Web Service Modeling Ontology

Dumitru Roman[1,⋆], Uwe Keller[1], Holger Lausen[1], Jos de Bruijn[1], Rubén Lara[1], Michael Stollberg[1], Axel Polleres[1], Cristina Feier[1], Cristoph Bussler[2], and Dieter Fensel[1,2]

[1] Digital Enterprise Research Institute Innsbruck (DERI Innsbruck), Institute for Computer Science, University of Innsbruck, Technikerstrasse 13, A-6020 Innsbruck, Austria
[2] Digital Enterprise Research Institute (DERI), University Road, Galway, Ireland

Abstract: The potential to achieve dynamic, scalable and cost-effective marketplaces and eCommerce solutions has driven recent research efforts towards so-called Semantic Web Services, that are enriching Web services with machine-processable semantics. To this end, 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.

In this paper we describe the overall structure of WSMO by its four main elements: *ontologies*, which provide the terminology used by other WSMO elements, *Web services*, which provide access to services that, in turn, provide some value in some domain, *goals* that represent user desires, and *mediators*, which deal with interoperability problems between different WSMO elements. Along with introducing the main elements of WSMO, we provide a logical language for defining formal statements in WSMO together with some motivating examples from practical use cases which shall demonstrate the benefits of Semantic Web Services.

Keywords: *Semantic Web services, Ontologies, Semantic Web, Web services*

⋆ Corresponding author - *Email address:* dumitru.roman@deri.org; *Phone:* +43 512 507 6463; *Fax:* +43 512 507 9872

1 Introduction

Web services [1] have added a new level of functionality to the current Web by taking a first step towards seamless integration of distributed software components using web standards. Nevertheless, current Web service technologies around SOAP [20], WSDL [10] and UDDI [5] operate at a syntactic level and, therefore, although they support interoperability (i.e. interoperability between the many diverse application development platforms that exist today) through common standards, they still require human interaction to a large extent: The human programmer has to manually search for appropriate Web services in order to combine them in a useful manner, which limits scalability and greatly curtails the added economic value of envisioned with the advent of Web services [18].

Recent research aimed at making web content more machine-processable, usually subsumed under the common term *Semantic Web* [7] are gaining momentum also, in particular in the context of Web services usage. Here, semantic markup shall be exploited to automate the tasks of Web service discovery, composition and invocation, thus enabling seamless interoperation between them [43] while keeping human intervention to a minimum.

The description of Web services in a machine-understandable fashion is expected to have a great impact in areas of e-Commerce and Enterprise Application Integration, as it is expected to enable dynamic and scalable cooperation between different systems and organizations: Web services provided by cooperating businesses or applications can be automatically located based on another business or application needs, they can be composed to achieve more complex, added-value functionalities, and cooperating businesses or applications can interoperate without prior agreements custom codes. Therefore, much more flexible and cost-effective integration can be achieved.

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) [18], refining and extending this framework and developing a formal ontology and set of languages.

The remainder of the paper is organized as follows. In Section 2 a general overview of WSMO is given. The top level elements of WSMO, namely Ontologies, Web services, Goals and Mediators are presented in detail in the subsequent Sections 3 to 6. Section 7 presents a formal language for defining logical statements in WSMO. Finally, Section 8 summarizes existing work in the area of Semantic Web Services and relates this work to WSMO, and Section 9 presents our conclusions and plans for further research.

2 WSMO: A Bird's-Eye View

WSMO provides ontological specifications for the core elements of Semantic Web services. In fact, Semantic Web services aim at an integrated technology for the next generation of the Web by combining Semantic Web technologies and Web services, thereby turning the Internet from a information repository for human consumption into a world-wide system for distributed Web computing. Therefore, appropriate frameworks for Semantic Web services need to integrate the basic Web design principles, those defined for the Semantic Web, as well as design principles for distributed, service-oriented computing of the Web. WSMO is therefore based on the following design principles:

- **Web Compliance** - WSMO inherits the concept of URI (Universal Resource Identifier) for unique identification of resources as the essential design principle of the Word Wide Web. Moreover, WSMO adopts the concept of Namespaces for denoting consistent information spaces, supports XML and other W3C Web technology recommendations, as well as the decentralization of resources.
- **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. Ontologies are a widely accepted state-of-the-art knowledge representation, and have thus been identified as the central enabling technology for the Semantic Web. The extensive usage of ontologies allows semantically enhanced information processing as well as support for interoperability; WSMO also supports the ontology languages defined for the Semantic Web.
- **Strict Decoupling** - Decoupling denotes that WSMO resources are defined in isolation, meaning that each 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** - As a complementary design principle to strict decoupling, mediation addresses the handling of heterogeneities that naturally arise in open environments. Heterogeneity can occur in terms of data, underlying ontology, protocol or process. WSMO recognizes the importance of mediation for the successful deployment of Web services by making mediation a first class component of the framework.
- **Ontological Role Separation** - Users, or more generally clients, exist in specific contexts which will not be the same as for available Web services. For example, a user may wish to book a holiday according to preferences for weather, culture and childcare, whereas Web services will typically cover

airline travel and hotel availability. The underlying epistemology of WSMO differentiates between the desires of users or clients and available services.

- **Description versus Implementation** - WSMO differentiates between the descriptions of Semantic Web services elements (description) and executable technologies (implementation). While the former requires a concise and sound description framework based on appropriate formalisms in order to provide concise semantic descriptions, the latter is concerned with the support of existing and emerging execution technologies for the Semantic Web and Web services. WSMO aims at providing an appropriate ontological description model, and to be compliant with existing and emerging technologies.
- **Execution Semantics** - In order to verify the WSMO specification, the formal execution semantics of reference implementations like WSMX as well as other WSMO-enabled systems provide the technical realization of WSMO.
- **Service versus Web service** - A Web service is a computational entity which is able to achieve a users goal by invocation. A service, in contrast, is the actual value provided by this invocation [4], [38] [3]. WSMO provides means to describe Web services that provide access (searching, buying, etc.) to services. WSMO is designed as a means to describe the former and not to replace the functionality of the latter.

The rest of this section gives a general overview of WSMO: the top level elements of WSMO are succinctly introduced in Section 2.1 and their conceptual relations are explained; in Section 2.2, by making use of the Meta Object Facility (MOF), the meta-language used for describing WSMO is presented. An example

[3] Note that [38] also distinguishes between a computational entity in general and Web service, where the former does not necessarily have a Web accessible interface. WSMO does not make this distinction.

is given in Section 2.3 that will be used throughout this document in order to define various elements of WSMO.

2.1 Top-level elements of WSMO

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 define Semantic Web Services:

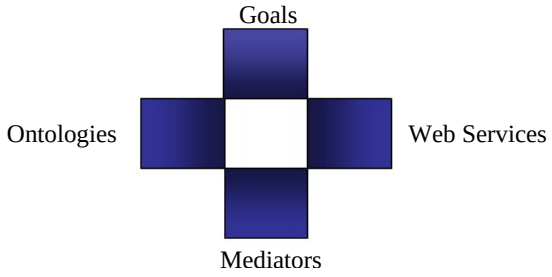

Fig. 1. WSMO core elements

Ontologies provide the terminology used by other WSMO elements to describe the relevant aspects of the domains of discourse. In contrast to mere terminologies that focus exclusively on syntactic aspects, ontologies can additionally provide formal definitions that are machine-processable and thus allow other components and applications to take actual meaning into account. They are described in detail in Section 3.

Web services represent computational entities able to provide access to services that, in turn, provide some value in a domain; Web service descriptions comprise the capabilities, interfaces and internal working of the service (as further described in Section 4). All these aspects of a Web 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 to define the used domain terminology, useful in describing 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 5.
Finally, **Mediators** decribe elements that handle interoperability problems between different WSMO elements. We envision mediators as the 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 Web services (protocol level) and on the level of combining Web services (process level). These are described in detail in Section 6.

2.2 Language for defining WSMO

WSMO is meant to be a meta-model for Semantic Web Services related aspects. For defining this model we make use of Meta Object Facility (MOF) [35] 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, Web services, goals, and mediators specifications constitute the model layer, and the actual data described by the ontologies and exchanged between Web services constitute the information layer. Figure 2 shows the relation between WSMO and the MOF layered architecture.

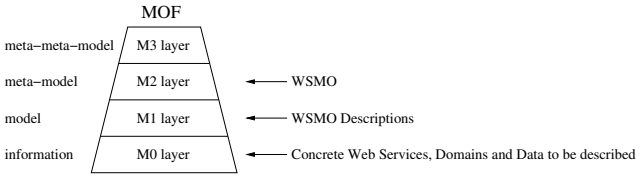

Fig. 2. The relation between WSMO and MOF.

