Semantic Web Services, State of the art

Juan Bernabé Moreno
University of Granada, Department of Computer Science and Artificial Intelligence

Abstract— **Semantic Web Services seem to bring closer to the reality the vision of software created on-the-fly and delivered and paid for as fluid streams of services as opposed to packaged products. This work introduces the SWS and presents three of the top approaches intended to bring the semantics to the web services technologies.**

***Index Terms*— Semantic Web services, WSMO, WSDL-S, OWL-S**

I. INTRODUCTION

Today, the web services present several problems that semantic web services strive for solving.

The interfaces of the web services nowadays are designed to enable the communication with a single program or with the same program.

Commonly it is about web browsers that don't process the information they get, but just display it enclosed within HTML tags.

The web browser knows nothing about the information between <H1></H1>, but only that this information should be displayed with a font-size higher.

Many services in the web have the problem that they can't be combined. A service can invoke another service only after big programming efforts (e.g.: parsing the output of the other service for a given token and applying some scraping techniques). The fact that almost all web services in the web have different output and this can vary along the time makes the integration a short of Sisyphus work.

On one hand we have the different standards that are developed by different companies to serve different purposes, which avoid the existence of uniform interfaces.

On the other hand, a web service can't describe itself to other programs and therefore, can't be easily found (discovered) without the intervention of a human user.

All together should work in the imminent future of the web services with the support of semantics.

Using web services with standardized interfaces should save a lot of time and money in the development of web-based applications. Semantic web services provides Internet with additional automatisms, which enables that one application running on the home computer autonomously searches, selects and invokes web services and even more, combines them to achieve a higher level functionality.

This is the dream of software agents come true.

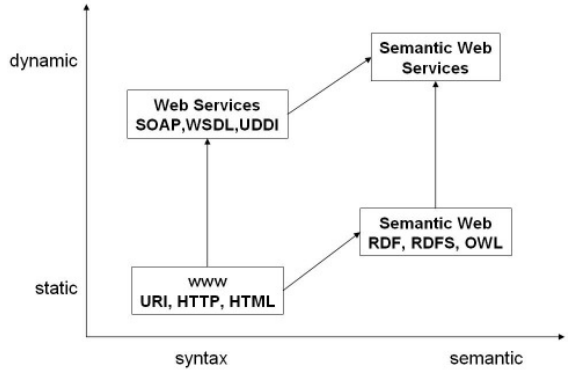

Figure 1 [10]

A. *What are Semantic Web Services good for?*

The goal of the semantic web services is the increase of automation in following web services processes:

- **Automatic discovery of web services:** the search and discovery of web services providing a given functionality in a Service Registry should be performed automatically, which is only possible by means of semantics
- **Automatic invocation of web services:** here are web services meant, that consists of several method calls (non-atomic) (e.g.: buying a CD in internet implies searching for it, selecting it, adding it to the shopping cart and paying for it). Without the support of semantics, this couldn't be possible
- **Automatic composition of web services:** should a service not be able of fulfilling the user requirements, then a composition of other web services is automatically created and performed to fulfil these requirements.

B. *Approaches towards the Semantic Web Services*

Depending on their starting point of the approach, we distinguish:

The top-down approach models the web service and its semantics independently on existing web services technologies. It takes place in an ontology language targeting the creation of an optimal web service description language. By means of the so called "grounding", where the mapping of semantic description elements to WSDL elements is specified, the relationship to the WSDL is established. We will discuss two prominent top-down approaches: the OWL-S and the WSMO [11]

Bottom-Up approaches pursue the semantic enrichment of existent technologies (especially WSDL and BPEL).

Therewith, WSDL descriptions are extended with semantic annotations of ontology concepts.

We will start by the bottom-up approach

II. WSDL-S [12]

The initiative the W3C organization is fostering the most represents the evolutionary and less ambitious approach, just because it relies on extending already existing components with semantic capabilities to overcome their limitations. The Web Services Definition Language (WSDL) is an extensible, platform independent XML language for "describing" services. It provides mainly functional information about the service: IDL description, access protocol and deployment details, etc… in general, all of the functional information needed to programmatically access a service, contained within a machine-readable format. WSDL does not include QoS, Taxonomies or Business information.

To put it simple, WSDL is a component definition language for Web service component

1) WSDL benefiting from semantics

Let's figure out the potential of adding semantic capabilities to the WSDL, referred to the web service life-cycle:

During *development*, the service provider can explicate the intended semantics by annotating the appropriate parts of the Web service with concepts from a richer semantic model.

