Semantic Web Services: A conceptual comparison of OWL-S, WSMO and METEOR-S approaches

Panagiotis Bouros

Department of Informatics and Telecommunications
National and Kapodistrian University of Athens (NKUA)
Panepistimiopolis, T.Y.P.A. Buildings,
GR-157 84 Ilisia, Athens, Greece
pbour@di.uoa.gr

Technical Report

1 Introduction

Today's distributed computing is strongly connected with the idea of Web Services. These are parts of programs that can be accessed over a network using well defined protocols. Thus, building a software can be carried out using loosely coupled, reusable components. Moreover, the notion of Web services can facilitate the interoperability between systems to accomplish inside business tasks realizing the system integration and also inter-organizational operations.

Current Web Services standards WSDL [19] and SOAP [11] introduced for describing, discovering and invoking Web Services, although they are well defined and accepted also by industry, they only describe the syntactic aspects of a Web Service. WSDL is less suitable for describing the semantics of a Web Service capability. This drawback affects not only the service discovery procedure but also service composition, invocation and interoperation [25]. Adding semantics to Web Services description standards can tackle these issues. The vision of Semantic Web Services is to describe the semantic aspects of a Web Service with a machine-interpretable and also understandable way enabling automatic interaction with services during their entire lifecycle. As far as publishing and discovery procedures are concerned, the provider can describe Web Service functionality and capability using terms derived form a semantic model. At the same time, requestors can use the same "tools" for describing requesting service desired functionality. Thus, afterwards, reasoning techniques can be used to find the semantic similarity between the service description and the request. During composition, the functional aspect of the annotations can be used to aggregate the functionality of multiple services to create useful service compositions. During invocation, mappings between the semantic annotated concepts use to describe Web Service are mapped into XML-based SOAP messages to enable service execution.

The description of Web Services in a machine-understandable fashion is ex-

pected to have a great impact in areas of e-Commerce and Enterprise Application Integration, as it can enable dynamic, scalable and reusable cooperation between different systems and organizations.

2 Objective

Semantic annotation of Web Service description has been the issue of many initiates, projects and languages introduced, the most significant among them are: the OWL-S Semantic Markup for Web Services [8], the Web Service Modeling Ontology (WSMO) [4] and the METEOR-S Web Service Annotation Framework [6].

This report presents a conceptual comparison of the latest stable version of the above three significant approaches proposed for adding semantics in Web Service description and standards, taking into consideration the OWL-S 1.1 version and the WSMO D2v1.2. Since the comparison is carried out in a conceptual level, it is our intention to identify the conceptual overlaps and differences among the three specifications according to discovery, interoperation, composition and invocation stages of Web Services lifecycle. More specifically, discovery issues involve the semantic representation and publishing of the services capability and requirements. However, in this report we wont deal with publishing services issues, i.e. with mechanisms of importing Semantic Web Services in registries like UDDI [13]. Interoperation is concerned with the operation of a service and the interaction with the other ones. Composition involves issues combining part of several Web Services resulting to composite ones. Finally, service invocation describes the mechanisms for actually using and executing the semantically described Web Services.

The rest of the report is organized as follows. In section 3, 4 and 5 we outline a short presentation of the three Semantic Web Services approaches dealing with discovery, interoperation, composition and invocation. Section 6 attempts to compare the specifications described in the previous sections and finally section 7 concludes the comparison report.

3 OWL-S

OWL-S is an ontology of services written in OWL [10] that enables users and software agents, with a high degree of automation, to discover, invoke, compose and monitor Web resources offering particular services and having particular properties. In the following sections, we present the overall structure of the OWL-S W3C submission [9] for OWL-S 1.1.

3.1 Upper Ontology

OWL-S's approach of semantically describing Web Services is denoted by the Upper Ontology pictured in Figure 1. Each instance of *Service* class:

- *presents* one or more *ServiceProfile* instances. The ServiceProfile answers the question of what the service does, publishing all the information needed for service-discovering.

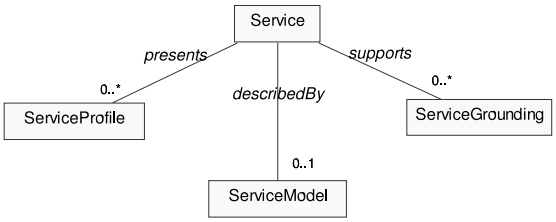

Figure 1: OWL-S upper ontology

- is *described by* at most one *ServiceModel*. The ServiceModel specifies how a client can use the service and how this service works, meaning what happens when the service is carried out and which processes are involved in case of a composite service.

- *supports* a number of *ServiceGrounding* instances. More specifically, when an instance of ServiceModel exists, the Service supports one or more instances of ServiceGrounding. ServiceGrounding defines the details of how an agent can access the service, by specifying communication protocols, message formats, port numbers and other service details.

ServiceModel along with ServiceGrounding provide all the necessary information for an agent to use a service.

3.2 Service Discovery

Service discovery is related with the procedure of identifying the services whose functionality meets the service request requirements. In order to describe the functionality of a service the OWL-S framework proposes a specification of ServiceProfile called *Profile*, representing the non-functional properties of the service provider, functional elements and some additional properties for example QoS features describing its characteristics. Figure 2 presents the conceptual model of ServiceProfile. Yet, the use of Profile is not mandatory within the framework, rather than using OWL subclassing, anybody can create specialized representations, acting as profiles. The non-functional properties of a service provide human-readable information about its name (serviceName), working requirements (textDescription) and the mechanism of referring to humans or individual responsible for the service (contactInformation).

Profile functionality description deals with issues of information transformation, representing the inputs and outputs of a service, and the behavior and the state change by the execution of it, specified by preconditions and effects. In order to represent the inputs and the outputs of a service the framework uses the specifications *Input* and *Output* of the *Parameter* class. In addition, as far effects and preconditions are concerned, a profile via *hasResult* and *hasPrecondition* is associated with a number of *Result* and *Condition* instances. The inputs, outputs, preconditions and results (IOPEs) contained in Profile description are referenced instances of ServiceModel Ontology IOPEs. This denotes the connection of Profile class with the ServiceModel and the corresponding process.

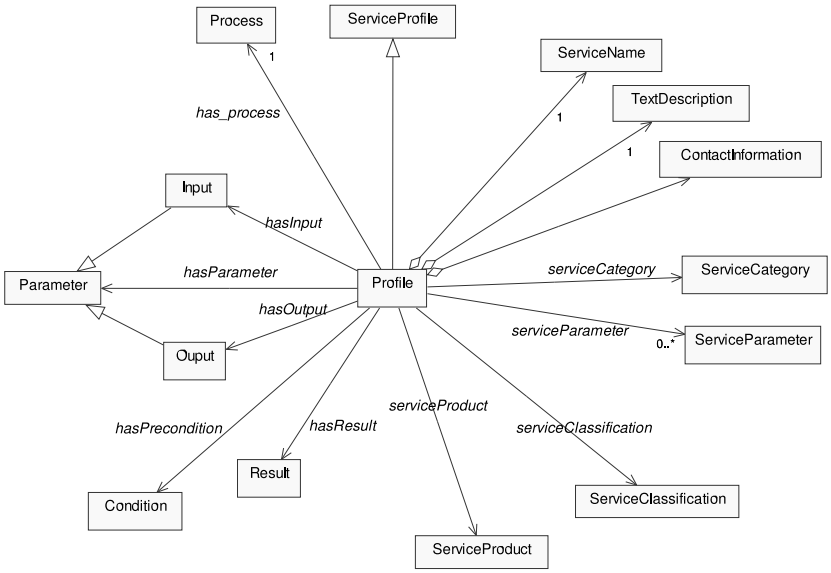

Figure 2: ServiceProfile conceptual model

For this purpose, the Profile class is associated via *has_Process* with a process of ServiceModel.

Finally Profile is associated through *serviceParameter* with a list of service parameters that accompanies the service description and with the *ServiceCategory* class over *serviceCategory* association, referring to a classification of services outside OWL-S or OWL, which may be on offer. In addition, the ServiceProfile defines *serviceClassification* and *serviceProduct* associations to specify the type of service provided and the products that are handled by the service. They are very alike to the ServiceParameter and ServiceCategory, but they define mappings to OWL ontologies of services like NAICS [7] and of products like UNSPSC [12].

