SAWSDL: Semantic Annotations for WSDL and XML Schema

Jacek Kopecký • *Digital Enterprise Research Institute, Innsbruck*
Tomas Vitvar • *Digital Enterprise Research Institute, Galway*
Carine Bournez • *W3C, Europe*
Joel Farrell • *IBM*

Web services are important for creating distributed applications on the Web. In fact, they're a key enabler for service-oriented architectures that focus on service reuse and interoperability. The World Wide Web Consortium (W3C) has recently finished work on two important standards for describing Web services — the Web Services Description Language (WSDL) 2.0 and Semantic Annotations for WSDL and XML Schema (SAWSDL). Here, the authors discuss the latter, which is the first standard for adding semantics to Web service descriptions.

Web services add a new level of functionality to the Web, a step toward an open environment of distributed applications. Although current Web service technologies built around SOAP and the Web Services Description Language (WSDL) form a solid foundation, scaling will be difficult without a proper degree of automation. In large-scale, open, and heterogeneous environments, Web services' success depends on resolving the fundamental challenges that existing integration technologies don't address — namely, search, integration, and mediation. XML descriptions of Web services support integration in rigid workflow or services configurations, but automation requires more than a description of the data structure and syntax. We can achieve increased automation using semantic technologies, such as those underlying the Semantic Web, as described by Tim Berners-Lee and colleagues in their seminal Semantic Web article.[1]

Building on WSDL, Semantic Annotations for WSDL and XML Schema (SAWSDL; www.w3.org/TR/sawsdl) adds hooks that let WSDL components point to their semantics (see Figure 1). SAWSDL itself doesn't provide any specific semantics; rather, it enables us to annotate the purely syntactic WSDL descriptions with pointers to semantic concepts. Software systems can interpret these concepts to (partially or fully) automate such tasks as service discovery, composition, selection, negotiation, mediation, and invocation.

Web Services and the Semantic Web

The Web supports distributed computing, and the term service-oriented architecture (SOA) was created in particular as a generalization of Web services technologies. The Web is already a vast data repository, and Web services are growing very quickly. To make sense of all this data and these services, the Semantic Web builds on the foundations of logic and knowledge representation to help computers find the right information for their users. Finding and combining information is only part of the vision, however. Computers must also be able to find and combine services on the Web to free users' hands and make the Web of services scale together with the Web of data.

SAWSDL is the World Wide Web Consortium's (W3C; www.w3.org) first step toward standardizing technologies for Semantic Web services (SWSs). As a standard, it provides a common

 1089-7801/07/$25.00

ground for the various ongoing efforts toward SWS frameworks, such as the Web Service Modeling Ontology (WSMO; www.wsmo.org)[2] and the OWL-based Web Service Ontology (OWL-S; www.daml.org/services/owl-s).

The major technologies for Web services are SOAP and WSDL. SAWSDL extends WSDL with pointers to semantics that are crucial for achieving automation. Figure 2 shows a stack of Web services description layers. The base is formed by WSDL and its bindings into lower-level communication technologies, especially HTTP and SOAP. On top of WSDL is a layer of semantic annotations that the higher layer of the service ontology uses. The top represents the application and domain-specific ontologies and other semantic descriptions.

SAWSDL Working Group

In 2004, the W3C started receiving submissions of specifications for semantic descriptions of Web services (OWL-S, WSMO, and others). In June 2005, it held the Workshop on Frameworks for Semantics in Web Services to discuss the proposed steps. The workshop identified a lack of agreement on what Semantic Web services should do; yet, most participants conceded that semantics are necessary in Web service descriptions and that building on WSDL, as WSDL-S[3] proposes, would be a good start.

In April 2006, the W3C formed a working group to standardize semantic annotations for WSDL, which resulted in the recommendation for SAWSDL, published in August 2007. SAWSDL builds mainly on WSDL 2.0,[4] but also supports the still-prevalent WSDL 1.1.[5] On the semantic side, SAWSDL is independent of any ontology technology and assumes that semantic concepts can be identified via URIs. For instance, SWS frameworks can use the Resource Description Framework (RDF) and Web Ontology Language (OWL) with SAWSDL to annotate Web services.

