# Semantic Web Service Automation with Lightweight Annotations*

Jacek Kopecký, Tomas Vitvar, and Dieter Fensel

Semantic Technology Institute (STI Innsbruck)
Innsbruck, Austria
⟨firstname.lastname⟩@sti2.at

**Abstract.** Web services, both RESTful and WSDL-based, are an increasingly important part of the Web. With the application of semantic technologies, we can achieve automation of the use of those services. In this paper, we present WSMO-Lite and MicroWSMO, two related lightweight approaches to semantic Web service description, evolved from the WSMO framework. WSMO-Lite uses SAWSDL to annotate WSDL-based services, whereas MicroWSMO uses the hRESTS microformat to annotate RESTful APIs and services. Both frameworks share an ontology for service semantics together with most of automation algorithms.

## 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 [3]. There are currently two major, largely complementary, ways to offer automated services on the Web: 1) the so-called "Web Services" specification stack, the keystone of Service Oriented Architectures (SOAs), based on SOAP and WSDL, and 2) machine-oriented Web applications and APIs, also called RESTful Web services, that conform to the REST architectural style [4] that underlies the architecture of the World Wide Web.

Semantics-based automation, the main goal of the Semantic Web, is supported by machine-readable formal semantic descriptions of both data and services. It has been researched under the name Semantic Web Services (SWS), and the existing efforts, such as WSMO [13] and OWL-S [15], focus mostly on WSDL and SOAP services. These efforts define complete frameworks for describing the semantics of services, while assuming that a service engineer first models the semantics (usually as ontologies, functional, nonfunctional, 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 SWS frameworks in a bottom-up fashion, that is, for building increments on top of

* This work is funded by the EU research project SOA4All, http://soa4all.eu/

existing service descriptions, gradually enhancing SOA capabilities with intelligent and automated integration.

Further, the current SWS frameworks do not support RESTful Web services, which have only recently started to be widely recognized as an approach viable for many distributed computing systems. RESTful Web services, also known as Web APIs, are the substance of so-called “mashups” (lightweight compositions of Web applications), an increasingly popular phenomenon.

In this paper, we present a unified lightweight approach for semantic description of both RESTful and WSDL-based services, which consists of a simple ontology that models Web services and their semantics, and two annotation mechanisms, one for WSDL and one for RESTful service descriptions, that apply this ontology to the respective Web service technologies.

WSMO-Lite is the annotation mechanism for WSDL-based services. It uses SAWSDL [11] to attach semantic concepts to WSDL and XML Schema components that describe various aspects of the service, and then it maps the structure of WSDL into the common service model from our ontology.

MicroWSMO, on the other hand, is an annotation mechanism for HTML descriptions of RESTful Web services. It extends the microformat hRESTS [8] that provides a basic machine-readable structure of service descriptions in unstructured HTML documentation. There are several alternatives to hRESTS for machine-readable description of Web APIs, e.g. the Web Application Description Language (WADL, [6]) and even WSDL 2.0 [17]. These are clearly-structured, detailed and extensible formats, but probably due to their perceived complexity, they do not seem to be gaining traction with API providers; service descriptions remain mostly in unstructured text; that is why we need a microformat like hRESTS.

The use of lightweight annotation mechanisms such as SAWSDL and hRESTS makes it possible to add semantics to service and API descriptions incrementally, without the need to adopt a complete framework such as OWL-S or WSMO. In this paper, we also show how the extent of semantic annotations is determined by the requirements on automation, i.e., by the tasks that must be automated. Currently, authoring semantic descriptons is a manual process; editing tools or (semi-)automated knowledge acquisition techniques may provide support for this crucial first step.

The content of this paper is structured as follows: in Section 2, we present the tasks whose automation is the subject of SWS research, and we list the kinds of service semantics that need to be captured in order to support such automation. In Section 3 we define the ontology for the service model and the different kinds of service semantics; followed by sections 4 and 5 where we show the mechanisms for semantic annotation of concrete service descriptions. In Section 6, we sketch algorithms that implement the various automation tasks. Finally, in Section 7, we discuss some related work, and Section 8 concludes the paper.

