WSMO-Lite: Lowering the Semantic Web Services Barrier with Modular and Light-weight Annotations *

Jacek Kopecký and Tomas Vitvar
Semantic Technology Institute (STI),
University of Innsbruck, Austria,
{firstname.lastname}@sti2.at

Abstract

Services are an increasingly important part of the Web, and they are a necessary component of the Semantic Web. Semantic Web Services (SWS) are a research effort towards automation of the use of Web services, enhancing existing SOA capabilities with intelligent and automated integration. Recently, we have introduced WSMO-Lite, a lightweight service ontology intended for semantic annotations of the Web Service Description Language WSDL. In contrast to preceding SWS frameworks such as OWL-S and WSMO, WSMO-Lite simplifies the semantic descriptions and enables bottom-up semantic annotation of Web services, but very importantly, it also relaxes the requirements on completeness of semantic descriptions, which enables building incremental layers of semantics on top of existing service descriptions. In this work, we describe various useful subsets of the extent of semantic annotation on Web services with respect to the requirements of SWS automation tasks; and we detail the means of validating SWS descriptions with flexible levels of strictness.

1 Introduction

The Semantic Web is not only an extension of the current Web with semantic descriptions of data; it also needs to integrate services that can be used automatically by the computer on behalf of its user [2]. A major technology for publishing services on the Web is the so-called *Web services*. Based on WWW standards HTTP and XML, Web services are gaining significant adoption in areas of application integration, wide-scale distributed computing, and business-to-business cooperation. Still, many tasks commonly performed in service-oriented systems remain manual (performed by a human operator).

In order to make Web services part of the Semantic Web, the research area of Semantic Web Services (SWS) aims to increase the level of automation around Web services. Typical tasks automated by SWS technologies are discovering available services and composing them to provide more complex functionalities. SWS automation is supported by machine-processable semantic descriptions that capture the important aspects of the meaning of service operations and messages.

The main current technologies for semantic descriptions of Web services, such as WSMO [8] and OWL-S [7], model services in a top-down fashion. They define complete frameworks for describing semantics for services while they assume that a service engineer first models the semantics (usually as ontologies, functional, non-functional, and behavioral descriptions) before grounding them in service invocation and communication technologies (e.g. WSDL and SOAP). This approach, however, does not fit well with industrial developments of SOA technology, such as WSDL and REST, where thousands of services are already available within and outside enterprises (i.e., on the Web). In other words, it is hard to use the semantic frameworks in a bottom-up fashion, that is, for building increments on top of existing services while at the same time enhancing SOA capabilities with intelligent and automated integration.

In 2007, the W3C finished its work on Semantic Annotations for WSDL and XML Schema (SAWSDL, [5]). The Recommendation defines simple extensions for the Web Services Description Language WSDL and for XML Schema; these extensions are used to link WSDL components with arbitrary semantic descriptions. It thus provides the grounds for a bottom-up approach to service modeling, pioneered by WSDL-S [1]: it supports the idea of adding small increments (and complexity) on top of WSDL, making it possible to cherry-pick results from various existing approaches. This way, SWS deployments can attain the 80/20 point: get 80% of the feasible automation with only 20% of the effort.

In [9], we defined WSMO-Lite, a lightweight framework

*This work is supported by the EU FP7 project SOA4All.