During *discovery*, the service requestor can describe the service requirements using terms from the semantic model. 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. More importantly, semantics can make it possible to specify mappings between data exchanged through XML-based SOAP messages, which would be extremely difficult to do with syntactic representation offered by the current standards (one of the core limitations of "syntactical" WSDL)

During *invocation*, mappings can be used for data transformations. Therefore, once represented, semantics can be leveraged by tools to automate service discovery, mediation, composition and monitoring.

WSDL-S aims at augmenting the expressivity of WSDL with semantics to describe the functional aspects of a web service.

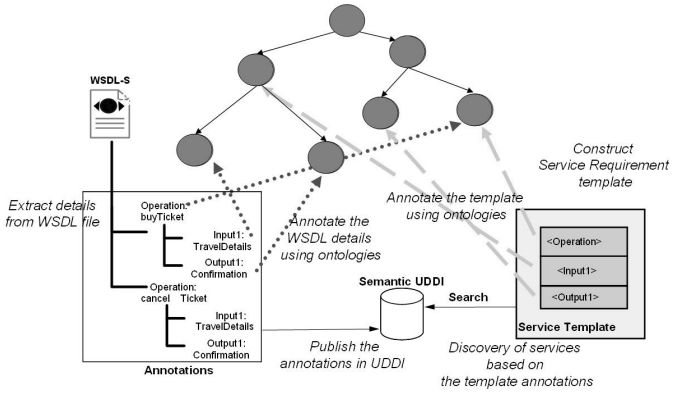

Figure 2

The Figure 2 shows the semantic publication steps, starting by the details extraction from WSDL, their annotation using ontologies and the publication of these annotations in UDDI, and the steps of the semantic discovery, starting by the construction of the service requirements template, the annotation of the template using ontologies and the service discovery based of template annotations.

2) Considerations adding semantics to WSDL

There are some core aspects to be taken into account when adding semantics to the WSDL

WS standards are becoming the preferred technology for application integration. Therefore, WSDL-S relies on the idea that any approach to adding semantics to Web Services should be specified in an upwardly compatible manner so as to not disrupt the existing install-base of Web Services.

Whatever mechanism chosen to add semantics to the WSDL should be independent of the semantic representation language. There are a number of potential languages for representing semantics such as OWL [OWL], WSMO [WSMO], and UML [UML]. Each language offers different levels of semantic expressivity and developer support. But tying the Web services standards to a particular semantic representation language would result into a lack of flexibility. Moreover, the mechanism should allow the association of multiple annotations written in different semantic representation languages, because service providers may choose to annotate their services in multiple semantic representation languages to be discovered by multiple discovery engines.

Another characteristic the annotation mechanism should have is the support of data types that are described in an XML schema-way. A common practice in Web services-based integration is to reuse interfaces that are described in XML. We believe that the semantic annotation of service inputs and outputs should support the annotation of XML schemas. An approach that does not address XML schema-based types will not be able exploit exiting assets or allow the gradual upgrade of deployed WSDL documents to include semantics.

The support for rich mapping mechanisms between Web Service schema types and ontologies is also a critical success factor. Provided the importance of annotating XML schemas in Web service descriptions, attention should be given to the problem of how to map XML schema complex types to

ontological concepts. For example, if the domain model is represented in OWL, the mapping between WSDL XSD elements and OWL concepts can be represented in any language of user's choice such as: RDF, OWL, XSLT, XQuery or any other arbitrary language as long as the chosen language is fully qualified with its own namespace.

3) What is WSDL-S?

Offer an evolutionary and compatible upgrade of existing Web services standards.

WSDL-S externalizes the semantic domain models, due to its ontology representation languages independency. WSDL-S allows for reusing existing domain models and the annotation using multiple ontologies (from the same or from different domains)

The semantic annotations are done by means of two entities:

Annotating message types (XSD complex types and elements)

- extension attribute : modelReference (semantic association)
- extension attribute : schemaMapping (schema/data mapping)

Annotating operations

- extension elements : precondition and effect (child elements of the operation element)
- extension attribute : category (on the interface element)
- extension attribute : modelreference (action) (on operation element)

