A Conceptual and Formal Framework for Semantic Web Services (v1)

Holger Lausen, Francisco Martin-Recuerda, Jos de-Bruijn, and Michael Stollberg (University of Innsbruck)

Abstract.
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB Deliverable D2.4.5v1 (WP2.4)

Current activities in WP 2.4 include web service discovery, composition and interoperation. Although significant contributions has been done, a conceptual and formal framework for Semantic Web services is required to provide a consensual grounding in which further progress in WP2.4 will be achieved. Following guidelines of SDK Cluster (http://www.sdkcluster.org/), the Web Service Modeling Ontology (WSMO) provides the conceptual underpinning and a formal language for semantically describing all relevant aspects of web services in order to facilitate the automatization of discovering, combining and invoking electronic services over the web. A revised version of this document will be submitted at the end of 2005.

Keyword list: web services, semantic web services, semantic web service discovery, semantic web service composition
Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

University of Innsbruck (UIBK) - Coordinator
Institute of Computer Science
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

École Polytechnique Fédérale de Lausanne (EPFL)
Computer Science Department
Swiss Federal Institute of Technology
IN (Eculbens), CH-1015 Lausanne
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

France Telecom (FT)
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Freie Universität Berlin (FU Berlin)
Takustrasse 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l’Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

National University of Ireland Galway (NUIG)
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

The Open University (OU)
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk
Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fl.upm.es

University of Karlsruhe (UKARL)
Institut für Angewandte Informatik und Formale Beschreibungsverfahren - AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Liverpool (UniLiv)
Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Manchester (UoM)
Room 2.32. Kilburn Building, Department of Computer Science, University of Manchester, Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

University of Trento (UniTn)
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Amsterdam (VUA)
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be
Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to writing parts of this document:

École Polytechnique Fédérale de Lausanne
Freie Universität Berlin
National University of Ireland Galway
Universität of Innsbruck
University of Manchester
University of Trento
## Changes

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>20.04.05</td>
<td>Holger Lausen</td>
<td>creation</td>
</tr>
<tr>
<td>0.3</td>
<td>30.05.05</td>
<td>Holger Lausen</td>
<td>restructuring</td>
</tr>
<tr>
<td>0.6</td>
<td>15.06.05</td>
<td>Francisco Martin-Recuerda</td>
<td>refinement</td>
</tr>
<tr>
<td>1</td>
<td>26.06.05</td>
<td>Holger Lausen</td>
<td>final version</td>
</tr>
</tbody>
</table>
Executive Summary

The deliverable provides a description of a conceptual and formal framework for Semantic Web services based on WSMO (http://www.wsmo.org/) and WSML (http://www.wsmo.org/wsml/) initiatives. We include a detailed description of both proposals and an overview of related initiatives.
# Contents

## 1 Web Service Modeling Ontology
1.1 Design Principles ........................................ 1  
1.2 WSMO from a birds view ................................. 2  
1.2.1 WSMO top level elements ............................ 2  
1.2.2 Meta-Object Facility ................................ 3  
1.3 Ontologies .................................................. 4  
1.3.1 Non-functional properties ............................. 5  
1.3.2 Imported Ontologies .................................... 5  
1.3.3 Used mediators ......................................... 6  
1.3.4 Concepts ............................................... 6  
1.3.5 Relations ................................................. 7  
1.3.6 Functions ............................................... 7  
1.3.7 Instances ............................................... 8  
1.3.8 Axioms ................................................... 8  
1.4 Services .................................................... 8  
1.4.1 Capability .............................................. 11  
1.4.2 Interfaces .............................................. 12  
1.5 Goals ....................................................... 14  
1.6 Mediators .................................................. 15  
1.6.1 WSMO Mediator Types ................................. 17  

## 2 WSML - A Language for WSMO
2.1 WSML Layering .............................................. 19  
2.2 General WSML Syntax ..................................... 21  
2.2.1 Identifiers in WSML .................................... 22  
2.2.2 Conceptual Syntax ..................................... 22  
2.2.3 Logical Expression Syntax ............................ 26  
2.2.4 WSML Web Syntaxes ................................... 29  
2.3 Key Features of WSML .................................... 29  
2.4 Conclusions and Future Work ............................. 30  

## 3 Related Work
3.1 Semantic Web Service Frameworks ....................... 32
CONTENTS

3.2 Languages for Semantic Web Services ........................................ 33

4 Conclusions and Future Work .................................................. 37
Chapter 1

Web Service Modeling Ontology

The Web Service Modeling Ontology (WSMO) aims at describing all relevant aspects related to general services which are accessible through a web service interface with the ultimate goal of enabling the (total or partial) automation of the tasks (e.g. discovery, selection, composition, mediation, execution, monitoring, etc.) involved in both intra- and inter-enterprise integration of web services. WSMO has its conceptual basis in the Web Service Modeling Framework (WSMF) [FB02], refining and extending this framework and developing a formal ontology and set of languages.

1.1 Design Principles

Semantic Web services aim at realizing the vision of the Semantic Web, i.e. turning the Internet from an information repository for human consumption into a world-wide system for distributed Web computing. Therefore, WSMO is based on the following design principles that integrate Web design principles, Semantic Web design principles, as well as design principles for distributed, service-oriented computing for the Web.

Web Compliance: WSMO inherits the concept of IRIs (Internationalized Resource Identifier) for unique identification of resources as the essential design principle of the Web. Moreover, WSMO adopts the concept of Namespaces for denoting consistent information spaces, and supports XML as well as other W3C Web technology recommendations.

Ontology-Based: Ontologies are used as the data model throughout WSMO, meaning that all resource descriptions as well as all data interchanged during service usage are based on ontologies. Following the idea of the Semantic Web, this allows semantically enhanced information processing as well as support for semantic interoperability.

Goal-driven Architecture: User requests are formulated as goals independently of available Web services. Thereby, the underlying epistemology of WSMO differentiates between the desires of clients and available Web services.
1. WEB SERVICE MODELING ONTOLOGY

**Strict Decoupling:** Each WSMO resource is specified independently, without regard to possible usage or interactions with other resources. This complies with the open and distributed nature of the Web.

