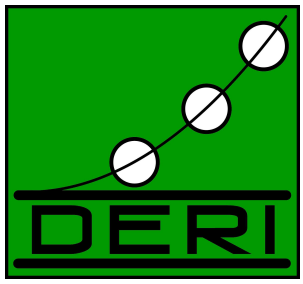
DERI – DIGITAL ENTERPRISE RESEARCH INSTITUTE




# THE WEB SERVICE MODELING LANGUAGE WSML: AN OVERVIEW

Jos De Bruijn Holger Lausen Axel Polleres
Dieter Fensel

DERI TECHNICAL REPORT 2005-06-16

JUNE 2005

**DERI Ireland**
University Road
Galway
IRELAND
www.deri.ie

**DERI Innsbruck**
Technikerstrasse 13
A-6020 Innsbruck
AUSTRIA
www.deri.ie

DERI – DIGITAL ENTERPRISE RESEARCH INSTITUTE

DERI TECHNICAL REPORT
DERI TECHNICAL REPORT 2005-06-16, JUNE 2005

# THE WEB SERVICE MODELING LANGUAGE WSML: AN OVERVIEW

Jos De Bruijn Holger Lausen Axel Polleres Dieter Fensel [1]

**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 description of Semantic Web Services, starting from the intersection of Datalog and the Description Logic $\mathcal{SHIQ}$. 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 facilitate 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.

**Keywords:** Semantic Web Services, Ontologies, WSMO, Conceptual Modeling, Logical Languages

[1]Digital Enterprise Research Institute. E-mail: {jos.debruijn, holger.lausen, axel.polleres, dieter.fensel }@deri.org

**Acknowledgements**: This work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, and ASG; by Science Foundation Ireland under the DERI-Lion project; by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects RW$^2$ and TSC.

# Contents

**1 Introduction** **1**

**2 WSML Layering** **1**

**3 General WSML Syntax** **3**
3.1 Identifiers in WSML . . . 3
3.2 Conceptual Syntax . . . 4
3.2.1 Ontologies . . . 4
3.2.2 Web Services . . . 6
3.2.3 Goals . . . 6
3.2.4 Mediators . . . 6
3.3 Logical Expression Syntax . . . 7
3.3.1 Particularities of the WSML Variants . . . 8
3.4 WSML Web Syntaxes . . . 9

**4 Key Features of WSML** **9**

**5 Related Work** **10**
5.1 RDFS . . . 10
5.2 OWL . . . 11
5.3 OWL-S . . . 11

**6 Conclusions and Future Work** **11**

# 1 Introduction

Web Services are pieces of functionality which are accessible over the Web. Current Web Service technologies 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 description of Web Services is required. Semantic Web technology enables formal description of Web content. A combination of Semantic Web with Web Service technology allows the formal description of Web Services. This combination is also called Semantic Web Services [12, 22]. In this context, the Web Service Modeling Ontology WSMO [24] provides a conceptual model for the description of various aspects related to Semantic Web Services. In particular, WSMO distinguishes four top-level elements:

**Ontologies** Ontologies [11] 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.

**Web Service descriptions** Web Service descriptions specify the functionality and the means of interacting with the Web Service in order to achieve the requested functionality.

