WSMO-Lite Annotations for Web Services[⋆]

Tomas Vitvar[1], Jacek Kopecký[2], Jana Viskova[3], and Dieter Fensel[2]

[1] Digital Enterprise Research Institute (DERI),
National University of Ireland, Galway
{firstname.lastname}@deri.org

[2] Semantic Technology Institute (STI),
University of Innsbruck, Austria,
{firstname.lastname}@uibk.ac.at

[3] Department of Information Networks,
University of Zilina, Slovakia,
viskova@kis.fri.utc.sk

Abstract Current efforts in Semantic Web Services do not sufficiently address the industrial developments of SOA technology in regards to bottom-up modeling of services, that is, building incremental layers on top of existing service descriptions. An important step in this direction has been made in the W3C by the SAWSDL WG proposing a framework for annotating WSDL services with arbitrary semantic descriptions. We build on the SAWSDL layer and define WSMO-Lite service ontology, narrowing down the use of SAWSDL as an annotation mechanism for WSMO-Lite. Ultimately, our goal is to allow incremental steps on top of existing service descriptions, enhancing existing SOA capabilities with intelligent and automated integration.

1 Introduction

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

In 2007, the W3C finished its work on Semantic Annotations for WSDL and XML Schema (SAWSDL)[1]. SAWSDL defines simple extensions for WSDL and XML Schema used to link WSDL components with arbitrary semantic descriptions. It thus provides the grounds for a bottom-up approach to service modeling: it supports the idea of adding

[⋆] *This work is supported by the SFI Grant No. SFI/02/CE1/I131, and the EU project Knowledge Web (FP6–507482).*

[1] http://www.w3.org/TR/sawsdl/

small increments (and complexity) on top of WSDL, allowing to adopt results from various existing approaches. As the basis for bottom-up modeling, SAWSDL is independent of any particular semantic technology, i.e., it does not define any types, forms or languages for semantic descriptions.

In this paper we describe the WSMO-lite service ontology, created as a recent result of the community effort in the WSMO WG[2]. WSMO-Lite is the next evolutionary step after SAWSDL, filling the SAWSDL annotations with concrete semantic service descriptions and thus embodying the semantic layer of the Semantic Service Stack. With the ultimatate goal to support realistic real-world challenges in intelligent service integration, WSMO-Lite addresses the following requirements:

– Identify the types and a simple vocabulary for semantic descriptions of services (a service ontology) as well as languages used to define these descriptions;
– Define an annotation mechanism for WSDL using this service ontology;
– Provide the bridge between WSDL, SAWSDL and (existing) domain-specific ontologies such as classification schemas, domain ontology models, etc.

The rest of this paper is structured as follows. In Section 2, we introduce the Semantic Service Stack along with the state-of-the-art technologies for services and Semantic Web languages used in the stack. In Section 3, we describe the WSMO-Lite Service Ontology and summarize the resolution of the major points from its development. In Section 4, we describe the WSMO-Lite semantic annotations for WSDL, and in Section 5 we describe some relevant aspects for WSMO-Lite applications.

2 Semantic Service Stack

As depicted in Figure 1, there are two levels in the Semantic Service Stack, namely *semantic* and *non-semantic* level. In addition, there are two types of stakeholders in the stack, namely a *service engineer* (human being) and a *client* (software agent). The service engineer uses Web services through the client, with particular tasks such as service discovery, selection, mediation, composition and invocation. Through these tasks the client or service engineer (depending on the level of automation) decide whether to bind with the service or not. In order to faciliate such decisions, services should describe their offers using so called *service contracts*. The Semantic Service Stack adopts the following general types of service contracts:

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

[2] `http://www.wsmo.org`

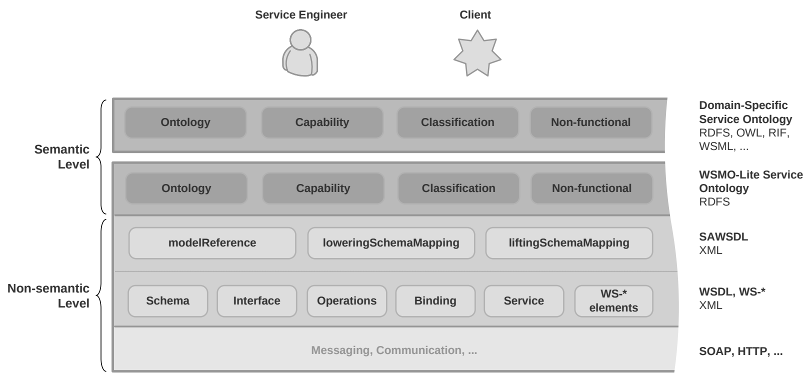

Figure 1. Semantic Service Stack

In the following sections, we show how the Semantic Service Stack represents the above general description types for service contracts at the two different levels.

