

THE PROCESS OF SEMANTIC ANNOTATION OF WEB SERVICES

Christoph Ringelstein, Thomas Franz, Steffen Staab
ISWeb, University of Koblenz-Landau, Germany
<http://isweb.uni-koblenz.de>
{cringel,franz,staab}@uni-koblenz.de

1 ABSTRACT

Web services are software components that are – in general – distributed over multiple organizations. They provide functionality without showing implementation details for the purpose of *abstracting* from implementation as well as for the purpose of *hiding* private, i.e. organization-internal, processes. Nevertheless, to use a web service one must know *some* of its details, i.e. what it does, what it requires, what it assumes, what it achieves and even, to some extent, how it achieves its purpose. The different web service standards, frequently summarized as WS*, allow web services to be *specified* with descriptions of such details. In this chapter, we argue that one should go beyond WS* and that it is preferable to provide *semantic descriptions*, i.e. specifications that can be understood and correctly interpreted by machines. Thereby, the particular focus of this contribution lies in analyzing the process of *semantic annotation*, i.e. the process of deriving semantic descriptions from lower level specifications, implementations and contextual descriptions. Hence, the concern of this chapter is really orthogonal to most other work which equates web service annotation with web service specification. We illustrate here that this is not the case.

2 INTRODUCTION

Web services are software components that are accessible as web resources in order to be reused by other web services or software. Hence, they function as middleware connecting different parties such as companies or organizations distributed over the Web. Thereby, a party providing a service may not be interested in exhibiting their organization-internal processes to the outside world. A second party consuming such a service may not be interested in analyzing a given web service in order to be able to use it. Therefore, an abstracting description of a web service is necessary to allow for its effective provisioning and use.

The description of a web service needs to include some bare technical information in order that it can be used. This includes:

- 1 *What* it does;
- 2 *How to invoke* the web service (i.e. the used communication protocol);
- 3 *What parameters* to provide to the web service (i.e. its signature);

To embed it into an organizational process it must also contain information about

- 4 *which protocol* should be followed when using the web service (e.g. “register user; then, book journey!”).

A web service user may have some expectations about a service’s

- 5 *properties*, concerning, e.g., security means (e.g. “always use secure communication for my bank account information!”).

This list is by no means complete. One may require further technical descriptions (e.g. transactions) or legal aspects (e.g. contractual issues). However, it indicates that two parties

that plan to cooperate via a web service need specifications of the service allowing them to share and *exploit* technical and non-technical descriptions.

Such sharing of descriptions is extremely difficult if the means of sharing are not standardized and descriptions boil down to verbose textual documents. To simplify use of web services, several properties of web services are described following standardized XML documents (e.g. SOAP, UDDI, WSDL, WS-Security), for further properties standardization activities for XML descriptions are underway (e.g. WS-Transaction, WS-BPEL, WS-Policy) and for yet others there is a discussion whether standardization should be initiated.¹

Exploitation of web service descriptions may occur in various ways. Technical and non-technical descriptions may be used (i) to select a service, (ii) to compose it with other web services, or (iii) to derive relevant properties about a composition of web services (e.g., combined cost or validity for a given specification).

In this chapter, we consider the process of provisioning data about a web service to constitute a **specification** of the web service. At this point, the question arises how a machine may attribute machine-understandable meaning to this metadata. The XML standards (WS*) listed above lack the formal semantics to achieve common interpretation and interoperability of web service annotations. Therefore, we argue for the use of ontologies for giving a formal semantics to web service annotations, i.e. we argue in favor of semantic web service annotations. A **web service ontology** defines general concepts such as *service* or *operation* as well as relations that exist between such concepts. The metadata describing a web service can instantiate concepts of the ontology (Patil, 2004). This connection supports web service developers to understand and compare the metadata of different services described by the same or a similar ontology. Consequently, ontology-based web service annotation leverages the use, reuse and verification of web services.

The *process of semantic web service annotation* in general requires input from multiple sources, i.e. legacy descriptions, as well as a labor-intensive modeling effort. Information about a web service can be gathered e.g. from the source code of a service (if annotation is done by a service provider), from the API documentation and description, from the overall textual documentation of a web service or from descriptions in WS* standards. Depending on the structuredness of these sources, semantic annotations may (have to) be provided manually (e.g. if full text is the input), semi-automatically (e.g. for some WS* descriptions) or fully automatically (e.g., if Java interfaces constitute the input). Hence, a semantic description of the signature of a web service may be provided by automatic means, while the functionality of web service operations or pre- and post-conditions of a web service operation may only be modeled manually.