## 2 Semantic Web Services Automation

SWS automation is implemented in a so-called semantic execution environment (SEE, for instance WSMX [7]). A user can submit a concrete *goal* to the SEE, which then

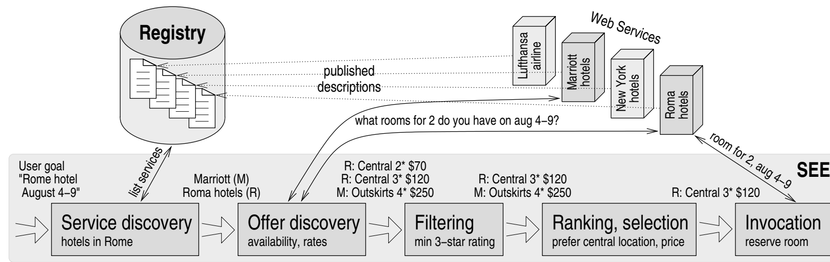


**Fig. 1.** Semantic Execution Environment (SEE) automation tasks

accomplishes it by finding and using the appropriate available Web services. SWS research focuses mainly on how the SEE "finds the appropriate Web service(s)", as illustrated in Figure 1 with the first four SEE tasks.

In the figure, the user wants to arrange a June vacation in Rome. There are four services with published descriptions: the airline Lufthansa, and hotel reservation services for New York, Rome, and one for the Marriott chain worldwide. The SEE first *discovers services* that may have hotels in Rome, discarding Lufthansa which does not provide hotels, and the New York service which does not cover Rome. Then the SEE *discovers offers* by interaction with the discovered services. The available offers are a 4* Marriott at the outskirts of Rome (judged by the client from the address of the hotel), and one 2* and one 3* hotel in the city center. Then the SEE *filters* the offers depending on the user's constraints and requirements (minimum 3-star rating), *ranks* them according to the user's preferences (central location, then price) and *selects* one offer, in the end *invoking* the respective service. Note that an actual implementation of the client may execute the tasks in different orders or even interleave them; e.g., discovery can be combined with filtering. This can be seen as optimization, not affecting the end result.

While our example comes from e-commerce, the process of discovery, filtering, ranking and invocation applies in general for any SWS system. In cases when the semantic description of a service is not sufficient for the SEE to determine whether a service will satisfy the user's goal, discovery is split into *service discovery*, which finds services that can *potentially* fulfill the goal, and *offer discovery*, which interacts with the discovered services and finds out about concrete offers.

SWS automation is supported by machine-processable semantic descriptions that capture the important aspects of the meaning of service operations and messages. Web services can be described in terms of the following general types of service semantics:

- *Information model* defines the semantics of input, output and fault messages.
- *Functional semantics* defines service functionality, that is, what a service can offer to its clients when it is invoked.
- *Nonfunctional semantics* defines any incidental details specific to the implementation or running environment of a service, such as its price or quality of service.
- *Behavioral semantics* specifies the protocol (ordering of operations) that a client needs to follow when consuming a service's functionality.

The different automation tasks have varying requirements on the extent of semantic descriptions necessary for automation. Table 1 shows what descriptions (Functional,

| Service Task | F | N | B | I |
|---|---|---|---|---|
| Service discovery | • | | | |
| Offer discovery | | | • | • |
| Filtering, ranking and selection | ○ | • | ○ | ○ |
| Operation invocation | | | | • |
| Service invocation | | | • | |

**Table 1.** Service usage tasks and the necessary types of semantics

Nonfunctional, Behavioral and Information model) are required (•) or useful but optional (○) for the various tasks. In the following list, we go through the service usage tasks and explain what semantic annotations are necessary.

- *Service discovery* finds services that can functionally satisfy a given goal. Therefore, it requires functional semantics. Note that this is an intentionally narrow view of service discovery, which helps us distinguish it from the other tasks.
- *Offer discovery* communicates with a discovered service and retrieves information about any available offers. Offer discovery deals with the data, therefore it requires information semantics; and it needs to invoke the service's operations in the appropriate sequence, therefore it needs behavioral semantics. Offer discovery incorporates the operation invocation task described below.
- *Filtering, ranking* and *selection,* based on user constraints and preferences, can use any available information. Most of the common ranking parameters fall into the category of nonfunctional semantics, which is therefore marked as required.
- *Operation invocation* exchanges messages with the service, therefore it needs information semantics to handle the message data.
- *Service invocation* attempts to execute the selected service to achieve the given goal; therefore it needs behavioral semantics in order to sequence the necessary operation invocations appropriately. Naturally, service invocation also incorporates operation invocation.
- *Mediation* is involved on any kind of semantics whenever the client encounters any mismatches and heterogeneities.