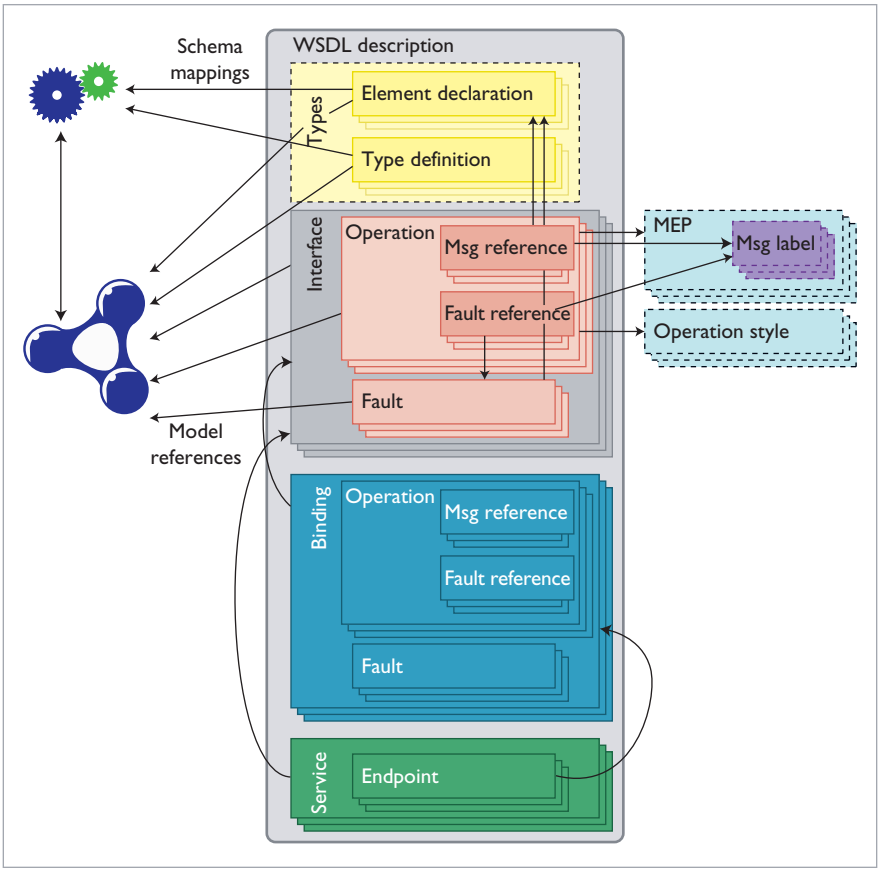

Figure 1. The Web Services Description Language (WSDL) with Semantic Annotations for WSDL and XML Schema (SAWSDL). The figure shows WSDL components and the SAWSDL annotations that point to semantic concepts for specifying semantics or to schema mappings for data transformations.

The working group included representatives from important Semantic Web research centers such as the Digital Enterprise Research Institute (DERI) Innsbruck (www.deri.at), DERI Galway (www.deri.ie), Wright State University (www.wright.edu), Georgia University (www.uga.edu), and Open University (www.open.ac.uk), as well as major industry organizations interested in Semantic Web services, such as IBM, ILOG (www.ilog.com), Telecom Italia (www.telecomitalia.it), CA (www.ca.com), and Scapa Technologies (www.scapatech.com).

Along with the SAWSDL specification, the working group produced a companion usage guide (www.w3.org/TR/sawsdl-guide/) to provide more examples on how SWS solutions can use SAWSDL. As part of the standardization process, the working group also produced a listing of already existing tools that use SAWSDL (see the sidebar "SAWSDL Implementations" for more information).