Benefits of semantic specifications of web services include a common framework that integrates semantic descriptions of many relevant web service properties. In contrast to WS* standards, such a semantic framework allows complex requirements to be queried for (e.g. “Give me all web shop services that offer customer care facilities and use 128-bit encoding of all communication”; cf. (Oberle, 2006)). The benefits harvested from semantic specifications must be traded off against manual semantic modeling, i.e. manual semantic annotation of web services. While the benefits of semantic specifications, i.e. the result of a semantic annotation process, are explored in many papers there are few papers that investigate the efforts that have to be carried out during modeling (cf. some very notable exceptions like (Sabou, 2005;

¹ Cf. http://www.oasis-open.org/committees/tc_cat.php?cat=ws and <http://www.w3.org/2002/ws/>

Zaremba, 2006; Patil, 2004; Hess, 2003)). It is the purpose of this chapter to explain the conceptual gap between legacy descriptions and semantic specifications and to indicate how this gap is to be bridged.

To clarify this point in even more detail: Merriam-Webster online defines “annotation” as

Function: *noun*

1 : a note added by way of comment or explanation

2 : the act of annotating

Most proposals about semantic annotation, such as ones to W3C like (Akkiraju et al. 2005; Battle et al. 2005) only consider “semantic annotation of web services” according to the first meaning of ‘annotation’, some others use ‘annotation’ as a synonym to ‘specification’ and do not intend to *add* to anything. The purpose of this contribution is to investigate the problem of ‘annotation’ in the second sense. For ease of writing and understandability, we never use ‘(semantic) annotation’ to refer to the first sense here, but use ‘(semantic) specification’ or ‘(semantic) description’ instead.

In the following, we start with an application example that functions as a run through example throughout this chapter. It demonstrates an application use case of building a web shop. It also shows how *semantic specifications resulting from semantic annotation may facilitate the development task* of building the web shop by providing a semantics-based development environment.

In the subsequent section, we analyze the purpose of web service specifications in such a development scenario as well as the legacy sources of information about a web service, including textual descriptions as well as the use of WS* documents. They help us to show (i) what semantic specifications we would like to provide and (ii) how to represent them. Eventually, we use this illustration as input for requirements to support the semantic annotation process, as this process needs more and more sophisticated support than is currently offered by semantic environments.

3 USE CASE

As use case scenario we consider the development of a web shop. In order to save time, reduce costs, and improve maintainability, existing software should be reused to implement the shop. Web services are chosen for the implementation as they meet such requirements, amongst others, allowing for easy use of services provided by other companies, e.g. payment services. The web shop should provide a web based user interface, but should also be reusable. Hence, the web shop will be a complex service composed of several other complex services. Ideally, the development process is supported by a web service development environment that can be used to organize the web services from which the web shop is built.

3.1 Architecture

Figure 1 shows parts of the web shop architecture using other web services.

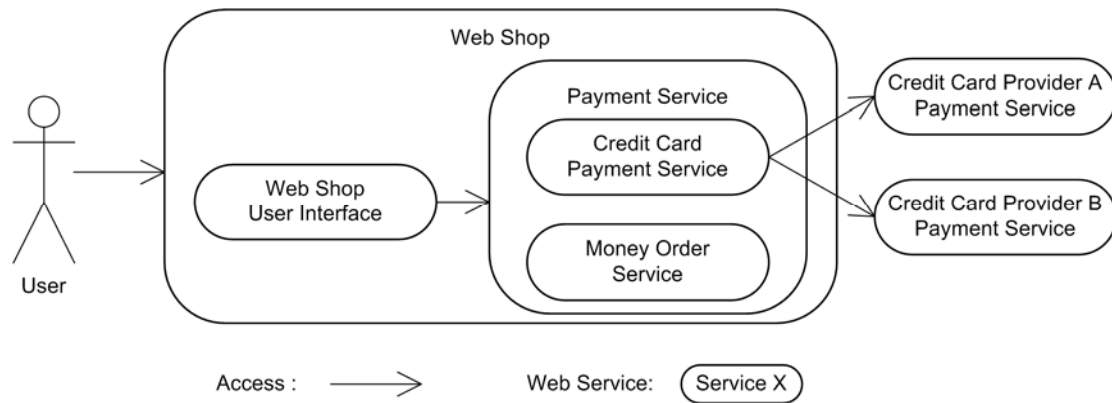


Fig. 1: Web Shop Architecture Example.

The payment service is a central component of the web shop. It is not an atomic service but an aggregation of several specialized services. The aggregated payment service provides a uniform interface partly independent of the interfaces of the underlying services. The specification of this general service has to merge the properties and conditions of all its subsidiary web services' specifications. For instance, a credit card number is essential for credit card payment, but not for money order payment. Internally, the payment service delegates to an appropriate subsidiary web service which can be part of the web shop (like the Money Order Service in the example), or which are services offered by other providers such as the Credit Card Payment Services of provider A.

3.2 Development Process

In order to build the web shop, the developer proceeds as follows: When he needs a specific web service, he queries a web service registry by specifying the desired properties of that web service. For instance, a query specifying the effect of a successful money transfer has as result a list of all payment services. A query with the condition that only credit card payment can be used will get as response all credit card payment services. In the ideal case, the query processor finds a service that matches all requirements, but in most cases it only finds services that fulfill some but not all requirements, so the engineer has to make some adjustments to use them.