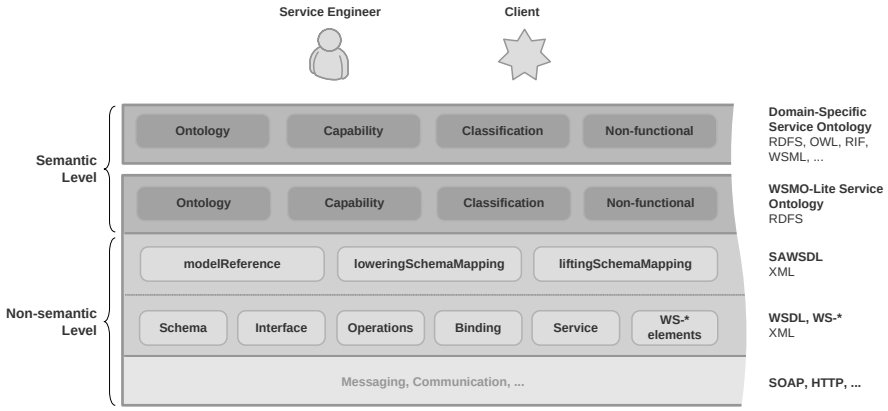

Figure 1. Semantic service description stack

for SWS annotations, introducing a service ontology primarily intended for use in SAWSDL. The most important difference between WSMO and WSMO-Lite is in their relation to WSDL: while WSMO hides the WSDL description behind the grounding mechanisms in service choreography, WSMO-Lite sees the WSDL description as its basis, directly annotating the various WSDL components with the appropriate semantics.

In this present paper, we explore a range of useful subsets of the extent of semantic Web service annotation with respect to the requirements of SWS automation tasks, and we detail the means of validating a SWS description against these subsets, with multiple levels of validation strictness.

The rest of this paper is structured as follows. Section 2 talks about the technologies underlying our work: Web service description technologies and WSMO-Lite. In Section 3, we discuss the various useful subsets of semantic Web service description. In Section 4, we show how SWS descriptions can be validated against the above subsets, and Section 5 concludes the paper.

2 Semantic Web service description

As depicted in Figure 1 and detailed in [9], there are two levels in the stack of semantic Web service description languages, namely a *semantic* and a *non-semantic* level. In addition, there are two types of stakeholders in the stack, namely a *service engineer* (human being) and a *client* (software agent). The service engineer uses Web services through the client, with particular tasks such as service discovery, selection, mediation, composition and invocation. Through these tasks the client or service engineer (depending on the level of automation) decide whether to bind with the service or not. In order to facilitate such decisions, services should describe their capabilities and interfaces in a machine-processable form.

Web services can be described in terms of the following general types of service semantics:

- *Information Model* defines the data model for input, output and fault messages.
- *Functional Descriptions* define service functionality, that is, what a service can offer to its clients when it is invoked.
- *Non-Functional Descriptions* define any incidental details specific to the implementation or running environment of a service.
- *Behavioral Descriptions* define external (public choreography) and internal (private workflow) behavior.

At the non-semantic level, the current SOA technology gives us de-facto and de-jure standards such as WSDL, SAWSDL, and related WS-* specifications. They all use XML as a common flexible data exchange format. Services are described as follows:

- *Information Model* is represented using XML Schema.
- *Functional Description* is represented using a WSDL Interface and its operations.
- *Non-Functional Description* is represented using various WS-* specifications, such as WS-Policy, WS-Reliability, WS-Security, etc., plus any technical details represented using WSDL Binding for message serializations and underlying communication protocols, such as SOAP, HTTP; and physical endpoint information specified using WSDL Service.
- *Behavioral Description* can be represented using the WS-* specifications of WS-BPEL[1] (for the workflow)

[1] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

and WS-CDL[2] (for the choreography).

SAWSDL is the bridge between the non-semantic layer and the semantic one; in other words, it is an essential part of the non-semantic level of the stack, providing the ground for the semantic layer. SAWSDL allows WSDL components to be annotated with semantics, using three extension attributes:

- `modelReference` for pointing to concepts that describe a WSDL component (so-called *reference annotations*),
- `loweringSchemaMapping` and `liftingSchemaMapping` for specifying the mappings between the XML data and the semantic information model (so-called *transformation annotations*).

At the semantic level, as shown in Figure 1, the WSMO-Lite service ontology describes Web services as follows:

- *Information Model* is represented using a domain ontology.
- *Functional Descriptions* are represented as *capabilities* and/or functionality *classifications*. A capability defines *conditions* which must hold in a state before a client can invoke the service, and *effects* which hold in a state after the service invocation. Classifications define the service functionality using some classification ontology (i.e., a hierarchy of categories).
- *Non-Functional Descriptions* are represented using an ontology, semantically representing some policy or other non-functional properties.
- *Behavioral Descriptions* are represented indirectly through functional annotations of service operations. See [9] for more details on behavioral descriptions in WSMO-Lite.