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

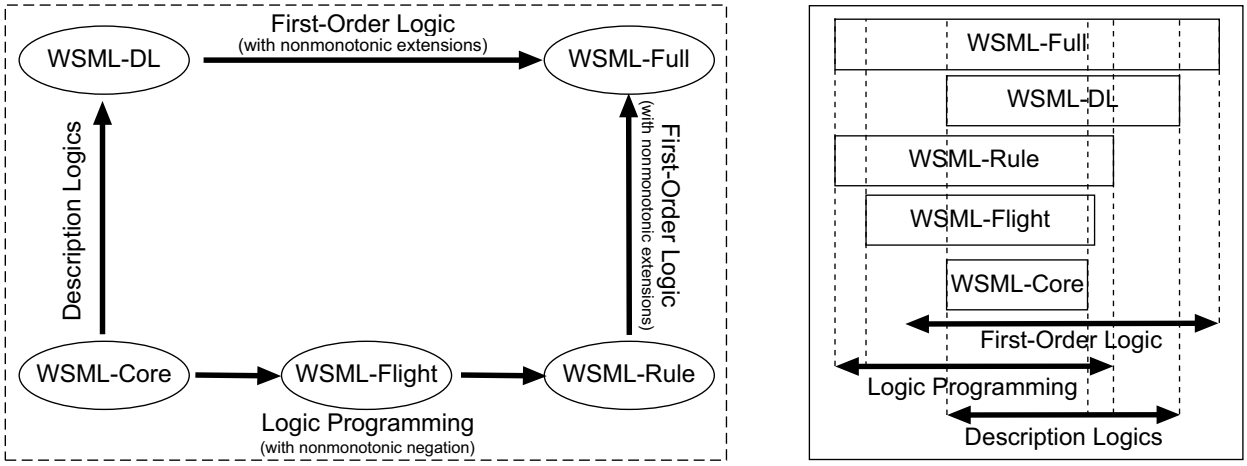
With WSML we provide a formal Web language based on the conceptual model of WSMO. The goal of WSML is to provide one coherent framework which brings together Web technologies with different well-known logical language paradigms in order to enable the description of Semantic Web Services. We take Description Logics [2], Logic Programming [20], and F-Logic [17], as starting points for the development of a number of WSML language variants. The core language is based on the intersection of Description Logics and Logic Programming [13]. This core language is extended in the directions of the mentioned language paradigms. Syntax-wise, WSML takes the user point of view with on the one hand its syntax for conceptual modeling and on the other hand allows full flexibility to specify arbitrary logical axioms and constraints using the logical expression syntax.[1]

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. The key features of WSML 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 on a per-application basis.

[1] For the full WSML specification and other WSML-related resources we refer to `http://www.wsmo.org/wsml-syntax`




(a) Language variants

(b) Layering

Figure 1: WSML Variants and Layering

**WSML-Core** is based on by the intersection of the Description Logic $\mathcal{SHIQ}$ and Horn Logic, based on Description Logic Programs [13]. 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 $\mathcal{SHIQ}(\mathbf{D})$, which is a major part of the (DL species of) OWL [8]. WSML-DL furthermore includes a datatype extension based on OWL-Eu [23] for richer datatype support.

**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 [17] and is semantically equivalent to Datalog with inequality and (locally) stratified negation.

**WSML-Rule** extends WSML-Flight with further features from Logic Programming such as the use of function symbols and unsafe rules.

**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, WSML-Core $\Rightarrow$ WSML-DL $\Rightarrow$ WSML-Full and WSML-Core $\Rightarrow$ WSML-Flight $\Rightarrow$ WSML-Rule $\Rightarrow$ 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 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 (Ontologies, Web Services, Goals, and Mediators) on the one hand and the specification of 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 particularities 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.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {_"http://www.example.org/example#",
  dc _"http :// purl .org/dc/elements/1.1/"}

  [... Goals, Ontologies, etc ...]
```

Listing 1: An Example Prologue of a WSML File

As can be seen from the example in Listing 1, the prologue of a WSML file consists of the following elements: an indication of the WSML language variant and the declaration of a number of namespace prefixes, as well as the default namespace for the file. After the prologue is the actual definition of the Ontology, Web Service, Goal or Mediator. Each of these definitions has a header part, which consists of meta-data in the form of non-functional properties, the declaration of imported ontologies and the specification of mediators used by the definition, illustrated in Listing 2. 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 describe 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) [9] 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 (e.g., [4]).

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 [4], where a QNames 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. Structured data values are constructed using datatype wrappers. Each datatype has a datatype wrapper associated with it. Datatype support in WSML is based on XML Schema, and 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.

### 3.2.1 Ontologies

An ontology in WSML consists of the elements **concept**, **relation**, **instance**, **relationInstance** and **axiom**. We start the description of WSML ontologies with an example which demonstrates the elements of an ontology, in Listing 2, and detail the elements below.