WSDL and SAWSDL

SAWSDL is a set of extensions for WSDL, which provides a standard description format for Web services. WSDL uses XML as a common flexible data-exchange format and applies XML Schema for data typing. It describes a Web service on three levels:

- *Reusable abstract interface* defines a set of operations, each representing a simple exchange of messages described with XML Schema element declarations.
- *Binding* describes on-the-wire mes-

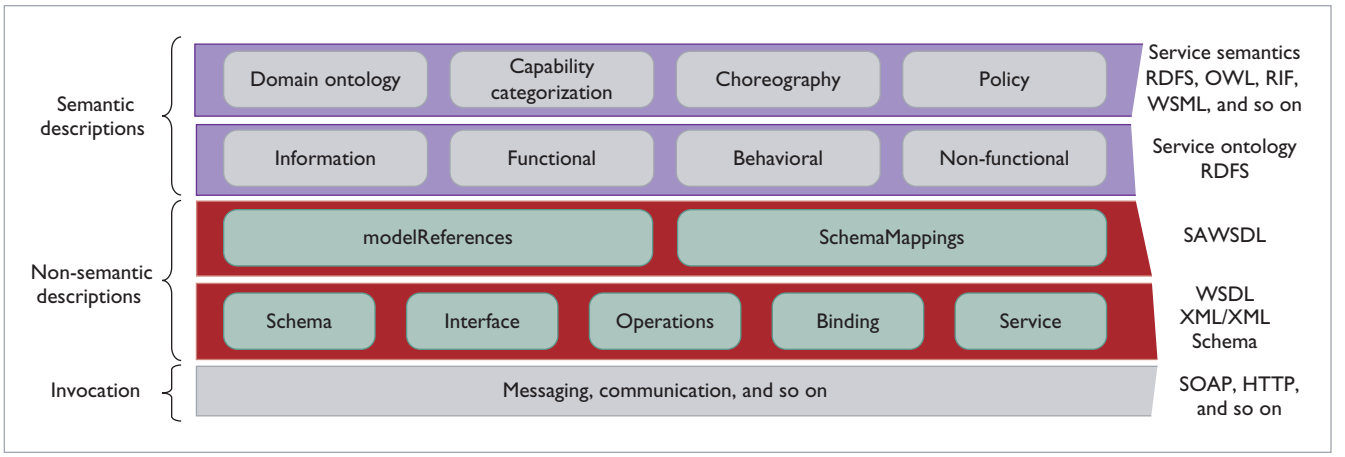

Figure 2. Extended Web service specification stack. The main description language, WSDL, is tightly bound to the underlying communication technologies. SAWSDL is a layer above WSDL that connects it to the semantic technologies — a service ontology that describes the common aspects of Web services and domain ontologies that fill in the actual application-specific details.

Table 1. Semantic Annotations for WSDL and XML Schema (SAWSDL) syntax summary.

Name	Description
`modelReference`	A list of references to concepts in some semantic models (XML attribute)
`liftingSchemaMapping`	A list of pointers to alternative data-lifting transformations (XML attribute)
`loweringSchemaMapping`	A list of pointers to alternative data-lowering transformations (XML attribute)
`attrExtensions`	Attaches attribute extensions where only element extensibility is allowed (XML element)

sage serialization; it follows the structure of an interface and fills in the necessary networking details (for instance, for SOAP or HTTP).
- *Service* represents a single physical Web service that implements a single interface; the Web service can be accessed at multiple network *endpoints*.

WSDL aims to describe the Web service on a *syntactic* level: it specifies what messages look like rather than what they mean. SAWSDL is a simple extension layer on top of WSDL that lets WSDL components specify their *semantics*. SAWSDL defines extension attributes that we can apply to elements both in WSDL and in XML Schema to annotate WSDL interfaces, operations, and their input and output messages.

The SAWSDL extensions take two forms: *model references* that point to semantic concepts and *schema mappings* that specify data transformations between messages' XML data structure and the associated semantic model. In Table 1, we summarize the complete syntax introduced by SAWSDL.

Model References

A model reference is an extension attribute, `sawsdl:modelReference`, that we can apply to any WSDL or XML Schema element in order to point to one or more semantic concepts. The value is a set of URIs, each one identifying some piece of semantics.

Model references generically refer to semantic concepts, thus serve as hooks for attaching semantics. As we'll illustrate later, we can use model references to describe the meaning of data or to specify the function of a Web service operation.

Schema Mappings

SAWSDL provides two attributes for attaching schema mappings: `sawsdl:liftingSchemaMapping` and `sawsdl:loweringSchemaMapping`. Lifting mappings transform XML data from a Web service message into a semantic model (for instance, into RDF data that follows some specific ontology), whereas lowering mappings transform data from a semantic model into an XML message.