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 paper we use listings with the MOF metamodel to illustrate the structure of WSMO where it supports the understanding of the overall structure. To be brief, some listings are shortened or omitted; the complete specification of WSMO in terms of MOF can be found in [41].

2.3 Illustrating Example

In order to illustrate WSMO we will briefly describe an application scenario based on [42]. Let us imagine a "Virtual Traveling Agency" (VTA for short) which is a platform providing eTourism services. These services can cover information services concerned with tourism such as events and sights in different areas and services that support booking of flights, hotels, rental cars, etc. By applying Semantic Web Services, a VTA can invoke Web services provided by several eTourism suppliers and aggregate them into new customer services in a (semi-)automatic fashion. In this paper we only focus on the description of a particular Web service in this scenario, we do not aim to explain the full application of such descriptions. As a concrete example we chose a service that is able to book and reserve a train ticket. The full example along with the supporting ontologies is available online[4]. When introducing the components of WSMO, we will occasionally refer to parts of this service description in the Web Service Modeling Language WSML, which will be described in more detail in Section 7.

3 Ontologies

An ontology is a formal explicit specification of a shared conceptualization [21]. From this conceptual definition we extract the essential components which constitute an ontology. They define an agreed common terminology by providing concepts, and relationships between the concepts.

[4] The full listing is available at http://cvs.deri.at/paper/wsmoExample.wsml

Although there are currently several standardizations efforts for ontology languages [23] [13] [24], none of them has 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. The semantics of this model are defined by logical expressions (cf. Section 7).

In the following we will present the conceptual model along with concrete examples. We will now 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

In the subsections which follow, we will describe all elements in more detail. The examples used for illustration purposes are given in WSML [12], a language specifically designed to express semantic descriptions according to the WSMO meta model. Explained in a nutshell, while WSMO provides the conceptual model for describing Web services, ontologies, goals and mediators semantically, WSML provides a formal language for writing, storing and communicating such descriptions. Although other concrete languages might be used to express our model too, we chose WSML for its close relationship to the meta model. A complete description of the language is out of the scope of this paper, but we will give an overview and some details on the logical language used in Section 7; necessary explanations are given throughout the text. But first, some general remarks are in order: WSML identifiers are URIs, for readability they are abbreviated using the QName [8] mechanism. URIs in the default namespace do

not need a prefix, other namespaces will be introduced in the explanatory text sections. A QName is written in the format prefix:localPart and will be expended to the full URI during processing.

3.1 Non-functional properties

Non functional properties are allowed in the definition of all WSMO elements. They are mainly used to describe non-functional aspects such as creator, creation date, natural language descriptions, etc. We take the elements defined by the Dublin Core Metadata Initiative [45] as a starting point and introduce other elements, for example the version of the ontology (other elements necessary for the description of other elements of WSMO, e.g web services, are introduced in their corresponding sections).

```
namespace <<http://example.org/wsmo#>>
  loc: <<http://wsmo.org/ontologies/location#>>
  xsd: <<http://www.w3.org/2001/XMLSchema#>>
  dc: <<http://purl.org/dc/elements/1.1#>>

ontology <<http://example.org/tripReservationOntology>>
  nonFunctionalProperties
    dc: title hasValue "An ontology describing trips and reservations"
    dc: creator hasValue "DERI Innsbruck"
    dc: publisher hasValue "DERI International"
    dc: contributor hasValues {<<www.deri.org/members/dumitrur>>,
      <<http://homepage.uibk.ac.at/~c703240/foaf.rdf>>}
    dc:date hasValue "2004-12-16"
    dc:format hasValue "text/x-wsml"
  endNonFunctionalProperties}
```

The example above illustrates the use of namespace declaration to QNames for readability. Note that those identifiers are only logical identifiers and not physical ones. The metadata in the non-functional properties can also refer to URIs (in case of dc:contributor to a foaf file or a homepage).

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 by modularization. Imported 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 (cf. Section 2.1) may use this import facility to include the logical definition of the vocabulary used.

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)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.

An example for mediator usage for terminology import is that for describing some WSMO element we need to merge a train connection ontology $\mathcal{O}_{ts}$ and a purchase ontology $\mathcal{O}_{pur}$, so that a train ticket in $\mathcal{O}_{ts}$ is understood as a subconcept of product in $\mathcal{O}_{pur}$. Defining a OO Mediator that merges $\mathcal{O}_{ts}$ and $\mathcal{O}_{pur}$ accordingly allows to apply the merged ontologies as the terminology definition for the element description. The concept of Mediators is described in detail in Section 6.

3.4 Concepts

Concepts constitute the basic elements of the agreed terminology for some problem domain. They represent classes of objects of a real or abstract world that have a specific shared property (e.g. being a ticket). Members of such a class are called *instances* of the corresponding concept. Formally, concepts are interpreted under set-theoretic semantics, that means they formally represent sets of elements.

The description of the single members of such a concept usually reveals a specific structure: attributes that are shared between all instances of a concept. Hence, from a high level perspective, a concept - described by a concept definition - provides attributes with names and types. Such a set of attribute definitions, i.e. pairs of attribute names and types, which apply to a specific concept C is called a *signature* of concept C. In particular, such definitions could for instance stem from imported definitions of a concept or its superconcepts.

Furthermore, a concept can be a subconcept of several (possibly none) direct superconcepts as specified by the "isA"-relation. Formally, this declares a specific relationship between the extensions of a concept and its superconcept: every instance of the subconcept is considered to be an instance of the superconcept as well. Hence, subconcepts refine their superconcepts in some way thus add some specific meaning to the superconcept which distinguishes the instances of the subconcept from the instances of the respective superconcepts.

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. That means for each concept C all attributes of all superconcepts can be applied to instances of C as well. Constraints (i.e. typing of attributes) must be refined. A detailed account to

inheritance in object-oriented and frame-based logics can be found in [28,47,48]. In the example below, a trip in Europe is defined as subconcept of a general trip; note that the attributes are repeated only for the reason of readability and would have been inherited anyway.

```
concept tripInEurope subConceptOf trip
  origin ofType loc: location
  destination ofType loc: location
  departure ofType xsd:dateTime
  arrival ofType xsd:dateTime
  definedBy
    forAll ?x (?x memberOf tripInEurope equivalent
      ?x memberOf trip and
      ?x. departure . loc : locateIn =loc:europe and
      ?x. origin . loc : locateIn =loc:europe).
```

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. Within the example the domain of the possible attribute values for *origin* is restricted to instances of the concept location (loc: is used as an abbreviation for the full logical identifer of this external ontology). Note that in WSMO/WSML we do not restrict ourselves to typing constraints, but also allow the types of slot fillers to be implied by the definition. This can be modeled in WSML by means of the keyword `impliesType` replacing `ofType`. An in-depth discussion of attributes and their semantics, in particular the difference between the two options `ofType` and `impliesType` in WSML, can be found in [9,12].

As every element in WSMO, a concept is ultimately defined by a logical expression it translates to. Additionally, within the conceptual model, axioms can be asserted to a concept that refines its meaning, e.g. with nuances that are not expressible by attributes or the concept hierarchy. A logical expression (cf.

Section 7) can be used to refine the semantics of the concept. More precisely, the logical expression defines (or restricts, as the case may be) the extension (i.e. the set of instances) of the concept. In the example we are using for illustration purposes here, the expression refines a trip within Europe, i.e. it restricts all attribute values of origin and destination to locations that have a *locatedIn* attribute value indicating that they are located in Europe.

More generally, we allow the following type of expression for concept definitions: If C is the identifier denoting the concept then the logical expression takes one of the following forms:

forAll ?x (?x **memberOf** C **implies** $lexpr$(`?x`))
forAll ?x (?x **memberOf** C **impliedBy** $lexpr$(`?x`))
forAll ?x (?x **memberOf** C **equivalent** $lexpr$(`?x`))

where $lexpr$(`?x`) is a logical expression with precisely one free variable `?x`. In the first case, there is a necessary condition for membership in the extension of the concept; in the second case, there is a sufficient condition and in the third case, there is a sufficient and necessary condition for an object being an element of the extension of the concept.

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.

relation airLineDistance **subRelationOf** distance
  from **ofType** loc: location
  to **ofType** loc: location
  distanceInMeter **ofType** xsd: integer

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 (i.e. the extension of the relation) is a subset of each of the extensions of the superrelations. In the example given, we define airline distance as a sub relation of the general distance relation.

