Chapter 3

Semantic Web Service Description

Matthias Klusch

3.1 Introduction

The convergence of Semantic Web with service-oriented computing is manifested by Semantic Web service (SWS) technology. It addresses the major challenge of automated, interoperable and meaningful coordination of Web services to be carried out by intelligent software agents. In this chapter, we briefly discuss prominent SWS description frameworks, that are the standard SAWSDL, OWL-S and WSML\(^1\). This is complemented by a critique, and selected references to further readings on the subject.

3.2 Issues of Semantic Service Description

Each semantic service description framework can be characterised with respect to (a) what kind of service semantics are described, (b) in what language or formalism, (c) allowing for what kind of reasoning upon the abstract service descriptions? Further, we distinguish between an abstract Web Service, that is the description of the computational entity of the service, and a concrete service as one of its instances or invocations that provide the actual value to the user [22]. In this sense, abstract service descriptions are considered complete but not necessarily correct: There might be concrete service instances that are models of the capability description of the abstract service but can actually not be delivered by the provider.

\(^1\)Due to space limitations other semantic service description frameworks like SWSL (Semantic Web Service Language) and the DIANE service description language are excluded.
Chapter 3. Semantic Web Service Description

3.2.1 Functional and Non-Functional Service Semantics

In general, the functionality of a service can be described in terms of what it does, and how it actually works. Both aspects of its functional semantics (or capability) are captured by a service profile, respectively, service process model. The profile describes the signature of the service in terms of its input (I) and output (O) parameters, and its preconditions (P) and effects (E) that are supposed to hold before or after executing the service in a given world state, and some additional provenance information such as the service name, its business domain and provider. The process model of atomic or composite services describes how the service works in terms of the interplay between data and control flow based on a common set of workflow or control constructs like sequence, split+join, choice, and others.

This general distinction between profile and process model semantics is common to structured Web service description frameworks, while differences are in the naming and formal representation of what part of service semantics. We can further differentiate between stateless (IO), respectively, state-based (PE) abstract service descriptions representing the set of its instances, that are concrete services providing value to the user. The non-functional service semantics are usually described with respect to a quality of service (QoS) model including delivery constraints, cost model with rules for pricing, repudiation, availability, and privacy policy.

3.2.2 Structured Representation of Service Semantics

A domain-independent and structured representation of service semantics is offered by upper (top-level) service ontologies and languages such as OWL-S and WSML with formal logic groundings, or SAWSDL which comes, in essence, without any formal semantics. Neither OWL-S nor WSML provide any agreed formal but intuitive, standard workflow-based semantics of the service process model (orchestration and choreography). Alternatively, for abstract service descriptions grounded in WSDL, the process model can be intuitively mapped to BPEL orchestrations with certain formal semantics.

3.2.3 Monolithic Representation of Service Semantics

The formal specification of service semantics agnostic to any structured service description format can be achieved, for example, by means of a specific set of concept and role axioms in an appropriate logic (cf. Section 3.6). Since the service capability is described by means of one single service concept, this representation of service semantics is called monolithic and allows to determine the semantic relations between service descriptions fully within the underlying logical formalism based on concept satisfaction, subsumption and entailment. However, it does not provide any further information on how the service actually works in terms of the process model nor any description of non-functional semantics.
3.2.4 Data Semantics

The domain-dependent semantics of service profile parameters (also called data semantics) are described in terms of concepts, roles (and rules) taken from shared domain, task, or application ontologies. These ontologies are defined in a formal Semantic Web language like OWL, WSML, or SWRL. If different ontologies are used, agents are supposed to automatically resolve the structural and semantic heterogeneities for interoperability to facilitate better Web Service discovery and composition. This process of ontology matching is usually restricted to ontologies specified in the same language, otherwise appropriate inter-ontology mappings have to be provided to the agents.

In subsequent sections, we briefly introduce prominent approaches to both types of service representation. For structured semantic service descriptions, we focus on OWL-S, WSML, and SAWSDL, and omit to discuss alternatives like DSD (DIANE service description format) and SWSL (Semantic Web service Language).

3.2.5 Reasoning about Semantic Service Descriptions

The basic idea of formally grounded descriptions of Web Services is to allow agents to better understand the functional and non-functional semantics through appropriate logic-based reasoning. For this purpose, it is commonly assumed that the applied type of logic reasoning complies with the underlying semantic service description framework. Further, the concept expressions used to specify the data semantics of service input and output parameters are assumed to build up from basic concepts and roles taken from formal application or domain ontologies which the requester and provider commonly refer to. We survey approaches to non-logic-based, logic-based, and hybrid reasoning means for Semantic Web Service discovery, and composition planning in the next chapter.

3.3 SAWSDL

The standard language WSDL for Web Services operates at the mere syntactic level as it lacks any declarative semantics needed to meaningfully represent and reason upon them by means of logical inferencing. In a first response to this problem, the W3C Working Group on Semantic Annotations for WSDL and XML Schema (SAWSDL) developed mechanisms with which semantic annotations can be added to WSDL components. The SAWSDL specification became a W3C candidate recommendation on January 26, 2007\(^2\), and eventually a W3C recommendation on August 28, 2007.