Lifting and lowering transformations are useful for communicating with a Web service from a semantic client – for example, the client software will *lower* some of its semantic data into a request message and send it to the Web service; when the client software receives the response message, it can *lift* the data contained in the message for semantic processing (see Figure 3a).

We can also use lifting and low-

SAWSDL Implementations

The W3C standardization process requires that every specification under development be tested for implementability before it becomes a final W3C recommendation. Every feature of the specification should be functional in at least one implementation, and, optimally, would be interoperable between at least two implementations. A working group wishing to proceed with a specification needs to create an implementation report. The Semantic Annotations for WSDL (SAWSDL) working group's implementation report (available at www.w3.org/2002/ws/sawsdl/CR) shows several implementation categories.

SAWSDL's direct implementations are parser APIs that make the annotations available to applications, and tools that let users annotate Web Services Description Language (WSDL) documents with semantic annotations. In the first category, the Woden API for WSDL 2.0 and the WSDL4J API for WSDL 1.1 were both extended to handle SAWSDL annotations. Many Java-based tools use these APIs, including Axis 2, the Apache Web services stack. Two GUI tools help annotate WSDL documents with semantics: Radiant from the University of Georgia and the Web Service Modeling Ontology (WSMO) Studio from Ontotext.

Because SAWSDL is a specification for hooks for attaching semantics, its value is mainly in actual applications that add semantics to Web service descriptions. The SAWSDL implementation report mentions several such applications. In particular, we can attach both OWL-S and WSMO, the major Semantic Web services frameworks, to WSDL service descriptions using SAWSDL, as specified in these frameworks' respective grounding definitions. On top of that, we can use SAWSDL for simple Web service discovery in a tool called Lumina from the University of Georgia and for semantic data matching and mediation in Semantic Tools for Web Services from IBM. Finally, we can use SAWSDL to add semantic annotations to Business Process Execution Language for Web Services (BPEL4WS), a use that the SAWSDL working group didn't directly anticipate; nevertheless, such unexpected uses are appropriate and well within the spirit of the SAWSDL specification.

ering annotations for XML data mediation through a shared ontology (see Figure 3b). An automated mediator can lift the data in one XML format to data in the shared ontology and then lower it to another XML format using the lifting annotation from the first format's schema and the lowering one from the second schema.

In XML Schema, we describe an XML element's content by a *type definition* and add the element's name as an *element declaration*. SAWSDL model reference and schema mapping annotations can be both on types and on elements; in fact, a type's annotations also apply to the elements of that type.

In particular, a SAWSDL processor merges the type's model references with the element's model references, and all of them apply to the element. Schema mappings, on the other hand, are only propagated from the type if the element doesn't declare any schema mappings of its own. This lets a type provide generic schema mappings and an element specify more concrete mappings appropriate for the type's specific use.

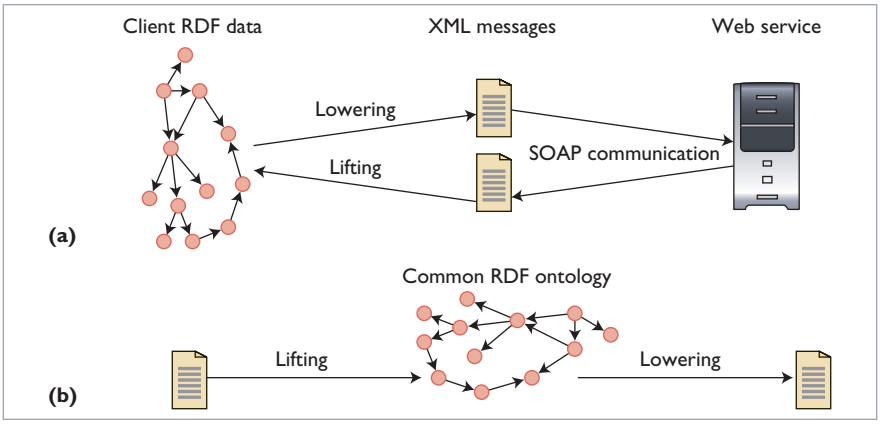

Figure 3. Lifting and lowering semantic data. We can use lifting and lowering transformations for (a) Web service communication and (b) XML data mediation.