2.1 Non-Semantic Level

In regard to SOA technology developments today, the Semantic Service Stack represents service contracts at the non-semantic level using the existing de-facto and de-jure standards: WSDL, SAWSDL, and related WS-* specifications. They all use XML as a common flexible data exchange format. Service contracts are represented as follows:

- *Information Model* is represented using XML Schema.
- *Functional Description* is represented using a WSDL Interface and its operations.
- *Non-Functional Description* is represented using various WS-* specifications, such as WS-Policy, WS-Reliability, WS-Security, etc.
- *Behavioral Description* is represented using the WS-* specifications of WS-BPEL[3] (for the workflow) and WS-CDL[4] (for the choreography).
- *Technical Description* is represented using WSDL Binding for message serializations and underlying communication protocols, such as SOAP, HTTP; and using WSDL Service for physical endpoint information.

In addition, while SAWSDL does not fall into any of the service contract descriptions, it is an essential part of the non-semantic level of the stack, providing the ground for the semantic layer. SAWSDL defines a simple extension layer that allows WSDL components to be annotated with semantics, using three extension attributes:

- *modelReference* for pointing to concepts that describe a WSDL component,
- *loweringSchemaMapping* and *liftingSchemaMapping* for specifying the mappings between the XML data and the semantic information model.

[3] `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`
[4] Choreography Description Language, `http://www.w3.org/TR/ws-cdl-10/`

2.2 Semantic Level

The Semantic Service Stack represents service contracts at the semantic level using the WSMO-Lite service ontology as follows (see Section 3 for a detailed description of WSMO-Lite):

- *Information Model* is represented using a domain ontology.
- *Functional Descriptions* are represented as *capabilities* and/or functionality *classifications*. A capability defines *conditions* which must hold in a state before a client can invoke the service, and *effects* which hold in a state after the service invocation. Classifications define the service functionality using some classification ontology (i.e., a hierarchy of categories).
- *Non-Functional Descriptions* are represented using an ontology, semantically representing some policy or other non-functional properties.
- *Behavioral Descriptions* are not represented explicitly in WSMO-Lite. In Section 5.1 we show how the public part of the behavioral description of a Web service may be derived from the functional descriptions of its operations.
- *Technical Descriptions* are not represented semantically in the service ontology, as they are sufficiently covered by the non-semantic description in WSDL.

In order to create or reuse domain-specific service ontologies on top of the Semantic Service Stack, a service engineer can use any W3C-compliant language with an RDF syntax. This preserves the choice of language expressivity according to domain-specific requirements. Such languages may include RDF Schema (RDFS), Web Ontology Language (OWL) [5], Rule Interchange Format (RIF)[5] or Web Service Modeling Language (WSML)[6] [10].

3 WSMO-Lite Service Ontology

Listing 1 shows the WSMO-Lite service ontology in RDFS, serialized in Notation 3[7]. Below, we explain the semantics of the WSMO-Lite elements:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix wl: <http://www.wsmo.org/ns/wsmo-lite#> .

