Semantic Web Services

Bearbeitet von
Dieter Fensel, Federico Michele Facca, Elena Simperl, Ioan Toma

1. Auflage 2011. Buch. xi, 357 S. Hardcover
ISBN 978 3 642 19192 3
Format (B x L): 15,5 x 23,5 cm
Gewicht: 719 g

Chapter 12
Lightweight Semantic Web Service Descriptions

Abstract The Web standardization consortium W3C has developed a lightweight bottom-up specification, Semantic Annotation for WSDL (SAWSDL), for adding semantic annotations to WSDL service descriptions. In this chapter, we describe SAWSDL, and then we present WSMO-Lite and MicroWSMO, two related lightweight approaches to Semantic Web Service description, evolved from the Web Service Modeling Ontology (WSMO) framework. WSMO-Lite defines an ontology for service semantics, used directly in SAWSDL to annotate WSDL-based services. MicroWSMO and its basis, hRESTS, are microformats that supplement WSDL and SAWSDL for unstructured HTML descriptions of services, providing WSMO-Lite support for the growing numbers of RESTful services.

12.1 Motivation

As the first step towards standardizing semantic descriptions for Web services, the Web standardization consortium W3C has developed a lightweight bottom-up specification for annotating WSDL service descriptions. The specification is called Semantic Annotations for WSDL and XML Schema (SAWSDL, [3, 6]).

Technologies such as WSMO (cf. Chap. 7) and OWL-S (cf. Chap. 11) embody the so-called *top-down* approach to semantic Web service modeling: a service engineer describes the semantics of the service independently of the actual realization in the service implementation; then the semantics is *grounded* in an underlying description such as WSDL. This approach separates the semantics from the underlying WSDL, on the principle that the semantics should not be influenced by implementation details.

In contrast, SAWSDL embodies a *bottom-up* modeling approach where the WSDL layer that describes the service on the implementation level forms the basis for adding semantics. With a bottom-up approach, semantics are added to WSDL descriptions in a modular fashion, as needed in a concrete deployment. SAWSDL by itself, however, does not specify any actual types of semantics; it only gives us hooks for attaching semantics to WSDL descriptions. SAWSDL is the ground stone on which semantic Web services frameworks should be built, and a catalyst for building them. We further describe SAWSDL in Sect. 12.2.1.

The standardization of SAWSDL led to the development of WSMO-Lite [4], an emerging technology that realizes the WSMO Semantic Web Services approach in

D. Fensel et al., *Semantic Web Services*, 279

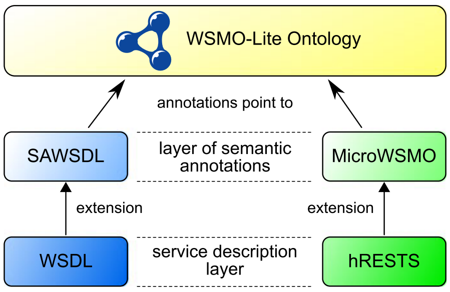


Fig. 12.1 WSMO-Lite and MicroWSMO layer cake

SAWSDL and RDF. WSMO-Lite identifies four types of semantics and defines a lightweight RDF Schema ontology for expressing them. Built on a simplified service model, WSMO-Lite applies directly to WSDL services, described in Chap. 4. We present the service semantics defined by WSMO-Lite in Sect. 12.2.2, and in Sect. 12.2.3, we show the specifics of how WSMO-Lite applies to WSDL descriptions.

The WSMO-Lite service model also maps naturally to RESTful services, described in Chap. 5. However, at the time of writing, there is no widely accepted standard for machine-readable descriptions of RESTful services, which are instead simply documented in HTML Web pages. Faced with HTML pages, we employ two *microformats* (an "adaptation of semantic XHTML that makes it easier to publish, index, and extract semi-structured information" [2]) to make the key pieces of the human-oriented service documentation machine-readable (hRESTS), and to add semantic annotations (MicroWSMO). Both microformats, defined in [5], are described in Sect. 12.2.4.