3.3 Interoperation

In order to give a detailed perspective on how a service operates, OWL-S framework defines as a subclass of ServiceModel, the *Process* class. Thus, according to the upper ontology of Figure 1 a service defined in OWL-S framework is related to a one specific process.

Figure 3 presents the conceptual model of service interoperation. Process class is associated to a number of IOPEs classes for defining the data transformation and the transition from one state to another. Part of or all the instances of the IOPEs are referenced by Profile class instance, as explained in the previous section.

An instance of the Process class can have inputs, outputs, locals, preconditions, results and participants.

- Participants are used to define entities involved into the process other than the client -the agent from whose point of view the process is described- and the server -the principal element of the service that the client deals with-.
- Inputs and outputs specify the data transformation produced by the process.
- Locals are parameters only used by atomic processes (see next section) in order to define variables whose scope is the process itself. All inputs, outputs and locals are subclasses of Parameter class.
- Preconditions are used to specify conditions that unless they are false the process can be performed successfully.
- Results represent the effect of a process the state of the world. For this purpose the Process class is associated with the Result class. Using the Result class the process model can describe the conditional outputs and the effects of the process. The *inCondition* associates the result with the conditions under which it occurs. The *withCondition* and the *hasResult* associations state what ensues when the condition is true, related to an output and an expression representing the effect.

Both preconditions and effects are represented as logical formulas. In order to integrate logical expressions into the OWL-S framework, there are treated as literals, either string or XML ones. The latter case is used for languages whose standard encoding is in XML, such as SWRL [22] or RDF [23]. The former case is for other languages such as KIF [16] and PDDL [20].

3.4 Composition

OWL-S framework specifies three types of processes, as subclasses of Process class, called *Atomic*, *Simple* and *Composite* (see Figure 3). An atomic process is a directly invocable process which can have no sub-processes. It is executed in a single step by passing the input message and resulting its output message. On the other hand, simple processes are not invocable, but they are used as elements of abstraction, either to provide a a specialized way of using some atomic process (*realizedBy* association), or a simplified representation of some composite process -for purposes of planning and reasoning- (*expand* association).

Every composite process is decomposable to non-composite or composite processes. The decomposition can be specified by the *ControlConstruct* class instances via *composedOf* association (Figure 4). The ControlConstruct class is associated, via *components* association, to an ordered list of control constructs or an unordered one, a bag, in order to define the nested control constructs from which it is composed. The composition mechanism can be considered as a tree whose nonterminal nodes are labelled with control constructs, each of which has children specified using components. The leaves of the tree are invocations of other processes, denoted as instances of class *Perform* indicating the sub-process that should be invoked. The model defines several specifications of ControlConstruct class: *If-Then-Else*, *Repeate-While*, *Repeate-Until*, *Choice*, *Split*, *Split+Join*, *Iterate*, *Unordered* and *Sequence*.

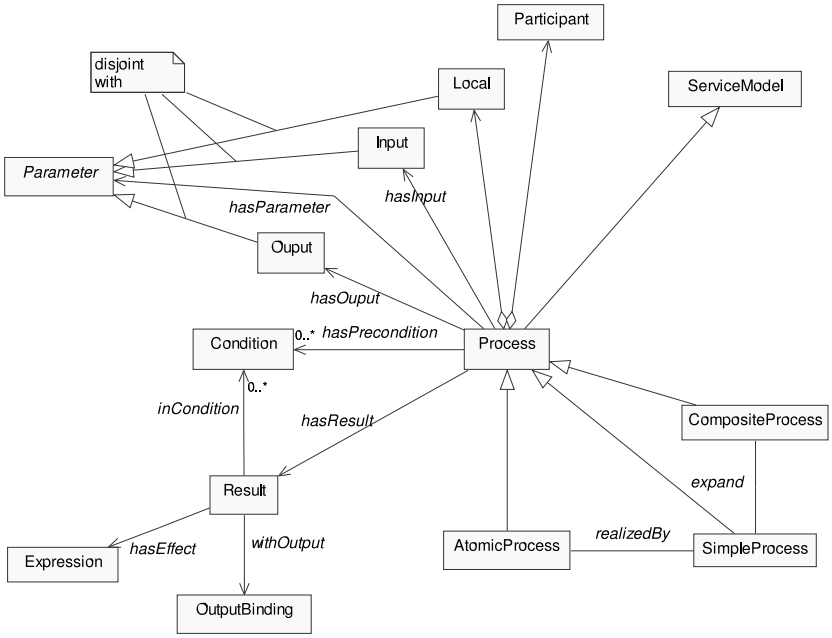

Figure 3: OWL-S process model and IOPEs

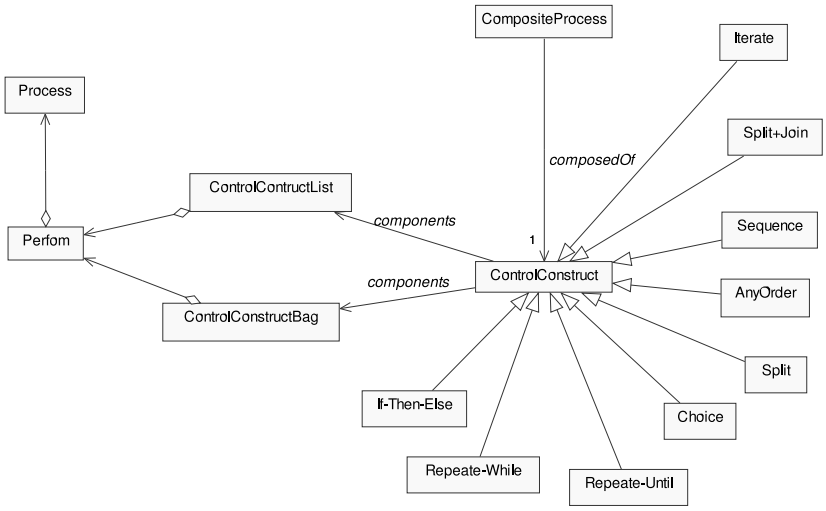

Figure 4: Process categorization and composition

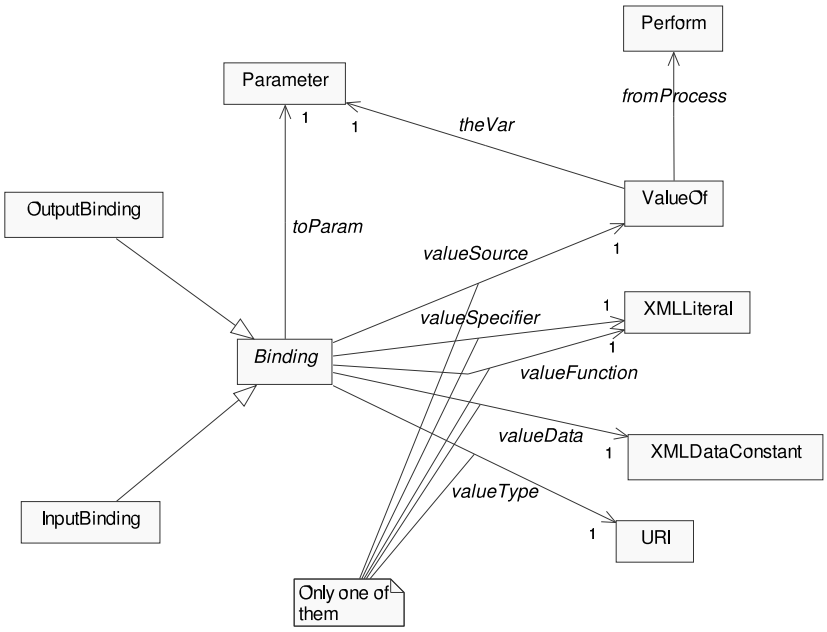

Figure 5: Binding mechanism

Apart from the control constructs presented before, OWL-S specifies constructs that can be used for the description of data flow. Using this mechanism we can state the relation of a process inputs to its outputs, conditions and results and the relation between the inputs and the outputs of a composite process with the inputs and the outputs of another. Moreover we can state regarding a composite process, sequence of a number of sub-processes, how the output of a sub-process is related to the input of another. Figure 5 pictures the OWL-S binding mechanism. According to the type of IOPE, *InputBinding* or *OutputBinding* sub-classes of abstract Binding class are instantiated. Binding class references via toParam association to the corresponding Parameter and to one of the ways to define the value of it: *ValueOf* (via *valueSource* association), an XMLLiteral (via *valueSpecifier* or valueFunction association), an XML data constant (*valueData* association) or an URI (via *vualeType* association). The *valueSource* association is a simple reference to a *ValueOf* object denoting from which parameter (via theVar association) in another process reference (*fromProcess* to Perform) in the same composite process, does the value of the parameter comes.