wl:Ontology rdf:type rdfs:Class;
   rdfs:subClassOf owl:Ontology.
wl:ClassificationRoot rdfs:subClassOf rdfs:Class.
wl:NonFunctionalParameter rdf:type rdfs:Class.
wl:Condition rdfs:subClassOf wl:Axiom.
wl:Effect rdfs:subClassOf wl:Axiom.
wl:Axiom rdf:type rdfs:Class.
```

Listing 1. WSMO-Lite Service Ontology

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

[6] For details on compliance of WSML with W3C languages see [3].

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

– *wl:Ontology* (lines 6–7) defines a container for a collection of assertions about the information model of a service. Same as *owl:Ontology*, *wl:Ontology* allows for meta-data such as comments, version control and inclusion of other ontologies. *wl:Ontology* is a subclass of *owl:Ontology* since as we already mentioned, it has a special meaning of the ontology used as the service information model.
– *wl:ClassificationRoot* (line 8) marks a class that is a root of a classification which also includes all the RDFS subclasses of the root class. A classification (taxonomy) of service functionalities can be used for functional description of a service.
– *wl:NonFunctionalParameter* (line 9) specifies a placeholder for a concrete domain-specific non-functional property.
– *wl:Condition* and *wl:Effect* (lines 10–12) together form a *capability* in functional service description. They are both subclasses of a general *wl:Axiom* class through which a concrete language can be used to describe the logical expressions for conditions and effects. We illustrate this on an example in Listing 2 (lines 26–42).

Below, we describe the resolutions of major points that came up while WSMO-Lite was under development in the WSMO WG:

1. Relation of WSMO-Lite to WSMO. WSMO-Lite has been created due to a need for lightweight service ontology which would directly build on the newest W3C standards and allow bottom-up modeling of services. On the other hand, WSMO is an established framework for Semantic Web Services representing a top-down model identifying semantics useful in a semantics-first environment. WSMO-Lite adopts the WSMO model and makes its semantics lighter in the following major aspects:

– WSMO defines formal user goals and mediators, while WSMO-Lite treats mediators as infrastructure elements, and specifications for user goals as dependent on the particular discovery mechanism used. They both can be adopted in the running environment in combination with WSMO-Lite.
– WSMO-Lite only defines semantics for the information model, functional and non-functional descriptions (as WSMO Service does) and only implicit behavior semantics (see below). If needed, an application can extend WSMO-Lite with its own explicit behavioral descriptions, or it can adopt other existing technologies.
– While WSMO uses the WSML language for describing domain-specific semantic models, WSMO-Lite allows the use of any ontology language with an RDF syntax (see Section 2.2 for more details).

2. WSMO-Lite does not define explicit behavioral descriptions. WSMO-Lite treats Web services as atomic, and it does not deal with the internals of services. For this reason, it contains no construct for describing a private behavior (internal workflow) nor does it deal with annotations of existing WS-BPEL processes. Semantic annotation of processes is an independent research effort led by the business process community[8] and its use in combination with WSMO-Lite services is an open research question. Nevertheless, WSMO-Lite allows to derive some behavioral descriptions from functional (capability) annotations of services. See Section 5.1 for more details.

[8] More details on this issue can be found for instance at `http://www.ip-super.org`

3. Dependency of WSMO-Lite on SAWSDL. As we already mentioned, WSMO-Lite has been created to address the need for a concrete service ontology as the next evolutionary step after SAWSDL. For this reason it might seem that WSMO-Lite is also SAWSDL-dependent. However, WSMO-Lite uses SAWSDL only as an annotation mechanism for WSDL (see Section 4) while the WSMO-Lite service ontology can be used with any machine-readable service descriptions in combination with an appropriate annotation mechanism.

4. WSMO-Lite annotations for RESTful services. In this paper we define how WSMO-Lite can be used on top of WSDL and SAWSDL. RESTful services (i.e., resource-oriented services using plain HTTP as the communication protocol) can also be described in WSDL, which has support for describing Web services that do not use SOAP as the communication protocol. Such descriptions can also be annotated using WSMO-Lite. On the other hand, there are efforts in defining an annotation mechanism for RESTful services without WSDL, such as SA-REST [11]. This work is complementary to WSMO-Lite and we expect to integrate it in the W3C SWS-Testbed XG[9].

5. Concrete semantics for conditions and effects. To work with conditions and effects, it is necessary to define the environment in which these axioms are evaluated. Such an environment depends on the particular logical language in which the axioms are expressed. WSMO-Lite does not prescribe any concrete language for functional service semantics, and therefore it cannot define semantics for confitions and effects as they are language-dependent.

4 WSMO-Lite Annotations for WSDL

Section 2.2 mentions briefly how the WSMO-Lite ontology is used for the semantic descriptions. In this section, we formally define the particular types of annotations supported by WSMO-Lite.

4.1 Definitions

Ontology The fundamental building block for all types of semantic descriptions offered by WSMO-Lite is the ontology. We use a general definition of the ontology

$$\Omega = (C, R, E, I) \tag{1}$$

where the sets C, R, E, I in turn denote classes (unary predicates), relations (binary and higher-arity predicates[10]), explicit instances (extensional definition), and axioms (intensional definition) which describe how new instances are inferred.