Like attributes for concepts, each relation has a possible empty set of named parameters. If no named parameters are given, an unnamed, ordered list is assumed. Each parameter has a single value, and can have a range restriction in form of a concept. Mixing named and unnamed parameters is not possible when defining a relation; either all parameters are named or all of them are named.

In the case of concepts a logical expression 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. If R is the identifier denoting the relation, and *impl* stands for implies, impliedBy or equivalent, then the logical expression takes one of the following forms:

forAll ?v1 ,...,? vn (R[p1 **hasValue** ?v1 ,..., pn **hasValue** ?vn]
impl *lexpr*(?v1,...,?vn))
forAll ?v1 ,...,? vn (R(?v1 ,...,? vn)
impl *lexpr*(?v1,...,?vn))

where *lexpr*(`?v1,...,?vn`) is a logical expression with precisely `?v1,...,?vn` as its free variables and `p1,...,pn` are the names of the parameters of the relation.

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 the definedBy property inherited from Relation.

The logical representation of a function is almost the same as for relations, whereby the result value of a function (i.e. the value of a function term) has to be represented explicitly: the function is represented by an (n+1)-ary predicate symbol with named arguments (where n is the number of arguments of the function) where the identifier of the function is used as the name of the predicate. In particular, the names of the parameters of the corresponding relation symbol are the names of the parameters of the function along with one additional parameter range for denoting the value of the function term with the given parameter values. If the parameters are not specified, the function is represented by a predicate symbol with ordered arguments, and by convention the first argument specifies the value of the function term with given argument values. The structure of the logical expression defining a function is inherited from Relation augmented with one additional parameter "range".

function kmToMiles
  km **ofType** xsd:integer
  range **ofType** xsd:integer
  definedBy
    forAll ?x,?y(
      kmToMiles(km **hasValue** ?x, range **hasValue** ?y) **equivalent**
        ?y = (?.x ∗ 0.621371192).

kmToMiles is an example for a function that constructs the integer value of miles corresponding to a given value in km. As with Relations, the name and the parameters are defined. Within the logical expression, built-in functions are used to perform operations on basic data types in this case, ∗ is used for the multiplication of numbers.

3.7 Instances

A concept represents a set of objects in a real or abstract world with a specific shared property. The objects themselves are called *instances*. The description of the single instances of a concept follows a common pattern: the signature imposed by the concept definition.

Instances are either defined explicitly or by a link to an instance store, i.e., an external storage of instances and their values. An explicit definition of instances of concepts is as follows:

```
instance Innsbruck memberOf loc:location
  locatedIn hasValue loc: austria
```

Besides the identifier of the instance (*Innsbruck*) the concept and the attribute values are given. These values have to be compatible with the corresponding type declaration in the concept 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 the explicit notation presented above. Especially where a huge number of instances exist, a link to a data store can be used [29]. Basically, the approach is to integrate large sets of instances which already exist on some storage device by sending queries to external storage devices or oracles.

3.8 Axioms

An axiom is considered to be a logical expression together with its non functional properties. A detailed discussion of axioms can be found in Section 7. Examples of logical expression have already been used and intuitively explained in the examples above.

4 Web services

The *Web 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 Web service, including its non-functional properties, its functionality, and the interfaces to obtain it. An unambiguous model of Web 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.

A WSMO Web service is a computational entity which is able (by invocation) to achieve a users goal. A service in contrast is the actual value provided by this invocation. Thereby a Web service might provide different services, such as for example Amazon can be used for acquiring books as well as to find out an ISBN number of a book.

Note that in WSMO the interaction with a service can be accomplished by using Web services in the WSDL [10] sense. However, we are not restricted to WSDL, but can use other methods also to interact with the service.

Figure 3 below shows the core elements that are part of the description of a WSMO Web service.

The main elements of a Web service description are: a capability describing the functionality of the Web service, and one or more interfaces in which the choreography and the orchestration of the Web service are described. The choreography specifies how the Web service achieves its capability by means of interactions with its user - i.e. the communication with the user of the Web service; the orchestration specifies how the service achieves its capability by making use of other services - i.e. the coordination of other Web services.

More precisely, the WSMO *Web service* element is defined as follows:

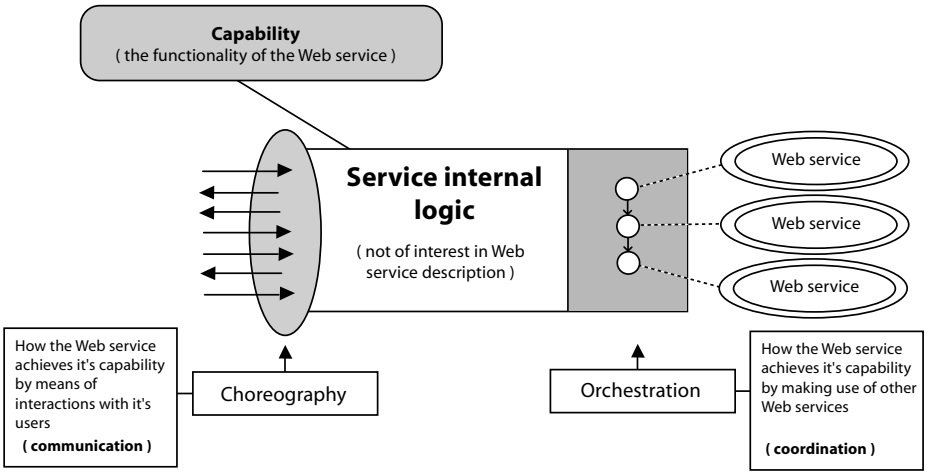

Fig. 3. WSMO Web service - general description.

Class webService
hasNonFunctionalProperty **type** nonFunctionalProperty
importsOntology **type** ontology
usesMediator **type** {ooMediator, wwMediator}
hasCapability **type** capability **multiplicity** = **single−valued**
hasInterface **type** interface

The **non functional properties** of a Web service are aspects of the Web service that are not directly related to its functionality; besides the non functional properties presented in Section 3.1, they consist of Web service specific elements like the following: Accuracy (the error rate generated by the Web service), Financial (the cost-related and charging-related properties of a service [36]), Network-related QoS (QoS mechanisms operating in the transport network which are independent of the Web service), Owner (the person or organization to which the Web service belongs), Performance (how fast a Web service request can be completed), Reliability (the ability of a Web service to perform its functions, i.e. to maintain its Web service quality), Robustness (the ability of the Web service to function correctly in the presence of incomplete or invalid inputs), Scalability (the ability of the Web service to process more requests in a

certain time interval), Security (the ability of a Web service to provide authentication, authorization, confidentiality, traceability/auditability, data encryption, and non-repudiation), Transactional (the transactional properties of the Web service), Trust (the trust-worthiness of the Web service), or Version[5] . The non functional properties are to be mainly used for the discovery and selection of Web 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 Web service (see Section 3.2).

A Web service **uses mediators** in the following situations:

- when using heterogeneous terminologies and conflicts between them arise; in these cases, a Web service can import ontologies using ontology mediators (ooMediators), as explained in Section 3.3.
- when it needs to cope with process and protocol heterogeneity when interacting with other Web services. In this case a *wwMediators* is used. For a more detailed description of mediators, see Section 6.

The **capability** describes the real Web service provided e.g. booking of train tickets. A more detailed description of capabilities is given in Section 4.1.

An **interface** describes the interface of the Web service to be used to achieve the described Web service. Further details are given in Section 4.2.

4.1 Capability

The functionality offered by a given Web service is described by its capability; it is expressed by the state of the world before the Web service is executed and the state of the world after successful Web service provision. The Web service capability is meant primarily for discovery and selection purposes i.e. the

[5] For a detailed description of the non functional properties, see reference [41].

capability is used by the requester to determine whether the Web service meets its needs.

