WSMO Deliverable

D11v0.2

WSMO-LITE: LIGHTWEIGHT SEMANTIC DESCRIPTIONS FOR SERVICES ON THE WEB

WSMO Working Draft – 22nd June 2007

Authors:
Tomas Vitvar
Jacek Kopecký
Dieter Fensel

Editors:
Tomas Vitvar
Jacek Kopecký

This version:
`http://www.wsmo.org/TR/d11/v0.2/20070622/`

Latest version:
`http://www.wsmo.org/TR/d11/v0.2/`

Previous version:
`http://www.wsmo.org/TR/d11/v0.2/20070421/`

Abstract

The current Web service technology brought a new potential to the Web of services. However, the success of Web services still depends on resolving three fundamental challenges, namely search, integration and mediation. In this deliverable we define an extended Web service stack enabling total or partial automation of web service provisioning process. With the goal of a maximal Web standards compliance, we describe various types of service semantics, use RDF Schema (RDFS) to define a pragmatic meaning for those descriptions, and use Semantic Annotations for WSDL and XML Schema (SAWSDL) to define a place for a semantic description in a Web service. We elaborate on the existing SAWSDL specifications and define precise rules for semantic annotations of Web services.

Contents

1 Introduction

Web services add a new level of functionality to the Web, stepping towards an open environment of distributed applications. While current Web service technology around SOAP, WSDL and UDDI brought a new potential to the Web of services, it will prove difficult to scale without a proper degree of automation. In large-scale, open, and heterogeneous environments, the success of Web services depends on resolving the fundamental challenges that existing integration technologies do not address, namely search, integration and mediation. Depending on XML descriptions, this technology only offers manual support for integration which usually operates on rigid configuration of workflows or services. Although flexible and extensible, XML only defines the structure and syntax of data limiting the possibilities for automation.

In order to address this drawback, we define an *extended Web service specification stack*, adding semantic layers which offer richer descriptions for Web services. Such richer descriptions enable partial or total automation of tasks including service discovery, composition, selection, negotiation, mediation and invocation. With the goal of a maximal Web standards compliance, we describe various types of service semantics, then we use RDF Schema (RDFS)[1] to define a pragmatic realization for those descriptions, and finally we use Semantic Annotations for WSDL and XML Schema (SAWSDL)[2] to define a place for the semantic descriptions of a Web service.

In this deliverable we use the Web Service Modeling Language (WSML) [4] as a certain style to express the semantic descriptions and also the language for capturing logical expressions. It is important to note that our goal is not to define yet another Web syntax for logics but to show how existing logic languages can be used to describe service semantics while at the same time preserving maximal compliance with Web standards. WSML is a language that can be used for that purpose. We want to allow the domain experts to define various types of service descriptions compliant with RDFS and to choose the types and expressivity for those descriptions according to specific domain requirements. A domain expert might consider the proper knowledge representation paradigm together with the proper logical language to serve the specific domain requirements, as well as the required tasks and the required reasoning for those tasks that should be performed with the services.

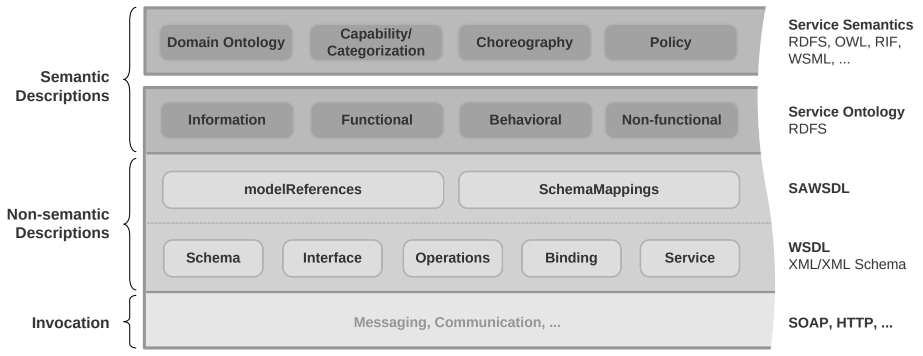

Figure 1: Extended Web Service Specification Stack

Figure 1 depicts the extended Web service specification stack, showing the standard specifications, Web languages and semantic extensions. In Section 2 we describe the underlying Web languages used for the non-semantic and semantic descriptions in the stack, as well as the underlying Web service standards. In Section 3 we define the various types of semantic descriptions and the service ontology for those descriptions. In Section 4 we specify how these semantic descriptions are linked with the non-semantic descriptions. In Section 5 we talk about related work and in Section 6 we conclude the deliverable and describe our future plans.

[1]`http://www.w3.org/TR/rdf-schema/`
[2]`http://www.w3c.org/ws/sawsdl`

2 Underlying Specifications

In this section, we briefly describe the underlying technologies for our work on the extended Web service specification stack depicted in Figure 1. We first describe the Semantic Web languages used for the semantic descriptions in the stack, and then the existing standards of Web services as the underlying layers of the stack that we later extend with semantic descriptions.

2.1 Semantic Web Languages

The W3C has produced several language recommendations for representation and exchange of knowledge on the Semantic Web. At the core, the Resource Description Framework (RDF)[3] represents information in graph-based models with so called *triples*, i.e. statements in the form ⟨subject, predicate, object⟩. The subjects and objects link the triples into a graph. RDF provides various syntaxes including RDF/XML[4] and Notation 3 (N3)[5]. RDF Schema (RDFS) defines constructs on top of RDF that allow the specification of lightweight ontologies: RDFS allows to define classes, properties as well as class and property hierarchies. Forming additional layers of expressivity on the top of RDF(S), the Web Ontology Language (OWL) [8] provides further vocabulary along with a formalism based on Description Logics. Last but not least, an ongoing effort aims to extend the existing languages with rules. In particular, the W3C Rule Interchange Format Working Group (RIF WG)[6] aims to produce a core rule language for the Semantic Web together with extensions that allow rules to be translated between different rule languages.