A particular axiom common in I is the *subclass* relationship: if c_1 is subclass of c_2 (written as $c_1 \subset c_2$), every instance of c_1 is also an instance of c_2. We call this axiom out because it is necessary for Definition 2 below.

[9] `http://www.w3.org/2005/Incubator/swsc/`

[10] Note that a minimal definition would combine the sets of classes and relations as a set of predicates, but we choose to split them, due to familiarity and also reuse in further definitions.

We distinguish several sub-types of ontologies: we denote an information model ontology as $O_I \equiv \Omega$; a functionality classification ontology with root $r \in C$ as $O_F(r) \equiv \Omega$; and an ontology for non-functional descriptions as $O_N \equiv \Omega$.

Capability Functional description of a service as a capability is defined here as

$$K = (\Sigma, \phi^{pre}, \phi^{eff}), \tag{2}$$

where $\Sigma \subseteq (\{x\} \cup C \cup R \cup E)$ is the signature of symbols, i.e., identifiers of elements from C, R, E of some ontology O_I complemented with variable names $\{x\}$; ϕ^{pre} is a condition which must hold in a state before the service can be invoked, and ϕ^{eff} is the effect, a condition which must hold in a state after the successful invocation. Conditions and effects are defined as statements in logic $\mathcal{L}(\Sigma)$.

In Definition 1 below, we specify a *restriction* relationship (partial ordering $\leq$) between capabilities, and in Definition 2 we define an analogous relationship between categories in a functionality classification. Practically, if a capability/category K_1 is a restriction of another capability/category K_2, any discovery algorithm that discovers K_1 as a suitable capability/category for some goal would also discover K_2 as such.

Definition 1 (capability restriction) A capability $K_1 = (\Sigma, \phi_1^{pre}, \phi_1^{eff})$ is a restriction of $K_2 = (\Sigma, \phi_2^{pre}, \phi_2^{eff})$ (written as $K_1 \leq K_2$) if the condition ϕ_1^{pre} only holds in states (denoted as s) where also ϕ_2^{pre} holds, and if the same is true for the effects:

$$\begin{aligned} K_1 \leq K_2 \iff \forall s : (&holds(\phi_1^{pre}, s) \Rightarrow holds(\phi_2^{pre}, s)) \wedge \\ &(holds(\phi_1^{eff}, s) \Rightarrow holds(\phi_2^{eff}, s)) \end{aligned} \tag{3}$$

Definition 2 (category restriction) For two functionality categories K_1 and K_2 from classification $O_F(r)$, K_1 is a restriction of K_2 (written as $K_1 \leq K_2$) if $K_1 \subset K_2$:

$$K_1 \leq K_2 \iff K_1 \subset K_2 \tag{4}$$

WSDL We denote an XML Schema in WSDL as S, a WSDL interface as I and a service as W. Further, we denote $\{x\}_S$ as the set of all element declarations and type definitions of S, and $\{op\}_I$ as the set of all operations of I. Each operation $op \in \{op\}_I$ may have one input message element $m \in \{x\}_S$ and one output message element $n \in \{x\}_S$ and a corresponding MEP[11] denoted here as *op.mep*.

Annotations According to SAWSDL, we distinguish two types of annotations, namely *reference annotations* and *transformation annotations*. A reference annotation points from a WSDL component to a semantic concept. This is denoted as the binary relation $ref(x, s)$ where $x \in (\{x\}_S \cup \{I\} \cup \{op\}_I)$ — any WSDL or Schema component; $s \in (C \cup R \cup E \cup \{K\})$ — an ontology element or a capability. SAWSDL represents *ref* using *modelReference* extension attribute on the WSDL or XML Schema component).

A transformation annotation specifies a data transformation called *lifting* from a component of schema S to an element of ontology O_I; and a reverse transformation

[11] Message Exchange Pattern, `http://www.w3.org/TR/wsdl20-adjuncts/#meps`

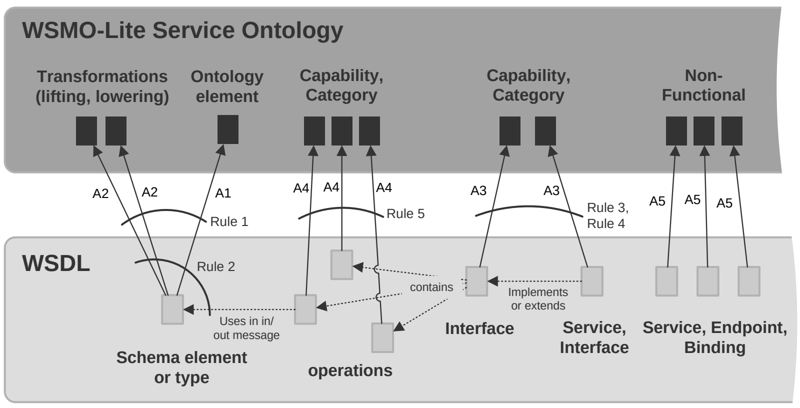

Figure 2. Illustration of Annotations and Rules

(from ontology to XML) called *lowering*. We denote these annotations as the binary relations *lower*$(m, f(c_1))$ and *lift*$(n, g(n))$, where $m, n \in \{x\}_S$. The function $f(c_1) = m$, where $c_1 \in (C \cup R)$, is a lowering function transforming data described semantically by c_1 to the XML message described by schema m (SAWSDL represents this annotation using *loweringSchemaMapping* extension attribute on m). Analogously, function $g(n) = c_2$, where $c_2 \in (C \cup R)$, is a lifting function transforming XML data from the message n to semantic data described by c_2 (SAWSDL represents this annotation using *liftingSchemaMapping* extension attribute on n).

4.2 Annotations and Rules

Figure 2 illustrates a set of annotations (marked *A1*...*A5*) and their associated rules (marked *Rule 1*...*Rule 5*). The rules have been refined from [13] to conform to the latest WSMO-Lite service ontology specification. The purpose of the rules is to ensure that the annotations are:

– *complete*, that is, no gaps are left in the semantic annotations, so that the client can see all the parts of the service description; for instance, all the operations should be semantically annotated so that they are reachable to automatic discovery.
– *consistent*, that is, no related annotations are contradictory; for instance the schema annotations by model reference need to point to concepts that are the outputs of the lifting schema mapping transformation, or inputs of the lowering one.

Listing 2 shows an example ontology we use to illustrate annotations. It defines a simple ontology for a telecommunication service (lines 9–24); the capability for a concrete Video on Demand subscription service (lines 26–39) (the condition says that the customer must have a network connection with some minimal bandwidth, the effect says that the customer is subscribed to the service); a non-functional property describing the pricing (lines 44–48); and a simple functionality classification (lines 50–53). We also define the *wsml:AxiomLiteral* data type (line 42) for WSML-Rule axioms so that a client can correctly process them according to the WSML specification.