The definition of the capability is given below:

```
Class capability
    hasNonFunctionalProperty type nonFunctionalProperty
    importsOntology type ontology
    usesMediator type ooMediator
    hasSharedVariables type sharedVariables
    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 3.1. **Imported Ontologies** and **used mediators** are defined as in Section 3.2 and Section 3.3 respectively.

Shared Variables represent the variables that are shared between of preconditions, postconditons, assumptions and effects. They are universally quantified variables in the formula that concatenates assumptions, preconditions, postconditions, and effects.

If `?v1,...,?vn` are the shared variables defined in a capability, and `pre(?v1,...,?vn)`, `ass(?v1,...,?vn)`, `post(?v1,...,?vn)` and `eff(?v1,...,?vn)`, are used to denote the formulae defined by the preconditions, assumptions, postconditions, and effects respectively, then the following holds:

```
forAll ?v1 ,...,? vn ( pre(?v1 ,...,? vn) and ass(?v1 ,...,? vn)
        implies post(?v1 ,...,? vn) and eff(?v1 ,...,? vn) ).
```

In our example, for two shared variables we will use:

```
sharedVariables ?trip , ?creditCard
```

The ?*trip* shared variable will be used to relate preconditions and postconditions/effects and the ?*creditCard* will be used to relate precondition/assumptions and effects.

Preconditions, in the description of the capability, specify the required state of the information space before the Web service execution; i.e. they specify what information a Web service requires, in order to provide its value. 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 Web service in a defined manner.

We extend the already described example and use the ontology presented in Section 3; in addition we reuse an ontology about purchases[6] which is assumed to be imported, and for brevity we assume *po* and *loc* as namespace prefixes. The following example presents a precondition saying that the information the Web service accepts must be an instance of the trip concept. A trip is described by its start and end locations, and its price. As restriction, the values of its origin and destination have to be in Italy or in Austria. Moreover a credit card is required for successful provision of this service.

```
precondition
  axiom preconditionBooking
  definedBy
    exists ?origin , ?destination
    (?trip [
      origin hasValue ?origin ,
      destination hasValue ?destination
     ]memberOf trip and
     (?origin . locatedIn = loc: austria
        or ?origin . locatedIn = loc: italy ) and
     (?destination . locatedIn = loc: austria
        or ?destination . locatedIn = loc: italy ) and
     ?creditCard memberOf po:creditCard).}
```

Assumptions in the description of the capability describe the state of the world which is assumed before the execution of the Web service. Otherwise, the successful provision of the Web service is not guaranteed. Unlike preconditions, assumptions are not necessarily checked by the Web service. We make this dis-

[6] http://wsmo.org/ontologies/purchase/

tinction in order to allow an explicit notion of conditions which exist in the real world, but which exist outside the information space.

Within our example we present an assumption saying that the service will be provided only if the provided credit card is valid. The validity of the credit card is specified using the *valid* relation. Although assumptions are not necessarily checked they can be helpful during the enactment of a service. E.g. When a requester has checked successfully the precondition (according to its input) but the service fails, the source of the failure will be stated in the assumptions. With the formalized additional requirements a requester can then do additional checks (for example with external services) to also address the requirements of the assumptions.

```
assumption
  axiom assumptionBooking
    definedBy
      valid (?creditCard).
```

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 Web service; it also describes the relation between the information that is provided to the Web service, and its results.

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

```
postcondition
  axiom postconditionBooking
    definedBy
      exists ?confirmation
        (?confirmation memberOf confirmation and
        ?confirmation.confirmationItem = ?trip).
```

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

The following example presents an effect saying that, after the execution of the Web service, the cost of the trip will be deducted from the balance of the credit card given as input.

```
effect
  axiom effectBooking
    definedBy
      ?creditCard.po:balance =
            ?creditCard.po: initialBalance  − ?trip. tripPrice .
```

Intuitive Semantics: Although we do not formally define them, we make the following intuitive assumptions about the semantics of Preconditions, Assumptions, Postconditions and Effects. Let $\mathcal{PRE}(?x_1, \ldots, ?x_n)$, $\mathcal{POST}(?x_1, \ldots, ?x_n)$, $\mathcal{ASS}(?x_1, \ldots, ?x_n)$, and $\mathcal{EFF}(?x_1, \ldots, ?x_n)$ be the formulae in the `definedBy` part of the respective description, with free variables in $\{?x_1, \ldots, ?x_n\}$. Slightly 'abusing' a situation calculus style notation (cf. for instance [39]), given the state s_0 before executing the Web service and $do(service, s_0)$ the state after executing the Web service, we could write down the relation between these formulae as follows:

$$\forall ?x_1, \ldots, ?x_n \; holds(\mathcal{PRE}(?x_1, \ldots, ?x_n) \wedge \mathcal{ASS}(?x_1, \ldots, ?x_n), s_0)$$
$$\rightarrow holds(\mathcal{POST}(?x_1, \ldots, ?x_n) \wedge \mathcal{EFF}(?x_1, \ldots, ?x_n), do(service, s_0))$$

Note that this implies that using the same free variables within different parts of the capability means referring to the same entity.

4.2 Interfaces

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

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

This distinction reflects the difference between communication and cooperation. The choreography defines how to communicate with the Web service in order to consume its functionality. The orchestration defines how the overall functionality is achieved by the cooperation of more elementary Web 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 Web 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 [10] 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

The set of **non functional properties** that can be attached to an interface is the one presented in Section 3.1. **Imported Ontologies** and **used mediators** are defined as in Section 3.2 and Section 3.3 respectively.

Choreography provides the necessary information to communicate with the Web service. From a business-to-business perspective, the choreography can be split into two distinct choreographies:

- execution choreography - defines the interaction protocol for accessing a Web service.
- meta choreography - defines the interaction protocol for negotiating an agreed service and for monitoring the agreed service level agreement during the execution of a service.

The general model for representing choreographies is a state-based mechanism and is inspired by the Abstract State Machines [22] methodology. ASMs have been chosen as underlying model of choreography and orchestration for the following three reasons:

- Minimality: ASMs provide a minimal set of modeling primitives, i.e., enforce minimal ontological commitments. Therefore, they do not introduce any ad hoc elements that would be questionable to be included into a standard proposal.
- Maximality: ASMs are expressive enough to model any a spect around computation.
- Formality: ASMs provide a rigid mathematical framework to express dynamics.

A WSMO choreography defines a state signature that is given by elements of the WSMO Ontology, and it remains unchanged during the execution of the Web service, a state that is given by a set of instance statements, and guarded transitions that express changes of states by means of rules, similar to ASM transition rules. For a more detailed description of WSMO choreography see reference [40].

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

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

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 Web services that would potentially satisfy user desires.

Note that WSMO completely decouples the objectives a requester has i.e. his goals, from the Web services that can actually fulfill such goals. Goals are to be resolved by selecting from the available Web services whichever described service provision best satisfies the goal (see [25] for more details).

The definition of a goal is given below:

Class goal
hasNonFunctionalProperty **type** nonFunctionalProperty
importsOntology **type** ontology
usesMediator **type** {ooMediator, ggMediator}
requestsCapability **type** capability **multiplicity** = **single−valued**
requestsInterface **type** interface

Given the fact that a goal can represent the Web 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 4). An extra non-functional property, the *Type of Match*, can be attached to a goal, 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 see reference [26]). A goal uses **imported ontologies** as the terminology to define the other elements that are part of the goal as long as no conflicts need to be resolved.

A goal **uses mediators** in the following situations:

- when using heterogeneous terminologies, conflicts between them might arise; in these cases, a Web service can import ontologies using ontology mediators (*ooMediators*), as explained in Section 3.3.
- 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 6).

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

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

The following example presents the goal of making a reservation for a trip between Innsbruck and Venice. More precisely, the information the user is looking

for is an instance of the confirmation concept, with the attribute value for the confirmed item corresponding to the trip, having the corresponding attribute values according to his specific desire.

```
goal havingATripConfirmation
  requestsCapability tripConfirmationCapability
    postcondition
      axiom postconditionGoalBooking
        definedBy
          ?confirmation memberOf confirmation and
          ?trip memberOf trip and
          ?trip . origin = loc:innsbruck and
          ?trip . destination = loc:venice and
          ?confirmation . confirmationItem = ?trip .
```

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 [46] 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 of generic, domain independent mediation facilities as well as reuse of mediators. Concerning the needs for mediation within Semantic Web Services, the WSMF [18] 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 ought 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 create a mediation-orientated architecture for Semantic Web Services, providing an infrastructure for handling heterogeneities that possibly arise between WSMO components and implementing the design concept of strong decoupling and strong mediation. A WSMO Mediator serves as a third party components that connects heterogeneous elements and resolves mismatches between them. The following specifies the general definition.