12.2 Technical Solution

Figure 12.1 shows the relative positioning of WSMO-Lite, WSDL with SAWSDL, and hRESTS with MicroWSMO. WSDL is the standard description language for Web services; SAWSDL extends it with annotations that can point to semantics, such as WSMO-Lite. hRESTS is analogous to WSDL—it describes the basic structure of RESTful services. MicroWSMO is then a direct realization of SAWSDL annotations over hRESTS. Both SAWSDL and MicroWSMO are annotation mechanisms enabling pointers to semantics; WSMO-Lite specifies a simple vocabulary for four types of semantics that fit in SAWSDL/MicroWSMO annotations.

Below, we first describe SAWSDL, the simple specification that is the cornerstone of lightweight semantic Web services frameworks. Then we proceed to detail WSMO-Lite, with its simple service model and four types of service semantics. Afterwards, we show exactly how WSMO-Lite semantics are applied in WSDL and SAWSDL. Finally, we extend the reach of WSMO-Lite also to RESTful services, using the microformats hRESTS and MicroWSMO.

12.2.1 SAWSDL

SAWSDL is a set of extensions for WSDL,[1] the standard description format for Web services. To briefly recap from Chap. 4, WSDL uses XML as a common flexible data exchange format, and applies XML Schema for data typing. WSDL describes a Web service in 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 the on-the-wire message serialization, following the structure of an interface and filling in the necessary networking details (for instance, for SOAP or HTTP).
- *Service* represents a single physical Web service which implements a single interface; the Web service can be accessible at multiple network *endpoints*.

The purpose of WSDL is to describe the Web service on a *syntactic level*: WSDL specifies what the messages *look like* rather than what the messages or operations *mean*. SAWSDL defines an extension layer over WSDL that allows the *semantics* to be specified on the various WSDL components. SAWSDL defines extension attributes that can be applied to elements both in WSDL and in XML Schema in order to annotate WSDL interfaces, operations and their input and output messages.

The SAWSDL extensions are of two forms: *model references* point to semantic concepts, and *schema mappings* specify data transformations between the XML data structure of messages and the associated semantic model.

12.2.1.1 Model Reference

A model reference is an extension attribute, `sawsdl:modelReference`, that can be applied 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 reference has the very generic purpose of referring to semantic concepts. As such, it serves as a *hook* where semantics can be attached. As illustrated in the examples presented later in this chapter, model reference can be used to describe the meaning of data or to specify the function of a Web service operation.

12.2.1.2 Schema Mappings

Schema mappings are represented by two attributes, `sawsdl:liftingSchema-Mapping` and `sawsdl:loweringSchemaMapping`. Lifting mappings transform XML data from a Web service message into a semantic model (for instance,

[1] Web Services Description Language, http://www.w3.org/TR/wsdl20/.

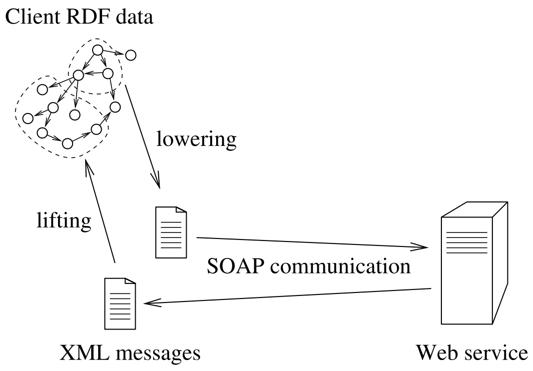


Fig. 12.2 RDF data lifting and lowering for WS communication

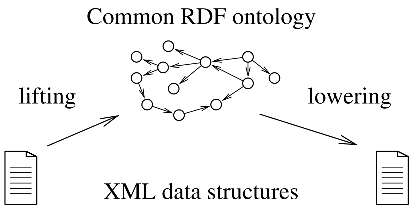


Fig. 12.3 RDF data lifting and lowering for data mediation

into RDF data that follows some specific ontology), and lowering mappings transform data from a semantic model into an XML message.

Lifting and lowering transformations are required for communication with the Web service from a semantic client: the client software will *lower* some of its semantic data into a request message and send it to the Web service, and when the response message is received, it can be *lifted* for semantic processing. This is illustrated in Fig. 12.2.

Lifting and lowering annotations can also be used for XML data mediation through a shared ontology, as shown in Fig. 12.3. The data in one XML format can be *lifted* to data in the shared ontology and then *lowered* to another XML format, using the lifting annotation from the schema for the first format, and the lowering annotation from the second schema.

