

# The Web Service Modeling Language WSML: An Overview

Jos de Bruijn<sup>1</sup>, Holger Lausen<sup>1</sup>, Axel Polleres<sup>1,2</sup>, and Dieter Fensel<sup>1</sup>

<sup>1</sup> Digital Enterprise Research Institute (DERI) Galway, Ireland and Innsbruck, Austria  
{jos.debruijn, holger.lausen, dieter.fensel}@deri.org

<sup>2</sup> Universidad Rey Juan Carlos, Madrid, Spain  
axel@polleres.net

**Abstract.** The Web Service Modeling Language (WSML) is a language for the specification of different aspects of Semantic Web Services. It provides a formal language for the Web Service Modeling Ontology WSMO which is based on well-known logical formalisms, specifying one coherent language framework for the semantic description of Web Services, starting from the intersection of Datalog and the Description Logic *SHIQ*. This core language is extended in the directions of Description Logics and Logic Programming in a principled manner with strict layering. WSML distinguishes between conceptual and logical modeling in order to support users who are not familiar with formal logic, while not restricting the expressive power of the language for the expert user. IRIs play a central role in WSML as identifiers. Furthermore, WSML defines XML and RDF serializations for inter-operation over the Semantic Web.

## 1 Introduction

Web Services<sup>1</sup> are pieces of functionality which are accessible over the Web. Current technologies such as WSDL allow to describe the functionality offered by a Web Service on a syntactical level only. For automation of tasks, such as Web Service discovery, composition and execution, semantic descriptions of Web Services are required. Since Semantic Web technology enables this formal description of Web content, the combination of Semantic Web with Web Service is the natural next step to be taken.

This combination is often referred to as Semantic Web Services [16]. In this context, the Web Service Modeling Ontology WSMO [17] provides a conceptual model for the description of various aspects of Services towards such Semantic Web Services (SWS). In particular, WSMO distinguishes four top-level elements:

**Ontologies** Ontologies provide formal and explicit specifications of the vocabularies used by the other modeling elements. Such formal specifications enable automated processing of WSMO descriptions and provide background knowledge for Goal and Web Service descriptions.

**Goals** Goals describe the functionality and interaction style from the requester perspective.

---

<sup>1</sup> Throughout this paper we use the terms “Service” and “Web Service” interchangeably.

**Web Service descriptions** Web Service descriptions specify the functionality and the means of interaction provided by the Web Service.

**Mediators** Mediators connect different WSMO elements and resolve heterogeneity in data representation, interaction style and business processes.

The Web Service Modeling Language WSML takes into account all aspects of Web Service description identified by WSMO. WSML comprises different formalisms in order to investigate their applicability to the description of SWS. Since our goal is to investigate the applicability of different formalisms to the description of SWS, it would be too restrictive to base our effort on existing language recommendations such as OWL [6]. A concrete goal in our development of WSML is to investigate the usage of different formalisms, most notably Description Logics and Logic Programming, in the context of Ontologies and Web services.

We see three main areas which benefit from the use of formal methods in service descriptions: *Ontology description*, *Declarative functional description of Goals and Web services*, and *Description of dynamics*. In its current version WSML defines a syntax and semantics for ontology descriptions. The underlying formalisms which were mentioned earlier are used to give a formal meaning to ontology descriptions in WSML. For the functional description of Goals and Web services, WSML offers a syntactical framework, with Hoare-style semantics in mind. However, WSML does not yet formally specify the exact semantics of the functional descriptions of services. The description of the dynamic behavior of Web services (choreography and orchestration) in the context of WSML is currently under investigation, but has not been integrated in WSML at this point. Thus, in this paper we primarily focus on ontology description in WSML, where it turns out that WSML already includes many potentially useful features lacking in previous approaches.

We give an overview of WSML and its language layering in Section 2. The normative human-readable syntax of WSML is described in Section 3, followed by key features of WSML which are described in Section 4. Section 5 describes related approaches for the description of Semantic Web Services and Ontologies. We draw conclusions and outline future work in Section 6.

## 2 WSML Layering

Figure 1(a) shows the different variants of WSML and the relationships between them. These variants differ in logical expressiveness and in the underlying language paradigms and allow users to make the trade-off between provided expressiveness and the implied complexity for ontology modeling on a per-application basis.

**WSML-Core** is based on by the intersection of the Description Logic *SHIQ* and Horn Logic, based on Description Logic Programs [8]. It has the least expressive power of all the WSML variants. The main features of the language are concepts, attributes, binary relations and instances, as well as concept and relation hierarchies and support for datatypes.