```
Class mediator
    hasNonFunctionalProperty type nonFunctionalProperty
    importsOntology type ontology
    hasSource type {ontology, goal, webService, mediator}
    hasTarget type {ontology, goal, webService, mediator}
    hasMediationService type {webService, goal, wwMediator}
```

As a Mediator can be provided as a Web service, the same **non functional properties** as for Web services are used (see Section 4 for what these non functional properties consist of).

Imported Ontologies allows to import ontologies as the explicit and formal terminology definitions for specifying mediators (see Section 3.2); especially, ontologies that define languages for mediation definitions are imported.

The **source** denotes the heterogeneous resources that are connected by the mediator and wherefore occurring heterogeneities are resolved; a mediator can have several source components.

The **target** denotes the element that receives the mediated source components, so that mismatches are resolved. This corresponds to the 'usedMediators' construct of WSMO elements that utilize a mediator.

The **mediation service** defines the mediation facility applied for resolving mismatches. A mediation service is comprised of mediation definitions that resolve mismatches, and a facility of executing this mappings. The link to the mediation service used by a specific mediator 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 with the mediator.

6.1 WSMO Mediator Types

In order to allow resolving of heterogeneities between the different WSMO components, WSMO defines different types of Mediators for connecting the different WSMO components and overcoming 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 an example for using the different mediator types is discussed 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 Meditator:

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 the creation of 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:

Class wgMediator **sub**−**Class** mediator
    usesMediator **type** ooMediator
    hasSource **type** {webService, wgMediator}
    hasTarget **type** {goal, ggMediator}

WG Mediators are used to pre-link Web services to existing Goals, or for handling 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:

Class wwMediator **sub**−**Class** mediator
  usesMediator **type** ooMediator
  hasSource **type** {webService, wwMediator}
  hasTarget **type** {webService, wwMediator}

A WW Mediator mediates between the choreographies of Web services that ought to interact, whereby 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.

6.2 Example for Using WSMO Mediators

In order to explain the usage of the different WSMO Mediator types, we refer to the example from the e-tourism domain introduced above. In this example, the following WSMO resources are defined:

- four domain ontologies. The three ontologies mentioned above: a Trip Reservation Ontology $\mathcal{O}_{ts}$, a Location $\mathcal{O}_{loc}$, a Purchase Ontology $\mathcal{O}_{po}$; and, in addition, a Payment Ontology $\mathcal{O}_{pay}$
- a Goal $\mathcal{G}_1$ that specifies "book a trip" which is used as a template for defining a concrete Goal $\mathcal{G}_2$ that states "book a trip from Innsbruck (Austria) to Venice (Italy) on date 2004-12-30", as stated in Section 5.
- a Web service $\mathcal{WS}_{vta}$ *VTA* described in Section 4.1 offered by a travel agency that provides an end-user Web service for booking trips for trains, planes, and long-distance bus connections; this Web service uses another Web service $\mathcal{WS}_{payment}$ for handling and processing payments.

Figure 4 shows the connections between the resources of this example. In the figure, full arrows to a mediator denote the sources (whereas full arrows to a

goal or a Web service denote ontology import) and dashed arrows the targets of the used mediators, with explanations below.

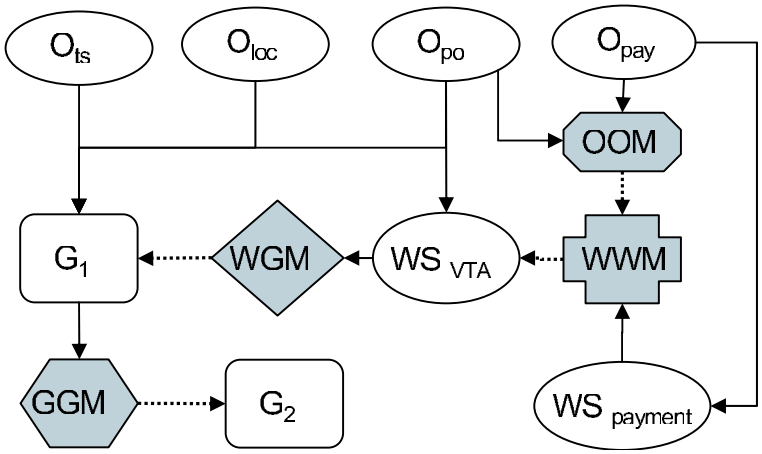

Fig. 4. WSMO Mediators - Usage Example

In order to define the Goal $\mathcal{G}_1$ and the Web service $\mathcal{WS}_{vta}$, the three domain ontologies $\mathcal{O}_{ts}$, $\mathcal{O}_{loc}$, and $\mathcal{O}_{po}$ are used as exemplified in the preceding sections. As there are no terminological mismatches to be resolved, no mediator is needed. Then, a WG Mediator $\mathcal{WGM}$ connects $\mathcal{G}_1$ and $\mathcal{WS}_{vta}$, whereby the information space is reduced to train trips only (the Web service offers trip booking for several means of transport, while $\mathcal{G}_1$ just requests a trip). A GG Mediator $\mathcal{GGM}$ defines that $\mathcal{G}_2$ is derived from $\mathcal{G}_1$ by inheriting all description notions and refining the objective specification to a concrete trip from Innsbruck to Venice on 2004-12-30; also, the WG Mediator $\mathcal{WGM}$ holds between $\mathcal{WS}_{vta}$ and $\mathcal{G}_2$ as it is inherited from $\mathcal{G}_1$.

The $\mathcal{WS}_{vta}$ defines in its orchestration the usage of a third party Web service $\mathcal{WS}_{payment}$ for processing online payment. The Web service description of $\mathcal{WS}_{payment}$ uses a different domain ontology $\mathcal{O}_{pay}$; in order to resolve the terminological interoperability between $\mathcal{WS}_{vta}$ and $\mathcal{WS}_{payment}$, an OO Mediator

$\mathcal{OOM}$ resolves the mismatches between $\mathcal{O}_{po}$ used by $\mathcal{WS}_{vta}$ and $\mathcal{O}_{pay}$ used by $\mathcal{WS}_{payment}$; this OO Mediator is used within a WW Mediator $\mathcal{WWM}$ that connects both Web services and, in addition, resolves possibly occurring mismatches on the protocol or process level.

This example exposes the need for different mediator types in order to establish interoperability in open and distributed environments. A mediator serves as a third party component that connects source elements and resolves mismatches between them, so that the target component receives homogeneous, mediated elements by usage of a mediator. Mediation definitions and execution facilities for these are provided by mediation services that are executed when invoking a mediator. The different mediator types allow to resolve heterogeneities that can appear between the different WSMO elements. OO Mediators are used to ensure semantic interoperability, while the other mediator types allow handling heterogeneities that can specifically occur between their respective source components: GG and WG Mediators support compatibility establishment between Goals and Web Services, and WW Mediators allow establishing interoperability of Web Services with respect to their service interfaces.

7 A Language for defining formal Statements in WSMO

For writing down WSMO descriptions of Web services, goals, ontologies and to some extend mediators, we have designed a formal language called WSML [11]. We have already seen sample fragments of this language in the previous sections. The correspondence with the conceptual model for the presented fragments is mostly self-explanatory since we used a human-readable syntax of WSML. More precisely, WSML is a *family* of formal description languages, that can be used for the precise specification of the single elements in the WSMO framework. Since this article aims to describe the conceptual model underlying our approach, we do

not describe the WSML family in-depth here for the sake of space restriction. The detailed semantics and different syntaxes (including an XML exchange syntax) of the variants of WSML are defined in [12,11].

However, some more detailed comments on the logical expressions inside axioms used almost everywhere in the WSMO model to capture specific nuances of meaning of modeling elements are in order. In the following, we give a definition of the syntax of an expressive formal language in the WSML family of languages that can be used for specifying these logical expressions. The language defined here basically is a first-order language, similar to First-order Logics [16] and Frame Logic (F-Logic, resp.) [28]. In contrast to classical First-order Logic, we use a language that provides object-oriented and frame-based modelling constructs which we believe is more suitable for modelling and knowledge representation for people with some background in computer science and programming. Many things that can directly be expressed in our language, e.g. the subconcept relation, must (and can) be artificially encoded in First-order Logic. Thus, we do not consider First-order logic as an adequate modelling language for our domain. In particular, we exploit the advanced object-oriented modeling constructs of F-Logic and reflect these constructs in our language. Another important aspect in the design of the WSML family of languages was that the single languages had to be based on existing Web technologies and standards, like URIs, namespaces, etc. The reason for not directly using OWL stems from the fact that OWL was designed as a language for the Semantic Web. It is suited for the annotation of semi-structured information with machine processable semantics. However, it was not developed with the design rationale in mind to define the semantics of processes that require rich definitions of their functionality.

Section 7.1 gives the definition of the basic *vocabulary* and the set of *terms* for building logical expression. Then we define in Section 7.2 the most basic