12.2.1.3 Annotation Propagation

In an XML Schema, the content of an element is described by a type definition, and the name of the element is added as an element declaration. SAWSDL model reference and schema mapping annotations can be both on types and on elements; in fact, the annotations of a type apply also to all elements of that type.

In particular, for a given pair of an element declaration and its type definition, the model references from the type are merged with the model references of the element, and all of them apply to the element. Schema mappings, on the other hand, are only propagated from the type if the element does not declare any schema mappings of its own. This allows a type to provide generic schema mappings, and the

Table 12.1 SAWSDL syntax summary

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

element to specify more concrete mappings appropriate for the specific use of the type. Table 12.1 provides a summary of the syntactical constructs of SAWSDL.

12.2.1.4 WSDL 1.1 Support

While SAWSDL is primarily built for WSDL 2.0, the older and more prevalent version WSDL 1.1 is also supported. Essentially, both model references and schema mappings apply in the same places in both versions of WSDL. However, the XML Schema for WSDL 1.1 only allows element extensions on operations, so a WSDL 1.1 document with the SAWSDL `sawsdl:modelReference` attribute on an operation would not be valid. To overcome this obstacle, SAWSDL defines an element called `sawsdl:attrExtensions` that is intended to carry extension attributes in places where only element extensibility is allowed. Instead of putting the model reference directly on the `wsdl11:operation` element, it is put on the `sawsdl:attrExtensions` element, which is then inserted into the `wsdl11:operation` element.

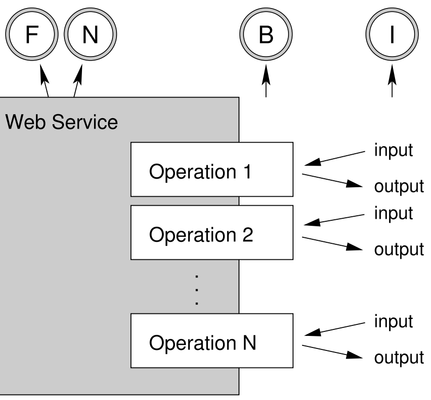
WSDL is a well-known and accepted language for describing Web service interfaces, and virtually all Web service enabled systems use it to advertise and use Web services. SAWSDL is a non-intrusive, simple extension of WSDL that enables semantic annotations in a way that does not invalidate any existing uses of WSDL. This makes it perfectly suitable as the basis for an ontology for describing Semantic Web Services, working towards automation in service-oriented systems.

12.2.2 WSMO-Lite Service Semantics

WSMO-Lite is a lightweight technology for semantic description of Web services. It distinguishes between the following four 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.

Fig. 12.4 Web service description model with attached semantics



- **Non-functional semantics** defines any incidental details specific to the implementation or running environment of a service, such as its price, location or quality of service.
- **Behavioral semantics** specifies the protocol (ordering of operations) that a client needs to follow when consuming a service.

These semantics apply to service descriptions, as illustrated in Fig. 12.4: functional and non-functional semantics belong to the Web service as a whole, behavioral semantics belong to the operations, and the information model describes the input and output messages. With SAWSDL, the Web service description can contain pointers to the semantics as appropriate.

The service model shown in Fig. 12.4 is very similar to the model of WSDL, but it also applies to RESTful Web services (as shown in Chap. 5). It can be seen as the minimal service model: a *Web service* provides several *operations*, each of which can have *input* and *output messages*, along with possible *input* and *output faults*.

Figure 12.5 lists the WSMO-Lite ontology. The ontology consists of 3 main blocks: a *service model* that unifies the views of WSDL-based and RESTful Web services, *SAWSDL annotation properties* for attaching semantics, and finally, *classes* for representing those semantics.

Note that instances of the service model are not expected to be authored directly; instead, the underlying technical descriptions (WSDL, hRESTS) are parsed in terms of this ontology, with SAWSDL pointers to instances of the service semantics classes in the third block.

The service model is rooted in the class wsl:Service. In contrast to WSDL, this service model does not separate the service from its interface, as such separation does not need to be kept to support SWS automation. A service is a collection of operations (class wsl:Operation). Operations may have input and output messages, plus possible fault messages (all in the class wsl: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.

