



WSML Deliverable  
D37 v0.1  
WSML/OWL MAPPING

WSML Working Draft – January 25, 2008

**Authors:**

Nathalie Steinmetz  
Jos de Bruijn

**Editors:**

Nathalie Steinmetz

**Reviewers:**

**This version:**

<http://www.wsmo.org/TR/d37/v0.1/20080125/>

**Latest version:**

<http://www.wsmo.org/TR/d37/v0.1/>



# Abstract

This document presents the mapping from WSML-DL to OWL DL. The mapping is applicable to ontologies and logical expressions only. It currently only maps the WSML surface syntax to OWL and not the WSML abstract syntax.



# Contents

<b>1</b>	<b>Mapping to OWL</b>	<b>4</b>
1.1	Mapping WSML-DL to OWL DL . . . . .	4
1.1.1	Pre-processing Steps . . . . .	4
1.1.2	Mapping Table . . . . .	5



# 1 Mapping to OWL

The mapping to OWL presented here is applicable to ontologies and logical expressions only; 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 of WSML-DL to OWL. 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 only maps the WSML surface syntax from D16.1 (<http://www.wsmo.org/TR/d16/d16.1/v0.3/>) to OWL and not the WSML abstract syntax from D16.3 (<http://www.wsmo.org/TR/d16/d16.3/v0.3/>).

## 1.1 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.

### 1.1.1 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 sQNames 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`” := `_string("string"); integer := _integer(integer); decimal := _decimal(decimal)`).
- Replace idlists with single ids (in the case of **ofType**, **impliesType**, **hasValue**, **memberOf**, **subConceptOf** and **subRelationOf**).  
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 [?, 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 rexr.*” is replaced by “*rexpr impliedBy lexr.*” and “*lexpr equivalent rexr*” is substituted by “*lexpr impliedBy rexr and rexr impliedBy lexr.*”
- 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 specification [?, 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))*”.

### 1.1.2 Mapping Table

Table 1.1 and Table 1.2 contain the mapping between the WSML-DL syntax and the OWL DL abstract syntax. The mapping is described through the mapping function  $\tau$ . In Table 1.2 we will introduce the functions  $\varepsilon$  and  $\alpha$ , 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 sQNames. The prefix ‘wsm1’ stands for ‘http://wsmo.org/wsm1/wsm1-syntax#’ and ‘owl’ stands for ‘http://www.w3.org/2002/07/owl#’.

WSML-DL	OWL-DL	Remarks
Mapping for ontologies		
$\tau(\text{ontology id}$ <i>header</i> <sub>1</sub> ... <i>header</i> <sub>n</sub> <i>ontology_element</i> <sub>1</sub> ... <i>ontology_element</i> <sub>n</sub> )	Ontology(id $\tau(\text{header}_1)$ ... $\tau(\text{header}_n)$ $\tau(\text{ontology\_element}_1)$ ... $\tau(\text{ontology\_element}_n)$ )	A header can contain <i>nonFunctionalProperties</i> , <i>usesMediator</i> and <i>importsOntology</i> statements. An <i>ontology_element</i> can be a <i>concept</i> , a <i>relation</i> , an <i>instance</i> , a <i>relation instance</i> or an <i>axiom</i> .