Listing 1 shows the WSMO-Lite service ontology in RDFS, serialized in Notation 3[3]. Below, we explain the semantics of the WSMO-Lite elements:

- `wl:Ontology` (lines 6–7) defines a container for a collection of assertions about the information model of a service. Same as `owl:Ontology`, `wl:Ontology` allows for meta-data such as comments, version control and inclusion of other ontologies. `wl:Ontology` is a subclass of `owl:Ontology` since as we already mentioned, it has a special meaning of the ontology used as the service information model.

[2]Choreography Description Language, `http://www.w3.org/TR/ws-cdl-10/`

[3]`http://www.w3.org/DesignIssues/Notation3.html`

```
1  @prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
3  @prefix owl: <http://www.w3.org/2002/07/owl#> .
4  @prefix wl: <http://www.wsmo.org/ns/wsmo−lite#> .
5
6  wl:Ontology rdf:type rdfs:Class;
7    rdfs:subClassOf owl:Ontology.
8  wl:ClassificationRoot rdfs:subClassOf rdfs:Class.
9  wl:NonFunctionalParameter rdf:type rdfs:Class.
10 wl:Condition rdfs:subClassOf wl:Axiom.
11 wl:Effect rdfs:subClassOf wl:Axiom.
12 wl:Axiom rdf:type rdfs:Class.
```

Listing 1. WSMO-Lite Service Ontology

- `wl:ClassificationRoot` (line 8) marks a class that is a root of a classification which also includes all the RDFS subclasses of the root class. A classification (taxonomy) of service functionalities can be used for functional description of a service.
- `wl:NonFunctionalParameter` (line 9) specifies a placeholder for a concrete domain-specific non-functional property.
- `wl:Condition` and `wl:Effect` (lines 10–12) together form a *capability* in functional service description. They are both subclasses of a general `wl:Axiom` class through which a concrete language can be used to describe the logical expressions for conditions and effects.

Finally, WSMO-Lite defines five types of semantic annotations:

A1: *Ontology annotations of XML Schema:* The schema used in WSDL to describe messages, i.e., the element declarations and type definitions, can carry reference annotations linking to the appropriate classes from the service information model ontology.

A2: *Transformation annotations of XML Schema:* To be able to communicate with a service, the client needs to transform data between its semantic model and the service-specific XML message structures. The schema may contain transformation annotations which specify the appropriate mappings.

A3: *Functional annotations of WSDL Interface and Service:* Functional descriptions (both capabilities and categories) apply both to concrete web services and to the reusable and abstract interfaces. A reference annotation points from a service or an interface to its appropriate functional description.

A4: *Functional annotations of WSDL Interface operations:* Functional descriptions (both capabilities and categories) apply also to interface operations, to indicate their

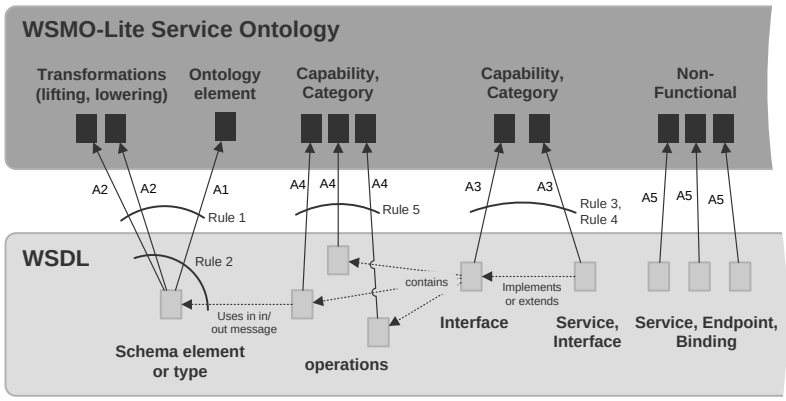

Figure 2. Illustration of Annotations and Rules

particular functionalities. A reference annotation points from an operation to its appropriate functional description.

A5: *Non-functional annotations of WSDL Service, Endpoints, and Binding:* Non-functional descriptions apply to a concrete instance of a Web service, that is, a Service, its Endpoints, or its Binding. A reference annotation can point from any of these components to a non-functional property.

Figure 2 shows these annotations along with the associated validation rules, explained in the following sections.