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

```
# namespace declarations (this line is a comment)
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .
@prefix wsl: <http://www.wsmo.org/ns/wsmo-lite#> .

# WSMO-Lite service model classes and properties
# this model is a simplified WSDL-based view of Web services
wsl:Service a rdfs:Class .
wsl:hasOperation a rdf:Property ;
   rdfs:domain wsl:Service ;
   rdfs:range wsl:Operation .
wsl:Operation a rdfs:Class .
wsl:hasInputMessage a rdf:Property ;
   rdfs:domain wsl:Operation ;
   rdfs:range wsl:Message .
wsl:hasOutputMessage a rdf:Property ;
   rdfs:domain wsl:Operation ;
   rdfs:range wsl:Message .
wsl:hasInputFault a rdf:Property ;
   rdfs:domain wsl:Operation ;
   rdfs:range wsl:Message .
wsl:hasOutputFault a rdf:Property ;
   rdfs:domain wsl:Operation ;
   rdfs:range wsl:Message .
wsl:Message a rdfs:Class .

# SAWSDL properties (included here for completeness)
sawsdl:modelReference a rdf:Property .
sawsdl:liftingSchemaMapping a rdf:Property .
sawsdl:loweringSchemaMapping a rdf:Property .

# WSMO-Lite service semantics classes
wsl:Ontology a rdfs:Class ;
   rdfs:subClassOf owl:Ontology .
wsl:FunctionalClassificationRoot rdfs:subClassOf rdfs:Class .
wsl:NonFunctionalParameter a rdfs:Class .
wsl:Condition a rdfs:Class .
wsl:Effect a rdfs:Class .
```

Fig. 12.5 Service Ontology, Captured in Notation 3. (A brief intro to the Notation 3 RDF syntax: comment lines start with "#", URIs are denoted as namespace-qualified names "a", is short for rdf:type, triples end with a period, and triples with the same subject can be grouped, having the next property–object pair after a semicolon instead of a period. More at http://www.w3.org/DesignIssues/Notation3.html)

To link the components of the service model with the concrete description of the functional, non-functional, behavioral and information semantics, as illustrated in Fig. 12.4, we adopt the SAWSDL properties sawsdl:modelReference, sawsdl:lifting-SchemaMapping and sawsdl:loweringSchemaMapping, defined by the SAWSDL RDF mapping in [6].

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 non-functional semantics; a model reference on an operation points to the operation's part of the behavioral semantics description; and a model reference on a message points to the message's counterpart(s) in the service's information model. The lifting and lowering schema mapping properties complement the model references on messages, to point to the appropriate data transformations.

A single component can have multiple model reference values, which all apply together; for example, a service can have a number of non-functional properties together with a pointer to its functionality description, or a message can be annotated with multiple ontology classes, indicating that the message carries instances of all the classes.

In the remainder of this subsection, we describe how the four types of service semantics are represented in the WSMO-Lite service ontology. In the example listings, we use the following namespace prefixes:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix wsl:   <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix ex:   <http://example.org/onto#> .
@prefix xs:   <http://www.w3.org/2001/XMLSchema#> .
@prefix wsml: <http://www.wsmo.org/wsml/wsml-syntax#> .
```

Information semantics are represented using domain ontologies. An ontology intended for information semantics may be marked as wsl:Ontology (as shown below on line 2), so that authoring tools can focus annotation suggestions more easily. The following listing is an example domain ontology for a video-on-demand service:

```
1  # ontology example
2  <> a wsl:Ontology.
3
4  ex:Customer a rdfs:Class .
5  ex:hasService a rdf:Property ;
6     rdfs:domain ex:Customer ;
7     rdfs:range ex:Service .
8  ex:Service a rdfs:Class .
9  ex:hasConnection a rdf:Property ;
10    rdfs:domain ex:Customer ;
11    rdfs:range ex:NetworkConnection .
12 ex:NetworkConnection a rdfs:Class .
13 ex:providesBandwidth a rdf:Property ;
14    rdfs:domain ex:NetworkConnection ;
15    rdfs:range xs:integer .
16 ex:VideoOnDemandService rdfs:subClassOf ex:Service .
```

Functional semantics are represented with *functionality classifications* and/or *preconditions* and *effects*.

Functionality classifications define the service's functionality using some classification taxonomy (i.e., a hierarchy of *categories*, such as the ecl@ss taxonomy[3]), and the class wsl:FunctionalClassificationRoot marks a class that is a root of a classification. The classification also includes all the RDFS subclasses of the root class. Functionality classifications are coarse-grained means for describing service functionality, due to the cost of creating shared taxonomies. An example functionality classification for subscription services is listed below:

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