**WSML-DL** captures the Description Logic *SHIQ(D)*, which is a major part of the (DL species of) OWL [6].

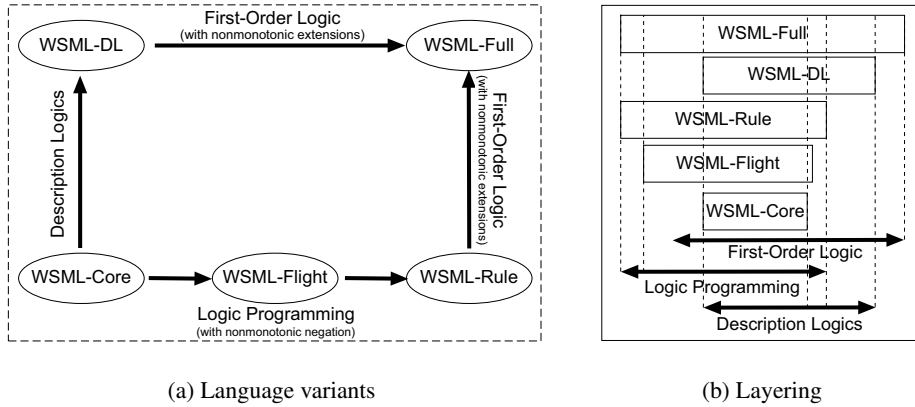


Fig. 1. WSML Variants and Layering

**WSML-Flight** is an extension of WSML-Core which provides a powerful rule language. It adds features such as meta-modeling, constraints and nonmonotonic negation. WSML-Flight is based on a logic programming variant of F-Logic [12] and is semantically equivalent to Datalog with inequality and (locally) stratified negation. WSML-Flight is a direct syntactic extension of WSML-Core and it is a semantic extension in the sense that the WSML-Core subset of WSML-Flight agrees with WSML-Core on ground entailments (cf. [11]).

**WSML-Rule** extends WSML-Flight with further features from Logic Programming, namely the use of function symbols, unsafe rules and unstratified negation under the Well-Founded semantics.

**WSML-Full** unifies WSML-DL and WSML-Rule under a First-Order umbrella with extensions to support the nonmonotonic negation of WSML-Rule. The semantics of WSML-Full is currently an open research issue.

As shown in Figure 1(b), WSML has two alternative layerings, namely,  $\text{WSML-Core} \Rightarrow \text{WSML-DL} \Rightarrow \text{WSML-Full}$  and  $\text{WSML-Core} \Rightarrow \text{WSML-Flight} \Rightarrow \text{WSML-Rule} \Rightarrow \text{WSML-Full}$ . For both layerings, WSML-Core and WSML-Full mark the least and most expressive layers. The two layerings are to a certain extent disjoint in the sense that inter-operation in WSML between the Description Logic variant (WSML-DL) on the one hand and the Logic Programming variants (WSML-Flight and WSML-Rule) on the other, is only possible through a common core (WSML-Core) or through a very expressive superset (WSML-Full).

### 3 General WSML Syntax

In this section we introduce the general WSML syntax which encompasses all features supported by the different language variants. We describe the restrictions imposed on this general syntax by the different variants. These restrictions follow from the logical language underlying the specific language variant, as described in the previous section.

WSML makes a clear distinction between the modeling of the different conceptual elements on the one hand and the specification of complex logical definitions on the other. To this end, the WSML syntax is split into two parts: the conceptual syntax and logical expression syntax. The conceptual syntax was developed from the user perspective, and is independent from the particular underlying logic; it shields the user from the peculiarities of the underlying logic. Having such a conceptual syntax allows for easy adoption of the language, since it allows for an intuitive understanding of the language for people not familiar with logical languages. In case the full power of the underlying logic is required, the logical expression syntax can be used. There are several entry points for logical expressions in the conceptual syntax, namely, axioms in ontologies and capability descriptions in Goals and Web Services.

We will first describe the use of Web identifiers and concrete data values in Section 3.1. The different kinds of WSML definitions and a general explanation of the conceptual syntax are given in Section 3.2. The logical expression syntax is described in Section 3.3. Finally, we briefly outline the XML and RDF serializations in Section 3.4.

### 3.1 Identifiers in WSML

WSML has three kinds of identifiers, namely, IRIs, sQNames, which are abbreviated IRIs, and data values.

An *IRI* (Internationalized Resource Identifier)<sup>2</sup> uniquely identifies a resource in a Web-compliant way. The IRI proposed standard is the successor of the popular URI standard and has already been adopted in various W3C recommendations. IRIs are delimited using an underscore and a double quote ‘\_’ and a double quote ‘”’, for example: `_”http://www.wsmo.org/wsml/wsml-syntax#”`.

In order to enhance legibility, an IRI can be abbreviated to an sQName, which is short for ‘serialized QName’, and is of the following form: *prefix#localname*. The prefix and local part may be omitted, in which case the name falls in the default namespace. Our concept of an ‘sQName’ corresponds with the use of QNames in RDF and is slightly different from QNames in XML, where a QName is not merely an abbreviation for an IRI, but a tuple `<namespaceURI, localname>`.