Sample

……

```
<xs:element name= "processPurchaseOrderResponse"
 type="xs:string
wssem:modelReference="POOntology#OrderConfirmation"/>
</xs:schema>
</types>
<interface name="PurchaseOrder">
<wssem:category name= "Electronics"
taxonomyURI=http://www.naics.com/ taxonomyCode="443112" />
<operation name="processPurchaseOrder" pattern=wsdl:in-out
  modelReference = "rosetta:#RequestQuote" >
<input messageLabel = "processPurchaseOrderRequest"
  element="tns:processPurchaseOrderRequest"/>
<output messageLabel ="processPurchaseOrderResponse"
  element="processPurchaseOrderResponse"/>
<!—Precondition and effect are added as extensible elements on an
operation>
<wssem:precondition name="ExistingAcctPrecond"
 wssem:modelReference="POOntology#AccountExists">
<wssem:effect name="ItemReservedEffect"
 wssem:modelReference="POOntology#ItemReserved"/>
</operation>
</interface>
```

Following annotation elements have been used in the example:

- extension element : *Precondition*: A set of assertions that must be satisfied before a Web service operation can be invoked ("must have an existing account with this company" or "only US customers can be served")
- extension element : *Effect*: defines the state of the world/information model after invoking an operation ("item shipped to mailing address", or "the credit card account will be debited")
- extension attribute : *Category*: models a service category on a WSDL interface element (category = "Electronics" Code = "naics:443112")
- extension element : *Action* annotated with a functional ontology concept.(action = "Rosetta:RequestQuote")

4) Why WSDL-S?

This approach is just the natural evolution of the existing well-consolidated WSDL dotted with more expressivity by employing concepts analogous to those in OWL-S.

Its key success factors are:

- Ease in adoption: as this approach is simple, light-weight and upwardly compatible with the existing WSDL standard
- Semantic representation language independency, which allows for the re-usage of domain models, the flexibility of modeling language choice and the annotation with multiple ontologies
- Ease in tool upgrades (e.g. wsif / axis invocation)

Even other more revolutionary approaches to the semantic web services are pursued, leveraging the existing WSDL and XML schemas for business documents and the set of tools to exploit them is and will be critical. WSDL-S already positioned itself as the best candidate to bridge the gap between those revolutionary approach and the

III. Web Services Modeling Ontology [2]

Web Service Modeling Ontology is a conceptual model for the web services description, in other words, one semantic web services core elements ontology

Similarly to other initiatives, the ultimate goal of WSMO is enabling the automatic service discovery and their execution, as well as paving the way to a holistic yet simple integration solution. This will allow for the automatic cooperation of non-dependant services to achieve a common functionality at a higher level.

A. The WSMO Working group

The WSMO was founded to achieve a mission consisting of the following 4 points:

- Strengthening European Research and Industry in Semantic Web and Semantic Web Services
- Working towards international standardization together with US-based DAML program
- Promoting research results to industry and academia through joint dissemination

- Strengthening world-wide research and standardization in Semantic Web and Semantic Web Services field

The research efforts are organized in three core subprojects, as shown in Figure 3:

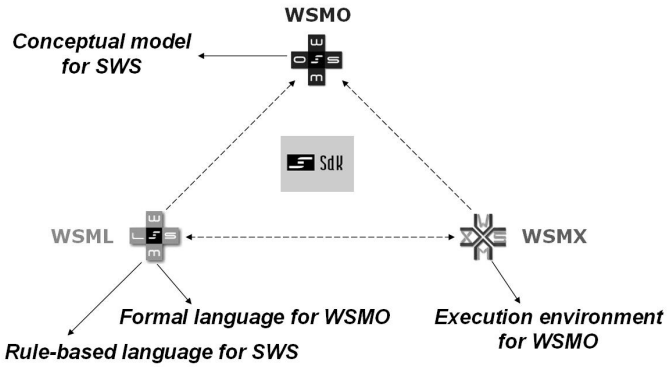

Figure 3

WSMX is an execution environment that enables discovery, selection, mediation, invocation and interoperation of SWSs. The development process for WSMX includes establishing a conceptual model, defining its execution semantics, developing the architecture of the system, designing the software and building a working implementation of the system. The research results for WSMX provide guidelines and justification for a general SWS architecture

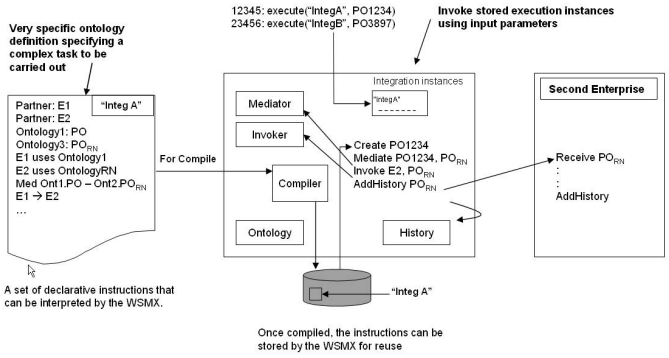

Figure 4

WSML [4] , the Web Service Modeling Language is a language for the specification of ontologies and different aspects of Web services. In this respect WSML provides a syntax and semantics for the WSMO. WSML uses well-known logical formalisms in order to enable the description of various aspects related to Semantic Web Services.