**Centrality of Mediation:** Mediation addresses the handling of heterogeneities that naturally arise in open environments like the Web. As a complementary design principle to strict decoupling, WSMO recognizes the importance of mediation for the successful deployment of Web services by making mediation a first class component of the framework.

**Description versus Implementation:** WSMO differentiates between the *description* and the *implementation* of Web services. The former denotes unambiguous description of Web services that is needed for automated usage of Web services; the latter is concerned with the internal implementation of the Web Service which is not of interest for Semantic Web service technologies.

**Execution Semantics:** The formal execution semantics of reference implementations like the Web service Execution Environment WSMX as well as other WSMO-enabled systems verify the WSMO design and specification.

1.2 WSMO from a birds view

Before we introduce all WSMO concepts in detail we give a high level overview of the WSMO elements and explain the model underpinning the formalization chosen.

1.2.1 WSMO top level elements

Following the key aspects identified in the Web Service Modeling Framework, WSMO identifies four top level elements as the main concepts which have to be described in order to describe Semantic Web Services:

![Figure 1.1: WSMO core elements](image)

- **Ontologies** provide the terminology used by other WSMO elements to describe the

**Goals**

**Mediators**

**Web Services**
relevant aspects of the domains of discourse. WSMO describes an epistemology of ontologies, i.e. the conceptual building blocks like concepts and relations.

**Services** represent services that could be requested, provided or agreed by service requesters and service providers; These descriptions comprise the capabilities, interfaces and internal working of the service (as further described in section 1.4). For the description all these aspects of a service are described using the terminology defined by the ontologies.

**Goals** describe aspects related to user desires with respect to the requested functionality; again Ontologies can be used in order to define the used domain terminology to describe the relevant aspects of goals. Goals model the user view in the Web Service usage process and therefore are a separate top level entity in WSMO described in detail in section 1.5.

Finally, **Mediators** describe elements that handle interoperability problems between different elements, for example two different ontologies or services. We envision mediators as a core concept to resolve incompatibilities on the data, process and protocol level, i.e. in order to resolve mismatches between different used terminologies (data level), in how to communicate between services (protocol level) and on the level of combining web services (and goals) (process level). These are described in detail in section 1.6.

### 1.2.2 Meta-Object Facility

WSMO is meant to be a meta-model for Semantic Web Services related aspects. For introducing this model we decided to make use of Meta Object Facility (MOF)\(^1\) specification which defines an abstract language and framework for specifying, constructing, and managing technology neutral meta-models.

MOF defines a metadata architecture consisting of four layers, namely:

- The *information layer* comprises the data we want to describe.
- The *model layer* comprises the metadata that describes data in the information layer.
- The *meta-model layer* comprises the descriptions that define the structure and semantics of the metadata.
- The *meta-meta-model layer* comprises the description of the structure and semantics of meta-metadata.

In terms of the four MOF layers, the language in which WSMO is defined corresponds to the meta-meta model layer, WSMO itself constitutes the meta-model layer, the actual ontologies, services, goals, and mediators specifications constitute the model layer, and

\(^1\)http://www.omg.org/technology/documents/formal/mof.htm
the actual data described by the ontologies and exchanged between web services constitute the information layer. Figure 1.2 shows the relation between WSMO and the MOF layered architecture.

![Figure 1.2: The relation between WSMO and MOF.](image)

The most used MOF meta-modeling construct in the definition of WSMO is the *Class* construct (and implicitly its class generalization (*sub-Class*) construct), together with its *Attributes*, the *type* of the *Attributes* and their *multiplicity* specifications. When defining WSMO, the following assumptions are made:

- Every *Attribute* has its *multiplicity* set to multi-valued by default; when an *Attribute* requires its *multiplicity* to be set to ”single-valued”, this will be explicitly stated in the listings where WSMO elements are defined.

- Some WSMO elements define *Attributes* taking values from the union of several types, a feature that is not directly supported by the MOF meta-modelling constructs; this can be simulated in MOF by defining a new *Class* as *super-Class* of all the types required in the definition of the *Attribute*, representing the union of the single types, with the *Constraint* that each instance of this new *Class* is an instance of at least one of the types which are used in the union; to define this new *Class* in WSMO, we use curly brackets, enumerating the *Classes* that describe the required types for the definition of the attribute.

In the remainder of this chapter we use listings with the MOF metamodel to illustrate the structure of WSMO where it supports the understanding of the overall structure. The complete specification is available online.

### 1.3 Ontologies

An ontology is a formal explicit specification of a shared conceptualization [Gru93]. From this conceptual definition we extract the essential components which constitute an ontology. They define a common agreed upon terminology by providing concepts and relationships among the concepts.

Although there are currently several standardizations efforts for ontology languages [Hay04] [DS04] [HPSB+04], none of them has the the desired expressivity and computational properties that are required to describe web services at a sufficient level of granularity. In the following we will define an epistemological model which is general enough to intuitively capture existing languages. In the next chapter of this book we will present a concrete language specifically designed to express this meta model.

Now we present the conceptual model along with concrete examples. We introduce the elements that constitute an ontology using MOF notation, defining the class ontology with the following attributes:

```
Class ontology
    hasNonFunctionalProperty type nonFunctionalProperty
    importsOntology type ontology
    usesMediator type ooMediator
    hasConcept type concept
    hasRelation type relation
    hasFunction type function
    hasInstance type instance
    hasAxiom type axiom
```

Listing 1.1: Ontology Definition

The above listing specifies the building blocks for an ontology. Note that all attributes are optionally, thus a ontology can have multiple instances of all elements, but does not have to.

1.3.1 Non-functional properties

Non-functional properties are allowed in the definition of all WSMO elements, however the following discussion is applicable for all elements where non-functional properties are allowed. Note, that in some cases WSMO specifically recommends certain properties applicable to an element, but this does not impact the general considerations presented here. Non-functional Properties are mainly used to describe non-functional aspects such as creator, creation date, natural language descriptions. The elements defined by the Dublin Core Metadata Initiative [WKLW98] are taken as starting point. Dublin Core is a set of attributes defining a standard for cross-domain information resource description. As WSMO they use URIs for identifications of elements, which are reused. WSMO proposes several extension to this set by introducing new attributes such as the version element that for example can contain information about the particular version of an element.