There are also several languages outside the W3C. For instance, WSML is a family of ontology languages compatible in many ways with the W3C recommendations and their underlying principles. WSML defines several variants which differ in the logical expressiveness and in the underlying language paradigms. The basic WSML variant called WSML-Core provides limited expressive power of language elements available in both Description Logics and Rule languages. WSML-Core is extended in two directions: Description Logics (WSML-DL) and Logic Programming (WSML-Flight, WSML-Rule). Some WSML variants (i.e. WSML-Core and WSML-DL) have direct mapping to OWL. In addition, WSML-Rule is the basis of the Web Rule Language (WRL)[7] specification which serves as an input for the RIF WG. Thus, RIF can be expected to be compatible with WSML-Rule, at least to a large extent. The detailed description of WSML and its compliance with standards can be found in [3].

2.2 Web Services

The Web Service Description Language (WSDL)[8] provides a standard description format for Web services, using XML as a common flexible data exchange format, and applying XML Schema for data typing. WSDL describes a Web service in three levels: 1) an XML-based reusable *abstract interface*, and the concrete details regarding 2) how and 3) where this interface can be accessed. The interface defines a set of operations, each representing a simple exchange of messages that follows a specific message exchange pattern (MEP) (e.g., ”In-Out”, where an input message is followed by an output message or a fault). Messages in operations reference XML Schema element declarations to describe their contents. In order to communicate with a Web service described by an abstract interface, a client must know how the XML messages are serialized on

[3] `http://www.w3.org/RDF/`
[4] `http://www.w3.org/TR/rdf-syntax-grammar/`
[5] `http://www.w3.org/DesignIssues/Notation3.html`
[6] `http://www.w3.org/2005/rules/`
[7] `http://www.w3.org/Submission/2005/08/`
[8] `http://w3.org/TR/wsdl20`

the network and where exactly they should be sent. In WSDL, on-the-wire message serialization is described in a *binding*, which generally follows the structure of an interface and specifies the necessary serialization details. WSDL predefines two binding types, one for SOAP (over HTTP) and one for plain HTTP. Finally, the *service* construct in WSDL represents a single physical Web service that implements a single interface. The Web service can be accessible at multiple *endpoints*, each potentially with a different binding: for example, one endpoint using an optimized messaging protocol with no data encryption for the secure environment of an intranet and a second endpoint using SOAP over HTTPS for access from the Internet.

While the purpose of WSDL is to describe the Web service on a *syntactic level* (WSDL specifies what the messages *look like* rather then what the messages or operations *mean*), the specification called Semantic Annotations for WSDL and XML Schema (SAWSDL) defines a simple extension layer over WSDL that allows the *semantics* to be specified on 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. Semantic annotations can be used, for instance, for associating WSDL interfaces with some taxonomical categories to help Web service discovery; for describing the purpose or applicability of WSDL operations to help discovery or composition; or for linking and mapping inputs, outputs and faults of WSDL operations to semantic concepts to help facilitate mediation, service discovery, composition and invocation.

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 a basis for our lightweight ontology for Semantic Web Services, working towards automation in service-oriented systems.

3 Semantics for Web Services

The major goal of adding semantics to web services is to increase automation of certain tasks which need to be performed with services before or during the invocation. Based on various efforts in Semantic Web Services and Service-Oriented Computing communities (e.g., [1, 6, 12, 13, 14, 17]), there are generally accepted types of semantic descriptions of *information*, *functional*, *non-functional*, and *behavioral* aspects of services, as well as general tasks of *discovery*, *negotiation*, *selection*, *composition*, *mediation* and *invocation*. The tasks are performed by a *semantic client*, i.e. a service requester or a middleware system both performing various combinations of tasks according to the requirements of a particular application. In this deliverable we use the term semantic client (or client) with no further definition, an interested reader can refer to e.g. [17] for more information about an intelligent middleware system for the Semantic Web Services.

In addition, different semantic descriptions are needed for different tasks. In the following paragraphs we briefly discuss each of the semantic descriptions and the related tasks that use those descriptions. In the subsequent sections of this deliverable we provide more elaborated definition for each of the semantic descriptions.

- **Information Semantics** is a semantic description of an information model (ontology together with instance data). This type of semantics is used by other descriptions and is usually needed when performing data mediation through ontology merging or mapping/aligning.

- **Functional Semantics** is a static description of the service capability, i.e. what the service can offer to its users. This type of semantics is usually required by service discovery comparing a need of a user (i.e. in the form of a formal goal) and the functional descriptions or classifications of services. Also, the capability is needed for composition when creating a plan for a given goal.

- **Non-Functional Semantics** is the description of additional constraints over service functionality not captured by its functional description; usually it is some policy. Non-functional semantics is needed to match discovered services against the preferences and constraints of the user.
- **Behavioral Semantics** is a description of a public and a private behavior of a service. For our work we only use the public behavior (called choreography) as a description of a protocol which must be followed by each client in order to consume the service's functionality. It is used during invocation of the service and negotiation with it. However, when the client uses a different choreography from the one defined by the service, the conflicts between both choreographies need to be resolved. This task is called a process mediation (see [2]) which operates on both choreography descriptions. Also, during discovery or composition, a compatibility check of client's and service's choreographies can be performed.