3.5 Invocation

The grounding of a service specifies the details of how to access and invoke a service. These details tackle protocol and message formats, serialization, transport and addressing. ServiceProfile and ServiceModel are considered as an abstract representation of a service, whereas ServiceGrounding is providing the concrete

specification of it. Thus, the main purpose of grounding is to present how the abstract inputs and outputs of the previous descriptions are to be realized as messages in a specific transmittable format.

Due to the existence of industry standard work already done for this purpose the OWL-S framework makes use of the Web Service Description Language (WSDL). Although WSDL and OWL-S do overlap in specifying the abstract part of Service definition, the WSDL/XSD is unable to capture the semantics of a service just like OWL-S is unable to express the binding information that WSDL specifies. Thus, for the definition of service grounding both OWL-S and WSDL are required. The grounding uses OWL classes as abstract types of messages types declared in WSDL and then WSDL binding specifies the format of these messages. Basically the *OWL-S/WSDL* grounding is based on three main correspondences between OWL-S and WSDL: a) an OWL-S atomic process corresponds to a WSDL (1.1) operation, b) OWL-S inputs/outputs correspond to the parts of an input/output message of a WSDL operation and c) OWL-S input and output types correspond to WSDL's extensible notion of abstract type.

Grounding OWL-S with WSDL and SOAP is achieved by constructing a WSDL document description with all the usual parts (types, message, operation, port type, binding, and service constructs). As far the types of WSDL message parts are concerned there are two solutions. The first sets WSDL as native speaker and so the types of message parts are OWL classes defined either inside the WSDL document or outside in other documents and referred to using the *owl-s-parameter*. The other solution, contains all the different approaches that can be used.

The previous paragraph denoted the way WSDL definitions may refer to the corresponding OWL-S declarations, Figure 6 presents the service grounding mechanism for referring WDSL constructs inside OWL-S. In this context, OWL-S framework defines a sub-class of ServiceGrounding called *WSDLGrouding*. An instance of a *WsdlGrouding* contains a list of *WsdlAtomicProcessGrouding* class instances, each of them providing the grounding for an OWL-S AtomicProcess instance (*owlsProcess* association). WsdlAtomicProcessGrouding instance refers to specific elements within the WSDL specification, including the *WsdlDocument* description, the *WsdlOperationRef* (the corresponding WSDL operation) and *WsdlInputMessage* and *WsdlOutputMessage* that contain the URI of the WSDL message definition for the input and the output of the atomic process. WsdlAtomicProcessGrounding instance is also associated through *wsdlInputs* and *wdslOutputs* with a list of mapping pairs, one for each WSDL input and output message part. Each pair is an instance of *WsdlMessageMap* containing two elements a) the message part as URI (class *WsdlMessagePart*), and b) the way this message part is derived from the inputs or outputs of the AtomicProcess instance. In the simplest cases, in which the message part corresponds to a single OWL-S input or output, is done via the *owlsParameter* association giving the URI of the parameter. Whereas in the more complicated cases an *XsltTransformation* script has to be used in order to derive the message part from the atomic process instance.

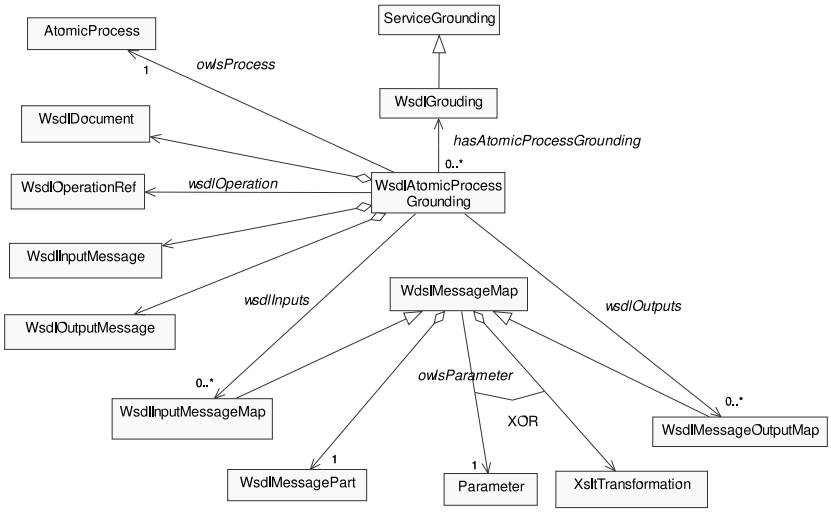

Figure 6: OWL-S/WSDL grounding

4 WSMO

The Web Service Modeling Ontology (WSMO) is a European initiative which aims to provide a standard for describing Semantic Web Services. It has been based on the work of Fensel and Bussler [21] and it is operated by the SDK Cluster.

4.1 Top level

Figure 7 pictures the top level of the WSMO ontology. All the subclasses of *WSMOTopLevelElement* class are further discussed in the following sections. Briefly, ontologies introduce terminology used in other elements, service contains the definition of services, goals describe problems that are addressed by those services and mediators are used to solve interoperability problems between either goals or ontologies or services. The non-functional properties are used in definition of the WSMO elements. The framework recommends most elements of [27]. Depending on which WSMO element is specified the corresponding non-functional properties are applied to it.

4.2 Service Discovery

Crucial issue for service discovery procedure is the introduction of a common terminology for the concepts representing service characteristics. In WSMO ontologies tackle this issue by providing concepts, relationships among these concepts and a set of expressions (the axioms), which explore the semantics of concepts and relationships. Figure 8 present the conceptual model of the WSMO Ontology class and its attributes.

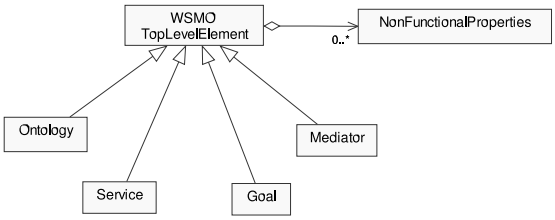

Figure 7: WSMO top level

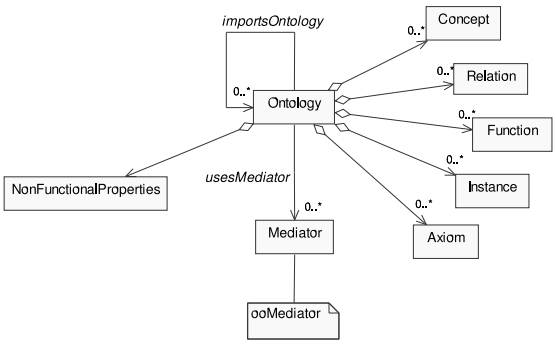

Figure 8: WSMO Ontology conceptual model

Ontology class is characterized by a number of non functional properties: *Contributor, Coverage, Creator, Date, Description, Format, Identifier, Language, Owner, Publisher, Relation, Rights, Source, Subject, Title, Type, Version*. Designing ontology for a domain of interest can be facilitated by importing already defined ontologies, as long as no conflicts need to be resolved. If so, then instances of ooMediators should be used.