```
// namespaces and prefixes
@prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
@prefix wl: <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#> .

// ontology example
<> rdf:type wl:Ontology.

ex:Customer rdf:type rdfs:Class .
ex:hasService rdf:type rdf:Property ;
    rdfs:domain ex:Customer ;
    rdfs:range ex:Service .
ex:Service rdf:type rdfs:Class .
ex:hasConnection rdf:type rdf:Property ;
    rdfs:domain ex:Customer ;
    rdfs:range ex:NetworkConnection .
ex:NetworkConnection rdf:type rdfs:Class .
ex:providesBandwidth rdf:type rdf:Property ;
    rdfs:domain ex:NetworkConnection ;
    rdfs:range xs:integer .
ex:VideoOnDemandService rdfs:subClassOf ex:Service .

// capability description example
ex:VideoOnDemandSubscriptionPrecondition rdf:type wl:Condition ;
    rdf:value "
        ?customer[hasConnection hasValue ?connection]
            memberOf Customer and
        ?connection[providesBandwidth hasValue ?y]
            memberOf NetworkConnection and
        ?y > 1000
    "^^wsml:AxiomLiteral .

ex:VideoOnDemandSubscriptionEffect rdf:type wl:Effect ;
    rdf:value "
        ?customer[hasService hasValue ?service]
    "^^wsml:AxiomLiteral .

// definition of the axiom for WSML language
wsml:AxiomLiteral rdf:type rdfs:Datatype .

// non−functional property example
ex:PriceSpecification rdfs:subClassOf wl:NonFunctionalParameter .
ex:VideoOnDemandPrice rdf:type ex:PriceSpecification ;
   ex:pricePerChange "30"^^ex:euroAmount ;
   ex:installationPrice "49"^^ex:euroAmount .

// classification example
ex:SubscriptionService rdf:type wl:ClassificationRoot .
ex:VideoSubscriptionService rdfs:subClassOf ex:SubscriptionService .
ex:NewsSubscriptionService rdfs:subClassOf ex:SubscriptionService .
```

Listing 2. Example of domain-specific service ontology

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

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

```
<xs:element name="NetworkConnection" type="NetworkConnectionType"
   sawsdl:modelReference="http://example.org/onto#NetworkConnection"
   sawsdl:loweringSchemaMapping="http://example.org/NetCn.xslt"/>
```

Listing 3. Example of annotations A1 and A2

Listing 3 shows an example of annotations *A1* and *A2* (the lowering transformation is omitted for brevity). Below, Rule 1 defines consistency of *A1* and *A2* annotations on schema components; Rule 2 defines completeness of these annotations on element declarations used as operation input and output messages.

Rule 1 (consistency) Let S be a schema, and O_I be an ontology. If for any $m \in \{x\}_S$ there exist the annotations $ref(m,c_1)$ (*A1*) and $lower(m, f(c_1))$ (*A2*), then it must hold that $f(c_1) = m$. Analogously, if for any $n \in \{x\}_S$ there exist the annotations $ref(n,c_2)$ (*A1*) and $lift(n, g(n))$ (*A2*), then it must hold that $g(n) = c_2$.

Rule 2 (completeness) Let S be a schema, and I be an interface. For each $m \in \{x\}_S$ where m is the input message element of any operation in $\{op\}_I$, the element must have consistent annotations $ref(m,c_1)$ (*A1*) and $lower(m, f(c_1))$ (*A2*). Analogously, for each $n \in \{x\}_S$ where n is the output message element of any operation in $\{op\}_I$, the element must have consistent annotations $ref(n,c_2)$ (*A1*) and $lift(n, g(n))$ (*A2*).

A3: Annotations of WSDL Interface and Service (functional). Functional descriptions (both capabilities and categories) apply both to concrete web services and to the reusable and abstract interfaces. A reference annotation points from a service or an interface to its appropriate functional description. Listing 4 shows an example of multiple *A3* annotations:

```
<wsdl:interface name="NetworkSubscription"
      sawsdl:modelReference="http://example.org/onto#VideoSubscriptionService
                              http://example.org/onto#VideoOnDemandSubscriptionPrecondition
                              http://example.org/onto#VideoOnDemandSubscriptionEffect" >
```

Listing 4. Example of annotations A3

Please note that a WSDL interface may be shared by multiple services, therefore the functional description of the interface should be general. A concrete functional description attached to the service then refines the functional description of the interface. Additionally, aggregate interfaces or services (i.e., those that combine multiple potentially independent functionalities) may be annotated with multiple functional descriptions. Rule 3 defines consistency for *A3* annotations: each functionality of a service must be a restriction of some functionality of the service's interface (see Definition 1 and Definition 2). This allows discovery to first find appropriate interfaces and then only check services that implement these interfaces. Rule 4 is analogous to Rule 3 with the difference that it applies to interface extension[12], ensuring that functionality cannot be lost through WSDL interface extension.

Rule 3 (consistency) Let W be a service and I be an interface such that W implements I. Then, for each annotation $ref(W,F)$ (*A3*) there must exist an annotation $ref(I,G)$ (*A3*) such that $F \leq G$.

[12] Interface extension is a feature of WSDL 2.0.

Rule 4 (consistency) Let I and J be some interfaces such that I extends J. Then, for each annotation $ref(I,F)$ (*A3*) there must exist an annotation $ref(J,G)$ (*A3*) such that $G \leq F$.

A4: Annotations of WSDL Interface operations (functional). Functional descriptions (both capabilities and categories) apply also to interface operations, to indicate their particular functionalities. A reference annotation points from an operation to its appropriate functional description.

Functional annotation of interface operations can be used for services whose interfaces are simply collections of standalone operations. For example, a network subscription service may offer independent operations for subscription to a bundle, cancellation of a subscription, or price inquiry. A client will generally only want to use one or two of these operations, not all three. This shows that service discovery can, in such cases, become operation discovery. Also, operation annotations can be used for defining the order in which the operations should be invoked (see Section 5.1).

Rule 5 defines completeness for *A4* annotations: all operations within an interface must be annotated with a functional description. This rule ensures that no operation is left invisible to the automated clients.

Rule 5 (completeness) For all $o \in \{op\}_I$ there must exist some functional description F (capability or category) such that $ref(o,F)$ is defined.

Please note that annotations *A3* and *A4* apply to both types of functional descriptions, i.e., a capability or a category from some functional classification. It is even possible to combine them for a service, interface and its operations.

A5: Annotations of WSDL Service, Endpoints, and Binding (non-functional). Non-functional descriptions apply to a concrete instance of a Web service, that is, a Service, its Endpoints, or its Binding. A reference annotation can point from any of these components to a non-functional property. Listing 5 shows an example of annotation *A5*:

