

An Introduction into Semantic Web Services

Torben Schreiter

Hasso-Plattner-Institute for IT Systems Engineering,
University of Potsdam, Germany
`torben.schreiter@hpi.uni-potsdam.de`

Abstract. The recent deployment of a large variety of web services available through the Internet identified a number of problems regarding ambiguity of service descriptions. Semantic web services are a promising approach towards machine-processability of web service's functional and non-functional capabilities.

This paper aims at introducing the general concepts of semantic annotation of web services. Therefore, ontologies will be explained first. The concrete ontology representation language OWL (Web Ontology Language) as well as its foundation, the Resource Description Framework (RDF), are described. Furthermore, by means of OWL-S as an upper ontology for services, it is illustrated how web services can be semantically described. Finally, the characteristics of semantic matching algorithms are defined and the architecture of the METEOR-S framework as an environment for adaptive and most optimal execution of web processes is explained.

1 Introduction

Web services have become very popular amongst providers of various commercial (e.g. B2B) and non-commercial services on the web due to the opportunity to interoperate with a large number of service requestors. This interoperability of different service requestors and providers has been facilitated by a (small) set of standardized technologies. Currently, the practical interoperability of web services is based on *syntactical* service descriptions using e.g. the widely spread *Web Services Description Language* (WSDL) [1]. However, it is desired to automate service discovery and service composition during the execution of web processes. Furthermore, the consideration of various Quality of Service (QoS) constraints is desirable. In general a more flexible and advantageous selection of services at run-time is expected to improve overall effectiveness and efficiency of the process since the range of services is also expected to change continuously. Regarding the process's entire lifecycle, it is e.g. conceivable that certain services are no longer available, whereas in other cases services might evolve, which are more suitable than those initially selected.

Current service descriptions do not provide any means of stating any information about the *semantics* of a service's characteristics and capabilities. Thus,

services cannot be unambiguously described. Problematically, as a requirement for automation of aspects like service discovery and service composition, machines have to be able to process services' descriptions in order to determine their applicability in certain (business) context. Currently, high research efforts are invested in the field of semantic annotation of web services based on ontologies in order to tackle this problem. The concept of semantic web services is based on the idea of the *Semantic Web*.

Originally, the Semantic Web was manifested in [2] as a vision for the future of the world wide web (WWW). This vision resulted in a project by the *World Wide Web Consortium* (W3C) in 2001 aiming at the development of technologies supporting machine-processability of web content. Within the scope of this project, the model of a semantic web language stack (see figure 1) was developed. The stack is based on well-known and mature concepts such as Unified Resource Identifiers (URIs), Unicode as well as XML documents (eXtensible Markup Language). Above of the foundation technologies, ontologies are intended to enable processing and reasoning on the knowledge. Ontology representation languages like the *Web Ontology Language* (OWL) [7,8] are utilized as a means for concrete representation of domain knowledge.

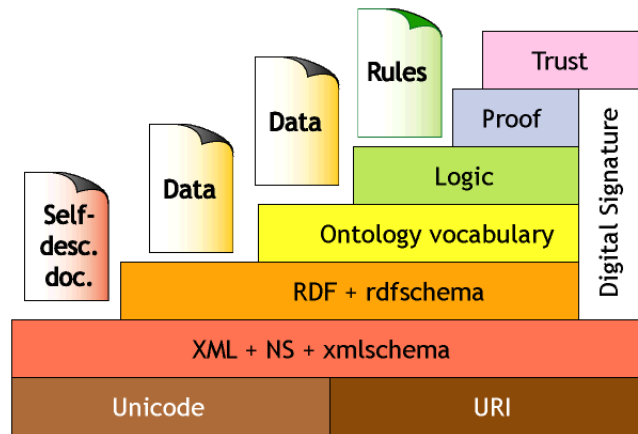


Fig. 1. W3C Semantic Web Language Stack. (Figure taken from [3])

Both concepts, Semantic Web and semantic web services, are based on the idea of machine-processable ontological domain knowledge forming the basis for reasoning. The major difference is, however, that the Semantic Web project intends to annotate passively presented web content, whereas semantic web services aim at annotating active services' capabilities. The remainder of this paper focuses on concepts and technologies related to semantic web services.

This paper is organized as follows: Section 2 introduces ontologies, provides some fundamental definitions and describes the *Resource Description Framework*

(RDF) as well as OWL as concrete means for ontological knowledge representation. Next, section 3 summarizes how web services in particular can be semantically described using an *Upper Ontology for Services* (OWL-S). The fourth section explains, which aspects are to be considered when semantically matching different services' capabilities. Section 5 briefly explains the architecture of the METEOR-S framework, which is capable of executing dynamic web processes based on semantic service descriptions. Finally, the paper is summarized and a conclusion is provided.

2 Ontologies

Ontologies are the foundation of semantic knowledge representation. They are crucial for reasoning on domain knowledge. This section starts giving a set of definitions in order to properly achieve a common understanding of what is meant by an ontology throughout this paper.

Definition 1. *Ontology is the metaphysical study of the nature of being and existence.*

Since the term "ontology" originates from the field of philosophy, definition 1 is to be seen in this context. Ontology tries to find answers for questions like the following ones:

What characterizes being?
Eventually, what is being?