formulae, the so-called *atomic formulae*. Based on atomic formulae and terms we can eventually define the set of *logical expressions* over a given vocabulary.

7.1 Basic Vocabulary and Terms

A language for defining statements about entities in a WSMO description basically is a set of expressions which is constructed from specific symbols according to a set of rules. In this section we define this set of basic symbols called *vocabulary* as well as the set of *terms* which are the most basic building blocks for statements.

Uniform Resource Identifiers and QNames are used to identify entities [6]. Everything in WSMO is by default denoted by a URI, except when it is a *Literal*, *Variable* or *Anonymous Id*. URIs can be expressed either as full URIs: e.g. http://www.wsmo.org/2004/d2/ or using qualified Names (QNames)[7] that are resolved to full URIs using namespace declarations

Literals are used to identify values such as numbers by means of a lexical representation. Anything represented by a literal could also be represented by a URI, but it is often more convenient or intuitive to use literals. Literals are either plain literals or typed literals. A Literal can be typed to a data type (e.g. to xsd:integer) [23].

Variable Names denote variables. Variable names are strings that start with a question mark followed by any positive number of symbols in {`a-z,A-Z,0-9,_,-`}, i.e. `?var` or `?lastValue_Of`. Each variable name represents a distinct symbol that can be used in vocabularies.

[7] For more details on QNames, see reference [8].

Anonymous Identifiers are used to denote objects that exist but do not need an explicit identifier. They can be numbered (_#1, _#2, ...) or unnumbered (_#). The same numbered Anonymous ID represents the same Identifier within the same scope. Otherwise, Anonymous IDs represent different identifiers [49].

The concept of anonymous IDs is similar to blank nodes in RDF [23]. However there are some differences. Blank nodes are essentially existentially quantified variables, where the quantifier has the scope of one document, while anonymous IDs are not existentially quantified variables, but constants. This allows two flavors of entailment: strict and relaxed entailment [49]. The relaxed entailment is equivalent to the behavior of blank nodes, while strict entailment is an isomorphic embedding, where named resource are mapped to identically named resources and anonymous IDs to anonymous IDs. In addition the entailment of proper instances is obtained by replacing one or more anonymous resources with named resources. This way strict entailment allows for an easier implementation. In addition, RDF defines different strategies for the union of two documents (merge and union), whereas the scope of one anonymous ID is a logical expression and the semantics of anonymous IDs do not require different strategies for a union of two documents or, respectively, of two logical expressions.

Definition 1. *The vocabulary V of our language $L(V)$ consists of the following symbols:*

- *A set of Uniform Resource Identifiers URI.*
- *A set of anonymous Ids AnID.*
- *A set of literals Lit.*
- *A set of variables names Var.*
- *A set of function symbols FSym which is a subset of URI.*
- *A set of predicate symbols PSym which is a subset of URI.*

- *A set of predicate symbols with named arguments PSymNamed which is a subset of URI.*
- *A finite set of auxiliary symbols AuxSym including* `(`,`)`,`ofType`,`ofTypeSet`, `memberOf`,`subConceptOf`,`subRelationOf` ,`hasValue`,`hasValues`,`false`,`true`,`[`,`]`.
- *A finite set of logical connectives and quantifiers including the usual ones from First-Order Logics, i.e.* `and`,`or`,`not`, `implies`,`impliedBy`,`equivalent`, `forAll`,`exists`.
- *All these sets are assumed to be mutually distinct (as long as no subset relationship has been explicitly stated).*
- *For each symbol S in FSym, PSym or PSymNamed, we assume that there is a corresponding function arity(S) defined which gives a non-negative integer specifying the number of arguments that are expected by the symbol S when building expressions in our language.*
- *For each symbol S in PSymNamed, we assume that there is a corresponding set of parameter names parNames(S) defined, which gives the names of the single parameters of the symbol that have to be used when building expressions in our language using these symbols.*

As usual, 0-ary function symbols are called constants. 0-ary predicate symbols correspond to propositional variables in classical propositional logic.

Based on a given vocabulary V we can define the set of terms $Term(V)$ which can be constructed from V. In general, terms can be used to describe computations in some domain. An additional interpretation of terms is that they denote objects in some universe and thus provide names for entities in some domain of discourse.

Definition 2. *Given a vocabulary V, we can define the set of terms Term(V) as follows:*

- *Any identifier $u \in URI$ is a term in Term(V).*
- *Any anonymous Id $i \in AnID$ is a term in Term(V).*
- *Any literal $l \in Lit$ is a term in Term(V).*
- *Any variable $v \in Var$ is a term in Term(V).*
- *If f is a function symbol from FSym with arity(f) = n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term in Term(V).*
- *Nothing else is a term.*

As usual, the set of ground terms GroundTerm(V) is the subset of terms in Term(V) which do not contain any variables.

7.2 Logical Expressions

We extend Definition 2 to the set $L(V)$ of logical expressions (or formulae, resp.) in two steps: first, we define the set $L_0(V)$ of *simple logical expressions* or atomic formulae which can be considered as the most basic expressions for representing statements. Then, we extend this set to complex logical expressions which give enough expressivity and freedom to ensure applicability of the languages $L(V)$ to a wide variety of domains.

Definition 3. *The set of simple logical expression in L(V) is inductively defined by*

- *If p is a predicate symbol in PSym with arity(p) = n and $t_1, \ldots, t_n$ are terms in Term(V), then $p(t_1, \ldots, t_n)$ is a simple logical expression in L(V) .*
- *If r is a predicate symbol with named arguments in PSymNamed with arity(p) = n, parNames(r) = $p_1, \ldots, p_n$ and $t_1, \ldots, t_n$ are terms in Term(V), then $r[p_1$ `hasValue` $t_1, \ldots, p_n$ `hasValue` $t_n]$ is a simple logical expression in L(V).*
- `true` *and* `false` *are simple logical expression in L(V).*
- *If P, A, T are terms in Term(V), then $P[A$ `ofType` $T]$ is a simple logical expression in L(V).*

- *If P, A, $T_1, \ldots, T_n$ (with $n \geq 1$) are terms in Term(V), then $P[A$* `ofTypeSet` *$(T_1, \ldots, T_n)]$ is a simple logical expression in L(V).*
- *If O, T are terms in Term(V), then O* `memberOf` *T is a simple logical expression in L(V).*
- *If C_1, C_2 are terms in Term(V), then C_1* `subConceptOf` *C_2 is a simple logical expression in L(V).*
- *If R_1, R_2 are both predicate symbols in PSym or both predicate symbols in PSymNamed with the same signature, then R_1* `subRelationOf` *R_2 is a simple logical expression on L(V).*
- *If O, V, A are terms in Term(V), then $O[A$* `hasValue` *$V]$ is a simple logical expression in L(V).*
- *If O, A, $V_1, \ldots, V_n$ (with $n \geq 1$) are terms in Term(V), then $O[A$* `hasValues` *$\{V_1, \ldots, V_n\}]$ is a simple logical expression in L(V).*
- *If T_1 and T_2 are terms in Term(V), then $T_1 = T_2$ is a simple logical expression in L(V).*
- *Nothing else is a simple logical expression.*

The set of simple logical expressions is denoted by $L_0(V)$.

The intuitive semantics for simple logical expressions is as follows:

- The semantics of predicates in $PSym$ is the common one for predicates in First-Order Logics, i.e. they denote basic statements about the elements of some universe which are represented by the arguments of the symbol.
- Predicates with named arguments have the same semantic purpose but instead of identifying the arguments of the predicate by means a fixed order, the single arguments are identified by a parameter name. The order of the arguments does not matter here for the semantics of the predicate but the corresponding parameter names. Obviously, this has consequences for unification algorithms.

- `true` and `false` denote atomic statements which are always true (or false, resp.).
- $C[A$ `ofType` $T]$ defines a constraint on the possible values that instances of class C may take for property A to values of type T. Thus, this is expression is a signature expression.
- The same purpose has the simple logical expression $C[A$ `ofTypeSet` $(T_1, \ldots, T_n)]$. It defines a constraint on the possible values that instances of class C may take for property A to values of types $T_1, \ldots, T_n$. That means all values of all the specified types are allowed as values for the property A.
- O `memberOf` T is true iff element O is an instance of type T, that means the element denoted by O is a member of the extension of type T.
- C_1 `subConceptOf` C_2 is true iff concept C_1 is a subconcept of concept C_2, that means the extension of concept C_1 is a subset of the extension of concept C_2.
- Similarly for R_1 `subRelationOf` R_2.