\(^2\)http://www.w3.org/2002/ws/sawSDL/
3.3.1 Annotating WSDL Components

Unlike OWL-S or WSML, SAWSDL does not specify a new language or top-level ontology for semantic service description but simply provides mechanisms by which ontological concepts that are defined outside WSDL service documents can be referenced to semantically annotate WSDL description elements. Based on its predecessor and W3C member submission WSDL-S\(^3\) in 2005, the key design principles for SAWSDL are that (a) the specification enables semantic annotations of Web Services using and building on the existing extensibility framework of WSDL; (b) it is agnostic to semantic (ontology) representation languages; and (c) it enables semantic annotations for Web Services not only for discovering Web Services but also for invoking them.

Based on these design principles, SAWSDL defines the following three new extensibility attributes to WSDL 2.0 elements for their semantic annotation:

- An extension attribute, named `modelReference`, to specify the association between a WSDL component and a concept in some semantic (domain) model. This `modelReference` attribute is used to annotate XML Schema complex type definitions, simple type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults. Each `modelReference` identifies the concept in a semantic model that describes the element to which it is attached.

- Two extension attributes (liftingSchemaMapping and loweringSchemaMapping) are added to the set of XML Schema element declarations, complex type definitions and simple type definitions. Both allow to specify mappings between semantic data in the domain referenced by `modelReference` and XML, which can be used during service invocation.

An example of a SAWSDL service, that is a semantically annotated WSDL service with references to external ontologies describing the semantics of WSDL elements, is given in Figure 3.1: The semantics of the service input parameter of type “OrderRequest” is defined by an equally named concept specified in an ontology “purchaseorder” which is referenced (URI) by the element tag “modelReference” attached to “OrderRequest”. It is also annotated with a tag a tag “loweringSchemaMapping” which value (URI) points to a data type mapping, in this case an XML document, which shows how the elements of this type can be mapped from the referenced semantic data model (here RDFS) to XMLS used in WSDL.

3.3.2 Limitations

Major critic of SAWSDL is that it comes, as a mere syntactic extension of WSDL, without any formal semantics. In contrast to OWL-S and (in part) WSML, there is

\[^3\)http://www.w3.org/Submission/WSDL-S/\]
3.3. SAWSDL

no defined formal grounding of neither the XML-based WSDL service components nor the referenced external metadata sources (via modelReference). Quoting from the SAWSDL specification: “Again, if the XML structures expected by the client and by the service differ, schema mappings can translate the XML structures into the semantic model where any mismatches can be understood and resolved.” This makes any form of logic-based discovery and composition of SAWSDL service descriptions in the Semantic Web rather obsolete but calls for “magic” mediators outside the framework to resolve the semantic heterogeneities.

Another problem with SAWSDL today is its –apart from the METEOR-S framework by the developers of SAWSDL (WSDL-S) and related ongoing development efforts at IBM– still very limited software support compared to the considerable investments made in research and development of software for more advanced frameworks like OWL-S and WSMO worldwide. However, the recent announcement of SAWSDL as a W3C recommendation does not only support a standardized evolution of the W3C Web service framework in principle (rather than a revolutionary technology switch to far more advanced technologies like OWL-S or WSML) but certainly will push software development in support of SAWSDL and reinforce research on refactoring these frameworks with respect to SAWSDL.
Chapter 3. Semantic Web Service Description

3.4 OWL-S

OWL-S is an upper ontology used to describe the semantics of services based on the W3C standard ontology OWL and is grounded in WSDL. It has its roots in the DAML Service Ontology (DAML-S) released in 2001, and became a W3C candidate recommendation in 2005. OWL-S builds on top of OWL and consists of three main upper ontologies: the Profile, the Process Model, and the Grounding (cf. Figure 3.2).

In the following, we briefly summarize the underlying standard ontology language OWL and then present each of the main elements of OWL-S service descriptions.

3.4.1 Background: OWL

The standard ontology language for the Semantic Web is OWL [2, 4, 12] which is formally grounded in description logics (DL). OWL has its roots in the joint initiative DAML+OIL of researchers from the US and Europe in 2000 to develop a formal annotation or mark-up language for the Web. Only three years later, OWL became a W3C recommendation, and has been widely adopted by both industry and academics since then. The current version of OWL is OWL 1.1.

Variants

OWL comes in several variants, that are OWL-Full, OWL-DL, and OWL-Lite. Each variant corresponds to a DL of different expressivity and complexity. OWL-Lite and OWL-DL are an abstract syntactic form of the description logic SHIF(D), respectively, SHOIN(D).

OWL-Full. The most expressive but undecidable variant OWL-Full provides full compatibility with RDFS and covers the expressivity of the description logic

Figure 3.2: OWL-S service description elements.
SHOIQ(D)* which offers not only simple data types (D) but inverse roles (I), roles as subroles (a role hierarchy H), role transitivity (S) and qualified role cardinality restrictions (Q), as well as derived classes (classes used as individuals) together with non-primitive roles (cf. figure 3.3). Since OWL-Full allows in particular non-primitive roles (which can either be transitive or have transitive subroles) in role cardinality restrictions (S*), it is undecidable [14].