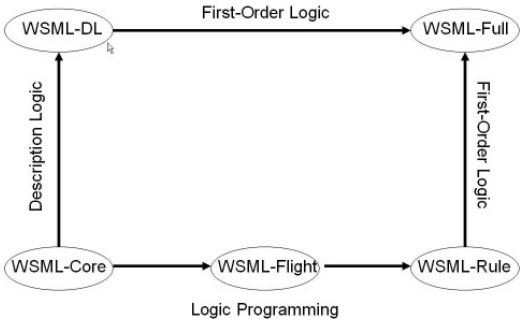

Figure 5

The Figure 5 depicts the five variants of WSDL and their relationships:

WSML-Core: this language is defined by the intersection of Description Logic and Horn Logic, based on Description Logic Programs. It has the least expressive power of all the languages of the WSML family and therefore has the most preferable computational characteristics. WSML-Core provides support for datatypes and datatype predicates

WSML-DL is an extension of WSML-Core which fully captures the Description Logic *SHIQ*(D), which captures a major part of the (DL species of the) Web Ontology Language OWL

WSML-Flight is an extension of WSML-Core with such features as meta-modeling, constraints and nonmonotonic negation.

WSML-Rule is an extension of WSML-Flight in the direction of Logic Programming. The language captures several extensions such as the use of function symbols and unsafe rules.

WSML-Full unifies WSML-DL and WSML-Rule under a First-Order umbrella with extensions to support the nonmonotonic negation of WSML-Rule. It is yet to be investigated which kind of formalisms are required to be achieved.

B. The WSMO in depth

In Figure 6 we can see the high level notions the WSMO is founded upon:

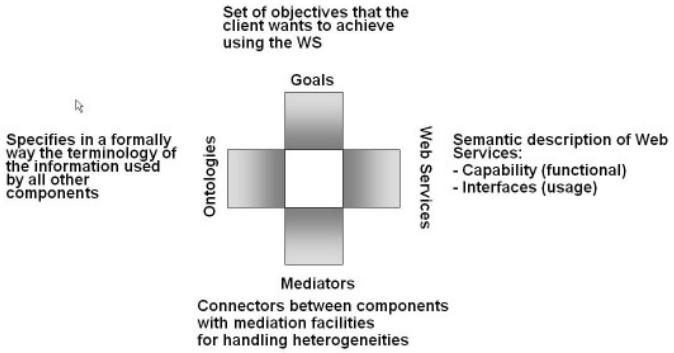

Figure 6

We will focus first on the Web services themselves and specifically on the way they are described.

WSMO separates the functional description of the web service or *capability* and their usage of the web service, which

requires the specification of their *interfaces*. There are two kinds of interfaces: choreography –when the ws is invoked- or orchestration –when the ws invokes other web services to perform its goals-, as we will see in further sections

1) Web services specification in a WSMO manner

The web services are specified by providing following elements:

- Non functional properties (conventional web service is added non-functional properties like the complete description of its elements, indicators about the quality of service (QoS), etc)
- Imported Ontologies
- Used mediators
  - *OO Mediator:* importing ontologies with mismatch resolution
  - *WG Mediator:* link to a Goal wherefore service is not usable a priori
- Pre-conditions
  What a web service expects in order to be able to provide its service. They define conditions over the input.
- Assumptions
  Conditions on the state of the world that has to hold before the Web Service can be executed
- Post-conditions
  Describes the result of the Web Service in relation to the input, and conditions on it
- Effects
  Conditions on the state of the world that hold after execution of the Web Service (i.e. changes in the state of the world)