Of course, this definition is not helpful concerning the field of semantic web services. Therefore, definition 2 is more appropriate for an ontology in the context of computer science.

Definition 2. *An Ontology in computer science is a controlled vocabulary and associated phrasings, i.e., representation language, used to express the content of a particular domain or field of knowledge.*

When using the term "ontology" in the following, it is meant in the context of computer science as defined in definition 2.

Additionally, the term "upper ontology" occasionally shows up. When speaking of an upper ontology, a special kind of ontology is meant. This kind of ontology usually describes very general concepts that are orthogonal to specific domains. The *Dublin Core Metadata Element Set* [4] is an example for general descriptions of resources including such elements as e.g. title, creator, subject, language, and date.

Generally, an ontology defines different types of elements. These are *concepts* and *individuals* (sometimes also referred to as *classes* and *instances*), *attributes*, and different kinds of *relations* (e.g. *is-a* relations, *part-of* relations, amongst others). Additionally, a set of grammar rules can be defined in order to express logical constraints.

Ontologies are usually related to a corresponding data model represented in XML. Reasoning engines and query languages operate on top of this data model. A reasoning engine is a software system capable of gaining further knowledge based on the information included in the ontologies using e.g. first-order logic. Query languages facilitate querying an ontology. An exemplary query might be: *What are the sub-concepts of person?* Please note that even though one particular representation language for an ontological data model is described later in this section (OWL), a detailed survey of reasoning engines and query languages is out of scope of this paper.

2.1 Related concepts in Knowledge Representation

As stated above, an ontology is a concept intended for knowledge representation. Additionally, there are a number of similar concepts for representing knowledge. These include *taxonomies* and *thesauri*. This section intends to demarcate the different concepts and to pinpoint their individual properties.

Taxonomy A taxonomy is a primarily hierarchically structured terminology. It contains a number of terms, which are arranged in a hierarchy. A common example for this is the biological taxonomy of species. In some cases of a taxonomy, it is permitted that one term is child of not only one other term, but of two or more.

Thesaurus A thesaurus is a hierarchical structure of terms similar to a taxonomy. Additionally, there is a very limited set of relation types between terms available. Common relation types are for instance the *similarity* and the *synonymity* of terms. Consequently, only information regarding the similarity and synonymity of two terms in the vocabulary can be expressed apart from the hierarchical relation.

Compared with taxonomies and thesauri, ontologies are the most expressive and powerful approach for knowledge representation. They allow an arbitrary number of different types of relations and, furthermore, allow to define logical constraints in order to define certain restrictions on the vocabulary.

2.2 The Resource Description Framework (RDF)

The Resource Description Framework (RDF) provides basic fact-stating facilities [5,6]. With RDF, any resource (identified through an URI¹) can be described. Similarly to natural languages, these descriptions follow the scheme *subject, predicate, object*. This scheme is called a *triple*.

¹ Uniform Resource Identifier

A basic fact-stating example is depicted in listing 1.1. The subject constitutes the resource to be described (in this case: *http://example.org/*). Next, the predicate expresses a relationship between subject and object (here: "has creator"²). Finally, the object is defined ("Torben").

```

1 <rdf:Description rdf:about=" http://example.org/">
2   <dc:Creator>Torben</dc:Creator>
3 </rdf:Description>

```

Listing 1.1. Simple example of stating a fact in RDF.

On top of RDF, there exists an extension called *RDF Schema*. RDF extended by RDF Schema (RDFS) can be seen as a vocabulary description language providing class- and property-structuring facilities. Hence, it is possible to define classes, sub-classes and properties of classes. This allows a more flexible way of structuring a vocabulary compared to simple fact-stating of RDF without RDFS.

Listing 1.2 shows an exemplary RDFS file (in XML syntax). First of all, the basic namespaces are defined. Then, three classes *Country*, *Person*, and *Referee* are defined, whereas *Referee* is a sub-class of *Person*. Next, the property *Nationality* is defined for a *Person* (domain) and has values of the class *Country* (range). Finally, two instances of the class *Country* are created.

```

1 <rdf:RDF
2   xmlns:rdf = " http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:rdfs = " http://www.w3.org/2000/01/rdf-schema#"
4   xml:base = " http://example.org/schemas/myschema">
5
6   <rdfs:Class rdf:ID=" Country">
7     <rdfs:label>country</rdfs:label>
8     <rdfs:comment>The class of all countries</rdfs:comment>
9   </rdfs:Class>
10
11  <rdfs:Class rdf:ID=" Person">
12    <rdfs:label>person</rdfs:label>
13    <rdfs:comment>The class of all people</rdfs:comment>
14  </rdfs:Class>
15
16  <rdfs:Class rdf:ID=" Referee">
17    <rdfs:label>referee</rdfs:label>
18    <rdfs:comment>The class of all referees</rdfs:comment>
19    <rdfs:subClassOf rdf:resource="#Person" />
20  </rdfs:Class>
21
22  <rdf:Property rdf:ID=" Nationality">

```

² The namespace *dc*: references the previously mentioned *Dublin Core* elements as defined in [4].

```

23 <rdfs:label>nationality</rdfs:label>
24 <rdfs:comment>Nationality of a person</rdfs:comment>
25 <rdfs:domain rdf:resource="#Person" />
26 <rdfs:range rdf:resource="#Country" />
27 </rdf:Property>

29 <Country rdf:ID="Brazil" />
30 <Country rdf:ID="TrinidadTobago" />

32 </rdf:RDF>

```

Listing 1.2. An RDFS example.