```
# classification example
ex:SubscriptionService  a  wsl:FunctionalClassificationRoot .
ex:VideoSubscriptionService rdfs:subClassOf ex:SubscriptionService .
ex:NewsSubscriptionService rdfs:subClassOf ex:SubscriptionService .
```

Preconditions and effects can be more fine-grained. They are logical expressions: the *preconditions* must hold in a state before the client can invoke the service, and the *effects* hold in a state after the service invocation. The current set of Web standards does not include a language for logical expressions, therefore the WSMO-Lite classes wsl:Condition (for preconditions) and wsl:Effect are simply placeholders. In our examples, we use WSML to express the logical expressions (cf. Chap. 8); we expect that in the future, the W3C Rule Interchange Format[4] will be a suitable logical expression language for preconditions and effects.

Non-functional semantics are represented using some ontology, semantically capturing non-functional properties such as the price, location or quality of service (QoS). The class wsl:NonfunctionalParameter marks a concrete piece of non-functional semantics, such as the example price specification below:

```
# non−functional property example
ex:PriceSpecification rdfs:subClassOf wsl:NonFunctionalParameter .
ex:VideoOnDemandPrice  a  ex:PriceSpecification ;
  ex:pricePerChange "30"^^ex:euroAmount ;
  ex:installationPrice "49"^^ex:euroAmount .
```

Behavioral semantics are represented by annotating the service operations with functional descriptions, i.e., with functionality classifications and/or preconditions and effects. Below, we show the preconditions and effects for a subscription operation on a video-on-demand service:

```
# precondition and effect examples
ex:VideoOnDemandSubscriptionPrecondition  a  wsl:Condition ;
   rdf:value """
      ?customer[ex#hasConnection hasValue ?connection]
         memberOf ex#Customer and
      ?connection[ex#providesBandwidth hasValue ?y]
         memberOf ex#NetworkConnection and
      ?y > 1000
   """^^wsml:AxiomLiteral .
ex:VideoOnDemandSubscriptionEffect  a  wsl:Effect ;
   rdf:value """
      ?customer[ex#hasService hasValue ?service]
         memberOf ex#Customer and
      ?service memberOf VideoOnDemandSubscription
   """^^wsml:AxiomLiteral .

# definition of the axiom for WSML language
wsml:AxiomLiteral  a  rdfs:Datatype .
```

[4]http://www.w3.org/2005/rules/.

```
 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="
14           http://example.org/onto#VideoOnDemandSubscriptionPrecondition
15           http://example.org/onto#VideoOnDemandSubscriptionEffect">
16       <wsdl:input element="NetworkConnection"/>
17       ...
18     </wsdl:operation>
19     ...
20   </wsdl:interface>
21   ...
22   <wsdl:service name="ExampleCommLtd"
23       interface="NetworkSubscription"
24       sawsdl:modelReference="http://example.org/onto#VideoOnDemandPrice">
25     <wsdl:endpoint name="public"
26       binding="SOAPBinding"
27       address="http://example.org/comm.ltd/subscription" />
28   </wsdl:service>
29  </wsdl:description>
```

Fig. 12.6 Various WSDL components with WSMO-Lite annotations

12.2.3 WSMO-Lite in SAWSDL

WSMO-Lite is based on the simplified service model shown in Fig. 12.4. In this section, we describe a set of recommendations on how to annotate actual WSDL descriptions with the four kinds of semantics, and a mapping from the annotated WSDL structure to the WSMO-Lite RDFS service model, which is then the input to semantic automation.

The listing in Fig. 12.6 shows WSMO-Lite annotations on an example WSDL document. WSDL distinguishes between a concrete service (line 22) and its abstract (and reusable) interface (line 10) that defines the operations (line 12). This structure is annotated using SAWSDL annotations with the examples of semantics shown in the preceding section.

The following paragraphs describe how the various types of semantics are attached to 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.

Non-functional semantics, by definition specific to a given service, are attached as model references directly to the WSDL service component (line 24).

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, while 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 13–15) with functional descriptions, so the client can then choose the appropriate operation to invoke at a certain point in time during its interaction with the service.

The mapping of a WSDL document into the WSMO-Lite RDFS service model is mostly straightforward: a WSDL `<wsdl:service>` element becomes an instance of WSMO-Lite wsl:Service, the WSDL operations of the interface implemented by the service are attached to the WSMO-Lite service as instances of wsl:Operation, with input, output and fault messages as specified in the WSDL operation. Model references from a WSDL service map directly to model references on the WSMO-Lite service instance; and similarly for model references on the operations.

Since WSMO-Lite does not represent a separate service interface, we combine the interface annotations with the service annotations-model references on an interface and added them to the model references of the service that implements this interface. And finally, any annotations (model references and lifting/lowering schema mappings) from the appropriate XML Schema components are mapped to annotations of the messages in our service model.

The listing in Fig. 12.7 shows a generated RDF form that captures the data in Fig. 12.6, using the lightweight WSMO-Lite RDFS ontology. Note the service, its operation and the operation's request message and the appropriate SAWSDL properties; Fig. 12.6 does not contain any additional data, therefore it is not shown in Fig. 12.7.

12.2.4 WSMO-Lite for RESTful Services

In the case of RESTful services and Web APIs, there is no widely accepted machine-readable service description language. WSDL 2.0 and WADL[5] are two proposals for such a language. However, the vast majority of public RESTful services are described in plain unstructured HTML documentation. In this section, we present two microformats, hRESTS and MicroWSMO, that enhance the HTML service documentation with machine-oriented structure and semantic annotations.

Microformats take advantage of existing XHTML facilities such as the `class` and `rel` attributes to mark up fragments of interest in a Web page, making the fragments more easily available for machine processing. For example, a calendar microformat marks up events with their start and end time and with the event title,

[5]https://wadl.dev.java.net/.