$\tau(\text{annotations } id_1 \text{ hasValue } value_1 \dots id_n \text{ hasValue } value_n \text{ endAnnotations})$	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.
$\tau(\text{importsOntology } id)$	Annotation(owl#import $id$ )	”id” stands for the identifier of a WSML file.
$\tau(\text{usesMediator } id)$	Annotation(wsml#usesMediator $id$ )	As OWL doesn’t have the concept of a mediator, a wsml#usesMediator annotation is used.
$\tau(\text{datatype.id}(x_1, \dots, x_n))$	$\tau_{serializer}(\text{datatype.id}(x_1, \dots, x_n))^{\wedge\wedge}$ $\tau_{datatypes}(\text{datatype.id})$	The function $\tau_{serializer}$ , which serializes the WSML representation of a data value to a string, is defined in Table 1.3. $\tau_{datatypes}$ maps WSML datatypes to XML Schema datatypes, according to [?, Table C.1].
$\tau(id)$	$id$	In WSML an IRI is enclosed by ‘_’ and ‘‘, which are omitted in OWL abstract syntax.
Mapping for axioms		
$\tau(\text{axiom } id \text{ log\_expr nfp})$	$\tau(\text{log\_expr})$	A log_expr can be a logical expression like the following. The axiom does not keep its non functional properties.
$\tau(id[\text{att\_id impliesType range\_id}])$	Class( $id$ restriction ( $att\_id$ allValuesFrom range_id ) ) ObjectProperty( $att\_id$ )	
$\tau(id[\text{att\_id ofType range\_id}])$	Class( $id$ restriction ( $att\_id$ allValuesFrom range_id ) ) DatatypeProperty( $att\_id$ )	
$\tau(id_1 \text{ subConceptOf } id_2)$	Class( $id_1$ partial $id_2$ )	
$\tau(id[\text{att\_id hasValue } value])$	Individual ( $id$ value( $att\_id \tau(value)$ ))	
$\tau(id_1 \text{ memberOf } id_2)$	Individual ( $id_1$ type( $id_2$ ))	
$\tau(?x[\text{att\_id2 hasValue ?y}] \text{ impliedBy } ?x[\text{att\_id hasValue ?y}])$	SubProperty( $att\_id$ $att\_id_2$ )	A left implication with attribute values as left-hand and right-hand sides is mapped to an OWL subProperty.



$\tau(?x[\text{att\_id hasValue ?y}] \text{impliedBy } ?x[\text{att\_id hasValue ?z}] \text{and } ?y[\text{att\_id hasValue ?z}])$	ObjectProperty( att_id Transitive )	Transitive Property
$\tau(?x[\text{att\_id hasValue ?y}] \text{impliedBy } ?y[\text{att\_id hasValue ?x}])$	ObjectProperty( att_id Symmetric )	Symmetric Property
$\tau(?x[\text{att\_id hasValue ?y}] \text{impliedBy } ?y[\text{att\_id2 hasValue ?x}])$	ObjectProperty( att_id inverseOf( att_id2 ) )	Inverse Property
$\tau(?x \text{ memberOf concept\_id2 impliedBy } ?x \text{ memberOf concept\_id})$	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”
$\tau(?x \text{ memberOf concept\_id impliedBy } ?x[\text{att\_id hasValue ?y}])$	ObjectProperty( att_id domain( concept\_id ) )	
$\tau(?y \text{ memberOf concept\_id impliedBy } ?x[\text{att\_id hasValue ?y}])$	ObjectProperty( att_id range( concept\_id ) )	
$\tau(\text{DES1 impliedBy DES2})$	$\alpha(\text{DES1})$ $\alpha(\text{DES2})$ $\text{subClassOf}(\varepsilon(\text{DES2}) \varepsilon(\text{DES1}))$	”A impliedBy B” can be written as “subClassOf(B,A)”.
$\tau()$		If $\tau$ is applied for a non-occurring production no translation has to be made.

Table 1.1: Mapping WSML-DL ontologies and axioms to OWL DL

Table 1.2 introduces us to the mapping of WSML-DL descriptions. Those are used inside of axioms, as you can see in Table 1.1. 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  $\tau$  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 ??.



WSML-DL	OWL-DL - concept expression $\varepsilon$	OWL-DL - axiom $\alpha$	Remarks
Mapping for descriptions (DES)			
$\tau(?x \text{ memberOf } id)$	id	Class(id)	Description. Membership molecule
$\tau(?x[att\_id \text{ hasValue } ?y])$	restriction ( att\_id allValuesFrom( owl:Thing))	ObjectProperty( att\_id )	Description. Attribute value molecule with ?y being an unbound variable withing the outer logical expression.
$\tau(?x[att\_id \text{ hasValue } ?y] \text{ and } ?y \text{ memberOf } id)$	restriction ( att\_id someValuesFrom( id))	Class(id) ObjectProperty( att\_id )	Description. Attribute value molecule with ?y being a bound variable.
$\tau(DES_1 \text{ and } \dots \text{ and } DES_n)$	intersectionOf ( $\varepsilon(DES_1)$ $\dots, \varepsilon(DES_n)$ )	$\alpha(DES_1)$ ... $\alpha(DES_n)$	Description. Conjunction
$\tau(DES_1 \text{ or } \dots \text{ or } DES_n)$	unionOf( $\varepsilon(DES_1), \dots, \varepsilon(DES_n)$ )	$\alpha(DES_1)$ ... $\alpha(DES_n)$	Description. Disjunction
$\tau(\text{neg } DES)$	complementOf( $\varepsilon(DES)$ )	$\alpha(DES)$	Description. Negation
$\tau(\text{exists } ?x (?y[ att\_id \text{ hasValue } ?x] \text{ and } DES))$	restriction ( att\_id someValuesFrom( $\varepsilon(DES)$ ))	$\alpha(DES)$ ObjectProperty( att\_id )	Description. Existential quantification
$\tau(\text{exists } ?x (?x[ att\_id \text{ hasValue } ?y] \text{ and } DES))$	restriction ( inverseOf( att\_id ) someValuesFrom( $\varepsilon(DES)$ ))	$\alpha(DES)$ ObjectProperty( att\_id )	Description. Existential quantification with inverse role.
$\tau(\text{forall } ?x (DES \text{ impliedBy } ?y[ att\_id \text{ hasValue } ?x]))$	restriction ( att\_id allValuesFrom ( $\varepsilon(DES)$ ))	$\alpha(DES)$ ObjectProperty( att\_id )	Description. Universal quantification
$\tau(\text{forall } ?x (DES \text{ impliedBy } ?x[ att\_id \text{ hasValue } ?y]))$	restriction ( inverseOf( att\_id ) allValuesFrom ( $\varepsilon(DES)$ ))	$\alpha(DES)$ ObjectProperty( att\_id )	Description. Universal quantification with inverse role.