**OWL-DL.** Unlike OWL-Full, the less expressive variant OWL-DL (SHOIN(D)) allows only for unqualified number (role cardinality) restrictions, and does not permit to state that a role \( P \) is transitive or the inverse of another role \( Q \neq P \). In particular, OWL-DL does not include relationships between (transitive) role chains which would cause its undecidability. That is, in role number restrictions, only simple roles which are neither transitive nor have transitive subroles are
allowed; otherwise we gain undecidability even in SHN [14]. OWL-DL also does not allow classes to be used as individuals (derived classes), or to impose cardinality constraints on subclasses.

**OWL-Lite.** The variant OWL-Lite (SHIF(D)) is even less expressive than OWL-DL. It prohibits unions and complements of classes, does not allow the use of individuals in class descriptions (enumerated classes, nominals O), and limits role cardinalities to 0 or 1 (F). However, it is possible to capture all OWL-DL class descriptions except those containing either individuals or role cardinalities greater than 1 by properly exploiting the implicit negations introduced by disjointness axioms, and introducing new class names [13]. In role cardinality restrictions, only simple roles are allowed; however, it is unknown whether SHF or SHIF becomes undecidable without this restriction [14].

The syntactic transformation from OWL-Lite and OWL-DL ontologies to corresponding DL knowledge bases is of polynomial complexity. What makes OWL a Semantic Web language is not its semantics (which are quite standard for a DL) but the use of URI references for names, the use of XMLS datatypes for data values, and the ability to connect to documents in the Web.

### Relation to RDFS

The abstract syntax of OWL can be mapped to the normative syntax of RDF. In general, OWL adds constructors to RDFS for building class and property (role) descriptions (vocabulary) and new axioms (constraints) with model-theoretic semantics. In particular, the use of intersection (union) within (sub-)class descriptions, or universal/existential quantifications within super-/subclasses in OWL is not possible in RDFS[13]. However, the variants OWL-DL and OWL-Lite are extensions of a restricted use of RDFS whereas OWL-Full is fully upward compatible with RDFS. As mentioned above, OWL-DL and OWL-Lite do not allow classes to be used as individuals, or to impose cardinality constraints on subclasses, and the language constructors cannot be applied to the language itself - which is possible in OWL-Full and RDFS.

It has been shown only recently in [21] that the formal semantics of a sublanguage of RDFS is compatible with that of the corresponding fragment of OWL-DL such that RDFS could indeed serve as a foundational language of the Semantic Web layer stack. Though checking whether a RDF graph is an OWL ontology and upgrading from RDFS to OWL remains hard in practice, and is topic of ongoing research. For a detailed treatment of this subject, we refer to [7]. The syntactic transformation from OWL-Lite and OWL-DL ontologies to corresponding DL knowledge bases is of polynomial complexity.

---

5RDFS statements are equivalent to DL axioms of the form $C \sqsubseteq D$, $\top \sqsubseteq \forall P : C$, $\top \sqsubseteq \forall P^- : C$, $P \sqsubseteq Q$, $a : C$ and $(a, b) : P$. 
3.4. **OWL-S**

**Complexity**

As mentioned above, for OWL-Lite and OWL-DL, entailment reduced to concept satisfiability and ABox consistency is decidable in EXPTIME (complete), respectively, NEXPTIME (complete) [11, 26]. Though SHOIQ(D) is intractably co-NEXPTIME hard [26], its variant with non-primitive transitive roles in role cardinality restrictions (S*), hence OWL-Full, is undecidable [14]. Reasoning with data types and values (D) can be separated from reasoning with classes and individuals by allowing the DL reasoner to access a datatype oracle that can answer simple questions with respect to data types and values; this way, the language remains decidable if data type and value reasoning is decidable, i.e., if the oracle can guarantee to answer all questions of the relevant kind for supported datatypes.

Efficient query answering over DL knowledge bases with large ABoxes (instance stores) and static TBoxes is of particular interest in practice. Unfortunately, OWL can be considered insufficient for this purpose in general: Conjunctive query answering (CQA) for SHIQ and SHIF underlying OWL-Lite is decidable but only in time exponential in the size of the knowledge base (taxonomic complexity) and double exponential in the size of the query [7] (query and combined complexity); the CQA complexity for OWL-DL is unknown.

Another important inference on OWL ontologies is defined in terms of ontology entailment: Ontology \( O_1 \) entails another \( O_2 \), \( O_1 \models O_2 \), iff all interpretations that satisfy \( O_1 \) also satisfy \( O_2 \) in the DL sense. For both OWL-DL (SHOIN(D)) and OWL-Lite (SHIF(D)), ontology entailment checking can be polynomially reduced to the checking of the satisfiability of the corresponding DL knowledge bases \( O_1, O_2 \) (ontology consistency checking) which is decidable for both variants. The main criticism of the standard Semantic Web ontology language OWL is that it only allows for static declarative knowledge representation of limited expressivity and reasoning support.

### 3.4.2 Service Profile

The OWL-S profile ontology is used to describe what the service does, and is meant to be mainly used for the purpose of service discovery. An OWL-S service profile or signature encompasses its functional parameters, i.e. hasInput, hasOutput, precondition and effect (IOPEs), as well as non-functional parameters such as serviceName, serviceCategory, qualityRating, textDescription, and meta-data (actor) about the service provider and other known requesters. Please note that, in contrast to OWL-S 1.0, in OWL-S 1.1 the service IOPE parameters are defined in the process model with unique references to these definitions from the profile (cf. Figure 3.4).

