification
$\tau(\textbf{forall}\ ?x\ (\text{DES}$ **impliedBy** $?x[$ att_id **hasValue** $?y]))$	**restriction** (inverseOf(att_id) **allValuesFrom** ($\varepsilon(\text{DES})))$	$\alpha(\text{DES})$ **ObjectProperty**(att_id)	Description. Universal quantification with inverse role.
$\tau(\textbf{exists}\ ?y_1,...,?y_n($ $?x[$ att_id **hasValue** $?y_1]$ **and** ... **and** $?x[$ att_id **hasValue** $?y_n]$ **and** DES **and** **neg**$(?y_1 :=: ?y_2)$ **and** ... **and** **neg**$(?y_{n-1} :=: ?y_n)$ $))$	**restriction** (att_id ' **minCardinality**(n))	$\alpha(\text{DES})$ **ObjectProperty**(att_id) **ObjectProperty**(att_id ' **range**$(\varepsilon(\text{DES})))$ **SubPropertyOf**(att_id' att_id) **Annotation**(wsml2owl: QCR hasValue att_id)	Description. (Qualified) minCardinality restriction.
$\tau(\textbf{forall}\ ?y_1,...,?y_{n+1}$ ($?y_1 :=: ?y_2$ **or** ... **or** $?y_n :=: ?y_{n+1}$ **impliedBy** $?x[$ att_id **hasValue** $?y_1]$ **and** ... **and** $?x[$ att_id **hasValue** $?y_{n+1}]$ **and** DES)	**restriction** (att_id ' **maxCardinality**(n))	$\alpha(\text{DES})$ **ObjectProperty**(att_id) **ObjectProperty**(att_id ' **range**$(\varepsilon(\text{DES})))$ **SubPropertyOf**(att_id' att_id) **Annotation**(wsml2owl: QCR hasValue att_id)	Description. (Qualified) maxCardinality restriction.

Table 2.2: Mapping WSML-DL descriptions to OWL DL

Table 2.3 shows the mapping from WSML data values to strings that can be used directly in OWL as lexical representation, followed by a URI reference to the XML Schema[1] datatype, as it's done with RDF typed literals[6] : lexicalFormRIreference. The mapping is defined through the function $\tau_{serializer}$.

WSML data value	String — OWL	Example
Mapping for data values		
$\tau_{serializer}$(xsd#string("any−character∗"))	'' any−character∗''	xsd#string("Mary Jones") is mapped to "Mary Jones"ˆˆ<xsd#string>
$\tau_{serializer}$(xsd#decimal('−' ?numeric+.numeric+))	'''−'? numeric+.numeric+''	xsd#decimal(-1.5) is mapped to "-1.5"ˆˆ<xsd#decimal>
$\tau_{serializer}$(xsd#integer('−' ?numeric+))	'''−'? numeric+''	xsd#integer(31) is mapped to "31"ˆˆ<xsd#integer>
$\tau_{serializer}$(xsd#float("'−' ?numeric+?.numeric+?e| E'−'?numeric+"))	"'−'?numeric+?.numeric+?e|E '−'?numeric+''	xsd#float("-60.5e-3") is mapped to "-0.0605"ˆˆ<xsd#float>
$\tau_{serializer}$(xsd#double("'−' ?numeric+?.numeric+?e| E'−'?numeric+"))	"'−'?numeric+?.numeric+?e|E '−'?numeric+''	xsd#double("58.5E-5") is mapped to "5.85E-4"ˆˆ<xsd#double>
$\tau_{serializer}$(xsd#boolean("true −or−false"))	'' true−or−false''	xsd#boolean("true") is mapped to "true"ˆˆ<xsd#boolean>
$\tau_{serializer}$(xsd#duration(year, month, day, hour, minute, second))	'' P1Y2M3DT10H30M20S''	xsd#duration(1, 2, 3, 5, 20, 10) is mapped to "P1Y2M3DT5H20M10S" ˆˆ<xsd#duration>
$\tau_{serializer}$(xsd#dateTime(year, month, day, hour, minute, second, timezone −hour, timezone− minute))	'' year−month−dayThour: minute:second−timezone −hour:timezone−minute''	xsd#dateTime(1977, 02, 07, 5, 20, 10, 12, 30) is mapped to "1977-02-07T5:20:10-12:30"ˆˆ<xsd#dateTime>
$\tau_{serializer}$(xsd#dateTime(year, month, day, hour, minute, second))	'' year−month−dayThour: minute:second''	xsd#dateTime(1977, 02, 07, 5, 20, 10) is mapped to "1977-02-07T5 :20:10"ˆˆ<xsd#dateTime>
$\tau_{serializer}$(xsd#time(hour, minute, second, timezone −hour, timezone− minute))	'' hour:minute:second−timezone −hour:timezone−minute''	xsd#time(5, 20, 10, 12, 30) is mapped to "5:20:10-12:30"ˆˆ<xsd#time>
$\tau_{serializer}$(xsd#time(hour, minute, second))	'' hour:minute:second ''	xsd#time(5, 20, 10) is mapped to "5:20:10"ˆˆ<xsd#time>
$\tau_{serializer}$(xsd#date(year, month, day, timezone− hour, timezone−minute))	'' year−month−day−timezone −hour:timezone−minute''	xsd#date(1967, 08, 16, 12, 30) is mapped to "1967-08-16-12:30"ˆˆ<xsd#date>
$\tau_{serializer}$(xsd#date(year, month, day))	'' year−month−day''	xsd#date(1967, 08, 16) is mapped to "1967-08-16"ˆˆ<xsd#date>