WSDL 1.1 Support

Although SAWSDL is built primarily for WSDL 2.0, it also supports the older and more prevalent version, WSDL 1.1. Essentially, both model references and schema mappings apply in the same places in both WSDL versions. However, the XML Schema for WSDL 1.1 allows only element extensions on operations, so a WSDL 1.1 document with the SAWSDL `modelReference` attribute on an operation wouldn't be valid. To overcome this obstacle, SAWSDL defines the element `attrExtensions` to carry extension attributes in places where only element extensibility is allowed. Instead of putting the model reference directly on the `operation` element, SAWSDL can put it on the `attrExtensions` element, then insert that into the `operation` element.

On Top of SAWSDL: Semantics for Web Services

SAWSDL alone isn't an actual framework for modeling Semantic Web services: it needs a concrete service

Table 2. Semantic annotations for Web Services Description Language (WSDL) components.

WSDL component	Semantics type	Description
Schema	Information	Ontology pointers, mappings
Interface	Functional	General, reusable capability or category
Interface operation	Functional	Concrete operation capability or category
Service	Functional	Concrete service capability or category
Interface	Behavioral	General, reusable behavioral description
Service	Behavioral	Concrete service behavior
Service and endpoint	Nonfunctional	Nonfunctional properties and policies
Binding (and subcomponents)	Nonfunctional	Operation-specific nonfunctional properties

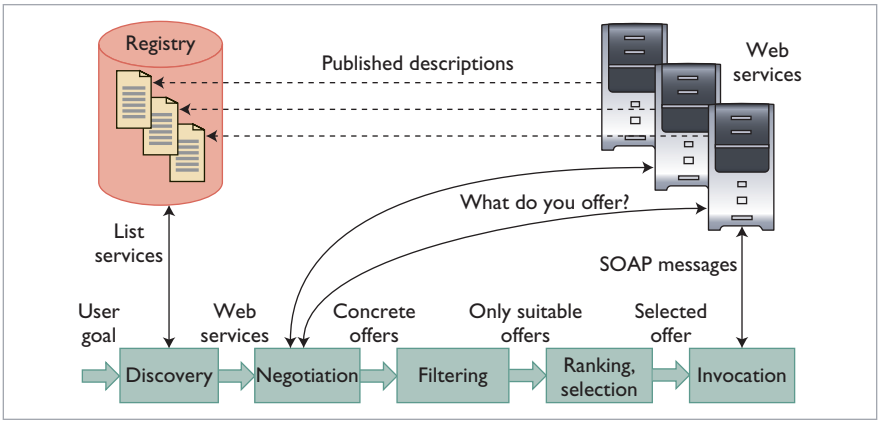

Figure 4. Tasks performed by a Semantic Web services client system. Starting with the discovery of Web services applicable to a given user goal, and ending with actual invocation of a selected Web service, these tasks can be automated using semantic technologies.

ontology that specifies what semantics we can attach to the SAWSDL hooks. This section shows what a very lightweight service ontology could look like.

Adding semantics to Web services mainly aims to automate certain tasks that must be performed with services before or during invocation. Based on various efforts in SWS and service-oriented computing communities (such as OWL-S and WSMO), the generally accepted tasks are discovery, negotiation, filtering, selection, and invocation (as Figure 4 shows), complemented by composition and interspersed with mediation. To perform these tasks, a user or an application has a *semantic client* – that is, a service requester or a middleware system that performs various combinations of tasks according to a particular application's requirements. More information about intelligent middleware systems for SWSs is available elsewhere.[6]

Different tasks require different semantic descriptions. We can describe four aspects of services semantically:

- *Information semantics* define an information model (an ontology together with instance data). Other semantic descriptions use information semantics and usually need them when performing data mediation through ontology merging or mapping/aligning.
- *Functional semantics* describe the service capability – that is, what the service can offer its users. They're used for service discovery, comparing a user's need (in the form of a goal description) and the functional descriptions of available Web services. Service composition also uses functional semantics (together with information semantics) when creating a plan for a given goal.
- *Nonfunctional semantics* define additional constraints and policies over service functionality that functional semantics don't capture. Nonfunctional semantics are needed to match discovered services against users' preferences and constraints.
- *Behavioral semantics* describe a service's public and private behavior. A description of the public behavior is a protocol that each client must follow in order to consume the service's functionality. It guides negotiation with the service as well as its invocation. Service discovery or composition might also incorporate compatibility checks or process mediation between the client's and the service's expected behaviors.