1.3.2 Imported Ontologies

Building an ontology for some particular problem domain can be a rather cumbersome and complex task. One standard way to deal with the complexity is modularization. Im-
ported ontologies allow a modular approach for ontology design and can be used as long as no conflicts need to be resolved between the ontologies. By importing ontologies all statements of the imported ontology will be virtually included in the importing ontology. Every WSMO top level entity may use this import facility to include the logical definition of the vocabulary used.

### 1.3.3 Used mediators

When importing ontologies in realistic scenarios, some steps for aligning, merging and transforming imported ontologies in order to resolve ontology mismatches are needed. For this reason and in line with the basic design principles underlying the WSMF, ontology mediators (ooMediator), which are described in detail in Section 1.6 are used when an alignment of the imported ontology is necessary. Such an alignment can be for example the renaming of concepts, attributes or similar. Just like the `importsOntology` statement the `usesMediator` statement is applicable to all top level elements, however depending on the element different mediators may be used.

### 1.3.4 Concepts

Concepts constitute the basic elements of the agreed terminology for some problem domain. From a high level perspective, a concept - described by a concept definition - provides attributes with names and types. More formally specified in MOF a concept is made up of the following elements.

```plaintext
Class concept
  hasNonFunctionalProperties type nonFunctionalProperties
  hasSuperConcept type concept
  hasAttribute type attribute
  hasDefinition type logicalExpression multiplicity = single–valued
```

Listing 1.2: Concept Definition

Furthermore, a concept can be a subconcept of several (possibly none) direct superconcepts as specified by the "isA"-relation. In the WSMO model each concept can have a finite number of concepts that serve as a superconcept for some concept. Being a subconcept of some other concept in particular means that a concept inherits the signature of this superconcept and the corresponding constraints.

A concept provides a (possibly empty) set of attributes that represent named slots for the data values for instances. An attribute specifies a slot of a concept by fixing the name of the slot as well as a logical constraint on the possible values filling that slot, which in the simple case can be another concept. Hence, this logical expression can be interpreted as a typing constraint.
Additionally a concept is defined by a logical expression. That means axioms can be asserted to a concept that refines its meaning, e.g. with nuances that are not expressible by attributes or the “isA” hierarchy. A logical expression can be used to refine the semantics of the concept. More precisely, the logical expression defines (or restricts respectively) the extension (i.e. the set of instances) of the concept.

### 1.3.5 Relations

Relations are used in order to model interdependencies between several concepts (respectively instances of these concepts). The arity of relations is not limited.

```
Class relation
  hasNonFunctionalProperties type nonFunctionalProperties
  hasSuperRelation type relation
  hasParameter type parameter
  hasDefinition type logicalExpression multiplicity = single--valued
```

Listing 1.3: Relation Definition

Every Relation can have a finite set of relations of which the defined relation is declared as being a subrelation. Being a subrelation of some other relation in particular means that the relation inherits the signature of this superrelation and the corresponding constraints. Furthermore, the set of tuples belonging to the relation (the extension of the relation, respectively) is a subset of each of the extensions of the superrelations. In the example given we define air-line distance as a sub relation of the general distance relation.

Similar to attributes for concepts, each relation has a possible empty set of named parameters. In case no named parameters are given a unnamed, ordered list is assumed. Each parameter is single valued and can have a range restriction in form of a concept.

As for Concepts, a logicalExpression defining the set of instances (n-ary tuples, if n is the arity of the relation) can be specified. If the parameters are specified the relation is represented by a n-ary predicate symbol with named arguments where the identifier of the relation is used as the name of the predicate symbol. Also similar to concepts a logical expression can be used to define the extension of the relation.

### 1.3.6 Functions

A function is a special relation, with a unary range and a n-ary domain (parameters inherited from relation), where the range specifies the return value. Function can be used for instance to represent and exploit built-in predicates of common datatypes. Their semantics can be captured externally by means of an oracle or it can be formalized by assigning a logical expression to a particular relation. The logical representation of a function is almost the same as for relations and functions. A typical example for the use of oracles
1. WEB SERVICE MODELING ONTOLOGY

are so called built-ins. Those are function like "round(1.3)" that are evaluated outside the logical theory.

1.3.7 Instances

Instances are either defined explicitly or by a link to an instance store, i.e., an external storage of instances and their values.

```
Class instance
  hasNonFunctionalProperties type nonFunctionalProperties
  hasType type concept
  hasAttributeValues type attributeValue
```

Listing 1.4: Instance Definition

Instances of relations (with arity n) can be seen as n-tuples of instances of the concepts which are specified as the parameters of the relation.

In general instances do not need to be specified using an explicit notation as introduced above. Especially for the case when a huge number of instances exist a link to a data store can be used [KOK04]. Basically, the approach is to integrate large sets of instances which are already existing on some storage devices by means of sending queries to external storage devices or oracles.

1.3.8 Axioms

An axiom is considered to be a logical expression together with its non-functional properties. A more detailed discussion can be found in chapter 2, that explains a concrete language implementing this meta model.

1.4 Services

The Service element of WSMO provides a conceptual model (a meta model in MOF terms) for describing in an explicit and unified manner all the aspects of a service, including its non-functional properties, functionality, and the interfaces to obtain it. An unambiguous model of services with well-defined semantics can be processed and interpreted by computers without human intervention, enabling the automation of the tasks involved in the usage of web services e.g. discovery, selection, composition, mediation, execution or monitoring.

As observed in [Pre04], the word service can be understood in different ways, with slightly different meanings: as provision of value in some domain, as a software entity
able to provide something of value, and as a means of interacting online with a service provider.

WSMO provides a unifying view of a service; the value the service can provide is captured by its capability, and the means to interact with the service provider to request the actual performance of the service, or to negotiate some aspects of its provision, is captured by the service interfaces. The software entity able to provide the service is transparent to us, and we are only concerned with its interaction style and with what other services are used to actually provide the value described in the capability.

Notice that in WSMO the interaction with a service can be realized by using web services in the WSDL [CCMW01] sense. However, we are not restricted to WSDL as the grounding of services. Figure 1.3 below shows the core elements that are part of the description of a WSMO service.