Please note that RDF(S) alone has not the complete expressiveness of a typical ontology representation language. A number of logical constraints cannot be defined using RDF(S). This results in the inability of expressing the following cases (amongst others):

- logical combinations of classes (intersection, union, complement)
- advanced attributes of properties (transitive, symmetric, functional, inverse)
- restrictions on local properties (e.g. one value must come from a particular class or at most 11 values are allowed)
- equivalence and disjointness of classes

As a result of the last issue, notion of contradiction might be impossible with the limited set of information expressible using RDF(S). Consider the following example: the range of a property is both the class *Metal* and the class *Plant*. Based on the information stated using RDF(S), the reasoner has to assume that all values of this property are members of the class of plants *and* of the class of metals. However, there can never be any value for this property since there exists no instance that can be member of both classes. Problematically about this is, that a reasoner is unable to detect this contradiction, because it is not possible to state that *Metal* and *Plant* are disjoint classes.

2.3 The Web Ontology Language (OWL)

The Web Ontology Language is an ontology representation language standardized by the W3C. Ontology representation languages have evolved over the years. Originally, two major projects were involved in the development of two separate ontology representation languages, which later were incorporated into OWL. These projects were those of the *Ontology Interchange Language* (OIL) and the *DARPA Agent Markup Language* (DAML). Both projects were merged into the *DAML+OIL* project. The resulting ontology representation language was submitted to the World Wide Web Consortium and was adopted by the *WebOnt* working group as the basis for their development of the Web Ontology Language.

OWL is conceptionally based on top of RDFS. Practically, it is realized as vocabulary extension of RDF(S). This can also be seen in figure 2. All major classes of OWL are derived from corresponding RDFS base classes.

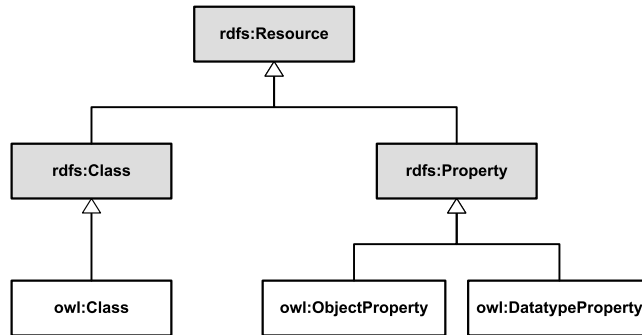


Fig. 2. Inheritance of OWL classes from RDFS pendants. (Figure adapted from [7])

Considering the example depicted in listing 1.3, OWL enables the definition of more advanced aspects than in RDF(S). Following the RDF header, the ontology itself is defined. Next, the classes *Country* and *Person* are declared to be disjoint. Two object properties *InNationalTeam* and *HasCitizen* are defined, whereas *HasCitizen* is additionally declared as the inverse property of *Nationality*³. Finally, a new class *NationalTeamMember* is defined, having precisely those members of *Player* that have exactly one value for the property *InNationalTeam*.

```

1 <rdf:RDF
2   xmlns:owl = "http://www.w3.org/2002/07/owl#"
3   xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4   xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
5   xmlns:xsd = "http://www.w3.org/2001/XMLSchema#">
6
7   <owl:Ontology rdf:about="">
8     <rdfs:comment>An example OWL ontology</rdfs:comment>
9     <rdfs:label>Soccer World Cup Ontology</rdfs:label>
10
11    <owl:Class rdf:about="Country">
12      <owl:disjointWith rdf:resource="#Person" />
13    </owl:Class>
14
15    <owl:ObjectProperty rdf:ID="InNationalTeam">
16      <rdfs:domain rdf:resource="#Player" />
17      <rdfs:range rdf:resource="#NationalTeam" />
18    </owl:ObjectProperty>
19
20    <owl:ObjectProperty rdf:ID="HasCitizen">
21      <rdfs:domain rdf:resource="#Country" />

```

³ Range and domain have, of course, been swapped compared to the property *Nationality*.

```

22     <rdfs:range rdf:resource="#Person" />
23     <owl:inverseOf rdf:resource="#Nationality" />
24 </owl:ObjectProperty>

26 <owl:Class rdf:about="#NationalTeamMember">
27   <rdfs:subClassOf>
28     <owl:Restriction>
29       <owl:onProperty rdf:resource="#InNationalTeam" />
30       <owl:cardinality rdf:datatype="http://www.w3.org
31         /2001/XMLSchema#nonNegativeInteger">
32         1
33       </owl:cardinality>
34     </owl:Restriction>
35   </rdfs:subClassOf>
36 </owl:Class>

37 </owl:Ontology>

39 </rdf:RDF>

```

Listing 1.3. An OWL example.