```
ontology _”http :// www.example.org/ontologies/example”
  nfp
    dc#title  hasValue ”WSML example person ontology”
  endnfp

  importsOntology {_”http://www.wsmo.org/ontologies/location”}

  concept Person
  nfp
    dc#relation  hasValue {personUncle}
  endnfp
    hasName ofType (0 1) _string
    hasChild inverseOf(hasParent) ofType Person

  relation  Marriage (ofType Person, ofType Person, ofType _date)
  nfp
    dc#description hasValue ”Relation between the participants and the date of the  marriage.”
  endnfp

  instance john memberOf Person
    hasName hasValue ”John”

  relationInstance Marriage(john,mary,_date(2005,03,03))

  axiom personUncle
  nfp
    dc#description hasValue ”The brother of a person’s parent is  that  person’s uncle.”
  endnfp
  definedBy
    ?x[hasUncle hasValue ?z] :– ?x[hasParent hasValue ?y] and ?y[hasBrother hasValue ?z].
```

Listing 2: An Example WSML Ontology

**Concepts** The notion of concepts (sometimes also called ‘classes’) plays a central role in ontologies [11]. 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 ‘Person’ in Listing 2. Identifiers of axioms related to a concept definition may be specified using the dc#relation non-functional property.

Attribute definitions may be of two forms, namely *constraining* (using **ofType**) and *inferring* (using **impliesType**) attribute definitions. Constraining attribute definitions define a typing constraint on the values for this attribute, similar to 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 [5] and OWL [8]. 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 WSML-Core and WSML-DL. Furthermore, the formalisms underlying WSML-Core and WSML-DL do not have the notion of local attribute definitions, and thus none of the attribute features may be used in WSML-Core and WSML-DL.

**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 share.

An instance may be member of zero or more concepts and may have a number of attribute values associated with it. 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.

Besides specifying instances of concepts, it is also possible to specify instances of relations, which correspond to tuples in a relation. Relation instances may have an identifier associated with them, but this is optional.

**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 knowledge or express constraints. For example, one could write a rule stating that the brother of a person's parent is that person's uncle (see the axiom personUncle in Listing 2). Logical expressions are explained in more detail in Section 3.3.

### 3.2.2 Web Services

A Web Service has a capability and a number of interfaces. The capability describes the Web Service functionality by expressing its pre- and post-state[2] using logical expressions, whereas interfaces describe how to interact with the service.

**Capabilities** Preconditions and assumptions describe the state before the execution of a Web Service. While preconditions describe conditions within the information space, i.e. conditions that can be directly checked by a service; assumptions describe condition over the state of world that 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 can be used in order to quantify variables over the complete capability and thus express relations between values in the pre- and post-states. Listing 3 describes a simple Web Service for credit card transactions: given a credit card and the cost of a product, the credit card limit needs to be higher than the cost of the product; after execution of the Web Service, the limit of the credit card has been decreased with the cost of the purchase.