![Figure 1.3: WSMO Service Description Overview.](image)

A service description consists of one capability, that describes the functional aspects of a service, non-functional properties, and one or more interfaces. An interface describes the choreography and the orchestration of the service. The choreography specifies how the service achieves its capability by means of interactions with its user - i.e. the communication with user of the service; the orchestration specifies how the service achieves its capability by making use of other services - i.e. the coordination of other services.
More precisely, the WSMO service element is defined as follows:

<table>
<thead>
<tr>
<th>Class</th>
<th>service</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasNonFunctionalProperty type</td>
<td>nonFunctionalProperty</td>
</tr>
<tr>
<td>importsOntology type</td>
<td>ontology</td>
</tr>
<tr>
<td>usesMediator type</td>
<td>(ooMediator, wwMediator)</td>
</tr>
<tr>
<td>hasCapability type</td>
<td>capability</td>
</tr>
<tr>
<td>multiplicity</td>
<td>single–valued</td>
</tr>
<tr>
<td>hasInterface type</td>
<td>interface</td>
</tr>
</tbody>
</table>

Listing 1.5: Service Definition

The non-functional properties of a service are aspects of the service that are not directly related to its functionality; besides the non-functional properties already presented, they consist of web service specific elements like Accuracy (the error rate generated by the service), Financial (the cost-related and charging-related properties of a service [OEtH02]), Network-related QoS (QoS mechanisms operating in the transport network which are independent of the service), Owner (the person or organization to which the service belongs), Performance (how fast a service request can be completed), Reliability (the ability of a service to perform its functions, i.e. to maintain its service quality), Robustness (the ability of the service to function correctly in the presence of incomplete or invalid inputs), Scalability (the ability of the service to process more requests in a certain time interval), Security (the ability of a service to provide authentication, authorization, confidentiality, traceability/auditability, data encryption, and non-repudiation), Transactional (the transactional properties of the service), Trust (the trust worthiness of the service), or Version\(^3\). The non-functional properties are to be mainly used for the discovery and selection of services; however they contain information that is also suitable for negotiation.

Imported Ontologies are used to import the explicit and formal vocabulary used in the specification of a service (see Section 1.3.2).

A service uses mediators in the following situations:

- when using heterogeneous terminologies and conflicts between them arise; in these cases, a service can import ontologies using ontology mediators (ooMediators), as explained in Section 1.3.3.

- when it needs to cope with process and protocol heterogeneity when interacting with other services. In this case a wwMediators is used. For a more detailed description of mediators, see Section 1.6.

The capability describes the real service provided e.g. booking of train tickets. A more detailed description of capabilities is given in Section 1.4.1.

An interface describes the interface of the web service to be used to achieve the described service. Further details are given in Section 1.4.2.

\(^3\)For a detailed description of the non-functional properties we refer to [LPR05].
1.4.1 Capability

The functionality offered by a given service is described by its capability; it is expressed by the state of the world before the service is executed and the state of the world after successful service provision. The service capability is meant primary for discovery and selection purposes i.e. the capability is used by the requester to determine whether the service meets its needs.

The definition of the capability is given below:

```plaintext
Class capability
  hasNonFunctionalProperty type nonFunctionalProperty
  importsOntology type ontology
  usesMediator type ooMediator
  hasPrecondition type axiom
  hasAssumption type axiom
  hasPostcondition type axiom
  hasEffect type axiom
```

The set of non-functional properties that can be attached to a capability is the one presented in Section 1.3.1. Imported Ontologies and used mediators are defined as in Section 1.3.2 and Section 1.3.3 respectively.

**Preconditions** in the description of the capability specify the required state of the information space before the service execution i.e. they specify what information a web service expects to provide its service. Preconditions constrain the set of states of the information space such that each state satisfying these constraints can serve as a valid starting state (in the information space) for executing the service in a defined manner.

**Assumptions** in the description of the capability describe the state of the world which is assumed before the execution of the service. Otherwise, the successful provision of the service is not guaranteed. As opposed to preconditions, assumptions are not necessarily to be checked by the service. We make this distinction in order to allow an explicit notion of conditions on the world state but outside the information space.

**PostConditions** in the description of the capability describe the state of the information space that is guaranteed to be reached after the successful execution of the service; it also describes the relation between the information that is provided to the service and its results.

The following example presents a postcondition saying that the information that the service provides is an instance of the confirmation concept, with the condition that the item that is confirmed is the trip initially requested.

**Effects** in the description of the capability describe the state of the world that is guaranteed to be reached after the the successful execution of the service i.e. if the preconditions and the assumptions of the service are satisfied.
1.4.2 Interfaces

An interface describes how the functionality of the service can be achieved (i.e. how the capability of a service can be fulfilled) by providing a twofold view on the operational competence of the service:

- choreography decomposes a capability in terms of interaction with the service.
- orchestration decomposes a capability in terms of functionality required from other services.

This distinction reflects the difference between communication and cooperation. The choreography defines how to communicate with the service in order to consume its functionality. The orchestration defines how the overall functionality is achieved by the cooperation of more elementary service providers.

The web service interface is meant primarily for behavioral description purposes of web services and is presented in a way that is suitable for software agents to determine the behavior of the service and reason about it; it might be also useful for discovery and selection purposes and in this description the connection to some existing web services specifications e.g. WSDL [CCMW01] could also be specified.

The definition of an interface is given below:

```
Class interface
  hasNonFunctionalProperty type nonFunctionalProperty
  importsOntology type ontology
  usesMediator type ooMediator
  hasChoreography type choreography
  hasOrchestration type orchestration
```

Listing 1.6: Interface Definition

The set of non-functional properties that can be attached to a capability is the one presented in Section 1.3.1. Imported Ontologies and used mediators are defined as in Section 1.3.2 and Section 1.3.3 respectively.

Choreography

WSMO Choreography deals with interactions of the Web service from the client’s perspective. We base the description of the behavior of a single service exposed to its client on the basic ASM model [Gur95]. WSMO Choreography interface descriptions inherit the core principles of such kind of ASMs, which summarized, are: (1) they are state-based, (2) they represents a state by a signature, and (3) it models state changes by transition rules that change the values of functions and relations defined by the signature of the algebra.
In order to define the signature we use a WSMO ontology, i.e. definitions of concepts, their attributes, relations and axioms over these. Instead of dynamic changes of function values as represented by dynamic functions in ASMs we allow the dynamic modification of instances and attribute values in the state ontology.