$\tau(\text{exists } ?y_1 \dots ?y_n ($ $\quad ?x[\text{att\_id hasValue } ?y_1] \text{ and}$ $\quad \dots \text{ and}$ $\quad ?x[\text{att\_id hasValue } ?y_n] \text{ and}$ $\quad \text{DES and}$ $\quad \text{neg}(?y_1 :=: ?y_2)$ $\quad \text{and}$ $\quad \dots \text{ and}$ $\quad \text{neg}(?y_{n-1} :=: ?y_n)$ $\quad ))$	<code>restriction ( att_id ' minCardinality (n ))</code>	$\alpha(\text{DES})$ <code>ObjectProperty( att_id )</code> <code>ObjectProperty( att_id ' range(<math>\varepsilon(\text{DES}))</math>)</code> <code>SubPropertyOf( att_id ' att_id )</code>	Description. (Qualified) minCardinality restriction.
$\tau(\text{forall } ?y_1 \dots ?y_{n+1}$ $\quad ($ $\quad ?y_1 :=: ?y_2 \text{ or}$ $\quad \dots \text{ or}$ $\quad ?y_n :=: ?y_{n+1}$ $\quad \text{impliedBy}$ $\quad ?x[\text{att\_id hasValue } ?y_1] \text{ and}$ $\quad \dots \text{ and}$ $\quad ?x[\text{att\_id hasValue } ?y_{n+1}]$ $\quad \text{and DES})$	<code>restriction ( att_id ' maxCardinality (n ))</code>	$\alpha(\text{DES})$ <code>ObjectProperty( att_id )</code> <code>ObjectProperty( att_id ' range(<math>\varepsilon(\text{DES}))</math>)</code> <code>SubPropertyOf( att_id ' att_id )</code>	Description. (Qualified) maxCardinality restriction.

Table 1.2: Mapping WSMML-DL descriptions to OWL DL

Table 1.3 shows the mapping from WSMML data values to strings that can be used in OWL. Two data values will not be mapped to strings, but to OWL elements: IRIs and sQNames. The mapping is defined through the function  $\tau_{\text{serializer}}$ .

WSMML data value	String — OWL	Example
Mapping for data values		
$\tau_{\text{serializer}}(\text{string}(\text{"any-character*"}))$	<code>"any-character*"</code>	<code>string("Mary Jones")</code> is mapped to <code>"Mary Jones"</code>
$\tau_{\text{serializer}}(\text{decimal}(\text{"-?numeric+.numeric+"}))$	<code>"-?numeric+.numeric+"</code>	<code>decimal(-1.5)</code> is mapped to <code>"-1.5"</code>
$\tau_{\text{serializer}}(\text{integer}(\text{"-?numeric+"}))$	<code>"-?numeric+"</code>	<code>integer(31)</code> is mapped to <code>"31"</code>
$\tau_{\text{serializer}}(\text{float}(\text{"-?numeric+.numeric+?e E'-?numeric+"}))$	<code>"-?numeric+.numeric+?e E'-?numeric+"</code>	<code>float("-60.5e-3")</code> is mapped to <code>"-0.0605"</code>
$\tau_{\text{serializer}}(\text{double}(\text{"-?numeric+.numeric+?e E'-?numeric+"}))$	<code>"-?numeric+.numeric+?e E'-?numeric+"</code>	<code>double("58.5E-5")</code> is mapped to <code>"5.85E-4"</code>
$\tau_{\text{serializer}}(\text{boolean}(\text{"true-or-false"}))$	<code>"true-or-false"</code>	<code>boolean("true")</code> is mapped to <code>"true"</code>