Data values in WSML are either strings, integers, decimals or structured data values, reflecting the XML Schema datatypes. WSML defines constructs which reflect the structure of data values. For example, the date “March 15th, 2005” is represented as: `_date(2005,3,15)`. In logical expressions, constructed data values can be used in the same way as constructed terms, with the difference that constructed terms may not be nested inside constructed data values.

### 3.2 Conceptual Syntax

The WSML conceptual syntax allows for the modeling of Ontologies, Web Services, Goals and Mediators. It is shared between all variants, with the exception of some restrictions which apply on the modeling of ontologies in WSML-Core and WSML-DL.

---

<sup>2</sup> IETF RFC 3987: <http://www.ietf.org/rfc/rfc3987.txt>

**Ontologies** An ontology in WSMML consists of the elements **concept**, **relation**, **instance**, **relationInstance** and **axiom**. Additionally, an ontology may have non-functional properties and may import other ontologies. We start the description of WSMML ontologies with an example which demonstrates the elements of an ontology in Listing 1, and detail the elements below.

```

wsmmlVariant _"http://www.wsmo.org/wsmml/wsmml-syntax/wsmml-flight"
namespace {_"http://example.org/bookOntology#",
  dc _"http://purl.org/dc/elements/1.1/" }
ontology _"http://example.org/bookOntology"
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.

```

**Listing 1.** An Example WSMML Ontology

*Concepts* The notion of concepts (sometimes also called ‘classes’) plays a central role in ontologies. Concepts form the basic terminology of the domain of discourse. A concept may have instances and may have a number of attributes associated with it. The non-functional properties, as well as the attribute definitions, are grouped together in one frame, as can be seen from the example concept book in Listing 1.

Attribute definitions can take two forms, namely *constraining* (using **ofType**) and *inferring* (using **impliesType**) attribute definitions<sup>3</sup>. Constraining attribute definitions define a typing constraint on the values for this attribute, similar to integrity constraints in Databases; inferring attribute definitions imply that the type of the values for the attribute is inferred from the attribute definition, similar to range restrictions on properties in RDFS [3] and OWL [6]. Each attribute definition may have a number of features associated with it, namely, transitivity, symmetry, reflexivity, and the inverse of an attribute, as well as minimal and maximal cardinality constraints.

Constraining attribute definitions, as well as cardinality constraints, require closed-world reasoning and are thus not allowed in WSMML-Core and WSMML-DL. As opposed

<sup>3</sup> The distinction between inferring and constraining attribute definitions is explained in more detail in [5, Section 2]

to features of roles in Description Logics, attribute features such as transitivity, symmetry, reflexivity and inverse attributes are local to a concept in WSML. Thus, none of these features may be used in WSML-Core and WSML-DL. For a motivation on the use of constraining attributes, see [5].

*Relations* Relations in WSML can have an arbitrary arity, may be organized in a hierarchy using **subRelationOf** and the parameters may be typed using parameter type definitions of the form (**ofType** *type*) and (**impliesType** *type*), where *type* is a concept identifier. The usage of **ofType** and **impliesType** correspond with the usage in attribute definitions. Namely, parameter definitions with the **ofType** keyword are used to check the type of parameter values, whereas parameter definitions with the **impliesType** keyword are used to infer concept membership of parameter values.

The allowed arity of the relation may be constrained by the underlying logic of the WSML language variant. WSML-Core and WSML-DL allow only binary relations and, similar to attribute definitions, they allow only parameter typing using the keyword **impliesType**.

*Instances* A concept may have a number of instances associated with it. Instances explicitly specified in an ontology are those which are shared as part of the ontology. However, most instance data exists outside the ontology in private databases. WSML does not prescribe how to connect such a database to an ontology, since different organizations will use the same ontology to query different databases and such corporate databases are typically not shared.

An instance may be member of zero or more concepts and may have a number of attribute values associated with it, see for example the instance `crimeAndPunishment` in Listing 1. Note that the specification of concept membership is optional and the attributes used in the instance specification do not necessarily have to occur in the associated concept definition. Consequently, WSML instances can be used to represent semi-structured data, since without concept membership and constraints on the use of attributes, instances form a directed labelled graph. Because of this possibility to capture semi-structured data, most RDF graphs can be represented as WSML instance data, and vice versa.

*Axioms* Axioms provide a means to add arbitrary logical expressions to an ontology. Such logical expressions can be used to refine concept or relation definitions in the ontology, but also to add arbitrary axiomatic domain knowledge or express constraints. The axiom `authorshipFromAuthor` in Listing 1 states that the relation `authorship` exists between any author and any book of which he is an author; consequently,  $\langle \text{dostoyesky}, \text{crimeAndPunishment} \rangle$  is in the relation `authorship`. Logical expressions are explained in more detail in Section 3.3.