3 Useful subsets of semantic descriptions

Top-down semantic Web service frameworks such as OWL-S and WSMO are built with the vision of complete automation. They provide fully-fledged models of service semantics, but they also marginalize WSDL, which would really become unimportant if all Web service use was automated. However, practice shows that complete automation is unattainable, and that WSDL, a technology well known to Web service engineers, needs to be central to any SWS framework that wants to succeed in the industry. The W3C standard SAWSDL is meant to support exactly such frameworks.

Apart from the centrality of WSDL, it is also important that SWS frameworks shouldn't require complete semantic descriptions of all the aspects of the available services; the semantic automation should be available in a piecemeal fashion.

WSMO-Lite succeeds on both accounts. It is built on top of SAWSDL, and it is modular, as we show in this section.

Table 1 provides a summary of how the various annotations of WSMO-Lite fit with the tasks common in service-oriented systems and amenable to automation. The symbol • marks the annotations required to automate a given task, and the symbol ○ marks annotations that are helpful but not absolutely required.

- *Service Discovery* aims to find services that may fulfill a given client goal. Service discovery mainly operates on functional descriptions (capabilities or categories), provided by annotations A3. A concrete discovery mechanism may check that the goal complies with the service's *conditions*, and that the service *effects* fit the goal. With *classification*, the goal specifies a need for services in a given category, so the discovery mechanism simply checks category (and subcategory) membership. Further, in certain settings (e.g. for data services), the suitability of a service to a given goal may be determined by its inputs and outputs, therefore annotations A1 are optional for this task.

- *Operation Discovery* is similar to service discovery, but it works on the granularity of individual service operations. Therefore, it requires annotations A4, and again, optionally it can use annotations A1. Operation discovery might be useful with interfaces that are collections of standalone, independent operations.

- *Offer Discovery* interacts with any discovered services to find out any concrete offers that are appropriate to the user's goal; this is an important step e.g. in e-commerce (cf. [4]). Offer discovery needs to interact

Service Task	A1	A2	A3	A4	A5
Service Discovery	○		•		
Operation Discovery	○			•	
Offer Discovery	•	•		•	
Composition	•		•	•	
Ranking and Selection	○		○	○	•
Operation Invocation		•			
Service Invocation	•	•		•	
Data Mediation	•	•			
Process Mediation	•	•		•	

Table 1. Service tasks and annotations

with any information-providing operations of the service (as determined by annotations A4), and for communication it requires annotations A1 and A2, as explained below under Operation and Service Invocation.

- *Composition* puts together multiple services in order to accomplish a task that no single available service can fulfill by itself. It uses capability descriptions, i.e., annotations A3 and A4 restricted to capabilities, along with input and output descriptions, provided by annotations A1, to put together a suitable execution plan.

- *Ranking and Selection* mainly processes non-functional descriptions, i.e., annotations A5, to select the service that most suits some particular requirements; however, other annotations (A1, A3, A4) can also be taken into account in comparing the relative suitability of different services.

- *Operation Invocation* is the invocation of a single operation, and it requires data transformations between the semantic model on the client and the service's XML message structure, provided by annotations A2.

- *Service Invocation* invokes the appropriate operations of the service, in a proper order. This task therefore uses the implicit interface choreography (cf. [9]) which is produced from annotations A4 and A1. On top of that, annotations A2 are required because service invocation involves operation invocation, described above.

- *Data Mediation* automates the mapping between heterogeneous data. This process uses data annotations (A1 and A2): assuming two different schemas correspond to a single shared ontology, the A1 annotations make it possible to discover such a correspondence, and the A2 annotations then enable the transformations of instance data: lifting from one schema and lowering to the other.

- *Process Mediation* is applied during conversation between two services mediating their choreographies and messages (cf. [3]). It combines data mediation and choreography processing and thus requires annotations A1, A2 and A4.

The different possible combinations of the annotations required for the various tasks demonstrate the modularity of WSMO-Lite. In the following section, we discuss the validation of WSMO-Lite descriptions.