Inputs and outputs relate to data channels, where data flows between processes. Preconditions specify facts of the world (state) that must be asserted in order for an agent to execute a service. Effects characterize facts that become asserted given a successful execution of the service in the physical world (state).
Chapter 3. Semantic Web Service Description

Whereas the semantics of each input and output parameter is defined as an OWL concept formally specified in a given ontology, typically in decidable OWL-DL or OWL-Lite, the preconditions and effects can be expressed in any appropriate logic (rule) language such as KIF, PDDL, and SWRL. Besides, the profile class can be subclassed and specialized, thus supporting the creation of profile taxonomies which subsequently describe different classes of services. An example of a Semantic Web service profile in OWL-S 1.1 is given in figure 3.5.

3.4.3 Service Process Model

An OWL-S process model describes the composition (choreography and orchestration) of one or more services, that is the controlled enactment of constituent processes with respective communication pattern. In OWL-S this is captured by a common subset of workflow features like split+join, sequence, and choice (cf. Figure 3.6). Originally, the process model was not intended for service discovery but the profile by the OWL-S coalition.

More concrete, a process in OWL-S can be atomic, simple, or composite. An atomic process is a single, black-box process description with exposed IOPEs. Simple processes provide a means of describing service or process abstractions which have no specific binding to a physical service, thus have to be realized by an atomic process, e.g. through service discovery and dynamic binding at runtime, or expanded into a composite process. The process model of the example OWL-S service above is provided in Figure 3.7.

Composite processes are hierarchically defined workflows, consisting of atomic, simple and other composite processes. These process workflows are constructed using a number of different control flow operators including Sequence, Unordered (lists), Choice, If-then-else, Iterate, Repeat-until, Repeat-while, Split, and Split+Join.
3.4. **OWL-S**

*Figure 3.5: Example of OWL-S 1.1 service profile.*

*Figure 3.6: OWL-S service process model.*
Chapter 3. Semantic Web Service Description

In OWL-S 1.1, the process model also specifies the inputs, outputs, preconditions, and effects of all processes that are part of a composed service, which are referenced in the profiles of the respective services. An OWL-S process model of a composite service can also specify that its output is equal to some output of one of its subprocesses whenever the composite process gets instantiated. Moreover, for a composite process with a Sequence control construct, the output of one subprocess can be defined to be an input to another subprocess (binding).

Unfortunately, the semantics of the OWL-S process model are left undefined in the official OWL-S documents. Though there are proposals to specify these semantics in terms of, for example, the situation calculus [19], and the logic programming language GOLOG based on this calculus [20].

3.4.4 Service Grounding

The grounding of a given OWL-S service description provides a pragmatic binding between the logic-based and XMLS-based service definitions for the purpose of facilitating service execution. Such a grounding of OWL-S services can be, in principle, arbitrary but has been exemplified for a grounding in WSDL to pragmatically connect OWL-S to an existing Web service standard (cf. Figure 3.8).

In particular, the OWL-S process model of a service is mapped to a WSDL
description through a thin (incomplete) grounding: Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of respectively named XML data types of corresponding input and output messages. Unlike OWL-S, WSDL cannot be used to express pre-conditions or effects of executing services. Any atomic or composite OWL-S service with a grounding in WSDL is executable either by direct invocation of the (service) program that is referenced in the WSDL file, or by a BPEL engine that processes the WSDL groundings of simple or orchestrated Semantic Web Services.

3.4.5 Software Support

One prominent software portal of the Semantic Web community is SemWebCentral\(^7\) developed by InfoEther and BBN Technologies within the DAML program in 2004 with BBN continuing to maintain it today. As a consequence, it comes at no surprise that this portal offers a large variety of tools for OWL and OWL-S service coordination as well as OWL and rule processing. Examples of publicly available software support of developing, searching, and composing OWL-S services are as follows.

\(^7\)http://projects.semwebcentral.org/
Chapter 3. Semantic Web Service Description

- **Development.**
  OWL-S IDE integrated development environment\(^8\), the OWL-S 1.1 API\(^9\) with the OWL-DL reasoner Pellet\(^10\) and OWL-S editors.

- **Discovery.**
  OWL-S service matchmakers OWLS-UDDI\(^11\), OWLSM\(^12\) and OWLS-MX\(^13\) with test collection OWLS-TC2.

- **Composition.**
  OWL-S service composition planners OWLS-XPlan\(^14\), GOAL\(^15\).

### 3.4.6 Limitations

Main critics of OWL-S concern its limited expressiveness of service descriptions in practice which, in fact, corresponds to that of its underlying description logic OWL-DL. Only static and deterministic aspects of the world can be described in OWL-DL, since it does not cover any notion of time and change, nor uncertainty. OWL-S allows specifying conditional effects, that are possible effects of the service each of which conditioned by its result (output) but not input. Besides, in contrast to WSDL, an OWL-S process model cannot contain any number of completely unrelated operations.

However, OWL-S bases on existing W3C Web standards, in particular the Web Services protocol stack: It extends OWL and has a grounding in WSDL. Furthermore, the large set of available tools and applications of OWL-S services, as well as ongoing research on Semantic Web rule languages on top of OWL such as SWRL and variants still support the adoption of OWL-S for Semantic Web Services, though this might be endangered by the choice of SAWSDL as a W3C standard just recently.