Taking the ASMs methodology as a starting point, a WSMO choreography is state-based and consists of three elements which are defined as follows:

```plaintext
Class choreography
  hasNonFunctionalProperties type nonFunctionalProperties
  hasStateSignature type stateSignature
  hasTransitionRules type transitionRules
```

Listing 1.7: Choreography Interface

**State Signature** The signature of the machine is defined by importing an ontology (possibly more than one) which defines the state signature over which the transition rules are executed, and a set of statements defining the modes of the concepts.

The state for the given signature of a WSMO choreography is defined by all ontology elements imported into the choreography. The elements that can change and that are used to express different states of a choreography, are instances of concepts and relations. These changes are expressed in terms of creation of new statements or changes to existing ones.

The concepts and relations of an ontology are marked to support a particular role (or mode). These roles are of five different types:

- **static** meaning that the extension of the concept cannot be changed. This is the default for all concepts and relations imported in the signature of the choreography.
- **controlled** meaning that the extension of the concept is changed only by a Web service instance.
- **in** meaning that the extension of the concept or relation can only be changed by the environment. A grounding mechanism for this item must be provided that implements write access for the environment.
- **shared** meaning that the extension of the concept or relation can be changed by the service and the environment. A grounding mechanism for this item may be provided that implements read/write access for the environment and the service.
- **out** meaning that the extension of the concept or relation can only be changed by the service. A grounding mechanism for this item must be provided that implements read access for the environment.

Since Web services deal with actual instance data, the classification inherits to instances of the respectively classified concepts and relations. I.e., instances of controlled...
concepts and relations can only be created and modified by the service, instances of in concepts can only be read by the service, instances of out concepts can only be created by the service but not read or further modified after its creation. Instances of shared concepts and relations are supposed to be read and written by both the service and possibly the environment, i.e. can also be modified after creation.

**Transition Rules**  The most basic form of rules deal with basic operations on instance data, such as adding, removing and updating instances to the signature ontology. To this end, we define the atomic update functions to add and delete, as well as a update instances, which allow us to add and remove instances to/from concepts and relations and add and remove attribute values for particular instances.

**Orchestration**

describes how the service makes use of other services in order to achieve its capability. In many real scenarios a service is provided by using and interacting with services provided by other applications or businesses. For example, the booking of a trip might involve the use of another service for validating the credit card and charging it with the correspondent amount and the user of the booking service may want to know with which other business organizations he is implicitly going to deal with.

WSMO introduces the orchestration element in the description of a service to reflect such dependencies. WSMO orchestration allows the use of statically or dynamically selected services. In the former case, a concrete service will be selected at design time. In the latter case, the service will only describe the goal that has to be fulfilled in order to provide its service. This goal will be used to select at run-time an available service fulfilling it (i.e. the service user could influence this choice).

**1.5 Goals**

Goals are used in WSMO to describe users desires. They provide the means to specify the requester-side objectives when consulting a web service, describing at a high-level a concrete task to be achieved.

Goals are representations of objectives for which fulfillment is sought through the execution of web services; they can be descriptions of services that would potentially satisfy the user desires.

Notice that WSMO completely decouples the objectives a requester has i.e. his goal, from the services that actually can fulfill such goal. Goals are to be resolved by selecting available services which described service provision satisfies the goal (see [KLP04] for more details).
The definition of a goal is given below:

<table>
<thead>
<tr>
<th>Class</th>
<th>goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasNonFunctionalProperty</td>
<td>type nonFunctionalProperty</td>
</tr>
<tr>
<td>importsOntology</td>
<td>type ontology</td>
</tr>
<tr>
<td>usesMediator</td>
<td>type {ooMediator, ggMediator}</td>
</tr>
<tr>
<td>requestsCapability</td>
<td>type capability multiplicity = single−valued</td>
</tr>
<tr>
<td>requestsInterface</td>
<td>type interface</td>
</tr>
</tbody>
</table>

Given the fact that a goal can represent the service that would potentially satisfy the user desires, the set of non-functional properties that can be attached to a goal is similar to the one attached to web services (see Section 1.4). An extra non-functional property can be attached to a goal, the Type of Match, which represents the type of match desired for a particular goal (under the assumption of a set based modelling this can be an exact match, a match where the goal description is a subset of the Web Service description or a match where the Web Service description is a subset of the goal description; for a detailed discussion refer to [KLP04]). A goal uses imported ontologies as the terminology to define the other elements that are part of the goal as long as no conflicts are needed to be resolved.

A goal uses mediators in the following situations:

- when using heterogeneous terminologies, conflicts between them might arise; in these cases, a service can import ontologies using ontology mediators (ooMediators), as explained in Section 1.3.2.

- when a goal reuses already existing goals, e.g. by refining them; for this, ggMediators are used (they are explained in more detail in Section 1.6).

The requested Capability in the definition of the goal describes the capability of the services the user would like to have.

The Interface in the definition of the goal describes the interface of the service the user would like to have and interact with.

### 1.6 Mediators

Mediation is concerned with handling heterogeneity, i.e. resolving possibly occurring mismatches between resources that ought to be interoperable. Heterogeneity naturally arises in open and distributed environments, and thus in the application areas of Semantic Web Services, WSMO defines the concept of Mediators as a top level notion.

Mediator-orientated architectures as introduced in [Wie94] specify a mediator as an entity for establishing interoperability of resources that are not compatible a priori by resolving mismatches between them at runtime. The aspired approach for mediation relies
on declarative description of resources whereupon mechanisms for resolving mismatches work on a structural, semantic level, in order to allow defining generic, domain independent mediation facilities as well as reuse of mediators. Concerning the needs for mediation within Semantic Web Services, the WSMF [FB02] defines three levels of mediation:

1. **Data Level Mediation** - mediation between heterogeneous data sources; within ontology-based frameworks like WSMO, this is mainly concerned with ontology integration.

2. **Protocol Level Mediation** - mediation between heterogeneous communication protocols; in WSMO, this mainly relates to choreographies of web services that are to interact.