Let's see one example:

namespace {_"http://example.org/CreditCardCharging#",
dc _"http://purl.org/dc/elements/1.1#",
po _"http://example.org/purchaseOntology#",
foaf _"http://xmlns.com/foaf/0.1/",
wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
ccci
_"http://www.example.org/CreditCardChargingInterfaceOntology#"}

webService _"http://example.org/CreditCardChargingWebService"
nonFunctionalProperties
dc:title **hasValue** „Credit Card Charging Web Service"
dc:creator **hasValue** „Association of all Credit Card Companies"
dc:description **hasValue** "web service for charging a credit card with an given amount and creating a remittance order for a given recipient"
dc:publisher **hasValue** " Association of all Credit Card Companies"
dc:date **hasValue** "2006-01-12"
dc:type **hasValue**
<<http://www.wsmo.org/2004/d2/#webservice>>
dc:format **hasValue** "text/html"
dc:language **hasValue** "en-us"
version **hasValue** "$Revision: 1.5 $"
endNonFunctionalProperties
importsOntology _"http://example.org/purchaseOntology"
capability CreditCardChargingCapability
interface CreditCardChargingInterface
importsOntology
_http://www.example.org/CreditCardChargingInterfaceOntology
choreography CreditCardChargingChoreography
orchestration CreditCardChargingOrchestration

2) Choreography and Orchestration

These concepts are intended to enable the automatic service execution

Choreography takes up how the user interacts with the service, to make use of its functionality

Orchestration handles how the functionality of the services is achieved by means of the aggregation of other web services.

Figure 7 explains better how orchestration and choreography take place.

3) Goals specification

WSMO predicates the ontological de-coupling of Requester and Provider, actually derived from task / problem solving methods/domain model.

The requests are therewith structured and reusable Requests may in principle not be satisfiable

Goals are linked to web services by means of ontological relationships and mediators to resolve the conceptual heterogeneity

To specify a goal, following elements are typically used:

- Non functional properties
- Used mediators:
  - A goal can import ontologies using ontology mediators.
  - A goal may be defined by reusing an already existing goal. This is achieved by using goal mediators.
  - Post-conditions describe the state of the information space that is desired.
  - Effects: describe the state of the world that is desired.

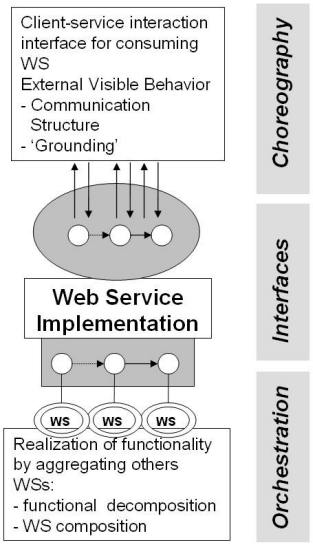

Figure 7

4) Mediation

In words of Christoph Bussler and Dieter Fensel, WSMF strictly enforces safe sex between components. They are never allowed to touch each other without a mediator in-between.

The Figure 8 provides a clear sample where mediation is required: Ontology 1 and Ontology 2 specify "address" in different ways; the application of a mediator will ensure that these concepts are understood uniformly.

The heterogeneity (or better said the lack of homogeneity) is the reason why for each invested dollar in programming, another 5 to 9 dollars are invested in integration [6]. Continuous mismatches at structural, semantic/conceptual level justify these economic efforts when the integration becomes a business requirement.

Thus, mediators are in plain English, components that resolve mismatches. Moreover, they allow for the declarative description of any arbitrary web service.

The types of mediation within Semantic Web Services are related to:

- Heterogeneous Data Sources
- Heterogeneous Communication patterns (protocol)
- Heterogeneous business processes

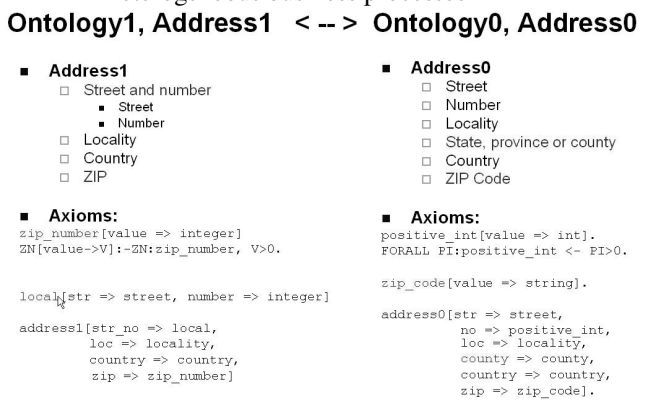

Figure 8

According to the entities they connect, mediators can be classified as (see Figure 9):

- *OO Mediators:* importing ontologies with heterogeneity resolution
- *GG Mediator:* goal definition by reusing an already existing goal allows definition of Goal Ontologies
- *WG Mediator connects web services and goals, which means that the web service/s is/are used to achieve the goal*
- *WW Mediator connect two web services.*