### 3.5 WSML

In this section, we informally introduce the reader to the basic elements of semantic service description in the Web service modeling language (WSML).

---

\(^8\) [http://projects.semwebcentral.org/projects/owl-s-ide/](http://projects.semwebcentral.org/projects/owl-s-ide/)
\(^11\) [http://projects.semwebcentral.org/projects/mm-client/](http://projects.semwebcentral.org/projects/mm-client/)
\(^12\) [http://projects.semwebcentral.org/projects/owlsm/](http://projects.semwebcentral.org/projects/owlsm/)
\(^14\) [http://projects.semwebcentral.org/projects/owl-s-xplan/](http://projects.semwebcentral.org/projects/owl-s-xplan/)
\(^15\) [http://www.smartweb-project.de](http://www.smartweb-project.de)
3.5. WSML

3.5.1 WSMO Framework

The WSMO (Web Service Modelling Ontology) framework\textsuperscript{16} provides a conceptual model and a formal language WSML (Web Service Modeling Language)\textsuperscript{17} for the semantic markup of Web services together with a reference implementation WSMX (Web Service Execution Environment). Historically, WSMO evolved from the Web Service Modeling Framework (WSMF) as a result of several European Commission funded research projects in the domain of Semantic Web Services like DIP, ASG, Super, TripCom, KnowledgeWeb and SEKT in the ESSI (European Semantic Systems Initiative) project cluster\textsuperscript{18}.

WSMO offers four key components to model different aspects of Semantic Web services in WSML (Web Service Modeling Language): Ontologies, goals, services, and mediators. Goals in goal repositories specify objectives that a client might have when searching for a relevant Web service. WSMO ontologies provide the formal logic-based grounding of information used by all other modeling components. Mediators bypass interoperability problems that appear between all these components at data (mediation of data structures), protocol (mediation of message exchange protocols), and process level (mediation of business logics) to "allow for loose coupling between Web services, goals (requests), and ontologies". Each of these components, called top-level elements of the WSMO conceptual model, can be assigned non-functional properties to be taken from the Dublin Core metadata standard by recommendation.

3.5.2 WSML Variants

The Web service modeling language WSML allows to describe a Semantic Web service in terms of its functionality (service capability), imported ontologies, and the interface through which it can be accessed for orchestration and choreography. The syntax of WSML is mainly derived from F-Logic extended with more verbose keywords (e.g., "hasValue" for $\texttt{hasValue} \rightarrow$, "p memberOf T" for \texttt{p:T} etc.), and has a normative human-readable syntax, as well as an XML and RDF syntax for exchange between machines. WSML comes in five variants with respect to the logical expressions allowed to describe the semantics of service and goal description elements. In the following, we informally introduce F-Logic and the WSML variants in very brief.

\textbf{F-Logic.} F-Logic is an object-oriented extension of first-order predicate logic with objects of complex internal structure, class hierarchies and inheritance, typing, and encapsulation in order to serve as a basis for object-oriented logic programming and knowledge representation. For modeling ontologies, it allows to

\textsuperscript{16}\url{http://www.wsmo.org/TR/d2/v1.4/20061106}
\textsuperscript{17}\url{http://www.wsmo.org/TR/d16/d16.1/v0.21/20051005/}
\textsuperscript{18}\url{http://www.sdkcluster.org/}
Chapter 3. Semantic Web Service Description

define, for example, is-a object class (or type) hierarchies through subclass relationships like person::human denoting class "person" as a subclass of "human", a class of objects with structured properties (object type signature) like person[name *⇒ string, children *⇒ person], and instances of classes (typed objects) like john:person as well as rules like (R:region :- R1:region, R::R1.) and (L:location :- L:R, R:region.) denoting that every subclass "R" of an object class "R1" of type "region" is a region and that every member L of a region "R" is also a location. Rules may also be used to define virtual classes like the rule (X:redcar :- X:car, X[color → red].) defining the virtual class "redcar".

F-Logic comes in two flavors with respective variants: A first-order F-Logic variant (F-Logic(FO)) that includes an (OWL-DL/WSML-DL) description logic subset of classical predicate logic, and a full logic programming (LP) variant (F-Logic(LP)) that is LP extended with procedural built-ins (functions), and non-monotonic default inheritance and negation-as-(finite)-failure\(^{19}\). Non-monotonic (default) inheritance of F-Logic(LP) allows to override default property values of classes inherited by subclasses. For example, a class Elephant[color *⇒ grey] with default value "grey" of property "color" has a subclass royalElephant[color *⇒ white] for which objects this default value of inherited property "color" is overridden by (default) value "white". Hence, one can assert object fred[color → grey] as member of class "Elephant" (but not "royalElephant"), and clyde[color → white] as member of both classes. Semantics of F-Logic(LP) are derived from Van Gelder's well-founded (fix-point-based, minimal model) semantics of the nonmonotonic part of logic programming [27]. F-Logic(LP) is more commonly used than F-Logic(FO) like in the LP-reasoners OntoBroker, Flora-2 and Florid. For more details on the syntax and semantics of F-Logic, we refer to [29, 28].