Variations of OWL Since the universal expressivity has a critical impact on the practical computability, three differently powerful variations of OWL are available. Therefore, the variations differ in terms of expressivity. These variations are:

- OWL Full
- OWL DL (Description Logic)
- OWL Lite

OWL Full includes all language primitives and, thus, poses the most powerful OWL variation. Unfortunately, this high level of expressiveness leads to computational undecidability. Advantageous about OWL Full is, however, that it is the only variation fully upward compatible with RDF. I.e. any correct RDF document is, at the same time, a correct OWL Full document.

OWL DL on the other hand is not fully compatible with RDF(S) anymore. The language is partially restrained, enabling efficient reasoning support. Some of the limitations compared to OWL Full are described in the following. Please note that the list of limitations is not exhaustive. For instance, in OWL DL it is disallowed to define cardinality restrictions on transitive properties. Moreover, the vocabulary has to be partitioned, meaning that a particular resource is only allowed to be either a class, a datatype, a datatype property, an object property, an individual, a data value or part of the built-in vocabulary at a time. Additionally, the partitioned resources have to be explicitly typed. If a certain resource is e.g. referred to as a class, it is necessary to explicitly define this resource as a class. Consequently, *all* named resources that are used in a OWL DL document have to be explicitly defined.

The least powerful variation is *OWL Lite*. OWL Lite is a real subset of OWL DL. It is supposed to be easier to learn and was also intended to push the development of tools. Again only a subset of all limitations is presented. Additionally to the limitations of OWL DL, cardinalities different from 0 or 1 are excluded as well as e.g. enumerated classes and disjointness statements. Please refer to [7] for further information on limitations of the OWL variations.

3 Semantic Service Descriptions

Any given service has a number of functional and non-functional properties. Furthermore, there usually exists additional information about the service such as e.g. legal information about the service provider that might be of interest when describing a service. As explained in the previous sections, it is a non-trivial challenge to semantically describe all of these capabilities of a web service and, therefore, to enable (semi-)automated service discovery and composition.

Different technologies exist that enable semantic annotation of web services in order to tackle this problem. Currently, none of the eligible technologies has been declared as standard (e.g. by W3C) so far. However, this paper intends to describe only one of the available technologies (OWL-S) in detail, whereas competing technologies such as WSDL-S (Web Service Semantics) [28] and WSMO (Web Service Modeling Ontology) [29] are only named. Please have a look at [16], which provides an excellent overview and an evaluation of the three most popular technologies enabling semantic description of web services: OWL-S, WSMO and WSDL-S.

3.1 OWL-S: An Upper Ontology for Services

The previous section described how ontologies can be defined in order to be processable by a computer. This definition of ontologies is not restricted to concrete domain ontologies capturing information about a certain business domain for instance. An ontology representation language such as OWL can also be used in order to define an upper ontology capturing different aspects that are orthogonal to specific domain aspects. OWL-S is an upper ontology for services. Therefore, it provides a set of concepts necessary for the semantic description of (web) services. OWL-S originates from DAML-S, and is developed and maintained by the OWL-S Coalition (former DAML-S Coalition).

According to OWL-S, a service is described by the following aspects, each of which is usually captured in a separate XML file.

- The *Service Profile* describes what the service does.
- The *Service Model* describes how to use the service.
- The *Service Grounding* describes how to access the service.

The different aspects reflect different views on a service defining a set of details relevant for this particular view. The specifics of each view is described in the following sections. Listing 1.4 shows an exemplary OWL-S service definition. The following sections contain listings continuing this example.

```

1 <service:Service rdf:ID="WCTicketPurchaseService">
2   <service:presents rdf:resource="Profile.owl#
      Profile_TicketPurchase_Service" />
3   <service:describedBy rdf:resource="Process.owl#
      BasicTicketPurchase" />
4   <service:supports rdf:resource="Grounding.owl#
      BasicTicketPurchaseServiceGrounding" />
5 </service:Service>

```

Listing 1.4. OWL-S definition of a World Cup ticket purchasing service (*Service.owl*).

Service Profile The Service Profile is intended to be used as an advertisement for a service in a service repository. It provides information about the service, which are rather general. Accordingly, this information is only used for selection of a service and not for detailed planning of interaction with it.

Information captured by the Service Profile includes *contact information* of the service provider, a *categorization* of the service, a set of *non-functional properties* such as QoS constraints, and, finally, a *functional description* of the service. The functional description specifies *inputs*, *outputs*, *preconditions*, and *effects*, which often are collectively labelled as *IOPEs*. An exemplary definition is shown in listing 1.5.

```

1 <profileHierarchy:TicketSelling rdf:ID="
      Profile_TicketPurchase_Service">
2   <service:presentedBy rdf:resource="WCTicketPurchaseService"
      />
3   <profile:has_process rdf:resource="Process.owl#
      BasicTicketPurchase" />
4   <profile:serviceName>WC_Ticket_Resale</profile:serviceName>
5   ...
6   <profile:contactInformation>...</profile:contactInformation>
7   <profile:hasInput rdf:resource="Process.owl#
      BasicTicketPurchaseFirstName" />
8   <profile:hasInput rdf:resource="Process.owl#
      BasicTicketPurchaseLastName" />
9   ...
10  <profile:hasPrecondition rdf:resource="Process.owl#
      BasicTicketPurchaseCCExists" />
11  ...
12 </profileHierarchy:TicketSelling>

```

Listing 1.5. Excerpt of an OWL-S Service Profile for the ticket purchasing service (*Profile.owl*).