```
<wsdl:service name="ExampleCommLtd"
      interface="NetworkSubscription"
      sawsdl:modelReference="http://example.org/onto#VideoOnDemandPrice">
   <wsdl:endpoint ...
</wsdl:service>
```

Listing 5. Example of annotation A5

Please note that non-functional descriptions are always specific to a concrete service, therefore, annotating interfaces or interface operations with non-functional properties is not defined. In case non-functional properties need to be specified on the operations (for example, different operations may have different invocation micropayment prices), a WSDL binding operation components (which mirror the operations of some interface) may be used to capture these properties. Due to the domain-specific nature of non-functional properties, WSMO-Lite cannot formulate any consistency or completeness rules for non-functional descriptions.

5 On Top of WSMO-Lite Annotations

WSMO-Lite annotations for Web services allow additional tasks on top: in particular we show implicit representation of a service choreography and illustrate the overall use of WSMO-Lite annotations for various SWS tasks essential for the client's automated decisions about services.

5.1 Implicit Choreography

In this section we show how WSMO-Lite interface operation annotations implicitly represent a choreography, understood according to [9] as a protocol from a single service's point of view[13], and formalized as an Abstract State Machine (ASM, [2]) as

$$X = (\Sigma, L) \tag{5}$$

where $\Sigma \subseteq (\{x\} \cup C \cup R \cup E)$ is the signature of symbols, i.e., variable names $\{x\}$ or identifiers of elements from C, R, E of some ontology O_I; and L is a set of rules. Further, we denote by Σ_I and Σ_O the input and output symbols of the choreography (subsets of $C \cup R \cup E$), corresponding to the input data sent to the service and the returned output data. Each rule $r \in L$ defines a state transition $r : r^{cond} \rightarrow r^{eff}$ where r^{cond} is an expression in logic $\mathcal{L}(\Sigma)$ which must hold in a state before the transition is executed; r^{eff} is an expression in logic $\mathcal{L}(\Sigma)$ describing a condition which holds in a state after the execution. And finally, we use ontology elements as conditions (as in $c_1 \in O_I : c_1 \wedge \phi^{pre}$ within the algorithm), by which we mean that there exists an entity in the knowledge base which fits the description of the ontology element; for example, if the ontology element c_1 is a class, the knowledge base contains an instance of this class.

We construct the choreography from capability annotations of interface operations, according to the following algorithm.

Inputs:

- An interface I with operations $\{op\}_I$, ontology O_I, and a set of capabilities $\{K\}$.
- Complete *A4* annotations using only capabilities from $\{K\}$ for all $\{op\}_I$.
- Consistent and complete *A1* and *A2* annotations using O_I for all input and output messages of operations $\{op\}_I$.

Output:

- Choreography X with Σ_I, Σ_O and L.

Algorithm:

```
1: for all $ref(op,K)$, $op \in \{op\}_I$, $K = (\phi^{pre}, \phi^{eff}) \in \{K\}$ do
2:   get $ref(m,c_1)$ where $m$ is the input message of $op$, $c_1 \in O_I$; add $c_1$ to $\Sigma_I$.
3:   get $ref(n,c_2)$ where $n$ is the output message of $op$, $c_2 \in O_I$; add $c_2$ to $\Sigma_O$.
4:   if $op.mep$ in $\{$in-out, in-only, out-only$\}$ then
5:     create the rule $r$: $r^{cond} = c_1 \wedge \phi^{pre}$, $r^{eff} = c_2 \wedge \phi^{eff}$; add $r$ to $L$.
6:   else if $op.mep$ in $\{$out-in$\}$ then
7:     create the rule $r_1$: $r_1^{cond} = \phi^{pre}$; $r_1^{eff} = c_2$; add $r_1$ to $L$.
```

[13] WS-CDL defines a different type of a choreography, i.e., as a common behavior of collaborating parties. The relationship of WSMO-Lite to WS-CDL is an open research question.