$\tau_{serializer}(\text{xsd\#gyearmonth}(\text{year, month}))$	''year−month''	xsd#gyearmonth(1977, 02) is mapped to "1977-02"^^<xsd#gyearmonth>
$\tau_{serializer}(\text{xsd\#gyear}(\text{year}))$	''year''	xsd#gyear(1977) is mapped to "1977"^^<xsd#gyear>
$\tau_{serializer}(\text{xsd\#gmonthday}(\text{month, day}))$	''month−day''	xsd#gmonthday(12, 06) is mapped to "12-06"^^<xsd#gmonthday>
$\tau_{serializer}(\text{xsd\#gday}(\text{day}))$	''day''	xsd#gday(24) is mapped to "24"^^<xsd#gday>
$\tau_{serializer}(\text{xsd\#gmonth}(\text{month}))$	''month''	xsd#gmonth(10) is mapped to "10"^^<xsd#gmonth>
$\tau_{serializer}(\text{xsd\#hexbinary}(\text{hexadecimal−encoding}))$	''hexadecimal−encoding''	xsd#hexbinary("0FB7") is mapped to "0FB7"^^<xsd#hexbinary>
$\tau_{serializer}(\text{xsd\#base64binary}(\text{hexadecimal−encoding}))$	''hexadecimal−encoding''	xsd#base64binary("R01G0DdhNgAPAATYyP") is mapped to "R01G0DdhNgAPAATYyP" ^^<xsd#base64binary>

Table 2.3: Mapping WSML data values to strings

2.3   Qualified Cardinality Restrictions

Cardinality restrictions are used to constrain the number of values of a particular property, irrespective of the value type. If we also want to specify the values to be of a particular type, we need "qualified cardinality restrictions" (QCR).

One famous example for QCRs in the human anatomy use case is mentioned by Alain Rector in [8]: "The normal hand has exactly five fingers of which one is a thumb". The fact that a hand has exactly five fingers is easy to describe via a simple cardinality constraint:

```
Class(NormalHand
     restriction (hasFinger  cardinality (5)))
```

But without QCRs we cannot describe that one of these fingers must be a thumb.

The Web Ontology Working Group has postponed the issue of full representation of QCRs, but has already proposed an OWL representation for them [10] and discussed a possible workaround that Alain Rector described in [8].

A W3C working draft [9] discusses a partial work-around within the OWL standard and a non endorsed extension of OWL, that allows to express QCRs correctly.

2.3.1   Use owl:someValuesFrom.

The construct `owl:someValuesFrom` is equivalent to a QCR with a minimum cardinality of 1. This means that the specified property should have at least one value of this type.

E.g., an Italian dinner should contain at least one antipasto:

```
Class( ItalianDinner   partial
    restriction (hasCourse someValuesFrom(AntiPasto)))
```

2.3.2   Workaround using rdfs:subPropertyOf.

The idea of the workaround is to express QCRs by having extra subproperties for each component whose number is to be specified. So we introduce a subProperty for the main property and then introduce an unqualified cardinality restriction on that subProperty.

Using this workaround, the "normal hand" example from human anatomy use case, mentioned before, would be represented by:

```
Class(Finger)
Class(Thumb partial Finger)
ObjectProperty(hasFinger range(Finger))
ObjectProperty(hasThumb range(Thumb))
SubPropertyOf(hasThumb hasFinger)
Class(NormalHand partial
     restriction (hasFinger  cardinality (5))
     restriction (hasThumb cardinality(1)))
```

Unfortunately   this   workaround   is   not   complete   (see [9]) and, although it may be ok for simple cases, it can lead to vast and cumbersome numbers of subproperties.