When considering another example of a service to buy a certain good (e.g. a ticket or a book), a means of payment is, usually, required. Let us, in this case,

consider the use of a credit card. So, the corresponding Service Profile would have to define the IOPEs for the use of a credit card. The credit card has to be valid in order for the precondition to be fulfilled. The service, then, takes the credit card number and its expiration date as input. As output, a receipt is generated by the service. The effect, after correct execution of the service, is that the credit card is charged.

Service Model (Processes) After selection of a service based on the Service Profile, the Service Model is consulted for information on the interaction with the service. An available interaction scenario with the service is called a *process*. Formally, a process is a sub-concept of Service Model. Therefore, the term process is often used instead of Service Model. The provided process description is used by the requestor to coordinate interaction with the service.

There are three different types of processes (interaction scenarios) available. These are:

- *Atomic* process
- *Composite* process
- *Simple* process

Atomic processes are directly invocable and execute in a single step. Thus, the interaction essentially consists of an input message containing values for the input parameters and an output message returning the results. A composite process specifies a more complex interaction with the service. Listing 1.6 partially shows the XML structure of an exemplary atomic process.

A composite process consists of a number of logical steps, which again represent any kind of process. In order to model control flow, a number of control constructs (*Sequence*, *Split*, *Split-Join*, *Any-Order*, *If-Then-Else*) is available to compose the sub-processes. The composite process is not directly invocable. Therefore, the subsequent steps have to be followed by the service requestor according to the defined control flow. Since any composite process is based on a set of concrete atomic processes, these processes have to be grounded using a *Service Grounding* in order to be invocable.

A simple process should be chosen in case it is desired to provide a different abstract view on an atomic process or to provide a simplified representation of a composite process. The definition of a simple process is to be seen as an abstraction mechanism. Such an abstraction can be useful for hiding certain details of a process model that may be either irrelevant for certain purposes or confidential.

Similarly to the Service Profile, the process description also contains information about IOPEs. However, these might be defined more fine granular than those in the Service Profile. In any case, consistency between the two descriptions has to be maintained either manually or by a supporting tool. Alternatively, it is possible to define the IOPEs once and, then, reference them (this can be seen in listing 1.5). Please note, that preconditions and effects have to be defined using logical formulas (using e.g. KIF [12], PDDL [13], SWRL [11]). [9] proposes a syntax for integration of expressions into OWL ontologies.

```

1 <process:AtomicProcess rdf:ID="BasicTicketPurchase">
2   <process:hasInput>
3     <process:Input rdf:ID="BasicTicketPurchaseFirstName">
4       <process:parameterType rdf:datatype="http://www.w3.org
5         /2001/XMLSchema#anyURI">
6         AnyOntology.owl#FirstName
7       </process:parameterType>
8     </process:Input>
9   </process:hasInput>
10  <process:hasInput>
11    ...
12  </process:hasInput>
13  <process:hasPrecondition>
14    ...
15  </process:hasPrecondition>
16  ...
17 </process:AtomicProcess>

```

Listing 1.6. Excerpt of an OWL-S Process definition for the ticket purchasing service (*Process.owl*).

Service Grounding Grounding means to map a service’s description to a concrete service. An OWL-S service description might apply to a number of different implementations. Hence, different languages and technologies such as e.g. WSDL [1] are involved and have to be interfaced. Of course, it is conceivable that multiple groundings are available for a service description, i.e. the service is implemented multiple times. In OWL-S a Service Grounding defines how the abstract semantic description of a service is to be interpreted in order to technically execute the actual service. This is accomplished by defining how to format inputs and outputs as messages and how to exchange these messages.

Grounding OWL-S on WSDL is very common. OWL-S suggests the complementary use of OWL-S and WSDL. That is, OWL-S aspects can be mapped onto aspects of WSDL. An OWL-S atomic process can be mapped onto a WSDL operation for instance. OWL-S in- and outputs can, furthermore, be mapped onto (parts of) WSDL in-/output messages of an operation. [10] describes in detail how OWL-S can be grounded on WSDL 1.1. Important is, however, that both service descriptions (WSDL and OWL-S) are consistent.

4 Semantic Matching

After a service is semantically described and its advertisement (e.g. OWL-S Service Profile) is stored in a central semantic service repository, it is possible to discover the service when searching for its capabilities (IOPEs plus QoS constraints). Service discovery is a step that, normally, precedes the task of (automated) service composition. When searching for services, *exact* matches of

capabilities is not necessarily the normal case. Thus, a matching engine has to consider inputs and outputs as well as preconditions and effects in order to determine the functional suitability of a service. Matching given QoS constraints usually takes place after a particular set of services has been identified as functionally suitable.