We can see that in order to be able to automate all the mentioned tasks, which is the intended application functionality of our service ontology, all four kinds of service semantics are required, but if only a subset of the tasks should be automated (e.g. service discovery and ranking, without offer discovery or automatic invocation), some of the semantics can be left unspecified.

## 3 Service Semantics Ontology

Figure 2 presents our simple service semantics ontology, serialized in Notation 3.[1] The ontology consists of 3 main blocks: a service model that unifies our view of WSDL-based and RESTful Web services, SAWSDL annotation properties for attaching semantics, and finally classes for representing those semantics.

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

```
1  # namespace declarations (this line is a comment)
2  @prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
3  @prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
4  @prefix owl: <http://www.w3.org/2002/07/owl#> .
5  @prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .
6  @prefix wsl: <http://www.wsmo.org/ns/wsmo−lite#> .
7
8  # service model classes and properties
9  wsl:Service rdf:type rdfs:Class .
10 wsl:hasOperation rdf:type rdf:Property ; rdfs:domain wsl:Service ; rdfs:range wsl:Operation .
11 wsl:Operation rdf:type rdfs:Class .
12 wsl:hasInputMessage rdf:type rdf:Property ; rdfs:domain wsl:Operation ; rdfs:range wsl:Message .
13 wsl:hasOutputMessage rdf:type rdf:Property ; rdfs:domain wsl:Operation ; rdfs:range wsl:Message .
14 wsl:hasInputFault rdf:type rdf:Property ; rdfs:domain wsl:Operation ; rdfs:range wsl:Message .
15 wsl:hasOutputFault rdf:type rdf:Property ; rdfs:domain wsl:Operation ; rdfs:range wsl:Message .
16 wsl:Message rdf:type rdfs:Class .
17
18 # SAWSDL properties (repeated here for completeness)
19 sawsdl:modelReference rdf:type rdf:Property .
20 sawsdl:liftingSchemaMapping rdf:type rdf:Property .
21 sawsdl:loweringSchemaMapping rdf:type rdf:Property .
22
23 # classes for expressing service semantics
24 wsl:Ontology rdf:type rdfs:Class ; rdfs:subClassOf owl:Ontology .
25 wsl:FunctionalClassificationRoot rdfs:subClassOf rdfs:Class .
26 wsl:NonfunctionalParameter rdf:type rdfs:Class .
27 wsl:Precondition rdf:type rdfs:Class .
28 wsl:Effect rdf:type rdfs:Class .
```

**Fig. 2.** Service Ontology, captured in Notation 3

The first block, the service model, is also illustrated in Figure 3. It is very similar to the model of WSDL, but it also applies to RESTful Web services (as we show in [8]). Instances of this ontology are not expected to be authored directly; instead, the underlying technical descriptions (WSDL, hRESTS) are parsed in terms of this ontology for processing in a SEE.

The service model is rooted in the class *Service*. In contrast to WSDL, our service model does not separate the service from its interface, as we do not need such separation to support SWS automation. A service is a collection of operations (class *Operation*). Operations may have input and output messages, plus possible fault messages (all in the class *Message*). Input messages (and faults[2]) are those that are sent by a client to a service, and output messages (and faults) are those sent from the service to the client.

Figure 3 also shows the semantic annotations: a service can be annotated with its functional and nonfunctional semantics, the behavioral semantics is represented on the operations, and the messages are connected to the information model. To link the concepts of the service model with the concrete description of the functional, nonfunctional, behavioral and information semantics, we adopt the standard SAWSDL properties *modelReference*, *liftingSchemaMapping* and *loweringSchemaMapping*, shortly described below.

A model reference can be used on any component in the service model to point to the semantics of that component. In particular, a model reference on a service can point to a description of the service's functional and nonfunctional semantics; a model reference on an operation points to the operation's part of the behavioral semantics de-

[2] Input faults are possible in some WSDL 2.0 message exchange patterns.