In this section we define the above mentioned types of service descriptions and use RDFS to model them as part of a *service ontology*. In particular, we define modeling elements based on RDFS for each type of the service description. We will show examples of the service ontology in Notation 3 and use the namespaces and their prefixes as shown in Listing 1.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix xs: <http://www.w3.org/2001/XMLSchema#> .

@prefix wl: <http://www.wsmo.org/wsmo-lite#> .
@prefix ex: <http://example.org/onto#> .
@prefix wsml: <http://www.wsmo.org/wsml-rdf-syntax#> .
```

Listing 1: Namespace declarations

3.1 Information Semantics

Information semantics is the formal definition of some domain knowledge used by the service in its *input* and *output* messages. We describe the information semantics as an ontology defining the terminology of the domain together with a knowledge base as the instantiation of the ontology. Formally, the information semantics is a structure

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

with a set of classes (unary predicates) C, a set of relations[9] (binary and higher-arity predicates) R, a set of explicit instances of C and R called E (extensional definition), and a set of axioms called I (intensional definition) that describe how new instances are inferred.

We use basic RDFS terms to express ontology, but other ontology languages (e.g. OWL, WSML) can also be used, especially to express domain ontologies. In fact, for some of the semantics detailed further in this deliverable we require a language that can express logical conditions. Table 1 shows the representation of information semantics in RDF and RDFS. Note that symbols such as c, r_1 etc. on the left-hand side are translated into URIs `c` and `r1` etc. on the right-hand side using a bijective naming function $N : symbol \rightarrow uri$. For instance instead of `r1` we could write $N(r_1)$, but we chose the former for readability.

[9]Note that the 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.

Table 1: Information semantics in RDFS

Information semantics construct	RDFS triples
$c \in C$	`c rdf:type rdfs:Class`
$c \in C \wedge c(e) \in E$	`e rdf:type c`
$r \in R$ r is a binary predicate	`r rdf:type rdf:Property`
$r \in R \wedge r(a,b) \in E$	`a r b`

$r \in R$
r is an n-ary predicate with parameters $r_1 \dots r_n$
	`r  rdf:type rdfs:Class`
	`r1 rdf:type rdf:Property`
	$\vdots$
	`rn rdf:type rdf:Property`

$r \in R \wedge r(a_1, \dots, a_n) \in E$	`_:x rdf:type r`
	`_:x r1 a1`
	$\vdots$
	`_:x rn an`

$(\forall a, \forall b : r(a,b) \Rightarrow c(a)) \in I$	`r rdfs:domain c`
$(\forall a, \forall b : r(a,b) \Rightarrow c(b)) \in I$	`r rdfs:range c`
$(\forall a : c_1(a) \Rightarrow c_2(a)) \in I$	`c1 rdfs:subClassOf c2`
$(\forall a, \forall b : r_1(a,b) \Rightarrow r_2(a,b)) \in I$	`r1 rdfs:subPropertyOf r2`
Other axioms are expressed in some rule language	

```
1   ...
2   ex:Bundle rdf:type rdfs:Class .
3   ex:NetworkConnection rdf:type rdfs:Class .
4   ex:Service rdf:type rdfs:Class .
5   ex:hasService rdf:type rdf:Property ;
6       rdfs:domain ex:Bundle ;
7       rdfs:range ex:Service .
8   ex:hasConnection rdf:type rdf:Property ;
9       rdfs:domain ex:Bundle ;
10      rdfs:range ex:NetworkConnection .
11  ex:providesBandwidth rdf:type rdf:Property ;
12      rdfs:domain ex:NetworkConnection ;
13      rdfs:range xs:integer .
14  ex:DSLConnection rdf:type rdfs:Class ;
15      rdfs:subClassOf ex:NetworkConnection .
16  ...
```

Listing 2: Example ontology

Our information semantics definition allows predicates with arity higher than two. RDFS only defines classes (unary predicates) and properties (binary predicates). For the higher-arity predicates, it is a common style to represent an n-ary predicate as a class, with attributes (properties with pre-set domain) representing the n parameters.

In Listing 2 we show a simple domain ontology in RDFS that describes the semantics of information needed for a telecommunication service. The *NetworkConnection* (line 3) stands for the class of all network connections which can be put in a hierarchy by means of *rdfs:subClassOf* predicate (lines 10–11). Classes can have properties, such as *Bundle* has a property *hasConnection* that points to the network connection which is part of the bundle. In some cases, it is required to model complex axiomatic information as part of the ontology for description of implicit knowledge. This requires a proper language that allows to define arbitrary complex logical expressions, e.g. RIF, WSML-Rule, etc.

3.2 Functional Semantics

Functional semantics is the formal description of service functionality, describing what a service can offer to its clients when it is invoked. We distinguish two types of service functionality description: (1) *capability* — the functionality defined using

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

Listing 3: Service Ontology: Functional Semantics Constructs

conditions which must hold before and after service invocation, and (2) *categorization* – the functionality defined using some classification schema[10], e.g. the United Nations Standard Products and Services Code (UNSPSC)[11], RosettaNet Technical Dictionary (RNTD)[12] etc. While a classification can be described as an ontology according to Equation 1, a capability is defined here as

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

where $\Sigma \subseteq (\{x\} \cup C \cup R \cup E)$ is is the signature of symbols, i.e. variable names $\{x\}$ or identifiers of elements from C, R, E of some information semantics Ω; ϕ^{pre} is a precondition 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. Preconditions and effects are defined as statements in logic $\mathcal{L}(\Sigma)$.

Below, in Definition 1, we specify a *restriction* relationship (partial ordering $\leq$) between capabilities that share the symbol signature Σ and the information semantics Ω. Practically, if a capability F_1 is a restriction of another capability F_2, any discovery algorithm that discovers F_1 as a suitable capability for some goal would also discover F_2 as such.

Definition 1 (capability restriction) A capability $F_1 = (\Sigma, \phi_1^{pre}, \phi_1^{eff})$ is a restriction of $F_2 = (\Sigma, \phi_2^{pre}, \phi_2^{eff})$ (written as $F_1 \leq F_2$) if the precondition 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:

$$\begin{aligned} F_1 \leq F_2 \quad \iff \forall s: \quad & (holds(\phi_1^{pre}, s) \Rightarrow holds(\phi_2^{pre}, s)) \land \\ & (holds(\phi_1^{eff}, s) \Rightarrow holds(\phi_2^{eff}, s)) \end{aligned} \tag{3}$$

Listing 3 shows the service ontology for functional semantics, introducing the class *Capability* with predicates *hasPrecondition* and *hasEffect*. The range of both these predicates is *Axiom*, meaning an arbitrary logical expression. The logical expression can be written in the syntax of any logical language, for instance WSML-Rule or RIF. Please note that we do not prescribe any constructs for functional semantics defined as a classification ontology, see Section 3.1 for discussion on expressing ontologies.

In order to demonstrate functional semantics on an example, in Listing 4 we show how precondition and effect are described using WSML-Rule for the telecommunication service whose information semantics is shown in Listing 2.

Here, a capability uses WSML-Rule to describe a functionality of the network subscription service (i.e. Video on Demand). The precondition specifies that the customer must have a minimal required bandwidth before the service can be invoked and the effect identifies a valid bundle having both the connection and the service defined as a result of the successful invocation. We use the *wsml:AxiomLiteral* datatype to capture rules expressed in the WSML syntax. A client can recognize this datatype and correctly process the axioms according to the WSML-Rule specification.

[10]In [7], Hepp develops the ontologized versions of various classifications.

[11]`http://www.unspsc.org/`

[12]`http://portal.rosettanet.org/cms/sites/RosettaNet/Standards/RStandards/dictionary/technical/`