First of all, all inputs and outputs of the query have to be matched against those of the service descriptions. [15] identifies four possible kinds of results, that might occur when matching web services' capabilities. The most straightforward ones are that there is no match at all (*fail*) or there is an *exact match*. Moreover, it is possible that a service provides a subset of the requested capabilities (*subsumes*). This matching result indicates a service that fulfills the needs of the requestor only partially. However, this does not mean that the service is not suitable at all. It is e.g. conceivable that a set of services collectively provide the capabilities originally requested. Similarly, also a superset of the requested capabilities can be provided (*plug in*). That is, a service call potentially returns more results than of particular interest for the requestor. Alternatively, in case of an input parameter, the service would potentially take more input values than provided by the requestor. In any case, this supersetted capabilities do not pose a problem in particular. Instead, by using an efficient matching algorithm, the service landscape can potentially be used more universal than without. This is because, a more advanced matching scheme enables services to be discovered and used that do not exactly fit the requirements but are, nevertheless, applicable to a certain extent.

Still, it is important to notice that pure matching of inputs and outputs alone is not sufficient. In order to illustrate the necessity for considering preconditions and effects as well, an example adapted from [30] is depicted in figure 3.

Scenario: A customer who already bought tickets, calls the callcenter again to modify some details concerning his order. The callcenter employee should, therefore, be able to oversee the customer's order as quickly as possible. A (web) service *GetTicketOrder* is capable of displaying a customer's order based on the customer's address. However, asking the customer for his address is quite time-intensive. Therefore, the phone number of the caller should be utilized in order to determine the caller's address first. Assuming that the caller is located at the billing address, this address should be used as input for the previously described web service *GetTicketOrder*. This simple case of a service composition involves the discovery of a suitable service determining a caller's location.

Now imagine, there are two different services advertised in the service repository taking a phone number as input and providing an address as output. If the matching engine would only match inputs and outputs, it would not matter which service is chosen⁴. In this case, it would be assumed that both services do exactly the same thing. The problem is: in- and outputs do not completely characterize a service's behavior. If we, additionally, consider preconditions and effects, it is possible to tell what the world looked like before the execution of the

⁴ In case QoS constraints are considered, maybe the cheaper or faster one would be chosen.

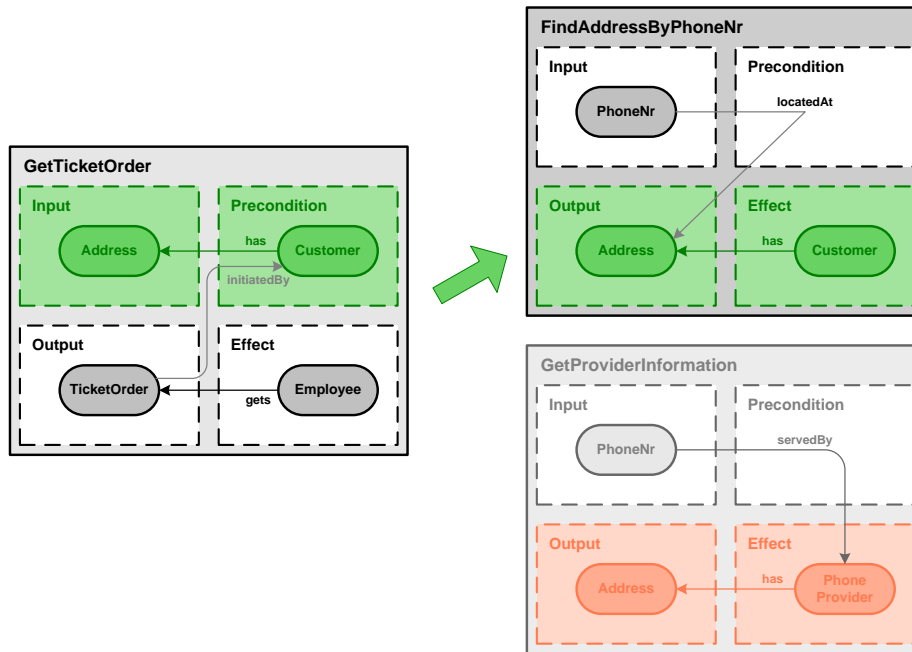


Fig. 3. An example for complete matching of inputs, outputs, preconditions and effects of web services.

service (precondition) and how it was changed (effect). The semantic description of preconditions and effects is, therefore, a proper means to distinguish the behaviour of services. Visualizing all IOPEs as in figure 3 shows that only the service *FindAddressByPhoneNr* is suitable for the given scenario. This is because the other service *GetProviderInformation* has an incompatible precondition and effect. It does not provide the address, where a certain phone number is located, but does return the address of the service provider hosting the phone line.

However, there is still a big challenge in the proper definition of preconditions and effects. Whereas it is rather straight-forward to define inputs and outputs, preconditions and effects have to be defined in a very balanced way. They should not be too general, but also not too specific. This can be very complex, since people involved in advertisement as well as in capturing the requirements for services to be used have to define preconditions and effects using the same granularity. Otherwise, semantic matching of the expressions can become problematic.