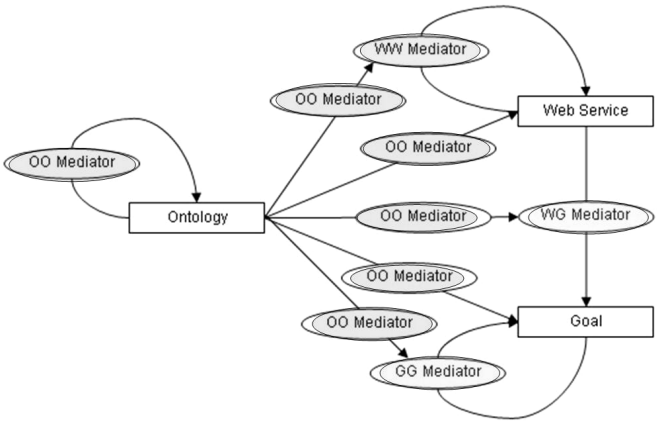

Figure 9

According to their function, they can be sub-classified in refiners and bridge:

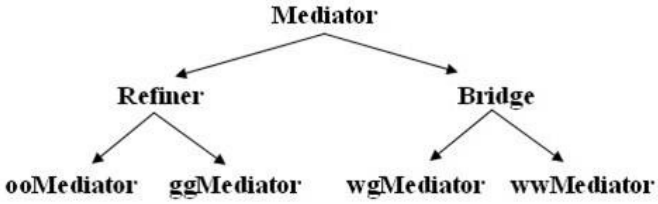

Figure 10

To explain what a goal refinement is about, we provide following example (see Figure 11)

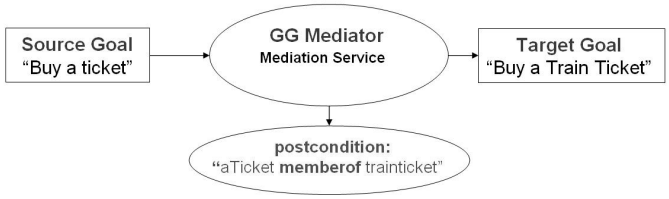

Figure 11

IV. OWL-S

The Web Ontology Language for Services (OWL-S) [7] is created upon the DAML-S, which is based on DAML+OIL (currently in Version 1.1).

The goal of the developers of OWL-S is the connection of Web Services and the semantic web to end up providing the Semantic web services.

To achieve this goal, OWL-S should provide following functionality:

– Automatic Web Service Discovery

– Automatic Web Service Invocation by a client or software agent. The execution is seen as a sequence of functional invocations and requires that the Software-Agent recognizes the interface semantic of the to-be-called WS

– Automatic Web Service composition and interoperation: given a request, the selection, composition and cooperation of web services to fulfill this request.

To provide the mentioned functionality, OWL-S is based on technologies already in place for Web Services (like SOAP and WSDL), adding types and classes to them, with the purpose of describing the web services functionality in a

machine understable way (covering not only the control and data flow, but also preconditions and effects)

Semantic Web Services (SWS) in OWL-S are described by four ontologies: “Service“, ”ServiceProfile“, ”ServiceModel“ and ”ServiceGrounding“

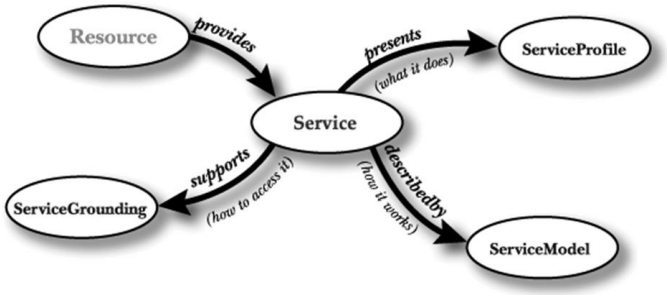

Figure 12

The motivation for structuring the Service-Ontology like shown in the Figure 12 resides in the need for describing three core characteristics of a Web Service:

- Which functionality is provided by the service?
- How is the serviced used?
- Which effects does the service have or how the user interacts with the service?

A. *Service Ontology*

For each service there is one ontology of the Service class describing the service.

One instance of the Service class supports a service grounding, is described by a service model and presents a service profile.

B. *Service Profile Ontology*

This ontology takes up the publishing of the service and describes at a certain level of abstraction the functionality the service is intended to provide. That enables that an agent decides if the service fulfills the required functionality in the required way (QoS, etc).