3. **Process Level Mediation** - mediation between heterogeneous business processes; this is concerned with mismatch handling on the business logic level of web services (related to the orchestration of web services).

WSMO Mediators realize a mediation-orientated architecture for Semantic Web Services, providing an infrastructure for handling heterogeneities that possibly arise between WSMO components and realizing the design concept of strong decoupling and strong mediation. A WSMO Mediator connects WSMO components and resolves mismatches between them with the following specifies the general definition.

As a Mediator can be provided as a service, the same non-functional properties as for services are used (see Section 1.4 for what these non-functional properties consist of).

**ImportedOntologies** are used to import the explicit and formal vocabulary used in the specification of a mediator (see Section 1.3.2).

The source component of a mediator defines the resources wherefore heterogeneities are resolved; a mediator can have several source components.

The target component of a mediator is the component that receives the mediated source components.

The mediation service defines the mediation facility applied for resolving mismatches. This can be defined in different ways: directly (i.e. explicitly linking to a mediation service); via a goal that specifies the desired mediation facility which is then detected by a discovery mechanism; or via another mediator when a mediation service is to be used that is not interoperable to the mediator.
1.6.1 WSMO Mediator Types

In order to allow resolving heterogeneities between the different WSMO components, WSMO defines different types of Mediators for connecting the different WSMO components and overcome heterogeneities that can arise between the components: OO Mediators, GG Mediators, WG Mediators, and WW Mediators. All mediators are subclasses of the general WSMO Mediator class defined above, whereby a prefix indicates the components connected by the mediator type. The following explains the different WSMO Mediator types, while a general example for using mediators is provided in the next section.

**OO Mediators**

OO Mediators resolve mismatches between ontologies and provide mediated domain knowledge specifications to the target component. The source components are ontologies or other OO Mediators that are heterogeneous and to be integrated, while the target component is any WSMO top level notion that applies the integrated ontologies. The following shows the description specialization of an OO Mediator:

```
Class ooMediator sub Class mediator
hasSource type {ontology, ooMediator}
```

OO Mediators are used to import the terminology required for a resource description whenever there is a mismatch between the ontologies to be used. The mediation technique related to OO Mediators is mainly ontology integration, i.e. merging, aligning, and mapping ontology definitions in order to retrieve integrated, homogeneous terminology definitions.

**GG Mediators**

A GG Mediator connects goals, allowing to create a new goal from existing goals and thus defining goal ontologies. GG Mediators are defined as:

```
Class ggMediator sub Class mediator
usesMediator type ooMediator
hasSource type {goal, ggMediator}
hasTarget type {goal, ggMediator}
```

A GG Mediator might use an OO Mediator to resolve terminology mismatches between the source goals. Mediation services for GG Mediators reduce or combine the descriptions of the source goals into the newly created target goal.
WG Mediators

A WG Mediator links a Web Service to a Goal, resolves terminological mismatches, and states the functional difference (if any) between both. WG Mediators are defined as follows:

```plaintext
Class wgMediator sub Class mediator
  usesMediator type ooMediator
  hasSource type (service, wgMediator)
  hasTarget type (goal, ggMediator)
```

WG Mediators are used to pre-link Services to existing Goals, or for handling of partial matches within Web Service discovery. As within GG Mediators, OO Mediators can be applied for resolving terminological mismatches.

WW Mediators

A WW Mediator is used to establish interoperability between Web Services that are not interoperable a priori. Its definition in the language of WSMO is as follows:

```plaintext
Class wwMediator sub Class mediator
  usesMediator type ooMediator
  hasSource type (service, wwMediator)
  hasTarget type (service, wwMediator)
```

A WW Mediator mediates between the choreographies of Web Services that are ought to interact, wherefore mediation might be required on the data, the protocol, and the process level. As within the other WSMO mediator types, OO Mediators can be applied for resolving terminological mismatches.
Chapter 2

WSML - A Language for WSMO

In this chapter we introduce the Web Service Modeling Language (WSML), a family of formal representation languages with its roots in Description Logics, First-Order Logic and Logic Programming. The conceptual modeling elements of WSML are based on the meta-model of WSMO as presented before.

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 [BCM+03], Logic Programming [Llo87], and F-Logic [KLW95], 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 [GHVD03]. 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

This chapter is organized as follows: we give an overview of WSML and its language layering in Section 2.1. The normative human-readable syntax of WSML is described in Section 2.2. The key features of WSML are described in Section 2.3. Section 3.2 describes related approaches for the description of Semantic Web Services and Ontologies. We draw conclusions and outline future work in Section 2.4.

2.1 WSML Layering

Figure 2.1(a) shows the different variants of WSML and the relationships between them. These variants differ in logical expressiveness and in the underlying language paradigms

¹For the full WSML specification and other WSML-related resources we refer to http://www.wsmo.org/wsml/wsml-syntax
and allow users to make the trade-off between provided expressiveness and the implied complexity on a per-application basis.

**WSML-Core** is based on the intersection of the Description Logic $\mathcal{SHIQ}$ and Horn Logic, based on Description Logic Programs [GHVD03]. 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 $\mathcal{SHIQ}$(D), which is a major part of the (DL species of) OWL [DS04]. WSML-DL furthermore includes a datatype extension based on OWL-Eu [PH05] 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 [KLW95] 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 2.1(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).

2.2 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 2.1: An Example Prologue of a WSML File

As can be seen from the example in Listing 2.1, 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 2.2. We will first describe the use of Web identifiers and concrete data values in Section 2.2.1. The different kinds of WSML definitions and a general explanation of the conceptual syntax are given in Section 2.2.2. The logical expression syntax is described in Section 2.2.3. Finally, we briefly describe the XML and RDF serializations in Section 2.2.4.

KWEB/2005/D2.4.5/v1       June 26, 2005       21
2.2.1 Identifiers in WSML

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

An IRI (Internationalized Resource Identifier) [DS05] 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., [BHLT04]).

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 [BHLT04], 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. 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.

2.2.2 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.

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 2.2, and detail the elements below.
Concepts The notion of concepts (sometimes also called ‘classes’) plays a central role in ontologies [Fen03]. 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 2.2. 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 [BG04] and OWL [DS04]. 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 2.2). Logical expressions are explained in more detail in Section 2.2.3.