5 The METEOR-S Framework

The previous sections introduced the fundamentals of semantic web services. Currently, implementations of frameworks supporting automated discovery and composition of semantic web services are, however, very rare. The *METEOR-S*

project [18,19,20] at the LSDIS Lab, University of Georgia develops one of the most popular and mature frameworks of its kind. Conceptually, the METEOR-S framework enables the definition of *web processes* comparable to workflows. These web processes are composed of web services. *Abstract* process definitions are, then, used to semantically define the behavior of the processes (*METEOR-S Composer*). These abstract definitions (in WS-BPEL [27]) consist of a number of *semantic (service) templates* defining particular abstract operations. Web services with semantic annotations are registered in a repository. During service discovery the requirements of the semantic templates are matched against the capabilities of the registered services (*METEOR-S Web Service Discovery Infrastructure*). Functional as well as non-functional properties of the services are considered. In order to find the services suiting a template best, the framework, first of all, identifies the set of services that functionally fulfill the template’s requirements. Next, the suitable services are ranked according to the non-functional constraints (QoS constraints). Non-functional constraints can either be quantitative or non-quantitative (qualitative) constraints. *Integer Linear Programming (ILP)* [18] is used for the analysis of quantitative constraints. An example for quantitative constraints is: $ProcessCosts \leq \$10000$. In order to satisfy non-quantitative constraints, the *Semantic Web Rule Language (SWRL)* [11] is used. Logical expressions such as preferences for certain service providers before others are, therefore, defined using SWRL.

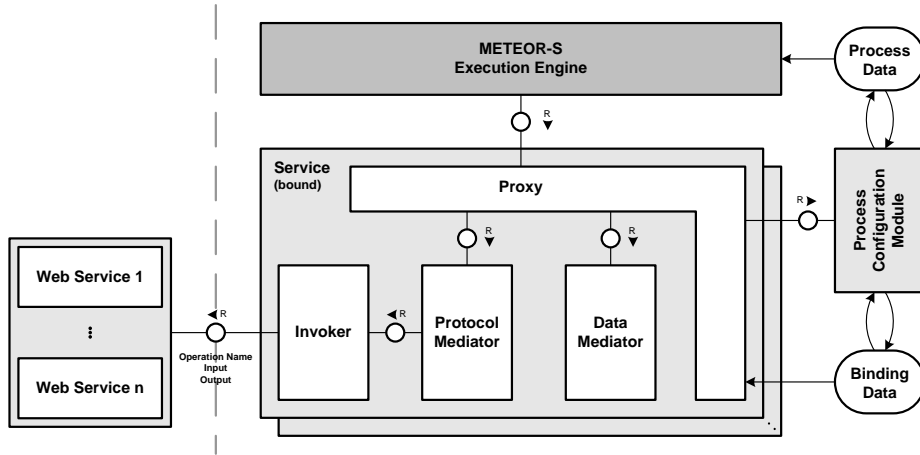


Fig. 4. FMC Block diagram showing the compositional structure of the METEOR-S Execution Environment. (Please see [31] for notational reference)

The remainder of this section aims at summarizing the software architecture of the *METEOR-S Execution Environment* as depicted in figure 4. The execution environment is responsible for dynamic configuration and execution of the process during the entire lifecycle. After the ranked set of services is avail-

able, the most optimal services are selected and bound to the process by the *Process Configuration Module*. Binding a service can be performed either static (at build-time), at deployment-time or at run-time. Most flexible is, of course, binding at run-time. However, losses in performances are to be expected. [18] provides a performance analysis of the different binding options. The *Execution Engine* can in some respects be compared to a workflow engine managing the execution of the processes. Invocation of a service is performed using a *Proxy*, which is dependent on the binding. Mediation takes place in case of protocol and data heterogeneities (*Protocol Mediator*, *Data Mediator*). Due to the use of proxies, the process can be dynamically re-configured on failure of a service.

6 Conclusion

We provided an overview of the field of semantic web services by introducing a reduced subset of available technologies and putting them into context of the concept of semantic web services. Firstly, we pointed out the commonalities as well as the differences between the Semantic Web project and the concept of semantic web services. Generally, this paper is not intended to provide a detailed comparison and evaluation of major existing technologies. Instead, we concentrated on one particular technology for each of the aspects introduced.

Ontologies are the foundation for semantic annotation of web services. They are defined using ontology representation languages. This paper provided a brief overview of the history of ontologies and their representation as well as of related concepts in knowledge representation. Furthermore, we introduced the Web Ontology Language and the Resource Description Framework, which, together, enable the development of domain ontologies. Both languages were illustrated using an example. Next, we provided an overview of OWL-S as an upper ontology for services. It enables the semantic description of web services. The different parts of an OWL-S description - Service Profile, Service Model and Service Grounding - were introduced and described. Important aspects concerning semantic matching of web services' capabilities were discussed in section 4. Finally, we described the architecture of the METEOR-S framework, which is capable of executing workflow-like web processes composed of semantically annotated web services. This framework provides support for dynamic re-configuration at run-time as well as consideration of QoS constraints for service selection. Data and protocol mediation is performed in order to guarantee interoperability of heterogeneous services.