```
@prefix ex: <http://example.com/onto#>
@prefix gen: <http://example.com/svc.wsdl#>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .
@prefix wsl: <http://www.wsmo.org/ns/wsmo-lite#> .

gen:ExampleCommLtd a wsl:Service ;
  rdfs:isDefinedBy <http://example.org/svc.wsdl> ;
  sawsdl:modelReference ex:VideoOnDemandPrice ;
  sawsdl:modelReference ex:VideoSubscriptionService ;
  wsl:hasOperation gen:SubscribeVideoOnDemand .

gen:SubscribeVideoOnDemand a wsl:Operation ;
  sawsdl:modelReference ex:VideoOnDemandSubscriptionPrecondition ;
  sawsdl:modelReference ex:VideoOnDemandSubscriptionEffect ;
  wsl:hasInputMessage gen:SubscribeVideoOnDemandRequest .

gen:SubscribeVideoOnDemandRequest a wsl:Message ;
  sawsdl:modelReference ex:NetworkConnection ;
  sawsdl:loweringSchemaMapping <http://example.org/NetCn.xslt> .
```

Fig. 12.7 RDF representation of the WSDL/SAWSDL data in Fig. 12.6, using the WSMO-Lite RDFS ontology

and a calendaring application can then directly import data from what otherwise looks like a normal Web page.[6]

As we have shown in Chap. 5, 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 operations with input and output messages. The hRESTS microformat 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 group of related Web resources, each of which provides a part of the overall service functionality. The interaction of a client with a RESTful service is a series of interactions where the client sends a request to a resource (using one of the HTTP methods GET, POST, PUT or DELETE), and receives a response that may link to further useful resources. While at runtime the client interacts with concrete resources, the service description may present a single operation that acts on many resources (e.g., getHotelDetails() which can be invoked on any hotel details resource). An operation description in hRESTS can therefore specify an address as an URI template whose parameters are part of the input data, and the HTTP method that implements the operation. To capture this information, hRESTS defines HTML classes `address` and `method`.

The definitions of the hRESTS HTML classes are summarized below, with pointers into the listing in Fig. 12.8, which shows an example hRESTS and MicroWSMO service description:

[6]Further details on how microformats work can be found at http://microformats.org.