**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.

\(^2\)Pre-state (post-state, respectively) refers to the state before (after, respectively) the execution of the Web 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 2.3 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 2.3: 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 [RSF05].

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.
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 Web Service.

2.2.3 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 [KLW95] 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 $A \text{memberOf} B$ or a subconcept molecule of the form $A \text{subConceptOf} B$ with $A$ and $B$ arbitrary terms.
• An object molecule is an attribute value expressions of the form $A[B \text{ hasValue } C]$, a constraining attribute signature expression of the form $A[B \text{ ofType } C]$, or an inferring attribute signature expression of the form $A[B \text{ ofType } C]$, with $A,B,C$ arbitrary terms.

WSML has the usual first-order connectives: the unary negation operator $\text{neg}$, and the binary operators for conjunction $\text{and}$, disjunction $\text{or}$, right implication $\text{implies}$, left implication $\text{impliedBy}$, and dual implication $\text{equivalent}$. Variables may be universally quantified using $\forall x$ or existentially quantified using $\exists x$. 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:

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall x (\exists y (x \text{ memberOf } \text{Person} \implies \text{father hasValue } y))$</td>
<td>every person has a father</td>
</tr>
<tr>
<td>$\exists x, y (x \text{ memberOf } y \text{ with some attribute called 'name' implies } \exists x,y (\text{john memberOf } x \text{ and } x \text{ ofType } y))$</td>
<td>John is member of a class which has some attribute called 'name'</td>
</tr>
</tbody>
</table>

Listing 2.4: Example of First-Order Formulae in WSML

Apart from First-Order formulae, WSML allows the use of the negation-as-failure symbol $\text{naf}$ on atoms, the special Logic Programming implication symbol $\text{:-}$ and the integrity constraint symbol $\text{!-}$. A logic programming rule consists of a $\text{head}$ and a $\text{body}$, separated by the $\text{:-}$ symbol. An integrity constraint consists of the symbol $\text{!-}$ followed by a rule body. Negation-as-failure $\text{naf}$ 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: $\text{and}$, $\text{implies}$, $\text{impliedBy}$, and $\text{equivalent}$. The following connectives are allowed in the body of a rule (or constraint): $\text{and}$, $\text{or}$, and $\text{naf}$. The following are examples of LP rules and database constraints:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall x (\text{father hasValue f(?y)})$ : $\text{:-} x \text{ memberOf } \text{Person}$</td>
<td>every person has a father</td>
</tr>
<tr>
<td>$\text{!- } x \text{ memberOf } \text{Man} \text{ and } x \text{ memberOf } \text{Woman}$</td>
<td>Man and Woman are disjoint</td>
</tr>
<tr>
<td>$x \text{ memberOf } \text{Bachelor} : \text{:-} x \text{ memberOf } \text{Person} \text{ and } \text{naf } \text{Marriage}(x,y,z)$</td>
<td>in case a person is not involved in a marriage, the person is a bachelor</td>
</tr>
</tbody>
</table>

Listing 2.5: Example of LP Rule and Constraint in WSML

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 2.2.3 presents a number of language features and indicates in which variant the feature can occur.
2. WSML - A LANGUAGE FOR WSMO

<table>
<thead>
<tr>
<th>Feature</th>
<th>Core</th>
<th>DL</th>
<th>Flight</th>
<th>Rule</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical Negation (neg)</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Existential Quantification</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Disjunction</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Meta Modeling</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Default Negation (naf)</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>LP implication</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Integrity Constraints</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Function Symbols</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Unsafe Rules</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 2.1: WSML Variants and Feature Matrix

**WSML-Core** allows only first-order formulae which can be translated to the DLP subset of $SHIQ(D)$ [GHVD03]. 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 2.2.3 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 $SHIQ(D)$. 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.
2.2.4 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 [dB05] and have additionally specified an XML Schema for WSML. 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 [dB05]. 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.

2.3 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 [LPR05] 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 [BG04] and OWL [DS04]. Already in the case of RDFS and OWL, we can see that layering languages on existing recommendations is not straightforward [HPSvH03; dBPLF05]; 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 and 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 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 [GHVD03].

**WWW Language** WSML has a number of features which integrate it seamlessly in the Web. WSML adopts the IRI [DS05] 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 [BM04] and datatype functions and operators are based on the functions and operators of XQuery [MMW05]. 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 [KLW95] 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.

## 2.4 Conclusions and Future Work

In this chapter 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 that has been presented in Chapter 1. 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 [dBPLF05; LRPF04]. 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. From the language development point of view, the semantics of WSML-Full has not yet been defined; we are currently looking into several non monotonic 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 [KLP04]. It might turn out, however, that different tasks need different operational semantics.
Chapter 3

Related Work

Within this section we review related work in both areas: Semantic Web Service Frameworks and Semantic Web Service Languages.

3.1 Semantic Web Service Frameworks

In this section we will review related work in the area of Semantic Web Service Frameworks. Other major initiatives in the area of Semantic Web Service are OWL-S, METEOR-S, and IRS-II.

OWL-S [MBH+04], part of the DAML program\(^1\), is an ontology for service description based on the Web Ontology Language (OWL) [DS04]. The OWL-S ontology consists of the following three parts: a service profile for advertising and discovering services; a process model, which describes a service’s operation in detail; the grounding, which provides details on how to interoperate with a service, via messages. The vocabulary defined by OWL-S may be used to provide semantic annotations of services, and automatic agents may process this information. The following major differences arise between OWL-S and WSMO: in OWL-S the language specification layers are not clearly separated using an MOF style; OWL-S relies on OWL combined with different notations and semantics for expressing conditions, but combinations with SWRL [HPSB+04] or the syntactical framework of DRS [McD04] lead to undecidability problems and leave the semantics open, respectively, and when combining OWL with KIF [Gen98] it is not clear how both interact. WSMO directly provides a family of layered logical languages which combines conceptual modeling with rules; WSMO orchestrations describe what other services have to be used or what other goals have to be fulfilled to provide a higher level service, while OWL-S does not model this aspect; WSMO allows the definition of multiple interfaces and, therefore, choreographies for a Web service, while OWL-S only

\(^1\)http://www.daml.org/
allows a single service model for a web service i.e. a unique way to interact with it; OWL-S uses a single modeling element for representing requests and services provided, while WSMO explicitly separates them by defining goals and web service capabilities; OWL-S does not explicitly consider the heterogeneity problem in the language itself, treating it as an architectural issue i.e. mediators are not an element of the ontology but are part of the underlying web service infrastructure [PSS04].