```
 8:      create the rule $r_2$: $r_2^{cond} = c_1 \wedge c_2$; $r_2^{eff} = \phi^{eff}$; add $r_2$ to $L$.
 9:    end if
10: end for
```

The algorithm creates the sets of choreography input and output symbols from the semantic representations of the input and output messages of all the operations (lines 2–3). In addition, it creates choreography rules where the conditions contain assertions about the input messages and the effects contain assertions about output messages of operations. The algorithm creates one rule for operations with the *in-out*, *in-only* or *out-only* MEPs (lines 4–5). Since the ASM rules always represent an *in-out* interaction, two rules need to be created for operations with the *out-in* MEP: one representing the output and one the following input interaction. In order to further illustrate the results of the algorithm, Table 1 shows the resulting rules for the four MEPs (please note, that we do not currently cover fault messages). Here, a transition rule $r^{cond} \rightarrow r^{eff}$ is represented as `if` r^{cond} `then` r^{eff}; the symbols `msg1...msg6` refer to schema elements used for input/output messages of operations; the symbols $c_1 \ldots c_6$ refer to identifiers of semantic descriptions of these messages; $ref(m,c)$ denotes the *A1* annotation, `w:` is a shortening for the URI `http://www.w3.org/ns/wsdl/` and `ex:` for some application URI `http://example.org/onto#`.

MEP and Rule	WSDL Operation

in-out:
if $c_1 \wedge$`cnd1` then $c_2 \wedge$`eff1`
$c_1 \in \Sigma_I$, *ref*(`msg1`, c_1)
$c_2 \in \Sigma_O$, *ref*(`msg2`, c_2)
	`<operation name="op1" pattern="w:in-out"`
	`    sawsdl:modelReference="ex:cnd1 ex:eff1">`
	`  <input  element="msg1"/>`
	`  <output element="msg2"/>`
	`</operation>`

in-only:
if $c_3 \wedge$`cnd2` then `eff2`
$c_3 \in \Sigma_I$, *ref*(`msg3`, c_3)
	`<operation name="op2" pattern="w:in-only">`
	`    sawsdl:modelReference="ex:cnd2 ex:eff2">`
	`  <input  element="msg3"/>`
	`</operation>`

out-only:
if `cnd3` then $c_4 \wedge$`eff3`
$c_4 \in \Sigma_O$, *ref*(`msg4`, c_4)
	`<operation name="op3" pattern="w:out-only">`
	`    sawsdl:modelReference="ex:cnd3 ex:eff3">`
	`  <output element="msg4"/>`
	`</operation>`

out-in:
if `cnd4` then c_5
if $c_5 \wedge c_6$ then `eff4`
$c_5 \in \Sigma_O$, *ref*(`msg5`, c_5)
$c_6 \in \Sigma_I$, *ref*(`msg6`, c_6)
	`<operation name="op4" pattern="w:out-in">`
	`    sawsdl:modelReference="ex:cnd4 ex:eff4">`
	`  <output element="msg5"/>`
	`  <input  element="msg6"/>`
	`</operation>`

Table 1. MEPs, Rules and WSDL operations

With a choreography constructed according to this algorithm, the client is able to automatically invoke a service, i.e., its operations in the correct and expected order.

5.2 Service Use Tasks

In this section, we discuss the use of the different types of WSMO-Lite annotations for automation of the various tasks that the client may perform with services. Not all annotations described in Section 4.2 are always needed, only those required by the tasks at hand in a particular domain-specific setting. Table 2 provides a summary, with *A1*. . . *A5* denoting the annotations and *R1*. . . *R5* denoting the rules. The symbol • marks the annotations and rules required to automate a given task, and the symbol ○ marks rules that are helpful but not absolutely required.

Service Task	**A1**	**A2**	**A3**	**A4**	**A5**	**R1**	**R2**	**R3**	**R4**	**R5**
Service Discovery			•					○	○	
Operation Discovery				•						○
Composition			•							
Ranking and Selection					•					
Operation Invocation	•	•				•	•			
Service Invocation	•	•		•		•	•			○
Data Mediation	•	•				•				
Process Mediation	•	•		•		•	•			○

Table 2. Service Tasks, Annotations and Rules

- *Service Discovery*, operating on functional descriptions (capabilities or categories), requires annotations *A3*. Rule 3 and Rule 4 help improve the scalability of the discovery through narrowing down a set of interfaces and services to be searched. If the discovery mechanism determines that an interface is not suitable, all the services implementing it and all the interfaces extended by it can immediately be discarded from further consideration.
- *Operation Discovery*, operating on functional descriptions of individual operations, requires annotations *A4*. Operation discovery might be useful with interfaces that are collections of standalone, independent operations. Rule 5 ensures that no operation is left invisible to this discovery process.
- *Composition* uses capability descriptions, i.e., annotations *A3* restricted to capabilities, to put together multiple services to achieve a complex goal.
- *Ranking and Selection* processes non-functional descriptions, i.e., annotations *A5*, to select the service that most suits some particular requirements.
- *Operation Invocation* is the invocation of a single operation, requiring data transformations between the semantic model on the client and the service's XML message structure. This requires *A1* and *A2* annotation, kept consistent by Rule 1. Rule 2 ensures that all operation messages have these annotations.
- *Service Invocation* requires the operations of the service to be invoked in a proper order. This task therefore uses the implicit interface choreography (Section 5.1) and requires annotations *A4*. Rule 5 ensures that no operation is omitted from the choreography.
- *Data Mediation* uses data annotations (*A1* and *A2*) — assuming two different schemas correspond to a single shared ontology, the *A1* annotations make it possible to discover such a correspondence, and the *A2* annotations then enable data mapping transformations: lifting from one schema and lowering to the other.