Concepts constitute the basic elements of the agreed terminology for some problem domain. Figure 9 presents the Concept class of WSMO framework. The non-functional properties that characterize Concept class are: *Contributor, Coverage, Creator, Date, Description, Identifier, Relation, Source, Subject, Title, Type, Version*. Apart from the non functional properties, the Concept class model also introduces the notion of inheritance for each concept. There can be a number of *super-concepts* for each concept inside ontology. Being a sub-concept of some other concept, means that the concept inherits the signature of his super-concept and the corresponding constraints. Moreover each concept consists of a set of *attributes* that represent named slots for data values for instances. The non-functional properties recommended for specifying Attribute class are: *Contributor, Coverage, Creator, Date, Description, Identifier, Relation, Source, Subject, Title, Type, Version*. Each attribute has as *range* a specific concept that serves as an integrity constraint on the values of it. In addition, each concept has a single *definition* as a logical expression, in order to formally specify the semantics of it.

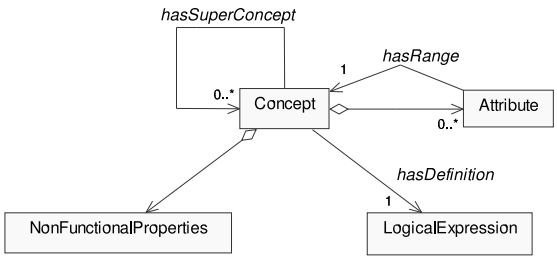

Figure 9: Concept conceptual model

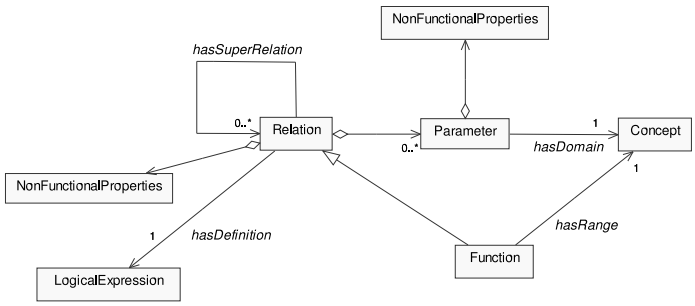

Figure 10: Relation conceptual model

When constructing an ontology and after defining concepts, the next step will be to define the relations and the interdependencies between them. Figure 10 pictures the *Relation* class model. The non-functional properties recommended for defining Relation class are: *Contributor, Coverage, Creator, Date, Description, Identifier, Relation, Source, Subject, Title, Type, Version.* This model also specifies the notion of inheritance. Thus, each relation can have a number of *super-relations* which means that it inherits the corresponding constraints and the signature of them. Relations can also have a list of *parameters* which are associated with a single concept constraining the possible values that they can take and they are specified by the *Contributor, Coverage, Creator, Date, Description, Identifier, Relation, Source, Subject, Title, Type, Version* non-functional properties. Finally, relations have a single *definition* as logical expression to define their set of instances.

Figure 10 also denotes the notion of function. *Function* class is a specialization of Relation class that has a unique set of possible returned values and an n-ary domain (parameters inherited from relation). A function element's set of returned values is related to its domain values.

Instances are defined for the basic entities of ontologies i.e. concepts and relations. There are either defined explicitly or by a link to an instance store, i.e., an external storage of instances and their values. The following descriptions concern the explicit definition of concept or relation instances. Figure 11 pictures the model of concept instance. The non-functional properties recommended for

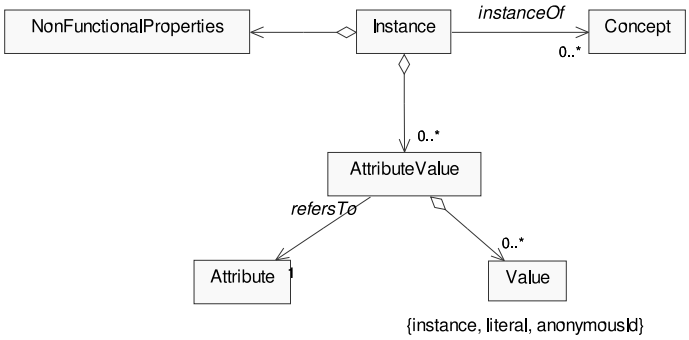

Figure 11: Instance conceptual model

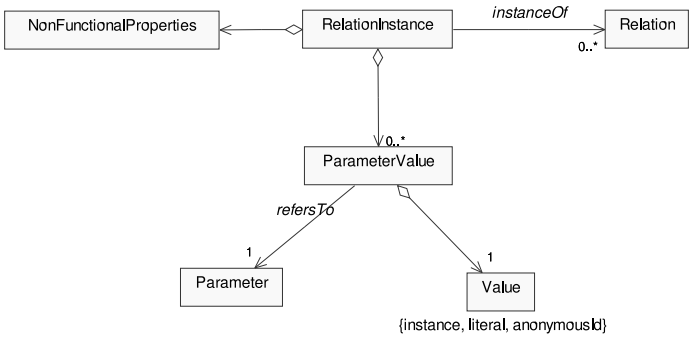

Figure 12: RelationInstance conceptual model

defining the Instance class are: *Contributor, Coverage, Creator, Date, Description, Identifier, Relation, Source, Subject, Title, Type, Version.* Class Instance is associated with the concepts of which it is instance. For each attribute defined on a concept the corresponding value is assigned through the instance and the *AttributeValue* class. Thus, each instance is related to a list of attribute values. Moreover each attribute value consists of the attribute referring to and its value, i.e. an instance, literal or anonymous ID.

Relation instances are treated as n-tuples instances of concepts, specified as the parameters of the relation. Figure 12 presents the definition of relation instances. The recommended non-functional properties are: *Contributor, Coverage, Creator, Date, Description, Identifier, Relation, Source, Subject, Title, Type, Version.* Alike instances of concepts, relation instances are associated with the relations belonging to. Each relation instance has a set of *parameter values* specifying the single instances that are related according to this relation instance. The parameter value contains the referring *parameter* and its *value*, i.e. an instance, literal or anonymous ID.

Finally, *axioms* are considered to be logical expressions for capturing the semantic properties of relations and concepts. Figure 13 presents the concep-

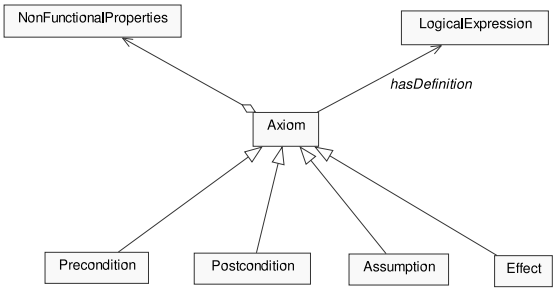

Figure 13: Axiom conceptual model

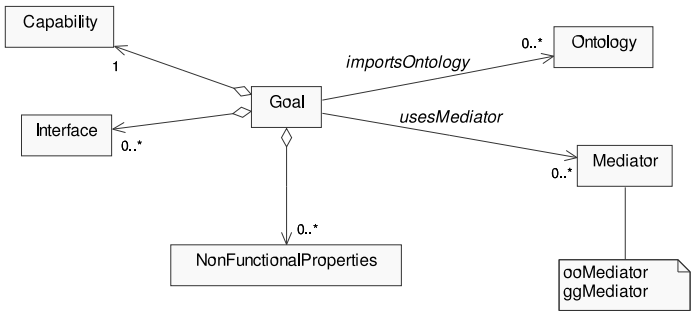

Figure 14: WSMO Goal class

tual model of Axiom class. The recommended non-functional properties are: *Contributor, Coverage, Creator, Date, Description, Identifier, Relation, Source, Subject, Title, Type, Version.* This model will be further explained in the following sections when discussing the capability of a service.

Goals are also related to WSMO service description concerning the service discovery procedure. They are representations of objectives that would be fulfilled through the execution of the service. Moreover, they can be descriptions of services that would potentially satisfy the user desires. Figure 14 presents the Goal class conceptual model. The non functional properties recommended are: *Accuracy, Contributor, Coverage, Creator, Date, Description, Financial, Format, Identifier, Language, Network-related QoS, Owner, Performance, Publisher, Relation, Reliability, Rights, Robustness, Scalability, Security, Source, Subject, Title, Transactional, Trust, Type, Type of Match, Version.* Goal class is associated with a specific capability (Figure 15) in order to define the desired service functionality. Furthermore Goal has a number of class Interface (Figure 17) instances for defining how the functionality of the goal will be achieved. Goals can import ontologies too, through instances of ooMediator in case of aligning, merging, and transforming issues. Finally, a goal can be defined by reusing one or several already-existing goals which is achieved by using goal mediators (ggMediator instance).