This workaround using the RDFS subPropertyOf statement is the one that we use for mapping WSML-DL to OWL DL in Section **??**.

2.3.3   Use a non-endorsed OWL extension.

Qualified Restrictions resemble regular restrictions but contain one extra triple in the RDF representation and an extra argument in the abstract syntax. This statement would be `valuesFrom`, which points to the value type being restricted.

Using this extension, the normal hand example can be described by:

```
Class(NormalHand partial
     restriction (hasFinger  cardinality (5))
     qualifiedRestriction (hasFinger valuesFrom (Thumb) cardinality(1)))
```

This representation is legal in OWL Full, but the semantics will only be treated correctly by parsers and classifiers that support QCRs.

3   Mapping OWL DL to WSML-DL

In the following we will provide the mapping table that allows us to translate OWL DL to WSML-DL.

OWL DL descriptions can mostly be used in a nested way. In the following we will call these descriptions "*ELEMENTs*". An *ELEMENT* will either be of type class or an anonymous class, i.e., a property restriction, an intersection, a union or a complement description. The translation to WSML works in an analogous way: an *ELEMENT* can be either a concept or a WSML-DL description that can be used in a nested way.

3.1   Mapping Table

Table 3.1 contains the mapping between the OWL DL abstract syntax and the WSML-DL surface syntax. The mapping is described through the mapping function τ.

Boldfaced words in the table refer to keywords in the WSML language. X and Y are meta-variables that are created in WSML from actual identifiers during the translation. IRIs are abbreviated by compact URIs. The prefix 'wsml' stands for '`http://wsmo.org/wsml/wsml-syntax#`', 'owl' stands for '`http://www.w3.org/2002/07/owl#`', 'xsd' stands for '`http://www.w3.org/2001/XMLSchema#`' and 'owl2wsml' stands for '`http://owl2wsmlTransformation#`'.

OWL-DL	WSML-DL	Remarks
Namespace		
Namespace (id = nid)	**namespace** id _"nid"	
Ontology Header		
Ontology (oid)	**ontology** _"oid"	
Annotation (annotID text)	**annotations** annotID **hasValue** "text" **endAnnotations**	
Annotation (owl:imports oid)	**importsOntology** _"oid" **annotations** owl2wsml#import **hasValue** oid **endAnnotations**	In OWL RDF/XML syntax the import statement looks like the following: <owl:imports rdf:resource="oid" \>.
Annotation (owl2wsml#import oid)	**importsOntology** _"oid"	If an ontology was imported initially in a wsml file. See Section 4.2 for details.

Classes		
Class (id **partial**)	**concept** id	
Class (id **partial** Element$_1$ ... Element$_n$)	**concept** id **subConceptOf** {Element$_1$ ... Element$_n$ }	
Class (id **complete** Element$_1$ ... Element$_n$)	?x **memberOf** id **equivalent** Element$_1$ **equivalent** ... **equivalent** Element$_n$	equivalence - complete statement
DisjointClasses (id$_1$ ... id$_n$)	?x **memberOf** id$_1$ **implies neg**(?x **memberOf** id$_2$) **and** ... **and neg**(?x **memberOf** id$_n$)	disjoint statement

PropertyRestriction		
restriction (prop_id **allValuesFrom** (id1)) **ObjectProperty**(prop_id **range**(id2))	**concept** id1 prop_id **impliesType** id2 **annotations** annotID **hasValue** owl2wsml#simpleClass **endAnnotations**	allValuesFrom
restriction (prop_id **allValuesFrom** (ELEMENT))	**forall** ?x(?y[prop_id **hasValue** ?x] **impliesType** ELEMENT)	allValuesFrom
restriction (prop_id **someValuesFrom** (ELEMENT))	**exists** ?x(?y[prop_id **hasValue** ?x] **and** ELEMENT)	someValuesFrom - minimal cardinality of 1
restriction (prop_id **value** (ELEMENT))	?x[prop_id **hasValue** ELEMENT]	value
restriction (prop_id **minCardinality** (n)) **ObjectProperty** (prop_id **range**(id))	**exists** ?x$_1$,..., ?x$_n$ (?x[prop_id **hasValue** ?x$_1$] **and** ... **and** ?x[prop_id **hasValue** ?x$_n$] **and** ?x$_1$ **memberOf** id **and** ... **and** ?x$_n$ **memberOf** id **and neg**(?x$_1$ = ?x$_2$) **and** ... **and neg**(?x$_{n-1}$ = ?x$_n$))	minCardinality
restriction (prop_id **maxCardinality** (n)) **ObjectProperty** (prop_id **range**(id))	**forall** ?x$_1$,..., ?x$_n$ (?x[prop_id **hasValue** ?x$_1$] **and** ... **and** ?x[prop_id **hasValue** ?x$_n$] **implies** (?x$_1$ = ?x$_2$) **or** ... **or** (?x$_{n-1}$ = ?x$_n$) **and** ?x$_1$ **memberOf** id **and** ... **and** ?x$_n$ **memberOf** id)	maxCardinality