- $O[A$ `hasValue` $V]$ is true iff the element denoted by O takes value V under property A.
- Similarly for the simple logical expression $O[A$ `hasValues` $V_1, \ldots, V_n]$: The expression holds if the set of values that the element O takes for property A includes all the values $V_1, \ldots, V_n$. That means the set of values of O for property A is a superset of the set $\{V_1, \ldots, V_n\}$.
- $T_1 = T_2$ is true iff both terms T_1 and T_2 denote the same element of the universe.

Finally, we extend the set $L_0(V)$ of simple logical expressions over vocabulary V to the set of logical expression $L(V)$.

Definition 4. *The set $L(V)$ of logical expressions (or formulae) over vocabulary V is inductively defined as follows:*

- *Every simple logical expression $s \in L_0(V)$ is a logical expression in L(V).*
- *If L is a logical expression in L(V) , then* `not` *L is a logical expression in L(V).*
- *If L_1 and L_2 are logical expressions in L(V) and* `op` *is one of the logical connectives in* {`or`,`and`,`implies`,`impliedBy`, `equivalent`}*, then L_1* `op` *L_2 is a logical expression in L(V).*
- *If L is a logical expression in L(V), x is a variable from Var and Q is a quantor in* {`forAll`,`exists`}*, then Qx(L) is a logical expression in L(V).*
- *Nothing else is a logical expression in L(V).*

The intuitive semantics for complex logical expressions is as follows:

- `not` L is true iff the logical expression L does not hold
- `or`,`and`,`implies`, `equivalent`,`impliedBy` denote the common disjunction, conjunction, implication, equivalence and backward implication of statements
- `forAll` x `(`L`)` is true iff L holds for all possible assignments of x with an element of the universe.
- `exists` x `(`L`)` is true iff there is an assignment of x with an element of the universe such that L holds.

Notational conventions. For notational convenience, we introduce the following set of abbreviations and conventions.

We use a precedence order $<$ on logical connectives as follows (where $op_1 < op_2$ means that op_2 binds stronger than op_1): `implies` , `equivalent`,`impliedBy` $<$ `or`, `and` $<$ `not`. The precedence order can be exploited when writing logical expressions in order to prevent extensive use of parenthesis. If there are ambiguities in evaluating an expression, parenthesis must be used to resolve the ambiguities.

The terms $O[A$ `ofTypeSet` $(T)]$ and $O[A$ `hasValues` $V]$ (that means for the case $n = 1$ in the respective clauses above) can be written in simplified form by omitting the parenthesis. A logical expression of the form `false impliedBy` L (commonly used in Logic Programming systems for defining integrity constraints) can be written as `constraint` L.

We allow the following syntactic composition of atomic formulas as a syntactic abbreviation for two separate atomic formulas: C_1 `subConceptOf` C_2 and $C_1[A$ `op` $V]$ can be syntactically combined to $C_1[A$ `op` $V]$ tt subConceptOf C_2. Additionally, for the sake of backwards compatibility with F-Logic, we allow the following notation for the combination of the two atomic formulae as well: C_1 `subConceptOf` $C_2[A$ `op` $V]$. Both abbreviations stand for the set of the two single atomic formulae. The first abbreviation is considered to be the standard abbreviation for combining these two kinds of atomics formulae.

Furthermore, we allow path expressions as a syntactical shortcut for navigation related expressions: $p.q$ stands for the element which can be reached by navigating from p via property q. The property q has to be a non-set-valued property (`hasValue`). For navigation over set-valued properties (`hasValues`), we use a different expression $p..q$. Such path expressions can be used like a term wherever a term is expected in a logical expression.

Semantically, the various modeling elements of ontologies as defined in Section 3 can be represented as follows: concepts can be represented as terms, relations as predicates with named arguments, functions as predicates with named arguments, instances as terms and axioms as logical expressions.

For further details and a precise account to the definition of the semantics of the languages $L(V)$, see reference [12]. Furthermore, [12] identifies several tractable subsets and extensions of the basic language $L(V)$ that we have defined here.

8 Related Work

Other major initiatives in the area of Semantic Web Service are OWL-S, METEOR-S, and IRS-II. OWL-S [43], part of the DAML program[8], is an ontology for service description based on the Web Ontology Language (OWL) [13]. 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 [24] or the syntactical framework of DRS [32] lead to undecidability problems or leave the semantics open, respectively, and when combining OWL with KIF [15] it is not clear how both interact, while WSMO directly provides a family of (properly) layered logical languages which combines conceptual modelling with rules. The various languages [11] (which have not been discussed in this article in detail) of the WSML family of ontology languages provide different expressiveness and different computational guarantees; these ontology languages are based to a large extend on research in the field of deductive databases. As a consequence, one particularly relevant reasoning task for the Semantic Web and its applications, namely query answering can be solved efficiently for the less expressive languages. Other reasoning tasks like logical entailment or satisfiability checking are decidable for some of the languages in the family but are undecidable for the most expressive ones as well. However, for concrete applications one can select the language

[8] http://www.daml.org/

which is adequate with respect to the needed expressiveness and computational characteristics. WSMO orchestrations describe what other Web 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 allows a single service model for a Web service i.e. a unique way to interact with it; OWL-S uses a single modelling 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 [37].

METEOR-S[9] aims at integrating Web service technologies such as Business Process Execution Language for Web services (BPEL4WS) [2], Web Service Description Language (WSDL) [10] and Universal Description, Discovery and Integration (UDDI) [5] 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 Web services and their related aspects.

The Internet Reasoning Service II (IRS-II) [34] is a Semantic Web Services framework, which allows applications to semantically describe and execute Web services. Compared to IRS-II, WSMO focuses more on the description elements that are needed to deal with Semantic Web Service. Conceptually, WSMO and IRS-II are not too different in the sense that both have common roots in UPML [19]. IRS-II and WSMO are expected to converge as future versions of IRS are planned to be WSMO compliant.

[9] http://lsdis.cs.uga.edu/Projects/METEOR-S/

9 Conclusions and Outlook

Semantic Web Services constitute one of the most promising research directions to improve the integration of applications within and across enterprise boundaries. In this context, WSMO aims to provide the conceptual and technical means to realize Semantic Web Services, improving the cost-effectiveness, scalability and robustness of current solutions.

The ontology presented in this paper provides the core elements that are needed to represent Semantic Web Services and related issues: ontologies, that provide the common terminology used by other WSMO elements, Web services which provide access to services that, in turn, provide some value in a domain, goals that are a description of problems that should be solved by Web services, and mediators, which deal with interoperability problems between different WSMO elements. For defining logical statements in WSMO we introduce a logical language.

In total, we believe that our framework consisting of an ontology and the language for describing Web services semantically sets a solid basis for solving the research issue on Semantic Web Services. A tutorial on WSMO, which explains WSMO in more depth can be found in the WSMO Primer [17]. Several use cases demonstrating how to use WSMO in a real-world setting can be found in the WSMO Use Case Modeling and Testing documents [42], use cases ranging from applications in travelling domain to applications in health domain (like one of the use cases of the EU funded project COCOON [44]). For the different subsets of the language for defining WSMO annotated Web services we refer to the WSML Family of Representation Languages [12]. A logical framework for Web service discovery in WSMO has been defined in [25].

Apart from theoretical results and investigations on how to apply WSMO in the usage process of Semantic Web Services [27,3,30,37,31,33,26,14] at the

current state, already a set of WSMO compliant tools have been developed or are under development: WSMX[10] - an execution environment for dynamic matchmaking, selection, mediation and invocation of Semantic Web Services based on WSMO, IRS-III[11] - a platform and infrastructure for creating WSMO-based Semantic Web Services, SWWS Studio[12] and the WSMO Studio [13] - WSMO compliant editors, wsmo4j[14] - an API and a reference implementation for building Semantic Web Services applications compliant with WSMO in Java.