```
capability
  sharedVariables {?x,?creditcard,?cost}
  precondition
    definedBy
      ?creditcard[
        limit hasValue ?x
      ]memberOf CreditCard
      and ?x >= ?cost.
  postcondition
    definedBy
      ?creditcard[
        limit hasValue ?x−?cost
      ].
```

Listing 3: A WSML Capability Definition

**Interfaces** Interfaces describe how to interact with a service, both from the requester (**choreography**) and the provider (**orchestration**) point of view. Choreography and orchestration are external to WSML; instead, WSML allows to reference any choreography or orchestration identified by an IRI. For a proposal on choreography and orchestration in WSMO, see [25].

### 3.2.3 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 a capability and a number of interfaces.

### 3.2.4 Mediators

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

[2] Pre-state (post-state, respectively) refers to the state before (after, respectively) the execution of the Web Service

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 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 [17] 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 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).

As usual, 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. The 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 Firs-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, WSML 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).

### 3.3.1 Particularities of the WSML Variants

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

Table 1: WSML Variants and Feature Matrix

| Feature | Core | DL | Flight | Rule | Full |
|---|---|---|---|---|---|
| Classical Negation (**neg**) | - | X | - | - | X |
| Existential Quantification | - | X | - | - | X |
| Disjunction | - | X | - | - | X |
| Meta Modeling | - | - | X | X | X |
| Default Negation (**naf**) | - | - | X | X | X |
| LP implication | - | - | X | X | X |
| Integrity Constraints | - | - | X | X | X |
| Function Symbols | - | - | - | X | X |
| Unsafe Rules | - | - | - | X | X |

**WSML-Core** allows only first-order formulae which can be translated to the DLP subset of $\mathcal{SHIQ}(\mathbf{D})$ [13]. This subset is very close to the 2-variable fragment of First-Order Logic, restricted to Horn logic. Although WSML-Core might appear in the Table 1 featureless, it captures most of the conceptual model of WSML, 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, disjunction and existential quantification.

**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 predicates 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 do not need to occur in the body.

**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.

### 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 [6] and have additionally specified an XML Schema for WSML[3]. Note that all WSML elements fall in the WSML namespace `http://www.wsmo.org/wsml/wsml-syntax#`.

WSML provides a serialization in RDF of all its conceptual modeling elements which can be found in [6]. 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.

## 4 Key Features of WSML

There are a number of features which make WSML stand out 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 coming from WSMO [24] and (2) *reuse* of existing well-known logical language paradigms. 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. There already exist language recommendations for certain aspects, such as the Ontology languages RDFS [5] and OWL [8]. Already in the case of RDFS and OWL, we can see that layering languages on existing recommendations is not straightforward [15, 7]; either the layering is not strict, or certain desirable features of a language, such as the ability to use existing efficient reasoners, are lost.

**Normative, human readable syntax** It has been argued that tools will hide language syntax from the user; however, as has been seen 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 an no or only limited knowledge of formal languages is required for the basic modeling of Web Services, Goals, Mediators and Ontologies. On the other hand, the logical expression syntax allows expert users to

[3] `http://www.wsmo.org/TR/d16/d16.1/v0.2/xml-syntax/wsml-xml-syntax.xsd`

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 formalism** 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. Furthermore it allows the reuse of tools already developed for these formalisms. Notably, WSML allows to reuse efficient querying engines developed for Datalog and efficient subsumption reasoners developed in the area of Description Logics. Interoperation between the paradigms is achieved through a common subset, depicted by DLP [13].

**WWW Language** WSML has a number of features which integrate it seamlessly in the Web. WSML adopts the IRI [9] 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 [3] and datatype functions and operators are based on the functions and operators of XQuery [21]. 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 and thus an integral part of the Semantic Web language stack.

**Frame-based syntax** Frame Logic [17] 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.

We believe the key features of WSML make it a flexible language for the description of Ontologies and Web Services.

# 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.

## 5.1 RDFS

RDFS [5] 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 useful in many cases [26, 7]. 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.

### 5.2 OWL

The Web Ontology Language OWL [8] 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 $\mathcal{SHIF}(\mathbf{D})$; OWL DL is a notational variant of the Description logic $\mathcal{SHOIN}(\mathbf{D})$ [15]. It turns out that OWL DL adds very little in expressiveness to OWL Lite [15]. The most expressive species of OWL, OWL Full, layers on top of both RDFS and OWL DL, and because these languages are so different, the semantics of OWL Full is not straightforward and is not a proper extension of the OWL DL semantics [7].