METEOR-S aims at integrating web service technologies such as Business Process Execution Language for Web Services (BPEL4WS) [ACD+03], Web Service Description Language (WSDL) [CCMW01] and Universal Description, Discovery and Integration (UDDI) [BCE+02] with Semantic Web technologies in order to automate the tasks of publication, discovery, description, and control flow of web services. Compared to WSMO, METEOR-S follows a much more technology centered approach, not providing a conceptual model for the description of services and their related aspects.

IRS-III The Internet Reasoning Service III (IRS-III) [DCH+04] is a Semantic Web Services framework, which allows applications to semantically describe and execute web services. Compared to IRS-III, WSMO focuses more on the description elements that are needed to deal with Semantic Web Service. Conceptually, WSMO and IRS-III are not too different in the sense that both have common roots in UPML [FBD+99].

3.2 Languages for Semantic Web Services

Although the choice for particular framework might already imply a choice for a particular formalism, we review related work in this area in a separated section (indeed WSMO claims to be language independent, while e.g. OWL-S allows for the syntactical plug-in for a variant of different languages.

RDFS RDFS [BG04] 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.
3. RELATED WORK

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 [Sch02; dBPLF05]. 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.

**OWL** The Web Ontology Language OWL [DS04] 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 \( SHIF(D) \); OWL DL is a notational variant of the Description logic \( SHOIN(D) \) [HPSvH03]. It turns out that OWL DL adds very little in expressiveness to OWL Lite [HPSvH03]. 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 [dBPLF05].

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 \( \text{IntersectionOf} \) and \( \text{SubClassOf} \), denoting concept intersection and concept subsumption, respectively. WSML uses first-order style modeling with constants, variables, predicates, and the usual logical connectives, e.g., \( \text{and} \) and \( \text{implies} \).

**SWRL** The Semantic Web Rule Language [HPSB⁺04] is an extension of OWL which adds support for Datalog syntax-style rules over OWL DL ontologies. Instead of arbitrary predicates (as in Datalog), SWRL allows arbitrary OWL DL descriptions in both the head and the body of rules, where a unary predicate corresponds to an OWL class and a binary predicate corresponds to an OWL property. While a subset of SWRL falls inside Horn Logic, a SWRL knowledge base easily goes beyond this fragment, because of the use of classical negation and existentially quantified variables and disjunction in the head of the rule. A set of Horn Logic formulae can be reduced to standard Logic Programming rules; the Horn Logic formulae and the Logic Programming rules entail exactly the same set
of ground formulae. Consequently, SWRL and standard rule languages differ in expressiveness. The advantage of common rule languages which are based on Horn Logic is the efficient reasoning support which has been developed for certain reasoning tasks like query answering. By going beyond the Horn fragment, SWRL loses this advantage.

SWRL is an expressive extension of OWL DL, which add important features to the standard, but is not a standard rule language, because OWL DL is not a rule language and only a subset can be translated to rules.

**SWSL** The Semantic Web Service Language is another recent proposal for a language for describing Semantic Web Services. SWSL has two parts, a rules language and a process ontology. The SWSL-Rules sublanguage is closely related to WSML-Rule. Both languages are largely based on F-logic and they mostly share the logical expression syntax. However, the two groups have pursued complementary goals. WSMO was focused on the end user and developed a "conceptual syntax" for top-level descriptions of services, which we believe might make the specifications easier to read. WSMO also paid special attention to the issue of OWL compatibility. To this end, we defined WSML-Core as a subset of both OWL and WSML, which will serve as a common ground for ontology interoperability. In contrast, SWSL's focus was on extending the functionality of their rule-based language. In particular, SWSL-Rules supports meta-reasoning with its HiLog and reification extensions. It also supports prioritized defaults and classical negation by incorporating Courteous Logic Programming.

On the process description front, WSMO and SWSL are more divergent, but they nonetheless cover complementary parts of the problem space. WSMO has focused on describing Web service choreography through ECA rules, which are viewed as abstract state machines. In contrast, SWSL has developed a first-order PSL-based process ontology, which allows the description of process orchestration as well as message exchange among processes. Further work is needed to determine to what extent the two approaches can be combined.

**KIF** The Knowledge Interchange Format (KIF) is a standards-proposal from the 1990s for the interchange of knowledge between knowledge bases. The language is constructed in such a way that it can be used to capture the semantics of most knowledge bases. As such, it is an extension of the first-order logic with reification. KIF currently only has a normal text syntax and thus each KIF expression in an OWL-S description consists of text and thus does not benefit from validation and parsing services offered by XML and RDF parsers/validators.

**DRS** DRS (Declarative RDF System) is an OWL ontology, which provides a vocabulary for writing down arbitrary formulas. DRS does not prescribe the semantics of formulas written down using its vocabulary. Thus, when using DRS to specify Web Services,
the user will have to find a way outside the language to agree on the semantics of the description.
Chapter 4

Conclusions and Future Work

In this deliverable we have presented the Web Service Modeling Ontology and the Web Service (WSMO) and the Web Service Modeling Language (WSML). Both together can be used as a Conceptual and Formal Framework for Semantic Web Services.

WSMO defines the conceptual elements necessary for the description of Services. It provides an Ontology that can be expressed using different languages. Despite the Mediators, Goals and Services also some emphasis has been made to define a meta model for ontologies, this allows the framework to be more flexible then if it would commit already on that meta model to a specific language.

WSML is a language developed for expressing the the meta model described in WSMO. 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. In difference to other proposals it does not directly layer on top of existing recommendation like OWL [DS04] but defines a mapping to them, having the advantage of layering that also allows to integrate Logic Programming based languages.

Future work in WSMO will be to define more clearly the notion of choreography and orchestration allowing the framework to express descriptions necessary for process mediation and composition.

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. 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 [KLP04]. It might turn out, however, that different tasks need different operational semantics.
Bibliography