```
...
ex:VideoOnDemanSubscription rdf:type wl:Capability ;
    wl: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 .
    wl:hasEffect "
        ?bundle[hasService hasValue ?service and
                hasConnection hasValue ?connection] memberOf Bundle
    "^^wsml:AxiomLiteral .
...
wsml:AxiomLiteral rdf:type rdfs:Class
   rdfs:subClassOf wl:Axiom
...
```

Listing 4: Capability Example

3.3 Behavioral Semantics

In general, behavioral semantics is the formal description which defines a service's *external* (public) and *internal* (private) behavior. The external behavior describes a protocol that can be used by the client to consume the service functionality. The internal behavior describes a workflow, i.e. how the functionality of the service is aggregated out of other services. However, internal behavior and its semantic description is outside the scope of this work; an interested reader can refer e.g. to the project SUPER[13].

For the purposes of our work, we call the public behavior of a service its *choreography*. The choreography is a description of a protocol from the service point of view, i.e. all the messages that are sent in to the service from the network and all the messages that are sent from the service out to the network.[14] We define the choreography X (read: chi) of the service using a state machine as

$$X = (\Sigma, L), \tag{4}$$

where $\Sigma \subseteq (\{x\} \cup C \cup R \cup E)$ is is the signature of symbols, i.e. variable names $\{x\}$ or identifiers of elements from C, R, E of some information semantics Ω; and L is a set of rules. Further, we distinguish dynamic symbols denoted as Σ_I (input), and Σ_O (output) and static symbols denoted as Σ_S. While the static symbols cannot be changed by the service invocation, the dynamic symbols correspond to input and output data of the service which can be changed by the invocation. Each rule $r \in L$ defines a state transition $r : r^{cond} \rightarrow r^{eff}$ where *cond* is defined as an expression in logic $\mathcal{L}(\Sigma_I \cup \Sigma_S)$ which must hold in a state before the transition is executed; *eff* is defined as an expression in logic $\mathcal{L}(\Sigma_I \cup \Sigma_O \cup \Sigma_S)$ describing how the state changes when the transition is executed. Σ_I and Σ_O correspond to the input and output data which is sent to the service and received back. Both input and output data is "linked" with input and output messages of an underlying Web service operation using the annotation mechanism described in Section 4. When the transition is executed, the input data is sent as a message to the underlying operation which then responds with a message containing the output data.

According to Definition 2, we define a *consistency* between a choreography and a capability that share the symbol signature Σ and an information semantics Ω.

Definition 2 (capability and choreography consistency) Let $X = (\Sigma, L)$ be a choreography, $F = (\Sigma, \phi^{pre}, \phi^{eff})$ be a capability, and $\tau = (s_1, s_2, ..., s_n)$ be some sequence of states where for each $s_i \in \tau, i = 1, ..., n-1$ exists $r_i \in L$ such that r_i^{cond} holds in s_i and r_i^{eff} changes the state to the state s_{i+1}. Then, X and F are consistent (denoted as $X \sim F$) iff ϕ^{pre} holds in s_1 and ϕ^{eff} holds in s_n.

[13]`http://www.ip-super.org`

[14]Please note that our notion of choreography is different from the one used in the Web Service Choreography Description Language (WS-CDL) (`http://www.w3.org/TR/ws-cdl-10/`)