4 Validating WSMO-Lite descriptions

As with any kind of formal, machine-processable descriptions, a validator is a very useful tool when developing semantic descriptions of Web services. Validation is a way of catching certain errors early in the development process. We distinguish four facets of validity (as a general term) of WSMO-Lite descriptions:

1. A *correct* description is such that truthfully models its underlying service. This is generally verified by "adding eyeballs". Apart from detecting inconsistencies (below), we do not attempt to specify an automatic correctness validator. A correct description enables automation, but even a correct description does not guarantee success in using the service, as we cannot eliminate run-time failures such as a product being out of stock, and even errors such as a network outage.

2. A *complete* description describes all the relevant aspects of the semantics of the service. Incompleteness is not necessarily a problem: for example, an order cancellation operation would not be needed for the goal of making an order, therefore even if the operation lacks data transformation annotations (A2), the invocation of the service, which technically requires these annotations, would still succeed (barring incorrectness or run-time errors). Therefore, an automatic validator encountering incompleteness may **report a warning** that the user should consider. Some cases of incompleteness may be acceptable to the user.

3. A *consistent* description does not contradict itself. An inconsistent description is clearly incorrect. If a validator encounters inconsistency, it must **report an error** that the user should fix. Even though the inconsistency might also not actually come into play in particular scenarios (e.g. if only the operation for order cancellation was annotated inconsistently), it should never be considered acceptable.

4. A *syntactically valid* description is captured using valid sentences of the underlying languages. This can be checked automatically by many existing tools; WSMO-Lite does not actually introduce any new syntax on top of existing standards. Nevertheless, a WSMO-Lite validator may include syntax validators, and then it must **report an error** on any syntactically invalid input documents.

Apart from the WSMO-Lite description, a validator tool also needs to consider the set of tasks for which the description is intended. This influences especially the completeness requirements; a description complete for service discovery need not be complete for service invocation. In fact, the consistency requirement mirrors the requirements for annotations — a task that requires some annotation (as shown in Table 1) also requires this annotation to be present on all the WSDL components on which it applies (with an

exception noted below); otherwise, the WSDL components might be inaccessible to the SWS automation processes.

The following is a list of consistency validation rules that can be checked on WSMO-Lite descriptions:

Consistency Rule 1 Every functionality (A3 annotation, a capability or a category) of a service must be a restriction of some functionality of the service's interface. (See [9] for the formal definitions of capability and category restriction.)

A concrete functional description attached to a service refines the functional description of the service's interface. For instance, the WSDL interface may be a general e-commerce interface, and a service may restrict it to only certain categories of products, e.g. music CDs. This allows discovery to first find suitable interfaces and then only check services that implement these interfaces. This is useful as an optimization for Service Discovery and Composition.

Consistency Rule 2 Similarly, every functionality of an interface must be a restriction of some functionality of any interface that extends it.[4]

This rule ensures that functionality cannot be lost through WSDL interface extension; discovery need not check the suitability of an interface that extends another interface already discovered as suitable. Similarly to the rule above, this one is useful as an optimization for Service Discovery and Composition.

The following are proposed consistency rules, but we do not have feasible algorithms for verifying them yet.

Consistency Rule 3 (not implemented) If an XML schema component has both a data annotation (A1) and a lifting annotation (A2), the lifting transformation must accept data valid according to the schema component, and produce instances valid according to the ontology element specified by the data annotation. Similarly, if an XML schema component has both a data annotation (A1) and a lowering annotation (A2), the lowering transformation must accept instances data valid according to the ontology element specified by the data annotation, and produce an XML element valid according to the schema component.

For illustration; a message may be described as being a Purchase Order. If it has a lifting annotation, the lifting transformation must accept an XML document that is valid according to the message schema, and the result of the transformation must contain an instance of Purchase Order; and if the message has a lowering annotation, the lowering transformation must take an instance of Purchase Order as its input and return an XML document that is valid according to the message schema.

There cannot be a general algorithm which checks this rule for powerful Turing-complete transformation languages such as XSLT, however there might be useful validators for some common cases, or for less powerful transformation languages.

Consistency Rule 4 (not implemented) The interface choreography produced from operation data annotations (A1) and functional annotations (A4) allows at least one successful execution coming from a state that fulfills the interface capability (A3) condition and ending in a state that fulfills the capability effect.