**Web Services** A Web Service has a capability and a number of interfaces. The capability describes the Web Service functionality by expressing conditions over its pre- and post-states<sup>4</sup> using logical expressions; interfaces describe how to interact with the

<sup>4</sup> Pre-state (post-state, respectively) refers to the state before (after, respectively) the execution of the Web Service

service. Additionally, WSML allows to specify non-functional properties of a Web Service. Listing 2 describes a simple Web Service for adding items to a shopping cart.

```
webService _"http://example.org/bookService"  
nonFunctionalProperties  
  dc:title hasValue "Example book buying service"  
  dc:description hasValue "A simple example web service for adding items to a shopping cart"  
endNonFunctionalProperties  
  
importsOntology _"http://example.org/bookOntology"  
capability  
  sharedVariables {?cartId, ?item}  
  precondition  
    definedBy  
      ?cartId memberOf _string and ?item memberOf book.  
  postcondition  
    definedBy  
      forall ?cart (?cart[id hasValue ?cartId] memberOf cart implies  
        ?cart[items hasValue ?item]).
```

**Listing 2.** A WSML Web Service description

*Capabilities* Preconditions and assumptions describe the state before the execution of a Web Service. While preconditions describe conditions over the information space, i.e., conditions over the input; assumptions describe condition over the state of world which can not necessarily be directly checked. Postconditions describe the relation between the input and the output, e.g., a credit card limit with respect to its values before the service execution. In this sense, they describe the information state after execution of the service. Effects describe changes in the real world caused by the service, e.g., the physical shipment of some good. The **sharedVariables** construct is used to identify variables which are shared between the pre- and postconditions and the assumptions and effects. Shared variables can be used to refer to the same input and output values in the conditions of the capability. Listing 2 describes a simple Web Service for adding items to a shopping cart: given a shopping cart identifier and a number of items, the items are added to the shopping cart with this identifier.

*Interfaces* Interfaces describe how to interact with a service from the requester point-of-view (**choreography**) and how the service interacts with other services and goals it needs to fulfill in order to fulfill its capability (**orchestration**), which is the provider point of view. Choreography and orchestration descriptions are external to WSML; WSML allows to reference any choreography or orchestration identified by an IRI.

**Goals** Goals are symmetric to Web Services in the sense that Goals describe desired functionality and Web Services describe offered functionality. Therefore, a Goal description consists of the same modeling elements as a Web Service description, namely, non-functional properties, a capability and a number of interfaces.

**Mediators** Mediators connect different Goals, Web Services and Ontologies, and enable inter-operation by reconciling differences in representation formats, encoding styles, business protocols, etc. Connections between Mediators and other WSML elements can be established in two different ways:

1. Each WSML element allows for the specification of a number of used mediators through the **usesMediator** keyword.
2. Each mediator has (depending on the type of mediator) one or more sources and one target. Both source and target are optional in order to allow for generic mediators.

A mediator achieves its mediation functionality either through a Web Service, which provides the mediation service, or a Goal, which can be used to dynamically discover the appropriate (mediation) Web Service.

### 3.3 Logical Expression Syntax

We will first explain the general logical expression syntax, which encompasses all WSML variants, and then describe the restrictions on this general syntax for each of the variants. The general logical expression syntax for WSML has a First-Order Logic style, in the sense that it has constants, function symbols, variables, predicates and the usual logical connectives. Furthermore, WSML has F-Logic [12] based extensions in order to model concepts, attributes, attribute definitions, and subconcept and concept membership relationships. Finally, WSML has a number of connectives to facilitate the Logic Programming based variants, namely default negation (negation-as-failure), LP-implication (which differs from classical implication) and database-style integrity constraints.

Variables in WSML start with a question mark, followed by an arbitrary number of alphanumeric characters, e.g., ?x, ?name, ?123. Free variables in WSML (i.e., variables which are not explicitly quantified), are implicitly universally quantified outside of the formula (i.e., the logical expression in which the variable occurs is the scope of quantification), unless indicated otherwise, through the **sharedVariables** construct (see the previous Section).