```
...
wl:Choreography rdf:type rdfs:Class .
wl:hasInput rdf:Type rdfs:Property ;
     rdfs:domain wl:Choreography ;
     rdfs:range rdfs:Class .
wl:hasOutput rdf:Type rdfs:Property ;
     rdfs:domain wl:Choreography ;
     rdfs:range rdfs:Class .
wl:hasRule rdf:Type rdfs:Property ;
     rdfs:domain wl:Choreography ;
     rdfs:range wl:Rule .
wl:Rule rdf:Type rdfs:Class .
...
```

Listing 5: Service Ontology: Choreography Constructs

```
1   ...
2   ex:PriceQuoteChoreography rdf:type wl:Choreography ;
3      wl:hasInput ex:ServiceQuoteRequest ;
4      wl:hasOutput ex:ServiceQuoteResponse ;
5      wl:hasRule “
6          if (?quoteRequest)
7              ?quoteRequest memberOf ServiceQuoteRequest
8          then
9              _# memberOf ServiceQuoteResponse
10     ”^^wsml:RuleLiteral .
11  ...
12  wsml:RuleLiteral rdf:type rdfs:Class ;
13    rdfs:subClassOf wl:Rule .
14  ...
```

Listing 6: Choreography Example

In Listing 5 the service ontology defines constructs for the choreography description where the *Choreography* class holds *hasInput*, *hasOutput* and *hasRule* properties. The *hasInput* and *hasOutput* refer to classes from information semantics and define input and output symbols for the behavioral description. The *hasRule* defines a transition rule with range defined as *rdfs:Class* leaving a particular form of transition rules on the application and a concrete language used.

Listing 6 shows the choreography description for the service in the example in Listing 4. In here, we use the WSML-Rule language to describe a rule (lines 6-9) allowing a requester to quote for a price of the network service (the quote response can be sent out from the service if the quote request has been previously received). The rule uses *in* and *out* classes which need to be defined as part of the information semantics (for brevity we do not show these definitions in this deliverable). In order to execute the state transition using this rule, the rule's condition (lines 6,7) defines what must be true in a state and the rule's effect (line 9) defines how the state changes while at the same time the data of *ServiceQuoteResponse* is sent out from a relevant service's operation. We use the *wsml:RuleLiteral* datatype to capture rules expressed in the WSML syntax. A client can recognize this datatype and correctly process the rules according to the WSML-Rule specification.

3.4 Non-Functional Semantics

Generally, non-functional properties are incidental details specific to the implementation or running environment of a service, independent of the actual purpose of the service but necessary for successful and interoperable communication. Policy languages (e.g. WS-Policy[15]) are often used to express various service constraints that fall within non-functional semantics of the service. In addition, there are works that focus on semantic representation of policies, e.g. [9, 10, 15, 16]. While some of these works are based on the WS-Policy framework (esp. [10]), others offer their own models. Se-

[15]http://w3.org/TR/ws-policy

mantic interoperability of the various policy languages is still an open issue.

Our service ontology does not prescribe any constructs to model non-functional semantics of the service and thus can be used with any of the proposed approaches. We allow a user to model non-functional semantics using any ontology language with RDF syntax, similarly as for the information semantics in Section 3.1. For instance, Listing 7 shows a simple non-functional property describing the price of a Video on Demand bundle change.