Consistency Rule 5 (not implemented) For every state that fulfills the interface capability (A3) condition, the interface choreography produced from operation data annotations (A1) and functional annotations (A4) allows at least one successful execution ending in a state that fulfills the capability effect.

These two rules both check the consistency of the interface functional annotations with the operation functional annotations (checking that the operations can deliver what the interface promises), with Rule 4 being a weaker variant of Rule 5. For instance, if the service functional description says the service can sell and deliver products, but there is no actual operation that allows for the client to specify delivery options, Rule 5 would be violated because delivery cannot be accomplished, but the service would comply with Rule 4 because if delivery is not necessary (for instance when purchasing music in a digital format, such as MP3), the service could be used successfully.

It is not clear to us yet whether either of these two rules can automatically be verified.

To summarize, a WSMO-Lite validator takes a semantic Web service description and a list of SWS tasks that the description should support, and returns a listing of errors and warnings, if any, according to these steps:

1. First, syntactical validity of the input description should be checked; in case of any errors, they are reported and the process stops.

2. If the list of intended SWS tasks contains Service Discovery or Composition, the input description is checked against the two consistency rules; if the consistency check fails, it is reported as an error.

3. The input description is checked for completeness of annotations required by the intended SWS tasks, any violations are reported as warnings.

[4] WSDL 2.0 allows interface extension.

The completeness requirements are in direct opposition to the requirements of "light-weight" and "incremental" annotation. This is why we suggest a validator to issue only warnings when validating incomplete descriptions; quite often "a little semantics can go a long way."

5 Conclusions and Future Work

In this paper, we have described the latest results from the development of WSMO-Lite, a lightweight ontology for Semantic Web Services, building on the newest W3C standards. WSMO-Lite fills in SAWSDL annotations with concrete semantic constructs, yet still stays open with respect to concrete ontology languages and the corresponding expressivity and complexity trade-offs. WSMO-Lite supports piecemeal and incremental annotation of existing Web services, which makes it much easier to adopt than larger frameworks, such as WSMO and OWL-S.

Prof. Amit Sheth, one of the main initiators of the SAWSDL work in W3C, points out in [6]: "Rather than look for a clear winner among various SWS approaches, I believe that in the post-SAWSDL context, significant contributions by each of the major approaches will likely influence how we incrementally enhance SAWSDL. Incrementally adding features (and hence complexity) when it makes sense, by borrowing from approaches offered by various researchers, will raise the chance that SAWSDL can present itself as the primary option for using semantics for real-world and industry-strength challenges involving Web services." WSMO-Lite adheres to the principles underlying Sheth's statement.

In our future work, we plan to extend WSMO-Lite towards semantic annotations for RESTful Web services, in order to embrace the booming ecosystem of Web 2.0 services and mash-ups; and we also expect to integrate WSMO-Lite alongside the larger WSMO framework in the system WSMX (Web Services Execution Environment, [3]).

References

[1] R. Akkiraju, *et al.* Web Service Semantics - WSDL-S, available at http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/. Tech. rep., LSDIS Lab, 2005.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[3] T. Haselwanter, *et al.* WSMX: A Semantic Service Oriented Middleware for B2B Integration. In *ICSOC*, pp. 477–483. 2006.

[4] J. Kopecký, E. Simperl, and D. Fensel. Semantic Web Service Offer Discovery. In *Proceedings of Service Matchmaking and Resource Retrieval in the Semantic Web Workshop, colocated with* 6^{th} *ISWC*. 2007.

[5] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, 11(6):60–67, 2007.

[6] D. Martin and J. Domingue. Semantic web services: Past, present and possible futures (systems trends and controversies). *IEEE Intelligent Systems*, 22(6), 2007.

[7] D. Martin *et al.* OWL-S: Semantic Markup for Web Services. Member submission, W3C, 2004. Available from: `http://www.w3.org/Submission/OWL-S/`.

[8] D. Roman, *et al.* Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.

[9] T. Vitvar, J. Kopecky, J. Viskova, and D. Fensel. WSMO-Lite Annotations for Web Services. In *Proceedings of the 5th European Semantic Web Conference (ESWC)*. 2008.