```
 1  <div class="service" id="svc">
 2   <h1><span class="label">ACME Hotels</span> service API</h1>
 3    <p>This service is a
 4     <a rel="model" href="http://example.com/ecommerce/hotelReservation">
 5       hotel reservation</a> service.
 6    </p>
 7   <div class="operation" id="op1">
 8    <h2>Operation <code class="label">getHotelDetails</code></h2>
 9    <p> Invoked using the <span class="method">GET</span>
10     at <code class="address">http://example.com/h/{id}</code><br/>
11     <span class="input">
12      <strong>Parameters:</strong>
13      <a rel="model" href="http://example.com/data/onto.owl#Hotel">
14        <code>id</code></a> – the identifier of the particular hotel
15        (<a rel="lowering" href="http://example.com/data/hotel.xsparql">lowering</a>)
16     </span><br/>
17     <span class="output">
18      <strong>Output value:</strong> hotel details in an
19      <code>ex:hotelInformation</code> document
20     </span>
21    </p>
22  </div></div>
```

Fig. 12.8 Example hRESTS and MicroWSMO service description

The **`service`** class on block markup (e.g., `<body>`, `<div>`), as shown in the example listing on line 1, indicates that the element describes a service API. A service contains one or more operations and may have a label (see below).

The **`operation`** class, also used on block markup (e.g., `<div>`), as shown in the listing on line 7, indicates that the element contains a description of a single Web service operation. An operation description specifies the address and the method used by the operation, and it may also contain description of the input and output of the operation, and finally a label.

The **`address`** class is used on textual markup (e.g., `<code>`, shown on line 10) or on a hyperlink (`<a href>`) and specifies the URI of the operation, or the URI template in case any inputs are URI parameters.

The **`method`** class on textual markup (e.g., `<span>`, shown on line 9) specifies the HTTP method used by the operation.

The **`input`** and **`output`** classes are used on block markup (e.g., `<div>` and also `<span>`), as shown on lines 11 and 17, to indicate the description of the input or the output of an operation.

The **`label`** class is used on textual markup to specify human-readable labels for services and operations, as shown on lines 2 and 8.

The definitions above imply a hierarchical use of the classes within the element structure of the HTML documentation. Indeed, the classes are to be used on nested elements, in a hierarchy that reflects the structure of the service model.

In summary, hRESTS can be used to structure the HTML documentation of a RESTful Web service according to the WSMO-Lite simple service model. With this structure in place, we now describe MicroWSMO, a microformat equivalent to SAWSDL. As shown earlier in this chapter, to annotate a service description with the appropriate semantics, a model reference on a service can point to a description of the service's functional and non-functional semantics; a model reference on an operation points to the operation's part of the behavioral semantics description; and a model reference on a message points to the message's counterpart(s) in the service's information semantics ontology, complemented as appropriate by a pointer to a lifting or lowering schema mapping.

SAWSDL annotations are URIs that identify semantic concepts and data transformations. Such URIs can be added to the HTML documentation of RESTful services in the form of hypertext links. HTML defines a mechanism for specifying the relation represented by link, embodied in the `rel` attribute; along with `class`, this attribute is also used to express microformats. In accordance with SAWSDL, MicroWSMO defines the following three new types of link relations:

- **`model`** indicates that the link is a model reference, as shown in the example listing in Fig. 12.8 on lines 4 and 13.
- **`lifting`** and **`lowering`** denote links to the respective data transformations, such as in the example listing on line 15.

Similarly to how SAWSDL annotations in WSDL can be mapped into the WSMO-Lite RDFS service ontology (cf. Figs. 12.6 and 12.7), also hRESTS and MicroWSMO descriptions can be transformed into such RDF. Note that hRESTS extends the WSMO-Lite service model with two properties: hr:hasAddress and hr:hasMethod, applicable to instances of wsl:Operation.

For microformats, the W3C has created a standard called GRDDL (Gleaning Resource Descriptions from Dialects of Languages, [1]) that specifies how HTML documents can point to transformations that extract RDF data. In accordance with GRDDL, an XHTML document that contains hRESTS (and MicroWSMO) data can point to an XSLT stylesheet[7] in its header metadata:

```
<head profile="http://www.w3.org/2003/g/data−view">
 <link rel="transformation"
        href="http://cms−wg.sti2.org/TR/d12/v0.1/20081202/xslt/hrests.xslt" />
 ... further metadata, especially page title ...
</head>
```

This header enables Web crawlers and other parsers to extract the RDF form of the service description data, even if they are not specifically aware of the hRESTS and MicroWSMO microformats. The MicroWSMO description from Fig. 12.8 is embedded in an XHTML document[8] that contains also the GRDDL transformation pointer. Figure 12.9 shows the RDF view of the document.

12.3 Extensions

In this section, we complete the picture of our lightweight SWS approach by sketching a number of algorithms for processing semantically annotated service descrip-

[7]The XSLT stylesheet for hRESTS/MicroWSMO is available at http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/.

[8]http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/example.xhtml.