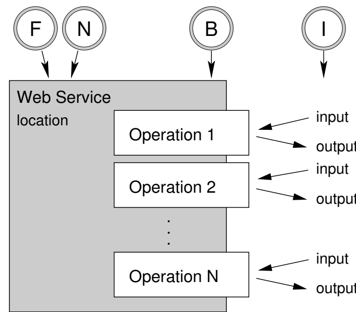


**Fig. 3.** Web service description model with attached semantics

scription; and a model reference on a message points to the message's counterpart(s) in the service's information model. A single component can have multiple model reference values, which all apply together; for example, a service can have a number of nonfunctional properties together with a pointer to its functionality description.

The lifting and lowering schema mapping properties are used to associate messages with the appropriate transformations between the underlying technical format such as XML and a semantic knowledge representation format such as RDF. The value of either property is a URIs of a document that describes the lifting or lowering transformation.

Finally, the four types of service semantics are represented in our service ontology as described in the following list. All the terms are formalized in [16].

- Information semantics are represented using domain ontologies.
- Functional semantics are represented as *capabilities* and/or functionality *classifications*. A capability defines logical expressions for *preconditions* which must hold in a state before the client can invoke the service, and *effects* which hold in a state after the service invocation. Functionality classifications define the service functionality using some classification ontology (i.e., a hierarchy of *categories*, such as the ecl@ss taxonomy[3]), and the class *FunctionalClassificationRoot* marks a class that is a root of a classification, which also includes all the RDFS subclasses of the root class.
- Nonfunctional semantics are represented using some ontology, semantically capturing a policy or other nonfunctional properties. The class *NonfunctionalParameter* marks a concrete piece of nonfunctional semantics.
- Behavioral semantics are represented by annotating the service operations with functional descriptions, i.e., with capabilities and/or functionality classifications. In Section 6, we briefly describe how the functional annotations of operations serve for ordering of operation invocations.

## 4 WSMO-Lite: Annotating WSDL Services

Initially, the service ontology presented in the preceding section was developed as part of WSMO-Lite, and applied directly to the WSDL model. However, to incorporate

[3] eCl@ss Standardized Material and Service Classification, `eclass-online.com`

```
1  <wsdl:description>
2    <wsdl:types><xs:schema>
3      ...
4      <xs:element name="NetworkConnection" type="NetworkConnectionType"
5         sawsdl:modelReference="http://example.org/onto#NetworkConnection"
6         sawsdl:loweringSchemaMapping="http://example.org/NetCn.xslt"/>
7      ...
8    </xs:schema></wsdl:types>
9    ...
10   <wsdl:interface name="NetworkSubscription"
11          sawsdl:modelReference="http://example.org/onto#VideoSubscriptionService" >
12     <wsdl:operation name="SubscribeVideoOnDemand"
13          sawsdl:modelReference="http://example.org/onto#VideoOnDemandSubscriptionPrecondition
14                                 http://example.org/onto#VideoOnDemandSubscriptionEffect">
15     ...
16   </wsdl:interface>
17   ...
18   <wsdl:service name="ExampleCommLtd"
19          interface="NetworkSubscription"
20          sawsdl:modelReference="http://example.org/onto#VideoOnDemandPrice">
21     <wsdl:endpoint name="public"
22          binding="SOAPBinding"
23          address="http://example.org/comm.ltd/subscription" />
24   </wsdl:service>
25 </wsdl:description>
```

**Fig. 4.** Various WSDL components with WSMO-Lite annotations

RESTful services, we have added the simplified service model and now WSMO-Lite comprises a set of recommendations on how to annotate WSDL with the four kinds of semantics, along with a mapping from the annotated WSDL structure to our simpler service model, which is then the input to most of the automation algorithms.

The listing in Figure 4 shows WSMO-Lite annotations on an example WSDL document. WSDL distinguishes between a concrete service (line 18) and its abstract (and reusable) interface (line 10) that defines the operations (line 12). This structure is annotated using SAWSDL annotations with examples of semantics. Due to space constraints, we refer the reader to [16] for the actual semantic definitions of these annotations; nevertheless, our example can be understood even without those detailed definitions.

The following paragraphs describe how the various types of semantics are attached in the WSDL structure:

Functional semantics can be attached as a model reference either on the WSDL service construct, concretely for the given service, or on the WSDL interface construct (line 11), in which case the functional semantics apply to any service that implements the given interface. Nonfunctional semantics, by definition specific to a given service, are attached as model references directly to the WSDL service component (line 20).