On the other hand, in order to define the provided functionality of a ser-

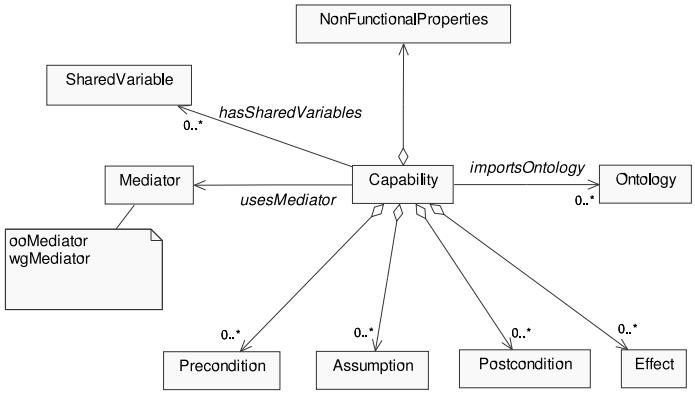

Figure 15: WSMO Capability class

vice, the notion *Capability* is introduced. Figure 15 pictures service capability model. The recommended non-functional properties are: *Accuracy, Contributor, Coverage, Creator, Date, Description, Financial, Format, Identifier, Language, Network-related QoS, Owner, Performance, Publisher, Relation, Reliability, Rights, Robustness, Scalability, Security, Source, Subject, Title, Transactional, Trust, Type, Version.* Apart from the nonFunctionalProperties, instances of Capability class can directly import ontologies, as long as no conflict is needed to be resolved, or by using instances of ooMediators. A capability can also be linked to a goal through a *wgMediator*. Moreover the functionality of a service is associated to a number of *preconditions, assumptions, post-conditions* and *effects* which are subclasses of Axiom class (see Figure 7). Preconditions and assumptions specify the world state before the execution of the service. Whereas, post-conditions and effects describe the state of the world after the execution of the service. Finally, capability is associated to a number of shared variables, which represent the variables that are shared between preconditions, assumptions, post conditions and effects.

4.3 Interoperation

Representing the service operation and data transformation, WSMO introduces the *Service* class element. More specifically, Figure 16 presents the conceptual model of Service class, which is used for service definition. The nonFunctional-Properties associated with the Service class can be used for defining attributes of a service are *Accuracy, Contributor, Coverage, Creator, Date, Description, Financial, Format, Identifier, Language, Network-related QoS, Owner, Performance, Publisher, Relation, Reliability, Rights, Robustness, Scalability, Security, Source, Subject, Title, Transactional, Trust, Type, Version.* Using the *importsOntology* association, Service can import ontologies as instances of Ontology class as long there are no conflicts to be resolved. Otherwise, the import of ontologies is carried out through mediators, specifically instances of ooMediator class to deal with merging, aligning and transforming issues. Moreover service use instances of *wwMediator* class to deal with process and protocol

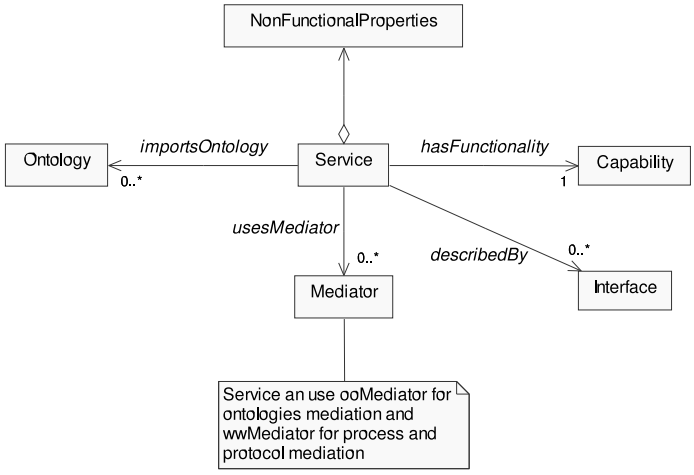

Figure 16: WSMO Service class

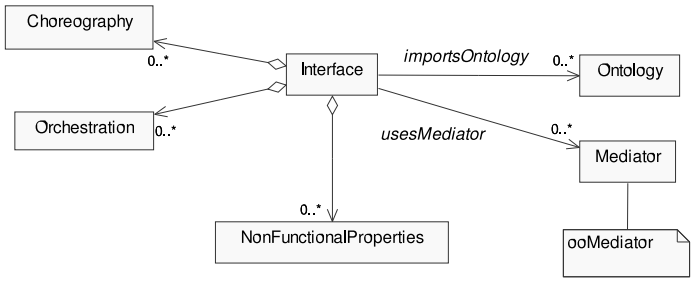

Figure 17: WSMO Interface class

mediation.

In addition, services are described by interfaces. The *Interface* class is used to denote how the capability of the service can be fulfilled. For this purpose there are two concepts introduced, as pictured in Figure 17. The *choreography* provides the necessary information for communicating with the service, whereas the *orchestration* describes how the service makes use of other services in order to achieve its capability. Interface can also import instances of Ontology class through ooMediators. Finally, the non-functional properties recommended for defining the Interface class are: *Accuracy, Contributor, Coverage, Creator, Date, Description, Financial, Format, Identifier, Language, Network-related QoS, Owner, Performance, Publisher, Relation, Reliability, Rights, Robustness, Scalability, Security, Source, Subject, Title, Transactional, Trust, Type, Version.*

As mentioned in the previous sections, WSMO framework includes the notion of mediators as a solution mechanism to the conflicts and interoperability issues possibly arising. There are four types of mediators used:

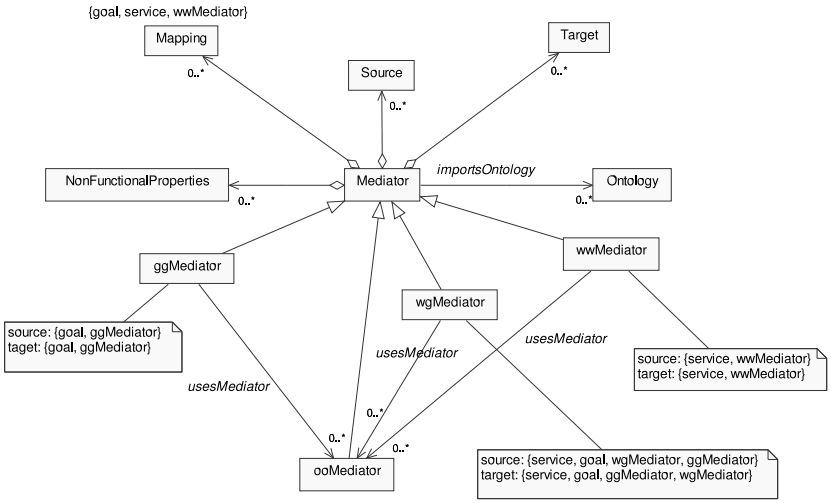

Figure 18: WSMO Mediator conceptual model

- *ooMediator*, which links two ontologies, resolving possible mismatch issues between them.
- *ggMediator*, which links two goals in the context of refinement of the source goal into the target goal.
- *wwMediator*, which links two services.
- *wgMediator*, which links services with goals, meaning that a service fulfills the goal that is linked to.

Figure 18 presents the conceptual model of Mediator class. The non-functional properties recommended for defining the Mediator class are: *Accuracy, Contributor, Coverage, Creator, Date, Description, Financial, Format, Identifier, Language, Network-related QoS, Owner, Performance, Publisher, Relation, Reliability, Rights, Robustness, Scalability, Security, Source, Subject, Title, Transactional, Trust, Type, Version.* The four types of mediators are subclasses of the Mediator class. Mediators can directly import ontologies as long as there are no conflicts to be resolved. Otherwise, wgMediators, ggMediators and wwMediators use ooMediators in order to resolve the mismatches. Moreover mediators have *sources* and *targets* as either services, goals or other mediators according to their kind. Finally, Mediator class is associated to a *mapping* which is either a goal that declarative describes it, or a service that actually implements it, or a wwMediator that links to a service that actually implements it.