However, the field of semantic web services is, generally, still a dedicated research topic. Consequently, there is a lack of accepted standards. Existing implementations comparable to METEOR-S are very rare and still in development. Additionally, the service landscape is, currently, in most domains still not rich enough in order to enable comfortable automated service composition. Moreover, it is questionable if (fully) automated service composition will *ever* be facilitated with a reasonable effort. There are a number of drawbacks, turning the semantic description of web services into a complex problem. First of all, high quality

domain ontologies do not exist in an acceptable quantity. They are difficult to develop and to maintain. It is not clear who could be responsible for standardization of the domain ontologies. Potential conflicts and inconsistencies between different ontologies are to be resolved. Next, defining preconditions and effects of services in a way that they are universally applicable is extremely difficult. Finding the correct granularity and 'completeness' are complex tasks.

Semi-automated service composition, on the other hand, seems to be a very promising approach. Semi-automatism would enable (human) process designers to efficiently compose a process of services supported by a tool. Such a tool could possibly present a set of preselected services for a certain purpose based on semantic annotations. Compared to fully automated composition, this would be much easier to accomplish. Of course, flexibility at run-time is limited.

References

1. Christensen, E. et al.: Web Services Description Language (WSDL) 1.1. (March 2001)
<http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
2. Berners-Lee, T. et al.: The Semantic Web. *Scientific American*. (2001)
<http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&ref=sciam>
3. Nykänen, O.: W3C FI & W3C Semantic Web. (2002)
<http://www.w3c.tut.fi/talks/2002/0923sw-vtt-on/>
4. Dublin Core Metadata Initiative: Dublin Core Metadata Element Set, Version 1.1: Reference Description. (2004)
<http://dublincore.org/documents/2004/12/20/dces/>
5. Beckett, D., McBride, B.: W3C: RDF/XML Syntax Specification. (February 2004)
<http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>
6. McBride, B.: The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS. in: *The Handbook on Ontologies in Information Systems*, S. Staab, R. Studer (eds.), Springer Verlag. (2003)
7. Antoniou, G., van Harmelen, F.: *Web Ontology Language: OWL*.
8. Horrocks, I., Patel-Schneider, P., van Harmelen, F.: *From SHIQ and RDF to OWL: The Making of a Web Ontology Language*. (2003)
9. OWL-S Coalition: *OWL-S: Semantic Markup for Web Services*.
10. Martin, D. et al.: *Describing Web Services using OWL-S and WSDL*. (2004)
<http://www.daml.org/services/owl-s/1.1/owl-s-wsdl.html>
11. Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosz, B., and Dean, M.: *SWRL - A semantic web rule language combining owl and ruleml*. (2003) <http://www.daml.org/2003/11/swrl/>
12. KIF: *Knowledge Interchange Format - Draft proposed American National Standard (dpans)*. Technical Report 2/98-004, ANS. (1998)
<http://logic.stanford.edu/kif/dpans.html>

13. Ghallab, M. et al.: PDDL - The Planning Domain Definition Language V.2. Technical Report, report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control. (1998)
14. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Importing the Semantic Web in UDDI. (2002)
15. Paolucci, M. et al.: Semantic Matching of Web Services Capabilities. (2002)
16. Schaffner, J.: Paving the Way for Automated Web Service Composition. (2005)
17. Naumenko, A., Nikitin, S., Terziyan, V., Veijalainen, J.: Using UDDI for Publishing Metadata of the Semantic Web. University of Jyväskylä, Finland. (2005)
18. Sheth, A. et al.: The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. (2005)
19. Sheth, A. et al.: METEOR-SWSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. (2005)
20. Sheth, A., Cardoso, J.: Semantic E-Workflow Composition. (2003)
21. Van den Heuvel, W., Maamar, Z.: Moving Toward a Framework to Compose Intelligent Web Services. (2005)
22. Object Management Group: Ontology Definition Metamodel. IBM, Sandpiper Software, Inc. (August 2005)
23. Sirin, E., Hendler, J., Parsia, B.: Semi-automatic Composition of Web Services using Semantic Descriptions. (2002)
24. McIlraith, S., Cao Son, T.: Adapting Golog for Composition of Semantic Web Services. (2002)
25. Ullrich, M., Maier, A., Angele, J.: Taxonomie, Thesaurus, Topic Map, Ontologie - ein Vergleich. v1.4 Ontoprise Whitepaper Series. (2004)
<http://www.ullri.ch/download/Ontologien/ttto13.pdf>
26. Pidcock, W.: What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology, and a meta-model?. Boeing. (2003)
<http://www.metamodel.com/article.php?story=20030115211223271>
27. IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems: Business Process Execution Language for Web Services version 1.1. (2005)
<http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>
28. Akkiraju, R. et al.: Web Service Semantics - WSDL-S. W3C Member Submission. (November 2005)
<http://www.metamodel.com/article.php?story=20030115211223271>
29. Roman, D. et al.: Web Service Modeling Ontology. Applied Ontology 1 (2005) 77106 77, IOS Press. (2005)
30. Weske, M.: Business Process Management Lecture Notes. Hasso-Plattner-Institute Potsdam. (2006)
31. Tabeing, P., Gröne, B., Knöpfel, A.: Fundamental Modeling Concepts - Effective Communication of IT Systems. John Wiley & Sons, Ltd. (2006)