```
ex:VideoOnDemandPrice rdf:type ex:PriceSpecification ;
   ex:pricePerChange ”30”^^ex:euroAmount ;
   ex:installationPrice ”49”^^ex:euroAmount .
```

Listing 7: Non-functional Property Example

4 Semantic Annotations for Web Services

In this section, we define how various types of semantic descriptions described in Section 3 can be applied to various WSDL components using SAWSDL attribute extensions. We define rules for consistency and completeness of the semantic descriptions and annotations of the WSDL components with these descriptions. For this purpose we use following notation for WSDL, SAWSDL and types of semantic descriptions:

- **WSDL:** *Schema* S and $\{x\}_S$ as a set of all element declarations and type definitions of schema S; *Interface* I and $\{o\}_I$ as a set of all operations of interface I, each operation $o \in \{o\}_I$ may have one input message element $m \in \{x\}_S$ and one output message element $n \in \{x\}_S$; *Service* E, *Binding*, *Endpoint*;
- **SAWSDL:** *modelReference*, $\mathit{ref}(x, \alpha)$ where x is a non-semantic description (any of the WSDL components) pointing to a semantic description α; *loweringSchemaMapping*, $\mathit{lower}(y, f(\beta))$ where y is an element or type from schema S, and $f(\beta) = y$ is a transformation function transforming some semantic description β to y; *liftingSchemaMapping*, $\mathit{lift}(z, g(z))$ where z is an element or type from S and $g(z) = \gamma$ is a transformation function transforming z to some semantic description γ;
- **Semantic Descriptions:** *Information* Ω (defined in Eq. 1 with $C(\Omega)$ as set of all classes), *Functional (Capability)* F (defined in Eq. 2), *Behavioral (Choreography)* X (defined in Eq. 4), and *Non-Functional*.

Please note that the SAWSDL specification only defines the use of annotations on *Schema* and the *Interface*, but other uses are intentionally not precluded. In addition, SAWSDL does not specify the type of semantics used for annotations, nor does it specify any rules where such semantics should be placed, leaving some flexibility for the application. SAWSDL uses URIs for all semantic references, expecting that the application either knows the referenced concept, or can find its definition. This section elaborates on the SAWSDL specifications and defines more precise rules, i.e., what kinds of semantics can be used for annotations and where they can be used. Table 2 shows the summary of how we apply the semantic descriptions to the WSDL components; in subsequent text of this section we describe these annotations in detail.

4.1 Information Semantics

Information semantics apply to the *Schema* that WSDL uses to describe messages, i.e. Element Declarations and Type Definitions. Both can carry *modelReferences* that

Table 2: Semantic annotations for 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 choreography
Service	Behavioral	Concrete service choreography
Schema	Behavioral	Pointers to input and output concepts in choreography signature
Service and Endpoint	Non-functional	Non-functional properties and policies
Binding (and sub-components)	Non-functional	For instance, operation-specific non-functional properties

link them to classes in the information semantics model Ω. Providing only references from the XML to the ontology is not sufficient, however, because at invocation time, the client needs to exchange data with the service, so the data needs to be transformed between the semantic model and the service-specific XML structure. For this, SAWSDL provides *liftingSchemaMapping* and *loweringSchemaMapping* annotations that link to the appropriate transformations. This is illustrated in Listing 8.

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

Listing 8: Schema linked to information semantics

According to Rule 1, all element declarations that are used as input messages must have consistent *modelReference* and *loweringSchemaMapping* annotations, and all element declarations that are used in output messages must have consistent *modelReference* and *liftingSchemaMapping* annotations. These mappings must exist for the client to understand and generate the messages in XML from ontology instances and vice-versa.

Rule 1 (completeness) Let S be a schema, $\{o\}_I$ be the set of operations of an interface I and Ω be a definition of information semantics. For each $m \in \{x\}_S$ where m is an *input message element* of any operation in $\{o\}_I$, there exists a class $c_1 \in C(\Omega)$ such that $ref(m, c_1)$ and $lower(m, f(c_1))$ with $f(c_1) = m$ are defined. Analogically, for each $n \in \{x\}_S$ where n is an *output message element* of any operation in $\{o\}_I$, there exists a class $c_2 \in C(\Omega)$ such that $ref(n, c_2)$ and $lift(n, g(n))$ with $g(n) = c_2$ are defined.

4.2 Functional Semantics

Functional semantics apply to the Web service, represented concretely by the *service* construct, and abstractly by the reusable *interface* construct. A SAWSDL *modelReference* is used to point from a service or an interface to its appropriate functional description, as shown in Listing 9. A WSDL interface may be shared by multiple services, therefore the functional description of the interface should be general, since it effectively applies to all services that implement that interface. 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.

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

Listing 9: WSDL interface linked to its capability

According to Rule 2 each functionality of a service must be a restriction of some functionality of the service's interface (see Definition 1). This is in particular useful to allow discovery to first find appropriate interfaces and then only deal with services that implement these interfaces. Rule 3 is analogical to Rule 2 with the difference that it applies to interface extension[16] when it is ensured that functionality cannot be lost through WSDL interface extension.

Rule 2 (consistency) Let F, G $(F \neq G)$ be some functional descriptions, E be the service and I be the interface such that E implements I. Then, if $ref(E, F)$ and $ref(I, G)$ are defined, then it must hold that $F \leq G$.

Rule 3 (consistency) Let F, G $(F \neq G)$ be some functional descriptions, I and J be some interfaces such that I extends J. Then, if $ref(I, F)$ and $ref(J, G)$ are defined, then it must hold that $G \leq F$.

Apart from describing the service (or the interface) as a whole, it is also possible to ascribe functional descriptions to the operations, again using *modelReference* pointers. Describing the functional semantics with operation capabilities is especially useful for Web services whose interface is simply a collection of standalone operations. For instance, 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 the operations, not all three. This shows that service discovery can, in such cases, become operation discovery. In this case, discovery and composition approaches may be used to select and order the invocations of the operations, and then the interface may not have a choreography description.

According to Rule 4, if an operation within an interface is not annotated with a capability, the interface must be annotated with a choreography that uses the operation. This rule ensures that no operation is left invisible to the semantic clients.

Rule 4 (completeness rule) Let $o \in \{o\}_I$ be the interface operation. If for any F, the $ref(o, F)$ is not defined where F is the functional description, then $ref(I, X)$ must be defined with $o \in X$ where X is the choreography description (cf. Section 4.3).

Please note that all the definitions above apply to both types of functional semantics defined in Section 3.2, i.e. capability defined on an abstract state space and categorization using some classification schema. It is even possible to combine both types of functional semantics for a service, interface and its operations. While the SAWSDL *modelReference* URI values do not 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.

4.3 Behavioral Semantics

Similarly to functional semantics, behavioral semantics (choreography) apply to the Web service, i.e. either to a WSDL service or to an interface. In this context, the purpose of a choreography is to define the order in which the client should invoke the operations of the Web service. Listing 10 shows the interface from Listing 9 with

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

the additional choreography annotation; a *modelReference* can contain any number of semantic concept URIs and it is up to the client to interpret them and to make use of them as appropriate. In fact, both services and interfaces can be annotated with multiple alternative choreographies.