4.4 Composition

The mechanism for describing WSMO choreographies and orchestrations is based on the Abstract State Machines (ASMs) methodology. A full description of the mechanism is presented in [1] and [2]. Figure 19 presents the WSMO

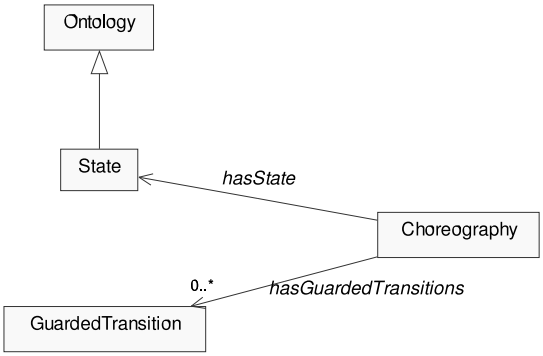

Figure 19: WSMO Choreography class

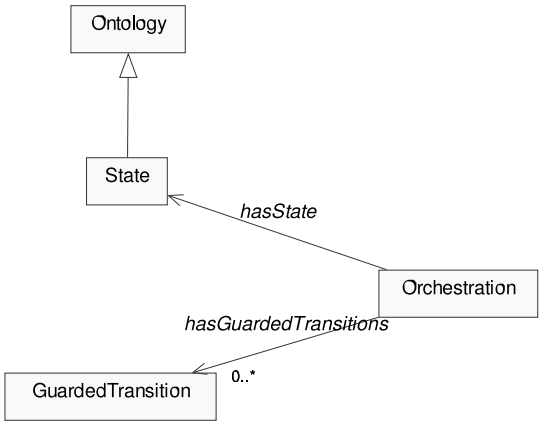

Figure 20: WSMO Orchestration class

Choreography model. The model abides with the following principles: (1) it is state-based, (2) it represents a state by an algebra, and (3) it models state changes by guarded transition rules that change the values of functions and relations defined by the signature of the algebra. The *State* class is a specification of the Ontology class with an additional non-functional property concerning the extensibility of the concept, relation, or function of the ontology. G*uarded transitions* are expressed in the form of: *if* <*condition*> *then* <*updates*>, where <*condition*> is an arbitrary WSML axiom, formulated in the given signature of the state and <*updates*> consist of arbitrary WSMO Ontology instance statements.

Figure 20 pictures the WSMO orchestration model. The model is alike to the one previous described for the choreography. The difference lies on the usage of mediators to link the orchestration to other goals or web services. Thus, the guarded transitions can be expressed as: *if* <*condition*> *then* <*mediator ID*>, with either wwMediators or wgMediators used.

4.5 Invocation

Grounding describes the mechanism that enables the service and the service requestor (client) to communicate with each other for the invocation process. Nowadays the majority of the services provided are syntactically described, so the majority of the clients are also implemented to deal with syntactic information concerning service description and invocation. Considering the Semantic Enabled Web Services and current state of the art, WSMO service grounding [3] explores the relationship between WSMO and the syntactical descriptions of a Web services. Thus, data in WSMO ontologies has to be bidirectionally mapped to XML data (usually described using XML Schema) and the functional and behavioral service descriptions in WSMO have to be related to the description construct present in WSDL.

With respect to WSMO/XML mapping, there are three approaches that can be followed: a) based upon transformations between the source XML data and an XML representation of the target ontology e.g. XSLT, b) ontological transformation between a semantic representation of the source XML tree and the target ontology, using an ontology mapping approach and c) direct mapping between the source XML data and the target ontology, using a mapping language specifically developed for this purpose. The first approach is working but the XSLT are intended for transformations between hierarchical structures rather than flat ones, like WSML [18]/XML documents. The third one is promising due to the simplicity of implementation. Finally the second approach considers the mappings between the WSML instances and the XML ones. The basic activities which this approach involves are a) the definition of a mapping from XML Schema Conceptual Model to the WSMO Ontology metamodel, b) application of the mapping in the first point to automatically create an ad-hoc WSMO ontology from a specific XML schema and c) the creation of bidirectional mapping rules to be used for the transformation between XML instances and WSML instances.

As far as the relation of WSMO with WSDL is concerned there are two different directions. In the first one WSDL elements extend the WSMO descriptions in order to specify the implementation details. On the other hand, in the second approach, WSMO elements are used to extend and semantically annotate the implementation-level WSDL. Considering the wide usage of Web Services standards, also by the industry, it's is preferable to exploit the already created WSDL descriptions of services and to extend them with semantic annotations.

The semantic annotation of WSDL descriptions is carried out in their abstract part, meaning that only interfaces, operations and operations are semantically extended. The syntax consists of a single element, *wsmo:webService* (the prefix *wsmo* stands for the namespace URI [15]) with a mandatory attribute named *ref* containing the URI identifier of a Web Service described in a WSMO document. First of all, WSDL interface is a set of operations each consisting of multiple messages (input and/or output) according to a message exchange pattern, whereas in WSMO Web Services have capabilities and interfaces consisting of orchestration and choreography. Assuming similarity between the intent of WSMO Web service and a WSDL service, an interface in WSDL is meant to provide a specific coarse-grained functionality. For discovery purposes, it is useful to ascribe a WSMO capability to a WSDL interface to specify the functionality it provides, and for invocation or composition purposes it is useful to ascribe

a WSMO choreography to a WSDL interface so that the possible and allowed orderings of invocations of the operations become known. Moreover WSMO choreography does not include the notions of inputs, outputs and operations that can be directly mapped to the corresponding elements of WSDL. Thus, for the purposes of invocation and composition of discovered services we can map WSDL interface operations to WSMO Web services, also specifying the capabilities of the operations. Finally the WSDL service element which specifies the access information can be extended with a WSMO Web service to indicate that this service only offers a subset of the capability of what is offered by its interface.

5 METEOR-S

The Managing End-To-End OpeRations (METEOR) [5] project at the LSDIS Lab, University of Georgia, focused on workflow management techniques for transactional workflows [26]. Its follow on project, which incorporates workflow management for semantic Web services is called METEOR-S. A key feature in this project is the usage of semantics by updating the today's well defined industry standards for the complete lifecycle of semantic Web processes, which represent complex interactions between Semantic Web Services.

5.1 Service Discovery

The basic directions for achieving semantic annotation of a Web Service's description, deal with annotating the service's operations, their inputs and outputs, introducing effects and preconditions and specifying its categorization. The mechanism used by the framework for achieving semantic annotation, is to relate the WSDL's constructs with concepts defined in a semantic model, i.e. ontology. Thus, METEOR proposes WSDL-S, a semantically enriched WSDL 2.0 document [14].

An annotation on an element or complex type is considered to be an input/output semantic annotation if the message is referenced by an element attribute of an input/output element. The semantic annotation is associated with the operation defined by the parent element of this input/output element or with each of the multiple operations that reference the input/output. If the annotation is applied to a complex type that is referenced by multiple messages from multiple messages, the semantic annotation applies each operation that references any of those messages for input/output. Figure 21 pictures the conceptual model for adding semantic information at the inputs/outputs of a Web Service. The *Element* class has at most one optional reference to a concept defined in a semantic model, i.e. ontology. Thus, the *ModelReference* object is used to handle one-to-one mappings of schema element to ontology concept. For the complex types there two alternatives for achieving semantic annotation: a) the bottom level approach and b) the top level one. In the first one, the annotation takes place on the leaf level, meaning that the elements contained in a complex type are semantically characterized using ModelReference. Whereas in the second approach, the whole complex type is annotated either via *SchemaMapping* or ModelReference. Schema mappings are used by the framework in order to handle many-to-one or one-to-many mappings with semantic models.