Terms are either identifiers, variables, or constructed terms. An atom is, as usual, a predicate symbol with a number of terms as arguments. Besides the usual atoms, WSML has a special kind of atoms, called *molecules*, which are used to capture information about concepts, instances, attributes and attribute values. There are two types of molecules, analogous to F-Logic:

- An *isa* molecule is a concept membership molecule of the form  $A$  **memberOf**  $B$  or a subconcept molecule of the form  $A$  **subConceptOf**  $B$  with  $A$  and  $B$  arbitrary terms
- An *object* molecule is an attribute value expressions of the form  $A$ [ $B$  **hasValue**  $C$ ], a constraining attribute signature expression of the form  $A$ [ $B$  **ofType**  $C$ ], or an inferring attribute signature expression of the form  $A$ [ $B$  **ofType**  $C$ ], with  $A, B, C$  arbitrary terms

WSML has the usual first-order connectives: the unary negation operator **neg**, and the binary operators for conjunction **and**, disjunction **or**, right implication **implies**, left implication **impliedBy**, and dual implication **equivalent**. Variables may be universally quantified using **forall** or existentially quantified using **exists**. First-order formulae are obtained by combining atoms using the mentioned connectives in the usual way. The following are examples of First-Order formulae in WSML:

```
//every person has a father
forall ?x (?x memberOf Person implies exists ?y (?x[father hasValue ?y])).
//john is member of a class which has some attribute called 'name'
exists ?x,?y (john memberOf ?x and ?x[name ofType ?y]).
```

Apart from First-Order formulae, WSMML allows the use of the negation-as-failure symbol **naf** on atoms, the special Logic Programming implication symbol **:-** and the integrity constraint symbol **!-**. A logic programming rule consists of a *head* and a *body*, separated by the **:-** symbol. An integrity constraint consists of the symbol **!-** followed by a rule body. Negation-as-failure **naf** is only allowed to occur in the body of a Logic Programming rule or an integrity constraint. The further use of logical connectives in Logic Programming rules is restricted. The following logical connectives are allowed in the head of a rule: **and**, **implies**, **impliedBy**, and **equivalent**. The following connectives are allowed in the body of a rule (or constraint): **and**, **or**, and **naf**. The following are examples of LP rules and database constraints:

```
//every person has a father
?x[father hasValue f(?y)] :- ?x memberOf Person.
//Man and Woman are disjoint
!- ?x memberOf Man and ?x memberOf Woman.
//in case a person is not involved in a marriage, the person is a bachelor
?x memberOf Bachelor :- ?x memberOf Person and naf Marriage(?x,?y,?z).
```

**Particularities of the WSMML Variants** Each of the WSMML variants defines a number of restrictions on the logical expression syntax. For example, LP rules and constraints are not allowed in WSMML-Core and WSMML-DL. Table 1 presents a number of language features and indicates in which variant the feature can occur.

Feature	Core	DL	Flight	Rule	Full
Classical Negation ( <b>neg</b> )	-	X	-	-	X
Existential Quantification	-	X	-	-	X
(Head) Disjunction	-	X	-	-	X
<i>n</i> -ary relations	-	-	X	X	X
Meta Modeling	-	-	X	X	X
Default Negation ( <b>naf</b> )	-	-	X	X	X
LP implication	-	-	X	X	X
Integrity Constraints	-	-	X	X	X
Function Symbols	-	-	-	X	X
Unsafe Rules	-	-	-	X	X

**Table 1.** WSMML Variants and Feature Matrix

- *WSMML-Core* allows only first-order formulae which can be translated to the DLP subset of *SHIQ(D)* [8]. This subset is very close to the 2-variable fragment of First-Order Logic, restricted to Horn logic. Although WSMML-Core might appear in the Table 1 featureless, it captures most of the conceptual model of WSMML, but has only limited expressiveness within the logical expressions.

- *WSML-DL* allows first-order formulae which can be translated to  $\mathcal{SHIQ}(\mathbf{D})$ . This subset is very close to the 2-variable fragment of First-Order Logic. Thus, WSML DL allows classical negation, and disjunction and existential quantification in the heads of implications.
- *WSML-Flight* extends the set of formulae allowed in WSML-Core by allowing variables in place of instance, concept and attribute identifiers and by allowing relations of arbitrary arity. In fact, any such formula is allowed in the head of a WSML-Flight rule. The body of a WSML-Flight rule allows conjunction, disjunction and default negation. The head and body are separated by the LP implication symbol. WSML-Flight additionally allows meta-modeling (e.g., classes-as-instances) and reasoning over the signature, because variables are allowed to occur in place of concept and attribute names.
- *WSML-Rule* extends WSML-Flight by allowing function symbols and unsafe rules, i.e., variables which occur in the head or in a negative body literal do not need to occur in a positive body literal.
- *WSML-Full* The logical syntax of WSML-Full is equivalent to the general logical expression syntax of WSML and allows the full expressiveness of all other WSML variants.