**WSML variants.** The formal semantics of WSML service description elements are specified as logical axioms and constraints in ontologies using one of five WSML variants: WSML-Core, WSML-DL, WSML-Flight, WSML-Rule and WSML-Full (cf. Figure 3.9).

Though WSML has a special focus on annotating Semantic Web services like OWL-S it tries to cover more representational aspects from knowledge representation and reasoning under both classical FOL and nonmonotonic LP semantics. For example, WSML-DL is a decidable variant of F-Logic(FO) with expressivity close to the description logic SHOIN(D), that is the variant OWL-DL of the standard ontology Web language OWL. WSML-Flight is a decidable Datalog variant of F-Logic(LP) (function-free, non-recursive and DL-safe Datalog rules) with (nonmonotonic) default negation under perfect model semantics [23] of locally stratified F-Logic programs with ground entailment. WSML-Rule is a fully-fledged

\(^{19}\)In nonmonotonic LP, like semi-decidable PROLOG and F-Logic(LP), the default negation of fact p (not p) means "p is true if p cannot be proven in a given knowledge base KB in finite time" (under closed-world assumption). This is nonmonotonic, i.e., truth values of asserted and implied knowledge in KB do not grow monotonically: (KB |= p) does not imply (KB ∪ {q} |= p), e.g., KB = \{(p :- not q)\} implies p true (KB |= p), but KB* = \{(q, p :- not q)\} implies p false.
logic programming language with function symbols, arbitrary rules with inequality and nonmonotonic negation, and meta-modeling elements such as treating concepts as instances, but does not feature existentials, strict (monotonic) negation, and equality reasoning. The semantics of WSML-Rule is defined through a mapping to undecidable (nonmonotonic, recursive) F-Logic(LP) variant with inequality and default negation under well-founded semantics [27]. WSML-Full shall unify the DL and LP paradigms as a superset of FOL with non-monotonic extensions to support nonmonotonic negation of WSML-Rule via Default Logic, Circumscription or Autoepistemic Logic. However, neither syntax nor semantics of WSML-Full have been completely defined yet.

3.5.3 Services in WSML

In general, the description of the semantics of a service and request (so-called goal) in WSML is structured into the parts of the service capability, the service interface used for orchestration and choreography, and the shared variables.

**Goal.** Like in OWL-S, a goal in WSMO represents the desired WSML service which is indicated with a special keyword “goal” instead of “webservice” in front of the service description. A goal refers to a desired state that can be described by help of a (world state) ontology. Such an ontology provides a basic vocabulary for specifying the formal semantics of service parameters and transition rules (TBox), and a set of concept and role instances (ABox) which may change their values from one world state to the other. It also specifies possible read-write access rights to instances and their grounding. A state is the dynamic set of instances of concepts,
relations and functions of given state ontology at a certain point of time. The interpretation of a goal (and service) in WSML is not unique: The user may want to express that either all, or only some of the objects that are contained in the described set are requested [16].

Figure 3.10 gives an example of a goal in WSML to find a service, which as a result of its execution, offers to reserve a ticket for the desired trip. In this case, the only element of the capability the user is interested in, is the postcondition of the desired service.

**Service Capability.** A WSML service capability describes the state-based functionality of a service in terms of its precondition (conditions over the information space), postcondition (result of service execution delivered to the user), assumption (conditions over the world state to met before service execution), and effect (how does the execution change the world state). Roughly speaking, a WSML service capability consists of references to logical expressions in a WSML variant that are named by the scope (precondition, postcondition, assumption, effect, capability) they intend to describe. It also specifies non-functional properties and all-quantified shared variables (with the service capability as scope) for which the logical conjunction of precondition and assumption entails that of the postcondition and the effect.

Figure 3.11 provides an example of a Web Service capability specified in WSML. This example service offers information about trips starting in Austria and requires the name of the person and credit card details for making the reservation. The assumption is that the credit card information provided by the requester must designate a valid credit card that should be of type either PlasticBuy or GoldenCard. The postcondition specifies that a reservation containing the details of a ticket for the desired trip and the reservation holder is the result of the successful execution of the Web Service. Finally, the effect in the world state is that the
credit card is charged with the cost of the ticket.

**Service Interface.** A WSML service interface contains the description of how the overall functionality of the Web service is achieved by means of cooperation of different Web service providers (orchestration) and the description of the communication pattern that allows to one to consume the functionality of the Web service (choreography). A choreography description has two parts: the state and the guarded transitions. As mentioned above, a state is represented by a WSMO ontology, while guarded transitions are if-then rules that specify conditional transitions between states in the abstract state space.

Figure 3.12 provides an example of a service interface with choreography, and a guarded transition rule which requires the following to hold: If a reservation request instance exists (it has been already received, since the corresponding concept in the state ontology currently has the mode “in”) with the request for a trip starting in Austria, and there exists a ticket instance for the desired trip in the Web service instance store, then create a temporary reservation for that ticket.

### 3.5.4 Software Support

The project web site www.wsmo.org provides, for example, a comprehensive set of links to software tools for developing WSMO oriented services (in WSML) most
of which available under open source related licenses at sourceforge.net. Examples include the WSMO4J API\(^{20}\), the WSMO studio\(^{21}\) with WSML service editor, WSML-DL and WSML-Rule reasoner, WSML validator, and the WSMX service execution environment\(^{22}\).