- *Process Mediation* combines data mediation and choreography processing and thus requires the combined annotations *A1*, *A2* and *A4* . As described in [4], process mediation is applied during conversation between two services mediating their choreographies and messages.

This provides certain modularity to WSMO-Lite, enabling different environments using this service ontology to mix and match the annotations as necessary for the required tasks. On top of already being light-weight, WSMO-Lite provides value even if only parts of it are used.

6 Related work

The major stream of related work is in the frameworks for Semantic Web Services (SWS), including WSMO [10], Semantic Markup for Web Services (OWL-S [12]) and Web Service Semantics (WSDL-S [1]). WSMO is a top-down conceptual model for SWS that defines four top-level components: ontologies, mediators, goals and web services. As we already mentioned, WSMO was the major input for WSMO-Lite. On the other hand, OWL-S was the first major ontology for SWS defining three interlinked ontologies: Service Profile (for the functional and non-functional descriptions), Service Model (for the behavioral descriptions), and Service Grounding (for physical Web service access). There are also recent works on OWL-S grounding that uses SAWSDL [8,7]. In comparison with that work, WSMO-Lite takes the additional step of simplifying the annotations into a lightweight ontology. WSDL-S was created in the METEOR-S[14] project as a specification of how WSDL can be annotated with semantic information. WSDL-S itself does not provide a concrete model for SWS, instead it makes the assumption that the concrete model will be expressible as annotations in WSDL and XML Schema documents. WSDL-S was taken as the basis for SAWSDL. In addition, there is a major orthogonal work to WSMO-Lite called SA-REST [11], aiming to enrich the informal descriptions of RESTful services, usually available in HTML, with RDFa[15] annotations. This work is complementary to WSMO-Lite as it could serve as an additional annotation mechanism for WSMO-Lite service ontology used for RESTful services. We will work on such integration withing the W3C SWS-Testbed XG.

7 Conclusion and Future Work

In this paper, we describe the latest results from the development of WSMO-Lite, a minimal lightweight ontology for Semantic Web Services, building on the newest W3C standards. WSMO-Lite fills in SAWSDL annotations, and thus enables the Semantic Service Stack, open for various customizations according to domain-specific requirements, languages of required expressivity and domain-specific ontologies. WSMO-Lite supports the idea of incremental enhancements of SAWSDL as Amit Sheth points out in [6]: "Rather than look for a clear winner among various SWS approaches, I believe that in the post-SAWSDL context, significant contributions by each of the major approaches will likely influence how we incrementally enhance SAWSDL. Incrementally adding features (and hence complexity) when it makes sense, by borrowing from approaches offered by various researchers, will raise the chance that SAWSDL can present itself as the primary option for using semantics for real-world and industry-strength challenges involving Web services."

[14] `http://lsdis.cs.uga.edu/projects/meteor-s/`

[15] `http://www.w3.org/TR/xhtml-rdfa-primer/`

In our future work we plan to work on validation of WSMO-Lite annotations, together with a compiler for WSMO-Lite descriptions. We also plan to integrate WSMO-Lite with other research efforts within the W3C SWS-Testbed XG (such as already mentioned SA-REST) and to support service mashups with the WSMO-Lite ontology. In addition, we plan to integrate the WSMO-Lite ontology with the results of the semantic business processes research.

References

1. R. Akkiraju, *et al.* Web Service Semantics - WSDL-S, available at http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/. Tech. rep., LSDIS Lab, 2005.
2. E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.
3. J. de Bruijn, D. Fensel, and H. Lausen. D34v0.1: The Web Compliance of WSML. Technical report, DERI, 2007. Available from: `http://www.wsmo.org/TR/d34/v0.1/`.
4. T. Haselwanter, *et al.* WSMX: A Semantic Service Oriented Middleware for B2B Integration. In *ICSOC*, pp. 477–483. 2006.
5. I. Horrocks. Owl: A description logic based ontology language. In *CP*, pp. 5–8. 2005.
6. D. Martin and J. Domingue. Semantic web services: Past, present and possible futures (systems trends and controversies). *IEEE Intelligent Systems*, 22(6), 2007.
7. D. Martin, M. Paolucci, and M. Wagner. Bringing Semantic Annotations to Web Services: OWL-S from the SAWSDL Perspective. In K. Aberer, *et al.*, (eds.) *The Semantic Web*, vol. 4825 of *LNCS*, pp. 340–352. Springer, 2007.
8. M. Paolucci, M. Wagner, and D. Martin. Grounding OWL-S in SAWSDL. In B. J. Krämer, K.-J. Lin, and P. Narasimhan, (eds.) *ICSOC*, vol. 4749 of *Lecture Notes in Computer Science*, pp. 416–421. Springer, 2007.
9. D. Roman and J. Scicluna. Ontology-based choreography of wsmo services. Wsmo d14 final draft v0.3, DERI, 2006. Available at: `http://www.wsmo.org/TR/d14/v0.3/`.
10. D. Roman, *et al.* Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.
11. A. P. Sheth, K. Gomadam, and J. Lathem. SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. *IEEE Internet Computing*, 11(6):91–94, 2007.
12. The OWL Services Coalition. OWL-S 1.1 Release, November 2004. Available at http://www.daml.org/services/owl-s/1.1/.
13. T. Vitvar, J. Kopecky, and D. Fensel. WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web. In *ECOWS*. 2007.