Information semantics are expressed in two ways. First, pointers to the semantic counterparts of the XML data are attached as model references on XML Schema element declarations and type definitions that are used to describe the operation messages (line 5). Second, lifting and lowering transformations need to be attached to the appropriate XML schema components: input messages (going into the service) need lowering annotations (line 6) to map the semantic client data into the XML messages, and output messages need lifting annotations so the semantic client can interpret the response data.

Finally, behavioral semantics of a service are expressed by annotating the service's operations (within the WSDL interface component, lines 13 and 14) with functional

descriptions, so the client can then choose the appropriate operation to invoke at the appropriate time during its interaction with the service.

A WSDL document with WSMO-Lite annotations can be validated for consistency and completeness, as described in [10]. When mapping such a WSDL document into our simplified service model, which does not represent a separate service interface, we combine the interface annotations with the service annotations. Similarly, annotations from the appropriate XML Schema components are then mapped to annotations of the messages in our service model. Otherwise, the mapping of WSDL to our service model is straightforward.

## 5 MicroWSMO: Annotating RESTful APIs

In the case of RESTful services and Web APIs, there is no widely accepted machine-readable service description language. WSDL 2.0 and WADL are two proposals for such a language, however, the vast majority of public RESTful services are described in plain unstructured HTML documentation. Therefore, in [8] we introduced hRESTS, a microformat for identifying the components of the service model in a machine-readable way in otherwise unstructured HTML service descriptions.

As we also show in [8], even though the interaction model of RESTful services (following links in a hypermedia graph) differs from that of SOAP services (messaging), the service model is actually the same: a service contains a number of largely independent operations with input and output messages. hRESTS captures this structure with HTML classes `service`, `operation`, `input` and `output` that identify the crucial parts of a textual service description.

In RESTful services, a service is a grouping of related Web resources, each of which provides a part of the overall service functionality. While a SOAP service has a single location address, each operation of a RESTful service must have an address of the concrete resource that provides this operation. Therefore, hRESTS also defines classes for marking the resource address and the HTTP method used on that resource (the classes `address` and `method`), which together uniquely identify a single operation.

hRESTS is an approach with wider goals than SWS automation, and it does not contain links for semantic annotations. Consequently, MicroWSMO extends hRESTS with SAWSDL-like annotations: the HTML class `mref` identifies the model reference annotations, and the link relations `lifting` and `lowering` identify the data grounding transformations. The concrete semantics are added analogously to how WSMO-Lite annotates WSDL documents: functional and nonfunctional semantics are model references on the service, behavioral semantics are captured using functional descriptions of operations, and information model links go on the input and output messages. An example hRESTS/MicroWSMO description can be seen in Figure 5, showing how the microformat is embedded in HTML.

If, in the future, a service description language such as WSDL 2.0 or WADL gains acceptance among the providers of RESTful services, we expect a straightforward application of SAWSDL to achieve the same effect as our current use of hRESTS and MicroWSMO.

```
<div class="service" id="svc">
 <p><span class="label">Example Comm Ltd.</span> is a
   <abbr class="mref" title="...#VideoSubscriptionService">
     video subscription</abbr> service.</p> ...
 <div class="operation" id="op1"><p> ...
   The operation <code class="label">SubscribeVideoOnDemand</code> is
   invoked using the method <span class="method">POST</span>
   at <code class="address">http://example.com/videoondemand/subscription</code>,
   by submitting <span class="input"> the
     <code class="mref" title="...#NetworkConnection">current network connection type</code> details
     (<a rel="lowering" href=".../networkConnection.xslt">lowering</a>).
   </span>.
   It returns ...
 </p></div>
</div>
```

**Fig. 5.** Example hRESTS description

## 6 Automation algorithms

In this section, we complete the picture of our lightweight SWS approach by sketching a number of algorithms for processing semantically annotated service descriptions, and thus automating the common tasks which are currently performed largely manually by human operators. Detailed realization of these algorithms remains as future work.

Automation is always guided by a given user goal. While we do not talk in this paper about concrete formal representation for user goals, the various algorithms need the goal to contain certain specific information. We describe this only abstractly, since we view the concrete representation of user goals as an implementation detail specific to a particular tool set.

***Service Discovery:*** for discovery (also known as "matchmaking") purposes, our approach provides functional service semantics of two forms: functionality classifications and precondition/effect capabilities, with differing discovery algorithms.