Table 2 summarizes how the various types of semantics fit into SAWSDL.

Information Semantics

Several options exist for expressing ontologies; they typically vary in language expressivity, and the available reasoning and querying algorithms range in performance and scalability. The standard options are

Semantic Web Languages

Semantic Annotations for WSDL and XML Schema (SAWSDL) lets us point to semantics from the Web Services Description Language (WSDL). But to represent the semantics, we need to reach for knowledge representation languages.

The W3C has produced several language recommendations for representing and exchanging knowledge on the Semantic Web. At the core, the Resource Description Framework (RDF; www.w3.org/RDF) represents information in graph-based models with so called *triples* — that is, statements in the form `<subject, predicate, object>`. The subjects and objects link the triples into a graph. RDF provides various syntaxes including RDF/XML and Notation 3 (N3).

RDF Schema (RDFS) defines constructs on top of RDF that enable the specification of lightweight ontologies: RDFS lets us define classes and properties as well as class and property hierarchies. On top of RDFS, the Web Ontology Language (OWL) provides a more expressive vocabulary along with a formalism based on description logics. Last but not least, an ongoing effort in the W3C Rule Interchange Format Working Group (RIF WG) aims to extend the existing languages with rules.

Several nonstandard languages are also available outside the W3C. For instance, the Web Services Modeling Language (WSML; www.wsmo.org/wsml) is an ontology language with a variant called WSML-Rule that's based on logic programming; the Semantic Web Rule Language (SWRL; www.daml.org/rules/proposal) is a proposed rule language combining OWL and RuleML. Both languages (and others) were submitted to W3C as inputs to the RIF work. Because RIF is in its early stages, we use WSML-Rule to capture logical expressions in some examples in this article.

RDF Schema (RDFS) and OWL, with its subsets OWL-Lite and OWL-DL (a fragment of Description Logics). We can draw further expressivity from rule languages, such as the upcoming W3C Rule Interchange Format (RIF), especially for capturing logical conditions. (See the "Semantic Web Languages" sidebar for more details on the available languages for expressing information semantics.)

Information semantics apply to the XML schema components that WSDL uses to describe messages — in particular, *element declarations* and *type definitions*, as Figure 5 shows (lines 6 through 8). Both can carry SAWSDL model references that link them to classes in the ontology. The model references are accompanied by lifting and lowering schema mappings to enable data exchange between the semantic client and the XML-based Web service.

Functional Semantics

Functional semantics is the formal description of service functionality, describing what a service can offer to its clients when they invoke it. We can describe this at two levels of granularity:

- *Categorization* — the service functionality falls within some category in an agreed classification schema, such as the United Nations Standard Products and Services Code (UNSPSC; www.unspsc.org) or the RosettaNet Technical Dictionary (RNTD; portal.rosettanet.org/cms/sites/RosettaNet/Standards/RStandards/dictionary).
- *Capability description* — we can also define the functionality using logical conditions that must hold before and after service invocation, so called preconditions and effects.

Here, we show a simple capability ontology that would be part of a larger service ontology:

```
sem:Capability
  rdf:type rdfs:Class.
sem:hasPrecondition
  rdf:type rdf:Property ;
  rdfs:domain sem:Capability ;
  rdfs:range sem:Axiom.
sem:hasEffect
  rdf:type rdf:Property ;
  rdfs:domain sem:Capability ;
  rdfs:range sem:Axiom.
sem:Axiom rdf:type rdfs:Class.
```

It introduces the class `Capability` with predicates `hasPrecondition` and `hasEffect`. Both these predicates' range is `Axiom`, meaning an arbitrary logical expression, which can be written in the syntax of any logical language, such as RIF or the Web Services Modeling Language (for example, WSML-Rule).

Functional semantics apply to the Web service, which is represented concretely by the `service` construct and abstractly by the reusable `interface` construct. We use a SAWSDL `modelReference` to point from a service or an interface to its appropriate capability, as we can see in Figure 5 (line 14, and lines 45 to 55). Multiple services might share a WSDL interface, thus its capability should be general because it effectively applies to all services that implement that interface. A concrete capability attached to the service then refines the interface's capability.