```
...
<wsdl:interface name="NetworkSubscription"
      sawsdl:modelReference="http://example.org/onto#NetworkSubscriptionCapability
                                    http://example.org/onto#NetworkSubscriptionBehavior" >
   ...
</wsdl:interface>
...
```

Listing 10: WSDL interface linked to its capability and choreography

According to Rule 5 each choreography of an interface (or a service) must be consistent (see Definition 2) with some capability of that interface (or that service).

Rule 5 (consistency) Let Z be an interface or a service and $\{X_i\}$ be all choreographies such that $ref(Z, X_i)$ is defined. Then, for each X_i some capability F must exist for which $ref(Z, F)$ is defined and $X_i \sim F$.

A reference to the choreography is complemented by information semantics annotations in the XML Schema, because the input and output symbols from the choreography signature need to be linked with the actual XML messages that will carry the data. Information semantics annotations are described in Section 4.1. In addition, according to Rule 6, for each input or output symbol used by the choreography of an interface (or service), there must be an element declaration used appropriately as an input or output message by an operation of the interface (or the interface implemented by the service) that has a *modelReference* to the input or output symbol.

Rule 6 (completeness) Let X be the choreography with Σ_{in} and Σ_{out} denoting the sets of input and output symbols of X, and let Z be an interface or a service such that $ref(Z, X)$ is defined. Further, let I be Z if Z is an interface, or the interface that Z implements, if Z is a service; then $\{o\}_I$ is the set of operations of I. For every concept α which is identified by an identifier from Σ_{in}, exists m as an input message in $\{o\}_I$ with defined $ref(m, \alpha)$. Analogically, For every concept β which is identified by an identifier from Σ_{out}, exists n as an output message in $\{o\}_I$ with defined $ref(n, \beta)$.

4.4 Non-Functional Semantics

While we do not provide a model for non-functional semantics (see the discussion in Section 3.4), we can describe where they can be attached in WSDL.

Non-functional semantics may be attached to the service as a whole, as shown in Listing 11, or to particular endpoints (for instance to indicate different security, reliability and price combinations offered by the different endpoints).