With functionality classifications, a service is annotated with particular functionality categories. We treat the service as an instance of these category classes. The user goal will identify a concrete category of services that the user needs. A discovery mechanism uses subsumption reasoning among the functionality categories to identify the services that are members of the goal category class ("direct matches"). If no such services are found, a discovery mechanism may also identify instances of progressively further superclasses of the goal category in the subclass hierarchy of the functionality classification. To illustrate: if the user is looking for a *VideoService*, it will find services marked as *VideoSubscriptionService* (presuming the intuitive subclass relationships) as direct matches, and it may find services marked as *MediaService* which are potentially also video services, even though the description does not directly advertise that.

For discovery with preconditions and effects, the user goal must specify the user's preconditions (requirements) and the requested effects. The discovery mechanism will need to check, for every available service, that the user's knowledge base fulfills the precondition of the service and that this precondition is not in conflict with the user's requirements, and finally that the effect of the service fulfills the effect requested by the user. This is achieved using satisfaction and entailment reasoning.

Discovery using functionality categorizations is likely to be coarse-grained, whereas the detailed discovery using preconditions and effects may be complicated for the users and resource-intensive. Therefore we expect to combine the two approaches, to describe the core functionality in general classifications, and only some specific details using logical expressions, resulting in better overall usability.

***Offer Discovery:*** especially in e-commerce scenarios, service discovery as described above cannot guarantee that the service will actually have the particular product that the user requests. For instance, if the user wants to buy a certain book, service discovery will return a number of online bookstores, but it cannot tell whether the book is available at these bookstores. Offer discovery is the process of negotiating with the service about the concrete offers pertinent to the user's goal.

An offer discovery algorithm uses the behavioral and information model annotations of a Web service to select and invoke the appropriate offer inquiry operations. In Web architecture [2], there is a concept of *safe interaction*, mostly applied to information retrieval. In particular, HTTP GET operations are supposed to be safe, and WSDL 2.0 contains a flag for safe SOAP operations. As detailed in [9], we implement offer discovery through the use of AI (Artificial Intelligence) planning over the inputs and outputs of the safe operations of a given service.

***Filtering, Ranking, Selection:*** these tasks mostly deal with the nonfunctional parameters of a service. The user goal (or general user settings) specifies constraints and preferences (also known as hard and soft requirements) on a number of different aspects of the discovered services and offers. For instance, service price, availability and reliability are typical parameters for services, and delivery options and warranty times can accompany the price as further nonfunctional parameters of service offers.

Filtering is implemented simply by comparing user constraints with the parameter values, resulting in a binary (yes/no) decision. Ranking, however, is a multidimensional optimization problem, and there are many approaches to dealing with it, including aggregation of all the dimensions through weighted preferences into a single metric by which the services are ordered, or finding locally-optimal services using techniques such as Skyline Queries [14].

Selection is then the task of selecting only one of the ranked services. With a total order, the first service can be selected automatically, but due to the complexity of comparing the different nonfunctional properties (for instance, is a longer warranty worth the slightly higher price?), often the ordered list of services will be presented to the user for manual selection.

***Invocation:*** service invocation involves the execution of the various operations of the selected service in the proper order so that the user goal is finally achieved.

To invoke a single operation, the client uses the information model annotations plus the technical details from the WSDL or hRESTS description to form the appropriate request message, transmit it over the network to the Web service, and to understand the response. If multiple operations must be invoked, the client can use AI planning techniques with functional semantics, and on RESTful services, the hypermedia graph can guide the client in its invocations, as the client gets links to further operations in the response to the last operation invoked.

## 7 Related Work

Our work is most directly related to SWS frameworks such as WSMO and OWL-S. In fact, we started working on WSMO-Lite as a reaction to the existence of SAWSDL, rethinking the WSMO conceptual model from the point of view of WSDL service descriptions. In comparison with WSMO in particular, our service semantics ontology, shared by both WSMO-Lite and MicroWSMO, drops the support for expressing service orchestration (the inner workings of the service in terms of other involved services), and we also do not describe user goals and mediators, considering these concepts out of scope of semantic service description. On the other hand, our service semantics ontology adds the possibility of specifying service functionality using a classification, which is a very straightforward and simple approach not directly supported in WSMO.