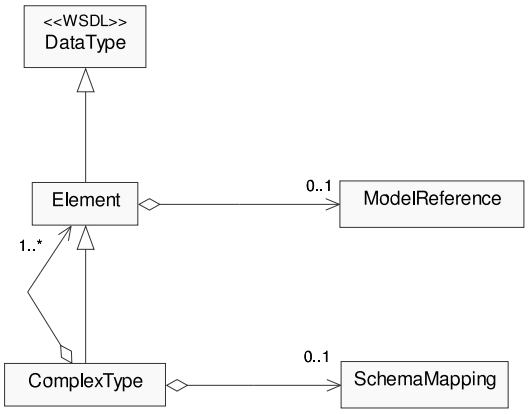

Figure 21: Annotating the inputs/outputs of a Web Service via their types

The framework introduces the concepts of *Precondition* and *Effect* that are associated with a Web Service operation in a portType. The preconditions specify the requirements that must be fulfilled in order for an operation to be invoked. On the other hand, the effects describe the changes in the state of the "world" that are expected after the invocation of the operation. A request-response operation is associated with at most one precondition and zero or more effects. Figure 22 presents the conceptual model of a Web Service operation associated with its preconditions and effects. For both of Precondition and Effect the ModelReference specifies the URI of the part of a semantic model that describes them. Moreover, both preconditions ad effects are associated with an *expression* for defining them. The format of these expressions is not specified by the WSDL-S framework, but is defined by the semantic representation language used to express the semantic model.

WSLD-S framework includes a mechanism to add categorization information to services which could be used while publishing services in registries such as UDDI and in service discovery by narrowing the range of candidate services. Figure 23 presents the model of specifying the service categorization. Each Web Service PortType can be associated to a number of *Category* objects. For each Category, its name within a taxonomy, a URI reference to the taxonomy definition, the value and the code associated with a category in the taxonomy, are specified.

5.2 Composition

This stage of Web Service lifecycle involves creating a representation of Web processes. METEOR-S framework has chosen BPEL4WS [17] for creating the *abstract process* as it is the de facto industry standard and provides a rich set of constructs for modelling workflow patterns [28]. Design of abstract processes involves the following tasks.

1. Creating the flow of the process using the control flow constructs provided by BPEL4WS.

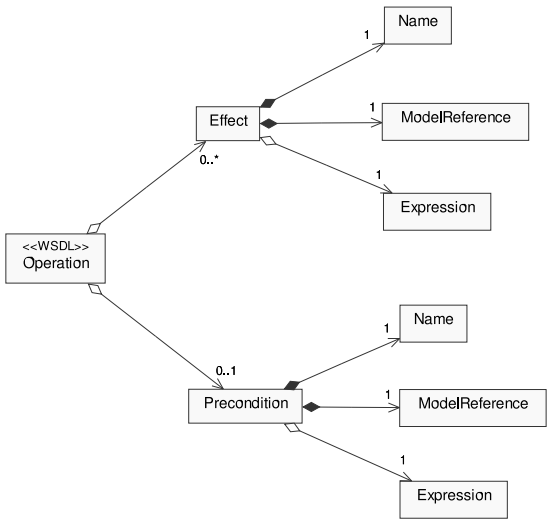

Figure 22: Specifying preconditions and effects

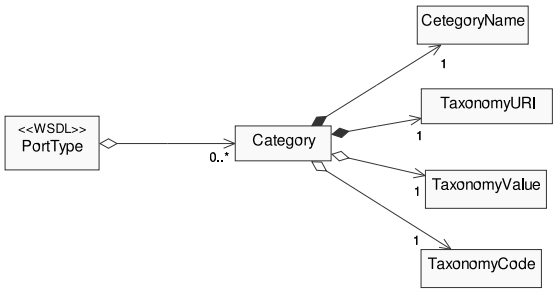

Figure 23: Service categorization

2. Representing the requirements of each service in the process by specifying service templates, which allow the process designer to either bind to a known Web service or specify a semantic description of the Web service.

3. Specifying process constraints for optimization.

5.3 Invocation

As far as accessing and invoking of a Web Service is concerned, METEOR-S framework uses the mechanism already defined in nowadays standards involving WSDL and SOAP definitions.

6 Comparison

In the previous sections we have presented the three more significant initiatives for semantically annotating Web Services descriptions. Before comparing them regarding the service discovery, interoperation, composition and invocation directions, we have to denote the general differences concerning each approach objectives and principles.

OWL-S purpose is to define an ontology for describing Web Services that enable users and software agents to automatically discover, invoke, compose and monitor Web resources offering services under specific constrains. Although, WSMO also aims at creating an ontology for describing various aspects related to Semantic Web Service, including discovery, composition and invocation, it is mainly focused on solving the interoperability problem. Both of these initatives are attempting to fulfill their goals by proposing a new framework for describing Web Services in a top down approach which only exploits Web Services standards regarding the grounding and invocation issues. More specifically they build a semantic framework describing a Web Service concepts which is grounded to nowadays standards, i.e. WSDL and SOAP. On the other hand, METEOR-S framework attempts to introduce semantic information to the complete Web process lifecycle by providing constructs for adding semantics to current industry standards.

Ontologies are the key for semantic annotation of Web Service description. According to OWL-S purpose and principles it is assumed that the language for specifying the taxonomy and the relations among the concepts used for semantically describe a Web Service is OWL. Moreover there is no mechanism specified for reusing already existed ontologies, that will also solve the arising heterogeneity problems. On the contrary, in WSMO ontologies are described in a meta-level, that allows describing all the elements of the ontologies used for the modeling of services. Thus, this mechanism offers more flexibility not dictating the formalism (i.e. the specific ontology language) to be used in describing imported ontologies. Furthermore the usage of mediators can solve the possible mismatching and heterogeneity problems of importing ontologies. In general, the notion of mediators in WSMO is very important not only for aligning ontologies but also goals, services and goals with services. It is actually strongly connected with the main purpose of the framework; solving the interoperability issue. Alike to WSMO, METEOR-S framework takes an agnostic approach to ontology representation languages. As the matter of fact, Web service developers are free to annotate their Web services with their choice of ontology language

Lifecycle part	OWL-S	WSMO	METEOR-S
Service discovery	Profile	Goal + Capability	WSDL-S
Interoperation	Process Model	Service	Abstract Process BPEL4WS
	Process	Capability	Abstract Process BPEL4WS
	Input	Precondition	Input
	Output	Postcondition	Output
	Precondition	Assumption	Precondition
	Result	Effect	Effect
Composition	Composite Process	Interface	BPEL4WS
	Control Construct	Choreography	BPEL4WS
		Orchestration	BPEL4WS
Invocation	OWL-S/WSDL grounding	WSMO/WSDL grounding	WSDL

Table 1: Conceptual correspodences

(such as UML or OWL). This is significant since the ability to reuse existing domain models expressed in modeling languages like UML can greatly alleviate the need to separately model semantics.

Finally, taking into consideration the previous presentations of the three frameworks and since WSMO is based on the conceptual work done in WSMF, it is more conceptually focused than OWL-S and METEOR-S and it is determined to follow from this conceptualization all the way to a formal execution model.

In the following sections we will discuss the differences of the examined initiatives in the bases of service discovery, interoperation, composition and invocation. Table 1 outlines the main conceptual correspondences among OWL-S, WSMO and METEOR-S frameworks.

6.1 Service discovery

In OWL-S service profile presents the intended purpose of the service, both describing the functionality offered by the provider and the one needed by the requester. Whereas WSMO distinguishes the provider point of view from the requester point of view, introducing two different concepts: goal and capability. More specifically, a goal in WSMO specifies the objectives that a client may have when she consults a Web Service. Furthermore, a Web Service capability defines the service by means of what the service offers to the requester i.e. what functionality the service provides. As far as cardinality issues are concerned, OWL-S framework does not introduces and restrictions, i.e. a service can be presented by none or more than one profiles. Web Services in WSMO can be associated to none or more goals by using the right mediators and to only one capability. Yet, the link of a capability to several goals by using different mediators can be seen as the definition of multiple service profiles in OWL-S.