$\tau_{serializer}(\text{duration}(\text{year}, \text{month}, \text{day}, \text{hour}, \text{minute}, \text{second}))$	"P1Y2M3DT10H30M20S"	<code>_duration(1, 2, 3, 5, 20, 10)</code> is mapped to "P1Y2M3DT5H20M10S"
$\tau_{serializer}(\text{dateTime}(\text{year}, \text{month}, \text{day}, \text{hour}, \text{minute}, \text{second}, \text{timezone} - \text{hour}, \text{timezone} - \text{minute}))$	"year-month-dayThour:minute:second-timezone-hour:timezone-minute"	<code>_dateTime(1977, 02, 07, 5, 20, 10, 12, 30)</code> is mapped to "1977-02-07T5:20:10-12:30"
$\tau_{serializer}(\text{dateTime}(\text{year}, \text{month}, \text{day}, \text{hour}, \text{minute}, \text{second}))$	"year-month-dayThour:minute:second"	<code>_dateTime(1977, 02, 07, 5, 20, 10)</code> is mapped to "1977-02-07T5:20:10"
$\tau_{serializer}(\text{time}(\text{hour}, \text{minute}, \text{second}, \text{timezone} - \text{hour}, \text{timezone} - \text{minute}))$	"hour:minute:second-timezone-hour:timezone-minute"	<code>_time(5, 20, 10, 12, 30)</code> is mapped to "5:20:10-12:30"
$\tau_{serializer}(\text{time}(\text{hour}, \text{minute}, \text{second}))$	"hour:minute:second"	<code>_time(5, 20, 10)</code> is mapped to "5:20:10"
$\tau_{serializer}(\text{date}(\text{year}, \text{month}, \text{day}, \text{timezone} - \text{hour}, \text{timezone} - \text{minute}))$	"year-month-day-timezone-hour:timezone-minute"	<code>_date(1967, 08, 16, 12, 30)</code> is mapped to "1967-08-16-12:30"
$\tau_{serializer}(\text{date}(\text{year}, \text{month}, \text{day}))$	"year-month-day"	<code>_date(1967, 08, 16)</code> is mapped to "1967-08-16"
$\tau_{serializer}(\text{gyearmonth}(\text{year}, \text{month}))$	"year-month"	<code>_gyearmonth(1977, 02)</code> is mapped to "1977-02"
$\tau_{serializer}(\text{gyear}(\text{year}))$	"year"	<code>_gyear(1977)</code> is mapped to "1977"
$\tau_{serializer}(\text{gmonthday}(\text{month}, \text{day}))$	"month-day"	<code>_gmonthday(12, 06)</code> is mapped to "12-06"
$\tau_{serializer}(\text{gday}(\text{day}))$	"day"	<code>_gday(24)</code> is mapped to "24"
$\tau_{serializer}(\text{gmonth}(\text{month}))$	"month"	<code>_gmonth(10)</code> is mapped to "10"
$\tau_{serializer}(\text{hexbinary}(\text{hexadecimal} - \text{encoding}))$	"hexadecimal-encoding"	<code>_hexbinary(0FB7)</code> is mapped to "0FB7"
$\tau_{serializer}(\text{base64binary}(\text{hexadecimal} - \text{encoding}))$	"hexadecimal-encoding"	<code>_base64binary("R01G0DdhNgAPAATyP")</code> is mapped to "R01G0DdhNgAPAATyP"

Table 1.3: Mapping WSML data values to strings



## Acknowledgements

The work is partially funded by the European Commission under the projects ASG, EASAIER, enIRaF, Knowledge Web, Musing, Salero, Seemp, Semantic-GOV, Super, SWING and TripCom; by Science Foundation Ireland under the DERI-Lion Grant No.SFI/02/CE1/I13; by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects Grisino, RW<sup>2</sup>, 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.