One missing piece in the various SWS technologies (incl. OWL-S, WSMO, and our approach) is *data grounding*, a way of specifying mappings (both lifting and lowering) between the higher-level semantic data used by the client software and the low-level messages required by the Web services. The suitability of different data grounding technologies depends on the kind of data that is exchanged by the service, therefore this area is usually left as an extension point. Various technologies can be plugged in to provide the grounding functionality: XSLT [18] as the basic standard XML transformation language; XSPARQL [1] as a new research proposal that combines the power of SPARQL for RDF manipulation with XQuery for handling XML data, simplifying the actual transformations; or approaches such as [12] by Necasky and Pokorny, based on a hub-and-spoke mapping approach with a central conceptual model.

Another area of related work is around the automation algorithms. While we sketch a number of algorithms in Section 6, it is not the purpose of this paper to provide deep details of these algorithms, or to provide a wide survey of existing approaches. We plan to perform implementation and evaluation of a number of these algorithms, supporting WSMO-Lite and MicroWSMO, in the scope of the EU research project SOA4All, whose use cases range from e-commerce to enterprise service delivery.

## 8 Conclusions

In this paper, we have described WSMO-Lite and MicroWSMO, our lightweight semantic Web service description approaches that build on SAWSDL and hRESTS to add semantics to Web services of both main kinds: SOAP/WSDL services and RESTful Web APIs. Our approach focuses on incremental and modular introduction of semantics into service descriptions, and thus complements the more expressive and well-known frameworks such as WSMO.

While lightweight, our semantic Web service description approach has sufficient expressivity to cover most of the known deployments of the larger frameworks. For instance, with the exception of service orchestration, WSMO-Lite could be used instead of WSMO in [5], in particular enabling better modularization of the extent of semantic annotations.

## References


1. W. Akhtar, J. Kopecký, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF worlds – and avoiding the XSLT pilgrimage. In S. Bechhofer and M. Koubarakis, editors, *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008*, volume 5021 of *Lecture Notes in Computer Science, LNCS*, pages 674–689, Tenerife, Spain, June 2008. Springer.
2. Architecture of the World Wide Web. Recommendation, W3C, December 2004. Available at http://www.w3.org/TR/webarch/.
3. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
4. R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000. Chair: Richard N. Taylor.
5. K. Furdik, J. Hreno, and T. Sabol. Conceptualisation and Semantic Annotation of eGovernment Services in WSMO. In *Proceedings of Znalosti 2008*, Bratislava, Slovakia, 2008.
6. M. J. Hadley. Web Application Description Language (WADL). Technical report, Sun Microsystems, November 2006. Available at https://wadl.dev.java.net/.
7. A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX – A Semantic Service-Oriented Architecture. *International Conference on Web Services (ICWS 2005)*, July 2005.
8. J. Kopecký, K. Gomadam, and T. Vitvar. hRESTS: an HTML Microformat for Describing RESTful Web Services. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence (WI-08)*, Sydney, Australia, 2008.
9. J. Kopecký and E. Simperl. Semantic Web Service Offer Discovery For E-commerce. In *Proceedings of the 10th International Conference on Electronic Commerce 2008, Innsbruck, Austria, August 19-22, 2008*, 2008.
10. J. Kopecký and T. Vitvar. WSMO-Lite: Lowering the Semantic Web Services Barrier with Modular and Light-Weight Annotations. In *Proceedings of the 2008 IEEE International Conference on Semantic Computing*, pages 238–244, Santa Clara, USA, 2008.
11. 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.
12. M. Necasky and J. Pokorny. Designing semantic web services using conceptual model. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, Fortaleza, Ceara, Brazil, 2008.
13. D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.
14. D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis. Serving the Sky: Discovering and Selecting Semantic Web Services through Dynamic Skyline Queries. In *Proceedings of the 2008 IEEE International Conference on Semantic Computing*, Santa Clara, USA, 2008.
15. The OWL Services Coalition. OWL-S 1.1 Release. Available at http://www.daml.org/services/owl-s/1.1/, November 2004.
16. T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel. WSMO-Lite Annotations for Web Services. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008*, pages 674–689, Tenerife, Spain, 2008. Springer.
17. Web Services Description Language (WSDL) Version 2.0. Recommendation, W3C, June 2007. Available at http://www.w3.org/TR/wsdl20/.
18. XSL Transformations. Recommendation, W3C, November 1999. Available at http://www.w3.org/TR/xslt.