The separation between conceptual and logical modeling allows for an easy adoption by non-experts, since the conceptual syntax does not require expert knowledge in logical modeling, whereas complex logical expressions require more familiarity and training with the language. Thus, WSML allows the modeling of different aspects related to Web services on a conceptual level, while still offering the full expressive power of the logic underlying the chosen WSML variant. Part of the conceptual syntax for ontologies has an equivalent in the logical syntax. This correspondence is used to define the semantics of the conceptual syntax. Notice that, since only parts of the conceptual syntax are mapped to the logical syntax, only a part of the conceptual syntax has a semantics in the logical language for ontologies. For example, non-functional properties are not translated (hence, the name ‘non-functional’). The translation between the conceptual and logical syntax is sketched in Table 2.

Conceptual	Logical
<b>concept</b> A subConceptOf B	A <b>subConceptOf</b> B.
<b>concept</b> A B <b>ofType</b> (0 1) C	A[B <b>ofType</b> C]. !– ?x <b>memberOf</b> A and ?x[B <b>hasValue</b> ?y, B <b>hasValue</b> ?z] and ?y != ?z.
<b>concept</b> A B <b>ofType</b> C	A[B <b>ofType</b> C].
<b>relation</b> A/n <b>subRelationOf</b> B	A(x <sub>1</sub> , ..., x <sub>n</sub> ) <b>implies</b> B(x <sub>1</sub> , ..., x <sub>n</sub> )
<b>instance</b> A <b>memberOf</b> B C <b>hasValue</b> D	A <b>memberOf</b> B. A[C <b>hasValue</b> D].

**Table 2.** Translating conceptual to logical syntax

### 3.4 WSML Web Syntaxes

The WSML XML syntax is similar to the human-readable syntax, both in keywords and in structure. We have defined the XML syntax through a translation from the human-readable syntax [4] and have additionally specified an XML Schema for WSML<sup>5</sup>. Note that all WSML elements fall in the WSML namespace <http://www.wsmo.org/wsmo/wsmo-syntax#>.

WSML provides a serialization in RDF of all its conceptual modeling elements which can be found in [4]. The WSML RDF syntax reuses the RDF and RDF Schema vocabulary to allow existing RDF(S)-based tools to achieve the highest possible degree of inter-operation. As a result, WSML can be seen as an extension of RDF(S).

## 4 Key Features of WSML

There are a number of features which make WSML unique from other language proposals for the Semantic Web and Semantic Web Services. These key features are mainly due to the two pillars of WSML, namely (1) a *language independent conceptual model* for Ontologies, Web Services, Goals and Mediators, based on WSMO [17] and (2) *reuse* of several well-known logical language paradigms in *one* syntactical framework. More specifically, we see the following as the key features of WSML:

**One syntactic framework for a set of layered languages** We believe different Semantic Web and Semantic Web Service applications need languages of different expressiveness and that no single language paradigm will be sufficient for all use cases. With WSML we investigate the use of Description Logics and Logic Programming for Semantic Web Services.

**Normative, human readable syntax** It has been argued that tools will hide language syntax from the user; however, as has been seen, for example, with the adoption of SQL, an expressive but understandable syntax is crucial for successful adoption of a language. Developers and early adopters of the language will have to deal with the concrete syntax. If it is easy to read and understand it will allow for easier adoption of the language.

**Separation of conceptual and logical modeling** On the one hand, the conceptual syntax of WSML has been designed in such a way that it is independent of the underlying logical language and no or only limited knowledge of logical languages is required for the basic modeling of Ontologies, Web Services, Goals, and Mediators. On the other hand, the logical expression syntax allows expert users to refine definitions on the conceptual syntax using the full expressive power of the underlying logic, which depends on the particular language variant chosen by the user.

**Semantics based on well known formalisms** WSML captures well known logical formalisms such as Datalog and Description Logics in a unifying syntactical framework, while maintaining the established computational properties of the original formalisms through proper syntactic layering. The variants allow the reuse of tools already developed for these formalisms. Notably, WSML allows to reuse efficient

---

<sup>5</sup> <http://www.wsmo.org/TR/d16/d16.1/v0.21/xml-syntax/wsmo-xml-syntax.xsd>

querying engines developed for Datalog and efficient subsumption reasoners developed in the area of Description Logics. Inter-operation between the paradigms is achieved through a common subset, WSML-Core, based on DLP [8].

**WWW Language** WSML has a number of features which integrate it seamlessly in the Web. WSML adopts the IRI standard, the successor of URI, for the identification of resources, following the Web architecture. Furthermore, WSML adopts the namespace mechanism of XML and datatypes in WSML are compatible with datatypes in XML Schema [2] and datatype functions and operators are based on the functions and operators of XQuery [15]. Finally, WSML defines an XML syntax and an RDF syntax for exchange over the Web. When using the RDF syntax, WSML can be seen as an extension of RDFS.