The set of properties to be specified per Web service can be separate into two groups:

- Non functional: not relevant for the semantic description of the service but crucial for its usage
- Functional: in/out parameters (“hasnput”, “hasoutput”), preconditions and effects.

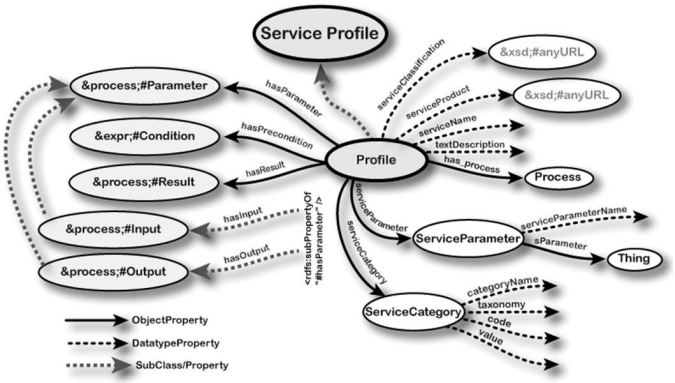

Figure 13

C. *Service Model Ontology*

This ontology provides a description on how the service is to be used and how it works. To do that, the web service control flow is modeled as a process. By means of inputs, outputs, preconditions and effects the description on how the web service tasks are carried out is given. A process can be seen as the specification of the way a client should interact with the service.

The web service flow consists of atomic processes, simple processes and composite processes.

An *atomic process* is just a process that doesn’t consist of sub-processes and can be bound to a WSDL operation and therefore can be invoked directly.

A *simple process* is just a layer of abstraction of atomic processes and has also in/output parameters, preconditions and effects. The only difference is that they are not bound with the grounding (although it is assumed, that simple processes can be executed)

Composite processes consist of simple and atomic processes whose flow is defined by using following operators: sequence, split and join, if-then-else and choice, any-order, repeat-while and repeat-until (see Figure 14)

D. *Service Grounding*

This ontology describes how the service is accessed and how the interaction with the services should happen. Communication protocols, messaging formats, serialization, transport, port number and location belong to the service grounding. The main task of this ontology consists of the serializing of data types and parameters of OWL-S and the packing of them into a concrete message, as well as enabling the communication between components.

It is done by means of WSDL as shown in Figure 15

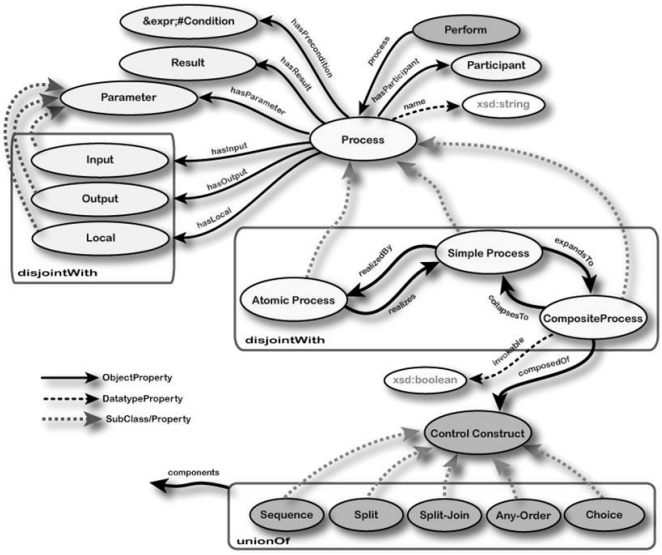

Figure 14

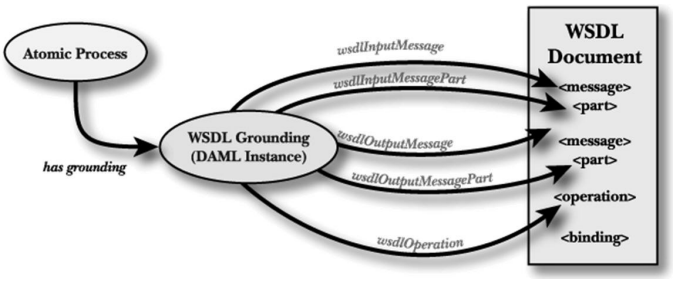

Figure 15

V. Comparison of WSMO and OWL-S

Even if both methods rely on the introduction of ontologies to enable the semantic web services, they radically differ in several aspects

OWL-S assists the developer by means of templates but at the same time, it brings some constraints that are not present in WSMO.