Remarkably, there are still neither implemented semantic WSML service composition planners nor full-fledged WSML service matchmakers available apart from a rather simple keyword-based and non-functional (QoS) parameter oriented WSML service discovery engine as part of the WSMX suite, and the hybrid matchmaker WSMO-MX. This situation of weak software support of services in WSML, however, could drastically improve in near future for various reasons of both politics and science.

### 3.5.5 Limitations

The WSMO conceptual model and its language WSML is an important step forward in the SWS domain as it explicitly overcomes some but not all limits of OWL-S. Unfortunately, the development of WSMO and, in particular, WSML has been originally at the cost of its connection to the W3C Web service standard stack at that time. This raised serious concerns by the W3C summarized in its

\(^{20}\)http://wsmo4j.sourceforge.net/

\(^{21}\)http://www.wsmostudio.org/download.html

\(^{22}\)http://sourceforge.net/projects/wsmx/
official response to the WSMO submission in 2005 from which we quote\textsuperscript{23}: 

“The submission represents a development, but one which has been done in isolation of the W3C standards. It does not use the RDFS concepts of Class and Property for its ontology, and does not connect to the WSDL definitions of services, or other parts of the Web Services Architecture. These differences are not clearly explained or justified. The notion of choreography in WSMO is obviously very far from the definition and scope presented in WS-CDL. The document only gives little detail about mediators, which seem to be the essential contribution in the submission.”

To date, however, the connection of WSML with WSDL and SAWSDL (WSDL-S) has been established in part, and is under joint investigation by both WSMO and SAWSDL initiatives in relevant working and incubator groups of the standardisation bodies OASIS and W3C.

Another main critic on WSML concerns the lack of formal semantics of service capabilities in both the WSMO working draft as of 2006, and the WSML specification submitted to the Web consortium W3C in 2005. Recently, this problem has been partly solved by means of a semi-monolithic FOL-based representation of functional service semantics over abstract state spaces and (guarded) state space transitions by service execution traces [24]. Though, the formal semantics of the WSML service (orchestration and choreography) interface part is still missing — which is not worse than the missing process model semantics of OWL-S.

Further, principled guidelines for developing the proposed types of WSMO mediators for services and goals in concrete terms are missing. Besides, the software support for WSML services provided by the WSMO initiative appears reasonable with a fair number of downloads but is still not comparable to that of OWL-S in terms of both quantity and diversity.

Finally, as with OWL-S, it remains to be shown whether the revolutionary but rather academic WSMO framework will be adopted by major business stakeholders within their service application landscapes in practice. In general, this also relates to the key concern of insufficient scaling of logic-based reasoning to the Web scale as mentioned in the previous chapter.

3.6 Monolithic DL-Based Service Descriptions

As mentioned above, an alternative to formally specifying the functional semantics of a Web service agnostic to any structured service description formats like OWL-S, SAWSDL, or WSML, is the pure DL-based approach: The abstract service semantics is defined through an appropriate set of concept and role axioms in a given description logic. Any instantiation of this service concept corresponds to a concrete service with concrete service properties. That is, the extension $S^I$ of a service concept $S$ representing the abstract service to be described in an interpretation $I$ of the concept over a given domain contains all service instances the provider of $S$ is willing to accept for contracting with a potential requester of

\textsuperscript{23}http://www.w3.org/Submission/2005/06/Comment
Chapter 3. Semantic Web Service Description

S. An example of a monolithic DL-based description of an abstract service and possible service instances is shown in Figure 3.13 ([8]).

In this example, the functional semantics or capability of the abstract Web service $S$ is described by a set $D_S$ of two DL concept axioms: The service concept $S$ for the shipping of items with a weight less than or equal to 50kg from cities in the UK to cities in Germany; the concept $Shipping$ (used to define $S$) which assures that instances of $S$ specify exactly one location for origin and destination of the shipping. Semantic relations between such monolithically described service semantics can be determined fully within the underlying logical formalism, that is by DL-based inferencing. For a more detailed treatment of this topic, we refer to [8].

3.7 Critique

Main critiques of Semantic Web services range from limitations of proposed frameworks via the lack of appropriate means of service coordination and software support to the legitimation of the research field as a whole. As one consequence, SWS technology still appears too immature for getting adopted by both common Web users and developers in practice, and industry for its commercial use on a large scale.

Do we really need formal service semantics? Some recent critics of SWS technology argue against the significance of its claimed benefits for practical Web service applications in general. Key justification of this argument, is related to the general critics on Semantic Web technologies. In fact, the need of having formal
logic-based semantics specified for Web Services in practical human-centred applications is often questioned: It is completely unclear whether the complete lack of formal service semantics turns out to be rather negligible, or crucial for what kinds of service applications for the common Web user in practice, and on which scale.

Just recently, van Harmelen and Fensel [6] argued for a more tolerant and scalable Semantic Web reasoning based on approximated rather than strict logic-based reasoning. This is in perfect line with experimental results available for hybrid SWS matchmakers that combine both logic and approximated reasoning like the OWLS-MX [17], the WSMO-MX [15] and the syntactic OWLS-iMatcher [1].