WSML-Core is an expressive subset of OWL Lite, whereas WSML-DL is expressively equivalent to OWL Lite. There are two major differences between ontology modeling in WSML and ontology modeling in OWL:

- WSML uses WSMO epistemology, whereas OWL uses Description Logics epistemology. The major differences are that (1) attribute definitions in WSML are local to a concept whereas property definitions in OWL are in principle global (although it is possible to define local restrictions on properties); and (2) WSML has a clear separation between the conceptual syntax and the logical expression syntax, whereas OWL has one syntax for all types of axioms.

- The modeling of arbitrary axioms is done in OWL using Description Logic-style primitives. The OWL abstract syntax defines such primitives as IntersectionOf and SubClassOf, denoting concept intersection and concept subsumption, respectively. WSML uses first-order style modeling with constants, variables, predicates, and the usual logical connectives, e.g., **and** and **implies**.

### 5.3 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 [18]. To overcome this limitation OWL-S allows the use of more expressive languages like SWRL [14], 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 the capability descriptions.

## 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 [24]. WSML brings together different logical language paradigms and unifies them in one syntactical framework based on the principles of strict language layering and reusing proven reasoning techniques and tools. Unlike other proposals for Semantic Web and Semantic Web Service languages, WSML 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. WSML overcomes some of the layering issues of other proposals and recommendations for Semantic Web (Service) languages, as well as some of the limitations of other languages with respect to conceptual modeling [7, 19]. 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.

Future work for WSML consists of the application of the language to various use cases and the implementation of several WSML tools WSML tools, such as editors and reasoners. There are ongoing efforts to implement WSML reasoners on top of KOAN2[4], OntoBroker[5], and FLORA-2[6]. 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. Finally, 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 [16]. It might turn out, however, that different tasks need different operational semantics.

## Acknowledgements

We would like to thank all members of the WSML working group for their input to this document, and especially Eyal Oren for his comments on earlier versions of this document.

This work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, and ASG; by Science Foundation Ireland under the DERI-Lion project; by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects RW$^2$ and TSC.

## References

[1] A. Ankolekar et al. *OWL-S 1.1 Release*. 2004.

[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[3] P. V. Biron and A. Malhotra, editors. *XML Schema Part 2: Datatypes*. W3C Recommendation 28 October 2004.

[4] T. Bray, D. Hollander, A. Layman, and R. Tobin, editors. *Namespaces in XML 1.1*. W3C Recommendation 4 February 2004.

[5] D. Brickley and R. V. Guha. RDF vocabulary description language 1.0: RDF schema. W3C Recommendation 10 February 2004

[6] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel. The web service modeling language WSML. WSML Final Draft D16.1v0.2, 2005.

[4] http://kaon2.semanticweb.org/
[5] http://ontobroker.semanticweb.org/
[6] http://flora.sourceforge.net/

[7] J. de Bruijn, A. Polleres, R. Lara, and D. Fensel. OWL DL vs. OWL Flight: Conceptual modeling and reasoning on the semantic web. In *Proceedings of the 14th International World Wide Web Conference (WWW2005)*, Chiba, Japan, 2005. ACM.

[8] M. Dean and G. Schreiber, editors. *OWL Web Ontology Language Reference*. 2004. W3C Recommendation 10 February 2004.

[9] M. Duerst and M. Suignard. Internationalized resource identifiers (IRIs). RFC 3987, IETF, 2005.

[10] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. In *Proc. of the International Conference of Knowledge Representation and Reasoning (KR04)*, 2004.

[11] D. Fensel. *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, 2nd edition*. Springer-Verlag, Berlin, 2003.

[12] D. Fensel and C. Bussler. The web service modeling framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[13] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proc. Intl. Conf. of the World Wide Web (WWW-2003)*, Budapest, Hungary, 2003.

[14] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission 21 May 2004.

[15] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[16] U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic Location of Services. *ESWC*, 2005.

[17] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843, 1995.

[18] R. Lara, H. Lausen, S. Arroyo, J. de Bruijn, and D. Fensel. Semantic Web Services: description requirements and current technologies. *Semantic Web Services for Enterprise Application Integration and e-Commerce workshop (SWSEE03), in conjunction with ICEC 2003*, Pittsburgh, USA, 2003.

[19] R. Lara, A. Polleres, H. Lausen, D. Roman, J. de Bruijn, and D. Fensel. A conceptual comparison between WSMO and OWL-S. WSMO Final Draft D4.1v0.1, 2005.

[20] J. W. Lloyd. *Foundations of Logic Programming (2nd edition)*. Springer-Verlag, 1987.

[21] A. Malhotra, J. Melon, and N. Walsh. XQuery 1.0 and XPath 2.0 functions and operators. Working draft, W3C, 2005.

[22] 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.

[23] J. Z. Pan and I. Horrocks. OWL-Eu: Adding Customised Datatypes into OWL. In *Proc. of Second European Semantic Web Conference (ESWC 2005)*, 2005.

[24] D. Roman, H. Lausen, and U. Keller, editors. *Web Service Modeling Ontology (WSMO)*. 2005. WSMO Final Draft D2v1.2.

[25] D. Roman, J. Scicluna, and C. Feier, editors. *Ontology-based Choreography and Orchestration of WSMO Services*. WSMO Final Draft D15v0.1, 2005.

[26] G. Schreiber. The web is not well-formed. *IEEE Intelligent Systems*, 17(2), 2002. Contribution to the section Trends and Controversies: Ontologies KISSES in Standardization.