The distinction between a general capability on the interface and a concrete, refined capability on the service enables an SWS discovery algorithm to first find appropriate interfaces and then deal only with services that implement them.

Apart from describing the service (or the interface) as a whole, we can also ascribe capabilities to the operations, again using `modelReference` pointers. Describing operation capabilities is especially useful for Web serv-

```
// the WSDL description
<wsdl:description...>
<wsdl:types>
<xs:schema...>
...
<xs:element name="NetworkConnection"
type="NetworkConnectionType"
sawsdl:modelReference="http://example.org/
onto#NetworkConnection"
sawsdl:loweringSchemaMapping="http://
example.org/NetConn.xslt" />
...
</xs:schema>
</wsdl:types>

<wsdl:interface name="NetworkSubscription"
sawsdl:modelReference="http://example.org/
onto#VideoOnDemandSubscription" >
...
</wsdl:interface>
...
<wsdl:service name="ExampleCommLtd"
interface="NetworkSubscription"
sawsdl:modelReference="http://example.org/
onto#VideoOnDemandPrice">
<wsdl:endpoint name="public"
binding="SOAPBinding"
address="http://example.org/comm.ltd/
subscription" />
</wsdl:service>
</wsdl:description>

// and now the ontological definitions
// ontology fragment
ex:Bundle rdf:type rdfs:Class.
ex:NetworkConnection rdf:type rdfs:Class.
ex:Service rdf:type rdfs:Class.
ex:hasService rdf:type rdf:Property ;
  rdfs:domain ex:Bundle ;
  rdfs:range ex:Service.
ex:hasConnection rdf:type rdf:Property ;
  rdfs:domain ex:Bundle ;
  rdfs:range ex:NetworkConnection.
ex:providesBandwidth rdf:type rdf:Property ;
  rdfs:domain ex:NetworkConnection ;
  rdfs:range xs:integer.
ex:DSLConnection rdf:type rdfs:Class ;
  rdfs:subClassOf ex:NetworkConnection.

// interface capability
ex:VideoOnDemandSubscription rdf:type
sem:Capability ;
  sem:hasPrecondition "
?customer[hasConnection hasValue
?connection] memberOf Customer and
?service[requiresBandwidth hasValue ?x] memberOf
Service and
?connection[providesBandwidth hasValue ?y]
memberOf NetworkConnection and
?y > ?x
"^^wsml:AxiomLiteral.
sem:hasEffect "
?bundle[hasService hasValue ?service and
hasConnection hasValue ?connection]
memberOf Bundle
"^^wsml:AxiomLiteral.

wsml:AxiomLiteral rdf:type rdfs:Class;
  rdfs:subClassOf sem:Axiom.

// service price
ex:VideoOnDemandPrice rdf:type
ex:PriceSpecification ;
  ex:pricePerChange "30"^^ex:euroAmount ;
  ex:installationPrice "49"^^ex:euroAmount.
```

Figure 5. Web Services Description Language (WSDL) with semantic annotations, followed by the relevant ontologies. This listing illustrates how simple semantic models can be attached to a WSDL document for use in a semantic Web service client system.

ices whose interface is simply a collection of stand-alone operations. For instance, a network subscription service might offer independent operations for subscribing to a bundle, canceling a subscription, or inquiring about pricing. A client will generally want to use only one or two of the operations, not all three. In such cases, instead of service discovery, the semantic client might need to do the more fine-grained *operation discovery*.

Categorization and capability description aren't exclusive alternatives — we can combine them. Although the SAWSDL `modelReference` URI values don't indicate whether the annotations go to capabilities or categories (or any other type of semantics, for that matter), the semantic model will make it clear.

Behavioral Semantics

In general, behavioral semantics is the formal description that defines a service's *external* (public) and *internal* (private) behavior. The external behav-

ior describes a protocol that the client can use to consume the service functionality. The internal behavior describes a workflow — that is, how the service's functionality is aggregated out of other services. However, internal behavior isn't directly interesting to the semantic client.