Web Service Modeling Ontology, driven by the SDK cluster[15], is being developed and adopted in several large scale research projects, where a number of international partners fruitfully contributed to its current state. Currently WSMO and its related specifications are being submitted for standardization to the World Wide Web Consortium[16] and is expected to have a high impact in the development of Semantic Web Services.

Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, InfraWebs, SEKT, SWWS, ASG and Esperonto; by Science Foundation Ireland under the DERI-Lion project; by the Vienna city government under the CoOperate programme and by the FIT-IT (Forschung, Innovation, Technologie - Informationstechnologie) under the projects RW^2 and TSC.

[10] http://sourceforge.net/projects/wsmx/
[11] http://kmi.open.ac.uk/projects/irs/#irsiii
[12] http://stronghold.sirma.bg/swws/
[13] http://www.wsmostudio.org/
[14] http://wsmo4j.sourceforge.net/
[15] http://www.sdk-cluster.org/
[16] http://www.w3.org/

The authors would like to thank to all the members of the WSMO[17], WSML[18], and WSMX[19] working groups for their advice and input to this document.

[17] http://www.wsmo.org/
[18] http://www.wsmo.org/wsml/
[19] http://www.wsmx.org/

References

1. Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services*. Springer, 2003.
2. Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business process execution language for web services version 1.1. Technical report, May 2003.
3. Sinuhé Arroyo, Christoph Bussler, Jacek Kopecký, Rubén Lara, Axel Polleres, and Michał Zaremba. Web service capabilities and constraints in WSMO. In *W3C Workshop on Constraints and Capabilities for Web Services*, Oracle Conference Center, Redwood Shores, CA, USA, October 2004.
4. Z. Baida, J. Gordijn, B. Omelayenko, and H. Akkermans. A shared service terminology for online service provisioning. In *ICEC '04: Proceedings of the 6th international conference on Electronic commerce*.
5. Tom Bellwood, Luc Clement, David Ehnebuske, Andrew Hately, Maryann Hondo, Yin Leng Husband, Karsten Januszewski, Sam Lee, Barbara McKee, Joel Munter, and Claus von Riegen. UDDI version 3.0. Technical report, July 2002.
6. Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform resource identifiers (uri): Generic syntax. RFC 2396, IETF, August 1998.
7. Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.
8. Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. `http://www.w3.org/TR/REC-xml-names`, 1999.
9. Jos De Bruijn, Axel Polleres, Rubén Lara, and Dieter Fensel. OWL DL vs. OWL Flight: Conceptual modeling and reasoning for the semantic web. Technical Report DERI-TR-2004-11-10, DERI, November 2004.
10. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, March 2001.

11. Jos de Bruijn, editor. *The Web Service Modeling Language WSML*. 2004. WSML Final Draft D16.1v0.2, March 2005. Available from http://www.wsmo.org/TR/d16/d16.1/v0.2/.

12. Jos de Bruijn, editor. *The WSML Family of Representation Languages*. 2004. WSMO Deliverable D16, WSMO Working Draft, 2004, latest version available at http://www.wsmo.org/TR/d16/.

13. Mike Dean and Guus Schreiber, editors. *OWL Web Ontology Language Reference*. 2004. W3C Recommendation 10 February 2004.

14. John Domingue, Liliana Cabral, Farshad Hakimpour, Denilson Sell, and Enrico Motta. Irs-iii: A platform and infrastructure for creating wsmo-based semantic web services. In *Proc. 1st WSMO Implementation Workshop (WIW)*, September 2004.

15. dpANS. KIF. knowledge interchange format: Draft proposed american national standard (dpANS). Technical Report NCITS.T2/98-004, 1998. available from http://logic.stanford.edu/kif/dpans.html.

16. H. B. Enderton. Degrees of computational complexity. *J. Comput. Syst. Sci.*, 6(5):389–396, 1972.

17. Cristina Feier, editor. *WSMO Primer*. 2004. WSMO Deliverable D3.1, WSMO Working Draft, 2004, latest version available at http://www.wsmo.org/TR/d3/d3.1/.

18. Dieter Fensel and Christoph Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.

19. Dieter Fensel, Enrico Motta, Frank van Harmelen, V. Richard Benjamins, Monica Crubezy, Stefan Decker, Mauro Gaspari, Rix Groenboom, William Grosso, Mark Musen, Enric Plaza, Guus Schreiber, Rudi Studer, and Bob Wielinga. The unified problem-solving method development language upml. *Knowl. Inf. Syst.*, 5(1):83–131, 2003.

20. XML Protocol Working Group. Soap version 1.2. Technical report, June 2003. W3C Recommendation.

21. Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

22. Yuri Gurevich. *Evolving algebras 1993: Lipari guide*, pages 9–36. Oxford University Press, Inc., 1995.

23. Patrick Hayes. Rdf semantics. `http://www.w3.org/TR/rdf-mt/`, February 2004.

24. Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. Available from http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/, May 2004.

25. Uwe Keller, Ruben Lara, and Axel Polleres, editors. *WSMO Web Service Discovery*. 2004. WSMO Web Service Discovery Working Draft D5.1v0.1, December 2004. Available from http://www.wsmo.org/TR/d5/d5.1/v0.1/.

26. Uwe Keller, Michael Stollberg, and Dieter Fensel. Woogle meets semantic web fred. In *Proc. 1st WSMO Implementation Workshop (WIW)*, September 2004.

27. Michael Kifer, Rubén Lara, Axel Polleres, Chang Zhao, Uwe Keller, Holger Lausen, and Dieter Fensel. A logical framework for web service discovery. In *ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications*, Hiroshima, Japan, November 2004.

28. Michael Kifer, Geord Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843, 1995.

29. Atanas Kiryakov, Damyan Ognyanov, and Vesselin Kirov. A framework for representing ontologies consisting of several thousand concepts definitions. DIP Deliverable D2.2, Ontotext Lab, 2004.

30. Rubén Lara, Dumitru Roman, Axel Polleres, and Dieter Fensel. A conceptual comparison of WSMO and OWL-S. In *Proceedings of the European Conference on Web Services (ECOWS 2004)*, Erfurt, Germany, September 2004.

31. Holger Lausen and Michael Felderer. Architecture for an ontology and web service modelling studio. In *Proc. 1st WSMO Implementation Workshop (WIW)*, September 2004.

32. Drew McDermott. DRS: A set of conventions for representing logical languages in RDF. Available from http://www.daml.org/services/owl-s/1.1B/DRSguide.pdf, January 2004.

33. Matthew Moran, Michal Zaremba, Adrian Mocan, and Christoph Bussler. Using wsmx to bind requester and provider at runtime when executing semantic web services. In *Proc. 1st WSMO Implementation Workshop (WIW)*, September 2004.

34. Enrico Motta, John Domingue, Liliana Cabral, and Mauro Gaspari. IRS-II: A framework and infrastructure for semantic web services. In *2nd International Semantic Web Conference (ISWC2003)*. Springer Verlag, October 2003.

35. Object Management Group Inc. (OMG). Meta object facility (MOF) specification v1.4, 2002.

36. Justin O'Sullivan, David Edmond, and Arthur ter Hofstede. What is a service?: Towards accurate description of non-functional properties. *Distributed and ParallelDatabases*, 12(2-3):117–133, 2002.

37. Massimo Paolucci, Naveen Srinivasan, and Katia Sycara. Expressing WSMO mediators in OWL-S. In David Martin, Ruben Lara, and Takahira Yamaguchi, editors, *Proc. 1st. Intl. Workshop SWS'2004 at ISWC 2004*, volume 119, CEUR-WS.org/Vol-119/, November 2004. CEUR-WS.org.

38. Chris Preist. A conceptual architecture for semantic web services. In *3rd International Semantic Web Conference (ISWC2004)*. Springer Verlag, November 2004.

39. Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

40. Dumitru Roman, editor. *WSMO Choreogrpahy*. 2004. WSMO Choreography Working Draft D14v0.1, November 2004. Available from http://www.wsmo.org/TR/d14/v1.0/.

41. Dumitru Roman, Holger Lausen, and Uwe Keller, editors. *Web Service Modeling Ontology (WSMO)*. 2004. WSMO Working Draft D2v1.1, December 2004. Available from http://www.wsmo.org/TR/d2/v1.1/.

42. M. Stollberg, H. Lausen, A. Polleres, and R. Lara, editors. *WSMO Use Case Modeling and Testing*. 2004. WSMO Deliverable D3.2, WSMO Working Draft,

2004, latest version available at http://www.wsmo.org/TR/d3/d3.2/.

43. The OWL Services Coalition. OWL-S 1.1 beta release. Available from http://www.daml.org/services/owl-s/1.1B/, July 2004.
44. Emanuele Della Valle, Dario Cerizza, and Lara Gadda, editors. *Use Case: Semantic Discovery of Community of Practice*. 2004. COCOON Working Draft, December 2004. Available from http://cocoon.cefriel.it/RD2/usecases/semantic-discovery-of-cop.
45. S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. Dublin core metadata for resource discovery. RFC 2413, IETF, September 1998.
46. Gio Wiederhold. Mediators in the architecture of the future information systems. *Computer*, 25(3):38–49, 1994.
47. Guizhen Yang and Michael Kifer. Well-founded optimism: Inheritance in frame-based knowledge bases. In *First International Conference on Ontologies, Databases and Applications of Semantics (ODBASE), Irvine, California*, 2002.
48. Guizhen Yang and Michael Kifer. Inheritance and rules in object-oriented semantic web languages. In *Second International Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML), Sanibel Island, Florida*, 2003.
49. Guizhen Yang and Michael Kifer. Reasoning about anonymous resources and meta statements on the semantic web. *Journal of Data Semantics*, 2800:69–97, 2003.