Information transformation is expressed in OWL-S using referenced instances of inputs and outputs which are defined in the corresponding service model. Moreover, the state change is also expressed by the referenced preconditions and results instances of service model. In WSMO sub-classes of Axiom class, precondition, postcondition, assumption and effect that are associated to the service capability, are used to express the information transformation and state change.

On the other hand, the analogous of OWL-S service profile and WSMO goal and capability is the abstract part of the WSDL 2.0 extension of METEOR-S called WSDL-S.

6.2 Interoperation

OWL-S process model represents how a service works. Each Web Service is viewed as a process and it is described by at most one process model. The process model specifies the functional properties of the process (service) together with the composition details in case of a composite one. In WSMO, the Service class corresponds to the service model concept of OWL-S. Service class describes the service functionality together with its interface for specifying the orchestration and the choreography of a composite service. Thus, WSMO capability can be considered as the analogous of the process concept of OWL-S, and more precisely of the atomic process, while simple process is not considered in WSMO. Regarding the cardinality issues, a Service can be associated with zero or one capability and none or more interfaces. Considering the METEOR-S approach the operation of a Web Service is described by the concrete part of the WSDL-S document and the Abstract Process model of BPEL4WS.

As far the information transformation and the state change are concerned the three initiatives introduce very similar concepts. Both the inputs and outputs of a OWL-S process and preconditions and postconditions of WSMO capability along with the semantic annotation of inputs and outputs of METEOR-S framework are describing the information transformation. Moreover for the description of state change, OWL-S introduces the preconditions and results of a process, WSMO the assumptions and effects of capability and METEOR-S the preconditions and effects.

Finally, in case of the OWL-S and WSMO framework the issue of profile/model consistency should also be considered. Information transformation and state changed are described both considering the service discovery and also service operation. In OWL-S, service profile IOPEs are a subset of the corresponding ones specified in service model. However, the profile's IOPEs descriptions and those in the model can be inconsistent without affecting the validity of the OWL expression. Yet, this inconsistency may result in a failure to provide the service functionality. Very alike, in WSMO the link between service capability and goal i.e. the wgMediator, does not contain any requirement on the consistency of the preconditions, post-conditions, assumptions and effects defined in the goal and the ones defined in the service capability.

6.3 Composition

As mentioned in the previous section OWL-S framework describes the functionality of a service using process concept. In case of a composite service, the

process model defines a composite process. In the same way WSMO introduces the interface class. OWL-S composite process describes the choreography of a service, meaning in which mechanism the contained processes - services (either atomic or composite ones) interoperate. On the other hand, in WSMO framework interface describes using a model based on Abstract State Machines technology, both the choreography and the orchestration of a composite service. An additional difference between OWL-S and WSMO lies on the cardinality restriction of process model. The choreography of a composite service can be described by at most one model, whereas in WSMO using multiple interfaces we can specify different models for service choreography and orchestration. Finally, in METEOR-S the composition mechanism is carried based on the abstract process model and the standards introduced by BPEL4WS.

6.4 Invocation

Considering the invocation issues, all three approaches are motivated by the same fact. Nowadays the majority of the services provided are syntactically described, so the majority of the clients are also implemented to deal with syntactic information concerning service description and invocation. Apart from the METEOR-S initiative which introduces an update of WSDL for semantically annotating its parts, both OWL-S and WSMO introduce grounding mechanisms based on WSDL. OWL-S/WSDL and WSMO/WSDL grounding models are annotating the WSDL parts with references to the corresponding ontologies concepts and in addition they relate parts of the service ontology introduced with right WSDL description parts, for example operations and message parts.

7 Conclusion

The next step in Web Services evolution is the exploitation of Semantic Web in Web Services framework. The vision of Semantic Web Services is to describe the semantic aspects of a Web Service with a machine-interpretable and understandable way enabling automatic interaction with services during their entire lifecycle, i.e. not only in service discovery procedure but also in service composition, invocation and interoperation. E-Commerce and Enterprise Application Integration are only two of the areas that would benefit from the usage of semantically described Web Services, as they can enable dynamic, scalable and reusable cooperation between different systems and organizations.

This survey tries to compare in a conceptual level [24] the three latest stable versions of the three more significant initiatives for Semantic Web Services, i.e. OWL-S 1.1, WSMO D2v1.2 and METEOR-S. Considering the differences among each framework purpose, principles and characteristics, most likely according to the needs of an application different approaches or combinations of them will be exploited.

References

[1] *D14v0.2. Ontology-based Choreography and Orchestration of WSMO Services*, http://www.wsmo.org/TR/d14/v0.2/.

[2] *D15v0.1. Orchestration in WSMO*, http://www.wsmo.org/TR/d16/d16.1/v0.2/.

[3] *D24.2v0.1. WSMO Grounding*, http://wsmo.org/TR/d24/d24.2/v0.1/.

[4] *D2v1.2. Web Service Modeling Ontology (WSMO)*, http://www.wsmo.org/TR/d2/v1.2/.

[5] *METEOR: Managing End-To-End OpeRations*, http://lsdis.cs.uga.edu/projects/past/METEOR/.

[6] *METEOR-S: Semantic Web Services and Processes*, http://lsdis.cs.uga.edu/Projects/METEOR-S/.

[7] *North American Industrial Classification System (NAICS)*, http://www.census.gov/epcd/www/naics.html.

[8] *OWL-S Home Page*, http://www.daml.org/services/owl-s/2003.

[9] *OWL-S W3C submission*, http://www.w3.org/Submission/OWL-S/.

[10] *OWL W3C submission*, http://www.w3.org/Submission/OWL/.

[11] *SOAP Version 1.2 Part 1: Messaging Framework*, http://www.w3.org/TR/2003/RECsoap12-part1-20030624/.

[12] *United Nations Standard Products and Services Code (UNSPSC)*, http://www.unspsc.org.

[13] *Universal Description, Discovery, and Integration (UDDI)*, http://www.uddi.org/.

[14] *Web services description language (wsdl) version 2.0*, http://www.w3.org/TR/wsdl20/.

[15] *WSMO namespace*, http://www.wsmo.org/TR/d24/d24.2/v0.1/.

[16] *KIF. Knowledge Interchange Format*, Tech. Report 2/98-004, 1998, Draft proposed American National Standard (dpans).

[17] *Business Process Execution Language for Web Services Version 1.1*, 2003, available at http://www106.ibm.com/developerworks/webservices /library/ws-bpel/.

[18] *The WSML Family of Representation Languages, WSMO Deliverable D16, DERI Working Draft*, 2004, latest version available at http://www.wsmo.org/2004/d16/.

[19] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, *Web Services Description Language (WSDL) 1.1*, http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.

[20] M. Ghallab et al, *PDDL-The Planning Domain Definition Language V. 2*, Tech. Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[21] D. Fensel and C. Bussler, *The Web Service Modeling Framework WSMF*, Electronic Commerce Research and Applications **1(2)** (2002).

[22] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabetand B. Grosof, and M. Dean, *Swrl: A semantic web rule language combining owl and ruleml*, 2003, Available at http://www.daml.org/2003/11/swrl/.

[23] G. Klyne and J. J. Carroll, *Resource description framework (RDF): concepts and abstract syntax*, 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210.

[24] R. Lara, D. Roman, A. Pollares, and D. Fensel, *A Conceptual Comparison of WSMO and OWL-S*, European Conference on Web Services (ECOWS 2004), Erfurt, Germany, September 27-30 2004.

[25] S. McIlraith, T. Son, and H. Zeng, *Semantic Web services*, IEEE Intelligent Systems (Special Issue on the Semantic Web) (2001).

[26] A. Sheth, K. Kochut, J. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch, and I. Shvchenko, *Supporting State-wide Immunization Tracking using Multi-Paradign WorkflowTechnology*, 22nd Intl. Conf. on Very Large Databases (VLDB96), September 1996.

[27] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf, *RFC 2413 - Dublin Core Metadata for Resource Discovery*, September 1998.

[28] P. Wohed, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede, *Pattern-based analysis of bpel4ws, qut*, Tech. Report FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002, available at http://tmitwww.tm.tue.nl/staff/wvdaalst/Publications/p175.pdf.