restriction (prop_id **cardinality** (n)) **ObjectProperty** (prop_id **range**(id))	**exists** $?x_1,...,?x_n$ (?x[prop_id **hasValue** $?x_1$] **and** ... **and** ?x[prop_id **hasValue** $?x_n$] **and** $?x_1$ **memberOf** id **and** ... **and** $?x_n$ **memberOf** id **and** **neg**($?x_1 = ?x_2$) **and** ... **and neg**($?x_{n-1} = ?x_n$)) **and forall** $?x_1,...,?x_n$(?x [prop_id **hasValue** $?x_1$] **and** ... **and** ?x[prop_id **hasValue** $?x_n$] **implies** ($?x_1 = ?x_2$) **or** ... **or** ($?x_{n-1} = ?x_n$) **and** $?x_1$ **memberOf** id **and** ... **and** $?x_n$ **memberOf** id)	cardinality
intersectionOf (ELEMENT$_1$ ... ELEMENT$_n$)	ELEMENT$_1$ **and** ... **and** ELEMENT$_n$	intersection
unionOf (ELEMENT$_1$ ... ELEMENT$_n$)	ELEMENT$_1$ **or** ... **or** ELEMENT$_n$	union
complementOf (ELEMENT)	**neg**(ELEMENT)	complement

Properties

ObjectProperty(prop_id **domain**(id1) **range**(id2))	**concept** id1 prop_id **impliesType** id2	ObjectProperty

DatatypeProperty(prop_id **domain**(id) **range**(xsd:Datatype))	**concept** id
	  prop_id **ofType**
	    xsd#Datatype
		DatatypeProperty. We do not explain the mapping of datatypes in detail as this mapping is straightforward: both WSML and OWL use XML Schema[1] Datatypes.

ObjectProperty(prop_id1 **super**(prop_id2) **domain**(id1) **range**(id2))	**concept** id1
	  prop_id1 **impliesType** id2
	
	axiom definedBy
	  ?x[prop_id2 **hasValue** ?y]
	    impliedBy ?x[prop_id1
	    hasValue ?y].
		ObjectProperty with super property

SubPropertyOf(prop_id1 prop_id2)	?x[prop_id2 **hasValue** ?y]
	  impliedBy ?x[prop_id1
	  hasValue ?y].
		subproperty statement

ObjectProperty(prop_id **Transitive domain**(id1) **range**(id2))	?x[prop_id **hasValue** ?z]
	  impliedBy ?x[prop_id
	  hasValue ?y] **and** ?y[
	  prop_id **hasValue** ?z].
		transitive statement

ObjectProperty(prop_id **Symmetric domain**(id1) **range**(id2))	?x[prop_id **hasValue** ?y]
	  impliedBy ?y[prop_id
	  hasValue ?x].
		symmetric statement

ObjectProperty(prop_id **Functional**)	**forall** ?x,?y,?z(?x[prop_id **hasValue** ?y] **and** ?x[prop_id **hasValue** ?z] **implies** (?y = ?z) **and** ELEMENT).	functional statement - $maxCardinality = 1$
ObjectProperty(prop_id1 **inverseOf**(prop_id2))	?x[prop_id1 **hasValue** ?y] **impliedBy** ?y[prop_id2 **hasValue** ?x].	inverse statement
ObjectProperty(prop_id1 **InverseFunctional**)	**forall** ?x,?y,?z(?y[prop_id **hasValue** ?x] **and** ?z[prop_id **hasValue** ?x] **implies** (?y = ?z) **and** ELEMENT).	inverse functional statement
EquivalentProperties (prop_id$_1$ ... prop_id$_n$)	?x[prop_id$_1$ **hasValue** ?y] **equivalent** ... **equivalent** ?x[prop_id$_n$ **hasValue** ?y].	equivalent property statement
Individuals		
Individual (id)	**instance** id	
Individual (id **type** (class_id))	**instance** id **memberOf** class_id	
Individual (id **type** (class_id) **value** (prop_id$_1$ **value**) ... (prop_id$_n$ **value**))	**instance** id **memberOf** class_id prop_id$_1$ **hasValue** value ... prop_id$_n$ **hasValue** value	
SameIndividual(id$_1$ ... id$_n$)	id$_1$ = id$_2$ **and** ... **and** id$_1$ = id$_n$.	same individual
DifferentIndividuals (id$_1$ ... id$_n$)	id$_1$ = **neg**(id$_2$) **and** ... **and** id$_1$ = **neg**(id$_n$).	different individuals
Enumerated Classes		
EnumeratedClass(class_id ind$_1$ ... ind$_n$)	**annotations** owl2wsml#enumeratedClass/ class_id/1 **hasValue** ind$_1$ ... owl2wsml#enumeratedClass/ class_id/n **hasValue** ind$_n$ **endAnnotations**	enumerated classes do not exist in WSML - we mark in an annotation to the ontology that we removed these from the resulting WSML ontology