Where are all the Semantic Web Services?  Another interesting question concerns the current reality of SWS technology in use. According to a recent survey of publicly available Semantic Web service descriptions in the surface Web [18], revealed that not more than around 1500 indexed semantic services in OWL-S, WSML, WSDL-S or SAWSDL are accessible in the Web of which only about one hundred are deployed outside special test collections like the OWLS-TC. Though we expect the majority of Semantic Web Services being maintained in private project repositories and sites of the deep Web [10], it certainly does not reflect the strong research efforts carried out in the SWS domain world wide.

Of course, one might argue that this comes at no surprise in two ways. First, SWS technology is immature (with a standard announced just recently, that is SAWSDL) which still provides insufficient common ground supporting its exploitation by end users. Though this is certainly true, the other related side of this argument is that massive research and development of the field around the globe should have produced a considerable amount of even publicly visible Semantic Web service descriptions within the past half dozen of years.

Second, one might argue that it is not clear whether the surface Web and academic publications are the right place to look for Semantic Web service descriptions, as many of them would be intended for internal or inter-enterprise use but not visible for the public. Though this is one possible reason of the low numbers reported above, it indicates some lack of visibility to the common Web user to date.

Where are the easy to use SWS tools for the public?  As with Semantic Web application building in general, apart from the project prototypes and systems there is hardly any easy to use software support off the shelves available to the common user for developing, reusing and sharing her own Semantic Web Services — which might hamper the current confluence of the field with the Web 2.0 into the so called service Web 3.0 in practice.

\footnote{projects.semwebcentral.org/projects/owls-tc/}
How to efficiently coordinate Semantic Web Services? Despite tremendous progress made in the field in European and national funded research projects like DIP, Super, CASCOM, Scallops and SmartWeb, there still is plenty of room for further investigating the characteristics, potential, and limits of SWS coordination in both theory and practice. The Semantic Web Services Challenge\textsuperscript{25} attempts to qualitatively measure the minimal amount of programming required to adapt the semantics of given systems to new services. This acknowledges that the complete automation of composing previously unknown services is impossible, rather being a kind of Holy Grail of modern semantic technologies. Besides, the comparative evaluation of developed SWS discovery tools is currently hard, if not impossible, to perform since the required large scale service retrieval test collections are still missing even for the standard SAWSDL. Related to this, there are no large scale experimental results on the scalability of proposed service coordination means in practice available.

Apart from the problem of scalable and efficient SWS discovery and composition, another open problem of SWS coordination as a whole is privacy preservation. Though there are quite a few approaches to user data privacy preservation for each of the individual coordination processes (discovery, composition, and negotiation), there is no integrated approach that allows to coherently secure SWS coordination activities.

3.8 Summary

This chapter briefly introduced prominent frameworks of describing services in the Semantic Web together with some major critics of the domain. Overall, the interdisciplinary, vivid research and development of the Semantic Web did accomplish an impressive record in both theory and applications within just a few years since its advent in 2000. Though we identified several major gaps to bridge before the still immature Semantic Web services technology will make it to the common user of the Web, the ongoing convergence of the Semantic Web, Web 2.0, and services into a so called service Web 3.0 indicates its potential for future Web application services. In the next chapter, we survey prominent approaches to semantic discovery and composition planning of services in the Semantic Web.

Further readings For more comprehensive information on Semantic Web Services in general, we refer to the accessible readings on the subject [25, 3, 5]. Examples of major funded research projects on Semantic Web Services are

- the European funded integrated projects DIP\textsuperscript{26} and ASG (Adaptive semantic services grid technologies)\textsuperscript{27}

\textsuperscript{25}http://sws-challenge.org
\textsuperscript{26}dip.semanticweb.org/
\textsuperscript{27}asg-platform.org
• SmartWeb — Mobile multi-modal provision of Semantic Web Services\textsuperscript{28},
• SCALLOPS\textsuperscript{29} — Secure Semantic Web Service coordination,
• the European funded specific targeted research projects CASCOM\textsuperscript{30}, ARTEMIS\textsuperscript{31} — Semantic Web Services for e-health applications (mobile, P2P)

For more information about Semantic Web service description frameworks, we refer to the respective documents submitted to the W3C:

• OWL-S\textsuperscript{32}
• WSMO\textsuperscript{33}
• SAWSDL\textsuperscript{34}
• Semantic Web Services Framework SWSF\textsuperscript{35} with SWSL-Rule\textsuperscript{36} for monolithic FOL-based service representation by means of different variants of rule languages (DLP, HiLog, etc).

References


Updated version at www.webont.org/ow1/1.1/tractable.html (6 April 2007)


\textsuperscript{28}http://www.smartweb-project.de/
\textsuperscript{29}http://www.ags.difi.uni-sb.de/klusch/scallops/
\textsuperscript{30}http://www.ist-cascom.org
\textsuperscript{31}http://www.srdc.metu.edu.tr/webpage/projects/artemis/
\textsuperscript{32}http://www.w3.org/Submission/OWL-S/
\textsuperscript{33}http://www.w3.org/Submission/WSMO/
\textsuperscript{34}http://www.w3.org/2002/ws/sawsdl/
\textsuperscript{35}http://www.daml.org/services/swsf/
\textsuperscript{36}http://www.w3.org/Submission/SWSF-SWSL/)


References