**Frame-based syntax** Frame Logic [12] allows the use of frames in logical expressions. This allows the user to work directly on the level of concepts, attributes, instances and attribute values, instead of at the level of predicates. Furthermore, variables are allowed in place of concept and attribute identifiers, which enables meta-modeling and reasoning over the signature in the rule-based WSML language variants.

## 5 Related Work

In this section we review existing work in the areas of Semantic Web and Semantic Web Services languages and compare it to WSML.

*RDFS* RDFS [3] is a simple ontology modeling languages based on triples. It allows to express classes, properties, class hierarchies, property hierarchies, and domain- and range restrictions. Several proposals for more expressive Semantic Web and Semantic Web Service descriptions extend RDFS, however there are difficulties in semantically layering an ontology language on top of RDFS:

1. RDFS allows the use of the language vocabulary as subjects and objects in the language itself.
2. RDFS allows the use of the same identifier to occur at the same time in place of a class, individual, and property identifier.

We believe that the number of use cases for the first feature, namely the use of language constructs in the language itself, is limited. However, the use of the same identifier as class, individual and property identifier (also called meta-modeling) is deemed useful in many cases. WSML does not allow the use of the language constructs in arbitrary places in an ontology, but does allow meta-modeling in its Flight, Rule and Full variants.

WSML is an extension of a significant part of RDFS; it does not allow the use of language constructs in the language itself and does not allow full treatment of blank nodes, because this would require reasoning with existential information, which is not allowed in the rule-based WSML variants. WSML provides a significant extension of RDFS through the possibility of specifying local attributes, range and cardinality constraints for attributes and attribute features such as symmetry, transitivity and reflexivity. Furthermore, WSML (in its rule-based variants) provides an expressive rule language which can be used for the manipulation of RDF data.

*OWL* The Web Ontology Language OWL [6] is a language for modeling ontologies based on the Description Logic paradigm. OWL consists of three species, namely OWL Lite, OWL DL and OWL Full, which are intended to be layered according to increasing expressiveness. OWL Lite is a notational variant of the Description Logic  $SHIF(\mathbf{D})$ ; OWL DL is a notational variant of the Description logic  $SHOIN(\mathbf{D})$ . The most expressive species of OWL, OWL Full, layers on top of both RDFS and OWL DL. We compare OWL with the ontology description component of WSML, since OWL does not offer means to describe Web Services, Goals and Mediators.

WSML-Core is a semantic subset of OWL Lite. WSML-DL is semantically equivalent to OWL DL. However, there is a major difference between ontology modeling in WSML and ontology modeling in OWL. WSML uses an epistemology which abstracts from the underlying logical language, whereas OWL directly uses Description Logics epistemology; WSML separates between conceptual modeling for the non-expert users and logical modeling for the expert user. Arguably, these properties could make WSML easier to use as an ontology language. This is, however, merely a conjecture and would require extensive user testing to verify its correctness.

WSML-Flight and WSML-Rule are based on the Logic Programming paradigm, rather than the Description Logic paradigm. Thus, their expressiveness is quite different from OWL. On the one hand, WSML-Flight/Rule allow chaining over predicates and non-monotonic negation, but do not allow classical negation and full disjunction and existential quantification. We conjecture that both the Description Logics and Logic Programming paradigms are useful on the Semantic Web (cf. [11]). With WSML we capture both paradigms in one coherent framework. Interaction between the paradigms is achieved through a common subset, WSML-Core.

*OWL-S* OWL-S [1] is an OWL ontology for the modeling of Semantic Web Services. It has been recognized that the expressiveness of OWL alone is not enough for the specification of Web Services (e.g. [13]). To overcome this limitation OWL-S allows the use of more expressive languages such as SWRL [9], KIF and DRS. However, the relation between the inputs and output described using OWL and the formulae in these languages is sometimes not entirely clear.

Comparing the language suggestions for WSML and OWL-S it turns out that while OWL-S aims at combining different notations and semantics with OWL for the description of service conditions and effects, WSML takes a more cautious approach: WSML does not distinguish between languages used for inputs/output and other description elements of the Web Service, but provides one uniform language for capability descriptions. Additionally, the languages suggested for OWL-S are all based on classical logic, whereas WSML also offers the possibility to use (nonmonotonic) Logic Programming.

Finally, WSML is based on the conceptual model of WSMO, which differs significantly from the OWL-S conceptual model for Web Service modeling. For a detailed comparison, see [14].

## 6 Conclusions and Future Work

In this paper we have presented the Web Service Modeling Language WSML, a language for the specification of different aspects related to Semantic Web Services, based

on the Web Service Modeling Ontology WSMO [17]. WSML brings together different logical language paradigms and unifies them in one syntactical framework, enabling the reuse of proven reasoning techniques and tools. Unlike other proposals for Semantic Web and Semantic Web Service languages, WSML has a normative human readable syntax that makes a separation between conceptual and logical syntax, thereby enabling conceptual modeling from the user point-of-view according to a language-independent meta-model (WSMO), while not restricting the expressiveness of the language for the expert user. With the use of IRIs (the successor of URI) and the use of XML and RDF, WSML is a language based on the principles of the Semantic Web and allows seamless integration with other Semantic Web languages and applications.