Table 3.1: Mapping OWL-DL ontologies to WSML-DL

4   Roundtripping Issues

In the following we will first outline some problems that appear while trying
to translate WSML-DL to OWL DL and the other way around.

4.1   Mapping Constraints

The mappings (from WSML-DL to OWL DL and the other way around) that
we presented in 2 and Chapter 3 is not complete. This is due to the fact that
both WSML-DL and OWL DL are based on different Description Logic dialects:
WSML-DL is based on $SHIQ(D)$, while OWL DL is based on $SHOIN(D)$.
[11] explains in more detail the similarities and differences between these two
dialects.

4.1.1   Qualified Cardinality Restrictions

Whereas qualified cardinality restrictions can be expressed in WSML-DL,
they cannot be expressed in OWL DL. There exist nevertheless some possible
workarounds for this in OWL. For a more elaborated discussion on these, see
Section 2.3. We have mapped the qualified cardinality restrictions from WSML-
DL to OWL DL using the workaround with the subPropertyOf statements,
although this is, as described in 2.3 not complete.

4.1.2   Enumerated Classes

WSML-DL does not support enumerated classes, as does OWL DL. This
allows one to write enumerations of individuals in OWL using the `oneOf` state-
ment:

oneOf(Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunnday)

4.2   Handling Roundtripping

We have introduced two namespaces that allows us to make annotations in
the ontology that is being newly created, be it a WSML or an OWL ontoloy:

- http://owl2wsmlTransformation

- http://wsml2owlTransformation

We use these namespaces in the mapping to indicate cases where we cannot
translate ontology elements in a straightforward way, as e.g. due to the mapping
constraints mentioned before in Section 4.1.

Acknowledgements

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

Bibliography

[1] Xml schema part 2: Datatypes. W3c recommendation, World Wide Web Consortium, 2001. Available at http://www.w3.org/TR/xmlschema-2/.

[2] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Owl web ontology language reference. Technical report, 2004. Available from: http://www.w3.org/TR/owl-ref/.

[3] Jos de Bruijn. Wsml abstract syntax and semantics. WSML Working Draft D16.3v0.3, WSML, 2008. Available from: http://www.wsmo.org/TR/d16/d16.3/v0.3/.

[4] Jos de Bruijn, Holger Lausen, Reto Krummenacher, Axel Polleres, Livia Predoiu, Michael Kifer, and Dieter Fensel. The web service modeling language WSML. WSML Final Draft D16.1v0.21, WSML, 2005. Available from: http://www.wsmo.org/TR/d16/d16.1/v0.21/.

[5] The WSML Working Group. Wsml language reference. WSML Working Draft D16.1v0.3, WSML, 2008. Available from: http://www.wsmo.org/TR/d16/d16.1/v0.3/.

[6] Graham Klyne and Jeremy J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. Recommendation 10 February 2004, W3C, 2004.

[7] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. Owl web ontology language semantics and abstract syntax. Technical report, 2004. Available from http://www.w3.org/TR/owl-semantics/.

[8] Alain Rector. Message to public-webont-comments@w3.org: "case for reinstatement of qualified cardinality restrictions", 2003. Available from: http://lists.w3.org/Archives/Public/public-webont-comments/2003Apr/0040.html.

[9] Alan Rector and Guus Schreiber. Qualified cardinality restrictions (qcrs): Constraining the number of values of a particular type for a property, 2006. W3C Working Draft, available from: http://www.cs.vu.nl/ guus/public/qcr-20060508.html.

[10] Guus Schreiber. Message to public-webont-wg@w3.org: "issue 3.2 qcr: proposal to postpone", 2003. Available from: http://lists.w3.org/Archives/Public/www-webont-wg/2003May/0072.html.

[11] Nathalie Steinmetz. Wsml-dl reasoner, 2006. Available from: http://www.sti-innsbruck.at/fileadmin/documents/thesis/dlreasoner.pdf.