```
...
<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>
...
```

Listing 11: WSDL service linked to its non-functional property

Non-functional semantics are always specific to a concrete service, therefore we do not recommend annotating interfaces with non-functional properties. In case non-functional properties need to be specified on the granularity of operations (for example, different operations may have different invocation micropayment prices), a WSDL binding or any of its sub-components may be used to capture these properties. With SAWSDL, non-functional properties are attached using *modelReference* from any of the WSDL components into the non-functional semantics model.

Some of the works on semantic models for non-functional properties are based on the WS-Policy framework. This framework contains an attachment specification, WS-PolicyAttachment[17], that defines mechanisms for associating policies with policy subjects. In WSDL, any component can be viewed as a policy subject. Using WS-PolicyAttachment for capturing the non-functional properties would be an alternative to using SAWSDL *modelReference*.

5 Related Work

There have been a number of approaches to automation of Web service usage with the help of semantic descriptions. Web Service Modeling Ontology (WSMO) and Semantic Markup for Web Services (OWL-S) are major frameworks that have achieved significant results; to our knowledge, at least WSMO continues to be actively developed. Web Service Semantics (WSDL-S) is a mechanism for adding semantic annotations to Web service descriptions, that was picked up by the W3C as input to the SAWSDL work. Additionally, WSDL 2.0 has an RDF mapping that expresses the WSDL model in RDFS and OWL. In this section, we focus on these efforts and their relation to our work presented in this deliverable; other approaches with more limited focus can be found, for instance, among the submissions to SWS Challenge[18].

OWL-S: Semantic Markup for Web Services [14] was the first major ontology for semantic description of Web services. In fact, OWL-S is a set of three interlinked ontologies: Service Profile captures the functional and non-functional semantics; Service Model details the behavioral semantics — it models the service choreography as a (composite) process with inputs and outputs and a rich variety of intermediate steps; and Service Grounding ties the process ontology with actual physical Web services. Information semantics is captured using the Web Ontology Language OWL.

Web Service Modeling Ontology WSMO [13] is a top-down conceptual model for semantic description of Web services. It has four top-level components: ontologies capture information semantics; goals describe what the user (or the system) wants to achieve; Web services model the properties of the available services; and mediators resolve any heterogeneities that might arise in a distributed system.

The structure of a Web service in WSMO has two aspects: *capability* describes the functionality of the service in terms of its preconditions and effects, and *interface* models the external interactions of the service. The interface is further split into *choreography* that captures the interaction between the service and its clients, and *orchestration* optionally describes how the service uses other Web services to achieve its function. In terms of the four kinds of semantics discussed in the Section 3, a Web service capability captures the functional semantics and the interface captures the behavioral semantics. The structure of the Web service description additionally allows non-functional properties which tie into some external model of non-functional semantics.

The structure of goals in WSMO follows the structure of Web services. One could say that a goal models the ideal requested service. OWL-S does not have an explicit model for goals, and also our work in this deliverable does not contain such a model, as goal description is a feature of a client system, whereas services should ideally be described semantically using a single standardized ontology.

[17]`http://w3.org/TR/ws-policy-attach`
[18]`http://sws-challenge.org/`

WSMO is realized in Web Service Modeling Language (WSML, [4]), a standalone language that is linked to existing Web service description standards, such as WSDL, using a so-called *grounding* (cf. [11]). Conceptually, grounding in WSMO is similar to that in OWL-S. A grounding layer makes the semantic model for Web services (WSMO or OWL-S) independent from any particular underlying communication or service description technology. WSMO or OWL-S systems can be built with plug-in support for any number of underlying technologies. In our work, we do not strive for such independence, instead we base our ontology on the Web services standard description language WSDL. Systems that support our lightweight semantic descriptions can be built as simple extensions of existing Web service systems, lowering the barrier of entry.

Web Service Semantics (WSDL-S, [1]) was created in the METEOR-S[19] project as a specification of how WSDL can be annotated with semantic information. WSDL-S itself does not provide a concrete model for semantic description of Web services, instead it makes the assumption that the concrete model will be expressible as annotations in WSDL and XML Schema documents. Parts of WSDL-S were taken as the basis for SAWSDL, which is discussed in the previous sections. Above those parts, WSDL-S also supported explicit constructs for categorizing WSDL interfaces, and for attaching preconditions and effects to interface operations. However, all this functionality can be achieved using the appropriate semantic models and the SAWSDL model reference.

Finally, the WSDL RDF mapping[20] represents the information from WSDL documents in RDF, in a simple OWL-based WSDL ontology. Also the SAWSDL annotations have their RDF representation. It is due to the existence of these RDF mappings that our lightweight semantic service ontology does not, in fact, need a class *Service* — the underlying model for Web services is taken from the WSDL ontology.

6 Conclusion and Future Work

In this deliverable we have proposed an open and lightweight approach to definition of service semantics with goal of the maximal compliance with Web standards. We have defined an extended specifications for Web service stack, i.e. semantic layer on the top of standard Web service specifications allowing to use and integrate not yet standardized rule languages for the Semantic Web. It is important to note that our goal is not to define yet another web syntax for logics but to show how existing rule languages can be reused for descriptions of some essential parts of services such as functional and behavioral descriptions. In addition, we have defined semantic annotations for Web Services using these descriptions together with the set of rules ensuring completeness and consistency of annotations. Our work takes into account the fact, that when annotating Web services with semantics, the domain expert might only want to use some of the semantic descriptions serving particular domain requirements of Web services' tasks and their automation. Using our approach it is possible to choose parts of service semantics, use them consistently for annotations of Web services and promote this way the automation of selected tasks in the service provisioning process.

In our future work we want to elaborate on the annotation of functional and behavioral descriptions for Web service interface. In particular we want to show how the behavioral semantics can be derived from annotations of interface operations. In addition, we want to make use of the safe methods and automatically determine a part of behavioral semantics relevant for fetching of data for service discovery with respect to description of a client goal [18]. Also, we plan to define other types of grounding to other Web service technology such as REST [5] and to use semantic annotations of services for enhancing mesh-ups, i.e. integration of different resources available through some API on the Web.

[19]`http://lsdis.cs.uga.edu/projects/meteor-s/`
[20]`http://w3.org/TR/wsdl20-rdf/`

Acknowledgements

This work is supported by the Science Foundation Ireland Grant No. SFI/02/CE1/I131, and the EU projects Knowledge Web (FP6-507482), SUPER (FP6-026850), and SemanticGov (FP6-027517).

References

[1] R. Akkiraju, J. Farrell, J.Miller, M. Nagarajan, M. M. Schmidt, Amit Sheth, and Kunal Verma. Web Service Semantics - WSDL-S, available at http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/. Technical report, LSDIS Lab, 2005.

[2] Emilia Cimpian and Adrian Mocan. Wsmx process mediation based on choreographies. In *Business Process Management Workshops*, pages 130–143, 2005.

[3] Jos de Bruijn, Dieter Fensel, and Holger Lausen. D34v0.1: The Web Compliance of WSML. Technical report, DERI, 2007. Available from: `http://www.wsmo.org/TR/d34/v0.1/`.

[4] Jos de Bruijn, Holger Lausen, Axel Polleres, and Dieter Fensel. The web service modeling language wsml: An overview. In *Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, volume 4011 of *Lecture Notes in Computer Science, LNCS*. Springer, 6 2006.

[5] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[6] Thomas Haselwanter, Paavo Kotinurmi, Matthew Moran, Tomáš Vitvar, and Maciej Zaremba. Wsmx: A semantic service oriented middleware for b2b integration. In *ICSOC*, pages 477–483, 2006.

[7] Martin Hepp. Products and services ontologies: A methodology for deriving owl ontologies from industrial categorization standards. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2(1):72–79, 2006.

[8] Ian Horrocks. Owl: A description logic based ontology language. In *CP*, pages 5–8, 2005.

[9] L. Kagal, T. Berners-Lee, D. Connolly, and D. J. Weitzner. Using Semantic Web Technologies for Policy Management on the Web. In *21st National Conference on Artificial Intelligence (AAAI)*, 2006.

[10] V. Kolovski, B. Parsia, Y. Katz, and J. A. Hendler. Representing Web Service Policies in OWL-DL. In *International Semantic Web Conference*, pages 461–475, 2005.

[11] Jacek Kopecký, Dumitru Roman, Matthew Moran, and Dieter Fensel. Semantic Web Services Grounding. In *Proc. of the Int'l Conference on Internet and Web Applications and Services (ICIW'06), Guadeloupe*, February 2006.

[12] A. Patil, S. Oundhakar, A. Sheth, and K. Verma. Semantic Web Services: Meteor-S Web Service Annotation Framework. In *13th International Conference on World Wide Web*, pages 553–562, 2004.

[13] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.

[14] The OWL Services Coalition. OWL-S 1.1 Release. Available at http://www.daml.org/services/owl-s/1.1/, November 2004.

[15] Andrzej Uszok, Jeffrey M. Bradshaw, Renia Jeffers, Matthew Johnson, Austin Tate, Jeff Dalton, and Stuart Aitken. Policy and Contract Management for Semantic Web Services. In *AAAI Spring Symposium on Semantic Web Services*, 2004.

[16] K. Verma, R. Akkiraju, and R. Goodwin. Semantic Matching of Web Service Policies. In *SDWP Workshop*, 2005.

[17] Tomas Vitvar, Adrian Mocan, Mick Kerrigan, Michal Zaremba, Maciej Zaremba, Matthew Moran, Emilia Cimpian, Thomas Haselwanter, and Dieter Fensel. Semantically-enabled service oriented architecture : concepts, technology and application. *Service Oriented Computing and Applications*, 2(2), 2007.

[18] Tomas Vitvar, Maciej Zaremba, and Matthew Moran. Dynamic discovery through meta-interactions with service providers. In *ESWC*, 2007.