We describe public behavior to define the order in which the client should invoke the Web service's operations. We can do this simply by describing the operations' capabilities (as we discussed in the previous section) and letting the client sort out what operations it needs to invoke, or we can describe the service's behavior explicitly using a dedicated language. An explicit behavioral description might be easier for the client to follow, especially if the client is a device with limited resources for logical reasoning.

Nonfunctional Semantics

Generally, nonfunctional properties are incidental details specific to a service's implementation or running environment, independent of its actual purpose, but necessary for successful and interoperable communication.

Nonfunctional semantics are always specific to a concrete service (see Figure 5, line 20); therefore, we don't recommend annotating interfaces with nonfunctional properties. In case they need to be specified on the granularity of operations (for example, different operations might have different invocation micropayment prices), we can use a WSDL binding or any of its subcomponents to capture these properties. With SAWSDL, we attach nonfunctional properties using `modelReference` from any of the WSDL components to the nonfunctional semantics model.

Some service constraints that fall within nonfunctional semantics are often expressed as *policies*. Policy languages such as WS-Policy (www.w3.org/TR/ws-policy) generally provide their own means for associating policies to policy subjects (such as WS-PolicyAttachment in the WS-Policy framework). Using a policy framework for capturing nonfunctional properties would be an alternative to using SAWSDL `modelReference`.

SAWSDL is the first step toward standardizing SWS. It forms the basis for interoperation between the various SWS efforts that previously couldn't seem to find any common ground. As we have shown, SAWSDL itself isn't a complete technology for allowing automation; indeed we must provide a service ontology and the appropriate domain ontologies to describe Web services. The major SWS frameworks (WSMO and OWL-S) have already started to embrace SAWSDL for *grounding* (connecting the semantic framework to the WSDL descriptions of Web services). The next step is to rework these frameworks with SAWSDL in mind, refactoring them into parts that can be attached using SAWSDL annotations. Such a refactored framework based on RDF and OWL would likely be the basis for further standardization in the W3C. You can discuss this article in the public mailing list at public-ws-semann@w3.org.

References

1. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific Am.*, vol. 284, no. 5, 2001, pp. 34–43.
2. D. Roman et al., "Web Service Modeling Ontology," *Applied Ontology*, vol. 1, no. 1, 2005, pp. 77–106.
3. K. Verma and A. Sheth, "Semantically Annotating a Web Service," *IEEE Internet Computing*, vol. 11, no. 2, 2007, pp. 83–85.
4. *Web Services Description Language (WSDL) v. 2.0*, World Wide Web Consortium (W3C) recommendation, June 2007; www.w3.org/TR/wsdl20.
5. "Web Services Description Language (WSDL) 1.1," World Wide Web Consortium (W3C) note, Mar. 2001; www.w3.org/TR/wsdl20.
6. T. Vitvar et al., "Semantically-Enabled Service-Oriented Architecture: Concepts, Technology and Application," *Service Oriented Computing and Applications*, vol. 2, no. 2, 2007, pp. 129–154.

Jacek Kopecký is a researcher at the Digital Enterprise Research Institute in Innsbruck, Austria. His research interests include Semantic Web services, and Web technologies in general. Kopecký has been involved with Web services standardization since 2001, and he chaired the W3C SAWSDL working group. Contact him at jacek.kopecky@deri.at.

Tomas Vitvar is a senior researcher at the Digital Enterprise Research Institute in Galway, Ireland. His research interests are in distributed systems and applications, including service-oriented computing, Semantic Web services, and enterprise computing. Vitvar has a PhD in computer science from the Czech Technical University. He is a member of the IEEE and of working groups in the W3C and the Organization for the Advancement of Structured Information Standards. Contact him at t.vitvar@ieee.org.

Carine Bournez is an engineer in the architecture domain at W3C. Her research interests include Web services and agents, associated semantics, and XML technologies. Bournez has an engineer degree and PhD in computer science from INSA Lyon, France. Her involvement in W3C Web services activity started with SOAP 1.2 and continued with Semantic Web services work, including SAWSDL. Contact her at carine@w3.org.

Joel Farrell is a senior technical staff member at IBM. His research interests include Web services, process automation through semantic reasoning, and situational applications. Farrell has an MS in computer science from Syracuse University. He is the chair of the Technical Steering Committee of the MedBiquitous Consortium. Contact him at joelf@us.ibm.com.