```
@prefix hr:      <http://www.wsmo.org/ns/hrests#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sawsdl:  <http://www.w3.org/ns/sawsdl#> .
@prefix wsl:     <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix ex:      <http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/example.xhtml#> .

ex:svc  a  wsl:Service ;
  rdfs:isDefinedBy <http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/example.xhtml> ;
  rdfs:label  "ACME Hotels" ;
  sawsdl:modelReference  <http://example.com/ecommerce/hotelReservation> ;
  wsl:hasOperation  ex:op1 .
ex:op1  a  wsl:Operation;
  rdfs:label  "getHotelDetails" ;
  hr:hasMethod  "GET" ;
  hr:hasAddress  "http://example.com/h/{id}"^^hr:URITemplate ;
  wsl:hasInputMessage  [
    a  wsl:Message ;
    sawsdl:modelReference <http://example.com/data/onto.owl#Hotel> ;
    sawsdl:loweringSchemaMapping  <http://example.com/data/hotel.xsparql>
  ] ;
  wsl:hasOutputMessage  [  a  wsl:Message  ] .
```

Fig. 12.9 RDF data extracted from Fig. 12.8 (with blank nodes in square brackets "[ ]")

tions, 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 about concrete formal representation for user goals in this chapter, 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. For example, 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 this is not directly advertised in the description. 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 preconditions of the service and that these preconditions are 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 the Web architecture, 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.

Filtering, Ranking, Selection These tasks mostly deal with the non-functional 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 non-functional 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. 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 non-functional 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 artificial intelligence 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.

12.4 Summary

In this chapter, we presented SAWSDL, and then we introduced WSMO-Lite and MicroWSMO, two related lightweight approaches to semantic Web service description, evolved from the WSMO framework. WSMO-Lite defines an ontology for service semantics, used directly in SAWSDL to annotate WSDLbased services. MicroWSMO and its basis, hRESTS, are microformats that supplement WSDL and SAWSDL for unstructured HTML descriptions of services, giving WSMO-Lite support for the growing numbers of RESTful services.

12.5 Exercises

Exercise 1 Referring to the scenario introduced in Sect. 4.3, create a simple taxonomy of travel services in RDF or OWL.

Exercise 2 Adopting WSML as language, define preconditions for BlueHotelService described in Sect. 4.3.

Exercise 3 Given the WSDL in Listing 4.13, add annotations from the above taxonomy, the precondition, and an ontology.

Exercise 4 Transform the annotated WSDL obtained in Exercise 3 into the WSMO-Lite RDFS form (make up identifiers for the service model components).

Exercise 5 Create from the RESTful version of the BlueHotelService presented in Sect. 5.3 and HTML description and annotate it to obtain the hRESTS.

Exercise 6 Enhance the obtained hRESTS with MicroWSMO pointers, reflecting the ones used in the SA-WSDL of Exercise 3.

Exercise 7 Transform the annotated HTML obtained in Exercise 6 into the WSMO-Lite RDFS form.

References

1. Gleaning resource descriptions from dialects of languages (GRDDL). Recommendation, W3C, (2007). Available at http://www.w3.org/TR/grddl/
2. Khare, R., Çelik, T.: Microformats: a pragmatic path to the Semantic Web (Poster). In: Proceedings of the 15th International Conference on World Wide Web, pp. 865–866 (2006)
3. Kopecký, J., Vitvar, T., Bournez, C., Farrell, J.: SAWSDL: semantic annotations for WSDL and XML schema. IEEE Internet Computing **11**(6), 60–67 (2007)
4. Kopecký, J., Vitvar, T., Fensel, D.: WSMO-Lite: lightweight semantic descriptions for services on the Web. CMS WG Working Draft D11 (2009). Available at http://cms-wg.sti2.org/TR/d11/
5. Kopecký, J., Vitvar, T., Fensel, D., Gomadam, K.: hRESTS & MicroWSMO. CMS WG Working Draft D12 (2009). Available at http://cms-wg.sti2.org/TR/d12/
6. Semantic annotations for WSDL and XML schema. Recommendation, W3C (2007). Available at http://www.w3.org/TR/sawsdl/