The definition of an inter-operability layer between the Description Logic and Rules paradigms, in the form of WSML-Core, enables the use and extension of the same core ontology for a number of different reasoning tasks supported by a number of different reasoners, most notably subsumption reasoning using Description Logic reasoners and query answering using Logic Programming reasoners.

Future work for WSML consists of the application of the language to various use cases and the improvement of WSML tools, such as editors and reasoners<sup>6</sup>. From the language development point of view, the semantics of WSML-Full has not yet been defined; we are currently looking into several nonmonotonic logics, such as Autoepistemic and Default Logic. There are approaches which combine expressive Description Logics with nonmonotonic logic programming without requiring the expressiveness of WSML-Full (e.g. ). Incorporating such approaches in WSML is a matter of ongoing investigation. We are working on defining the operational semantics for the Web Service capability. Such operational semantics is necessary for the automation of several Web Service related tasks, such as discovery [10]. It might turn out, however, that different tasks need different operational semantics. Finally, the Web Service choreography and orchestration are currently place-holders in WSML; work is ongoing to fill these place-holders.

## Acknowledgments

We would like to thank all members of the WSML working group, especially Eyal Oren and Rubén Lara, for their comments and input to this document. We thank the anonymous reviewers for useful feedback.

This work was funded by the European Commission under the projects ASG, DIP, KnowledgeWeb, and SEKT; by Science Foundation Ireland under Grant No. SFI/02/CE1/I13; and by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the project RW<sup>2</sup>.

## References

1. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDemott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan and K. Sycara. *OWL-S*:

<sup>6</sup> An overview of currently available tools can be found at: <http://tools.deri.org/wsml/>

- Semantic markup for web services.* Member Submission 22, W3C, November 2004. <http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/>.
2. P. V. Biron and A. Malhotra, editors. *XML Schema Part 2: Datatypes.* 2004. <http://www.w3.org/TR/xmlschema-2/>.
  3. D. Brickley and R. V. Guha. RDF vocabulary description language 1.0: RDF schema. Recommendation 10 February 2004, W3C, 2004. <http://www.w3.org/TR/rdf-schema/>.
  4. J. de Bruijn, editor. *The Web Service Modeling Language WSMML.* 2005. WSMO Final Draft D16.v0.21. <http://www.wsmo.org/TR/d16/d16.1/v0.21/>.
  5. J. de Bruijn, A. Polleres, R. Lara, and D. Fensel. OWL DL vs. OWL Flight: Conceptual modeling and reasoning on the semantic web. In *Proc. WWW2005*, Chiba, Japan. 2005.
  6. M. Dean and G. Schreiber, editors. *OWL Web Ontology Language Reference.* 2004. W3C Recommendation 10 February 2004.
  7. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. In *Proc. KR2004*, 2004.
  8. B. N. Groszof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proc. WWW2003*, Budapest, Hungary, 2003.
  9. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. Member submission 21 may 2004, W3C, 2004. <http://www.w3.org/Submission/SWRL/>.
  10. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic location of services. In *Proc. ESWC2005*. 2005.
  11. M. Kifer, J. de Bruijn, H. Boley, and D. Fensel. A realistic architecture for the semantic web. In *Proc. RuleML-2005*, Ireland, Galway. 2005.
  12. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843, 1995.
  13. R. Lara, H. Lausen, S. Arroyo, J. de Bruijn, and D. Fensel. Semantic web services: description requirements and current technologies. In *Semantic Web Services for Enterprise Application Integration and e-Commerce workshop (SWSEE03), in conjunction with ICEC 2003*, Pittsburgh, PA, USA.
  14. R. Lara, A. Polleres, H. Lausen, D. Roman, J. de Bruijn, and D. Fensel. A conceptual comparison between WSMO and OWL-S. Final draft D4.1v0.1, WSMO, 2004. <http://www.wsmo.org/TR/d4/d4.1/v0.1/>.
  15. A. Malhotra, J. Melon, and N. Walsh. Xquery 1.0 and xpath 2.0 functions and operators. Candidate Recommendation, W3C, 2005. <http://www.w3.org/TR/xpath-functions/>.
  16. S. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46–53, 2001.
  17. D. Roman, U. Keller, H. Lausen, R. L. Jos de Bruijn, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005.
  18. R. Rosati.  $\mathcal{DL} + \text{log}$ : Tight integration of description logics and disjunctive datalog. In *Proc. KR2006*, 2006.