OWL-S doesn't provide any mechanism to address the problem of the different types of web services, whereas WSML by the definition of mediators assumes the fact that web services can be heterogeneous.

The definition of mediators in WSMO motivates that investigate more on how OWL-S solves the compatibility problem. OWL-S provides the web service and the users with information on how to find already existent mediators or on how to generate them by using web service composition.

A clear disadvantage of OWL-S is the obligation of defining both the web service functionality and the user requirements on the web service in the service profile. WSMO separates both into two different ontologies (Web Service and Goals)

In terms of composition, OWL-S provides a well-defined choreography and an automatic orchestration, whereas WSMO has an automatic, half-automatic and fixed composition. Even if choreography and orchestration are also provided, the support for them is in OWL-S better.

Grounding is comparable, as both languages provide the standard assignment of classes to WSDL data types, and both predicate for a separation of the web service description from its interface implementation. The fact that OWL-S supports expressions like XSLT transformations or inference rules in languages like KIF (Knowledge Interface Format) or SWRL (Semantic Web Rule Language), that allows for re-using the already existing expressions without having to re-formulate them.

With WSMO several process flows can be specified for the same service, enabling the execution of the same service in different ways (indispensable requirement in pervasive environments or in situations where load balancing is required)

To summarize the WSMO vs OWL-S comparison I will provide a similarity and two differences:

- Both OWL-S and WSMO rely on the usage of ontologies as core components to enable the semantic web services
- OWL-S promotes a specializing/generalizing strategy, where atomic processes can be compounded into more complex processes
- WSMO strives for enabling the integration of non-dependant, separately developed, isolated solutions by means of mediators.

VI. Conclusion and Final Thought

The introduction of the semantic web services brings to the development community the big challenge of the ontology languages. To make use of their huge potential, a lot of efforts are still required to make people confident with the new fashion of web services.

WSDL-S, OWL-S and WSMO have already laid the cornerstone, but the lack of user friendly tools for the creation of SWS is still uncovered. Only if the average-user gets on board and the right expectations are set, the SWS will be accepted.

When a new technology emerges, it is crucial to set the right expectations. Otherwise, as several times demonstrated in the history of the AI, two weaknesses might lead it to break down: technology insiders being over-promising, and outsiders being over-optimistic (as stated Henry Thompson in the presentation of the XML Meta-Architecture)

Along the Artificial Intelligence History, a lot of efforts have been put in designing expressive notations to tackle the problem of knowledge representation. But those efforts result to be useless when taking a step ahead and trying to exploit the represented knowledge. To put it in other words, it is proven that designing an approach to knowledge representation without designing first an inference engine for this knowledge can be a waste of time.

Actually, we have been facing and are still facing the trade-off between using 1st order predicate logic and thereby getting a variety of well-understood inference engines, or using something user-friendlier and more expressive, but we are not able to exploit. The same thought can be applied to upcoming semantic web services and the emerging technologies melting pot.

References

[1] Web Sevices Definition Language WSDL available at http://www.w3.org/TR/wsdl

[2] Web Services Modeling Ontology available at http://www.wsmo.org

[3] Web Service Execution Environment (WSMX) Available at http://www.w3.org/Submission/WSMX/

[4] Web Services Modeling Language specified at http://www.wsmo.org/TR/d16/d16.1/v0.21/#part:variants

[5] Bussler, Christoph and Fensel, Dieter. 1st F2F meeting SDK cluster working group on Semantic Web Services. Wiesbaden, Germany, March 2004

[6] Domingue, John. Ontolog Semantic Web Service Ontology Standard Panel. The Open University, October 2005

[7] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S.,Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: Owl-s: Semantic markup for web services. Technical report, http://www.w3.org/Submission/OWL-S/ (2004)

[8] Knowledge Interchange Format, available at http://logic.stanford.edu/kif/kif.html
[9] Semantic Web Rule Language combining OWL and RuleML, available at http://www.w3.org/Submission/SWRL/
[10] Daskalova, H., Atanasova, T.: Semantic web services - where we are and where weare going. Technical report, Institute of Information Technologies - BAS, http://www.infrawebs-eu.org/get_file.php?id=277 (2005)
[11] de Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Keller, U., Kifer, M., König-Ries, B., Kopecky, J., Lara, R., Lausen, H., Oren, E., Polleres, A., Roman,D., Scicluna, J., Stollberg, M.: Web service modeling ontology (wsmo). Technical report, http://www.w3.org/Submission/WSMO/ (2005)
[12] Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.T., Sheth, A.,Verma, K.: Web service semantics - wsdl-s. Technical report (2005)