For the previously illustrated development process to work, the registry needs specific information about the services it manages. As an example of which information that is, we examine some relevant properties of the Payment Service. The Payment Service provides three operations. The first operation *MethodSupported* takes as input the kind of payment method and asks the service if a specific payment method (like credit card or money order) is supported. The output of this method is true or false depending on whether a service for the payment method exists and is reachable. The second operation *ValidityCheck* checks the validity of payment information. The inputs of the operation are the verifiable details of the payment (like the existence of accounts, the validity of credit cards, and so on) and the output is the validation of the information. The third one is the *Transaction*-operation that executes a specified transaction. For this purpose, it collects all details of the transaction to be executed and returns a transaction identification number. Since web services are stateless, the *Transaction*-operation needs all information, even if the *ValidityCheck*-operation has been called before with the same information.

3.3 The Up-and-Running Web Shop

In Figure 2, the workflow of the payment process is shown. The workflow starts at the user interface, where the first action is the user interface calling the *MethodSupported*-operation of the payment service using the interface description specified in the service specification of the payment service. The payment service returns an answer, which the user interface can interpret, because the answer format is also described in the specification. In the next step, the *ValidityCheck*-operation as specified in its interface description is called. The payment service itself calls the matching operation of the specific credit card payment service and passes response to the user interface. The payment service can successfully call the operation, because the input and output interfaces of the credit card payment service are also specified. The final call of the user interface calls the *Transaction*-operation handing over the result of the validity check. Because the output of one operation is required by the second one, the possible sequences of web service invocations are limited. The call of the *Transaction*-operation has the same invocation sequence as the call of the *ValidityCheck*-operation.

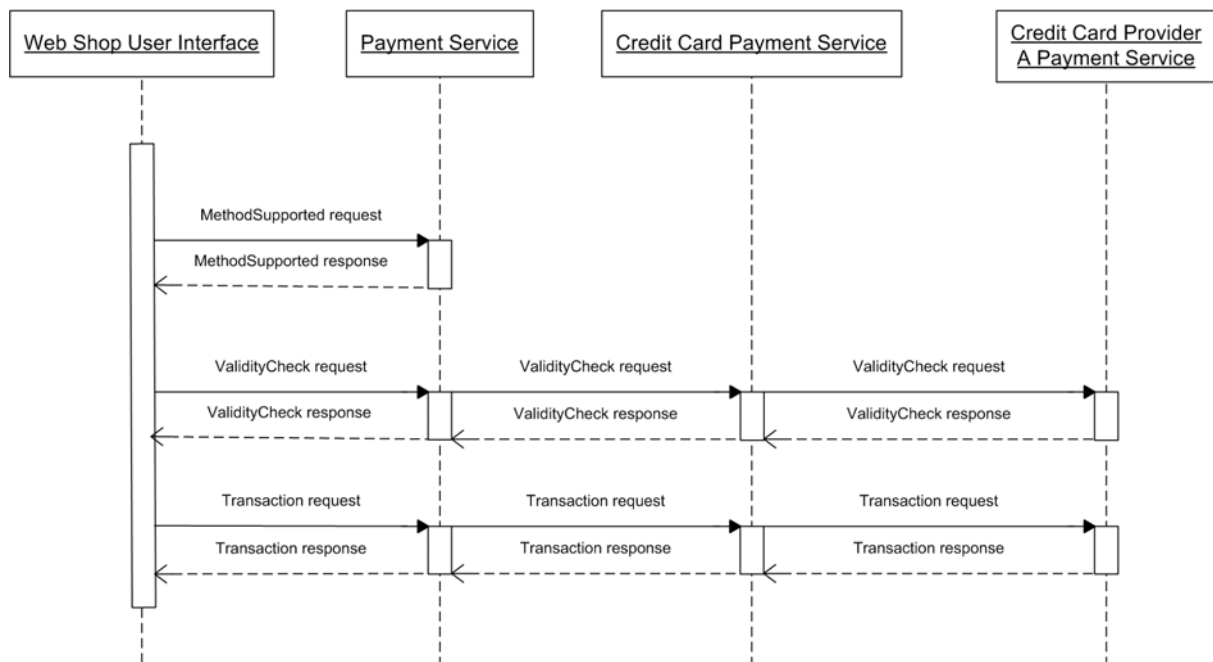


Fig. 2: Payment Workflow.

4 ANNOTATING WEB SERVICES

We now analyze different aspects of annotating **web services** by asking four questions relevant in this context:

- Which information sources containing information relevant to the specification of web services exist?
- Which descriptions are needed to manage and reuse web services?
- How should the metadata be represented?
- How can the description be exploited?

In the following, we give answers to these questions and illustrate central aspects by means of application example.

4.1 Purpose of Web Service Annotation

The goal of the **annotation** of web services is to simplify the management of services in the web by partial automation of management tasks like composition or invocation of web services. A challenge in automating web service management is that most information about web services is in formats only humans understand, e.g. documentations. Thus, a machine interpretable web service description is needed that fills the semantic gap between existing information about web services and the information needed. Semantic descriptions are a means to bridge this gap in the form of semantic metadata describing the semantics of an item of information in a machine interpretable format. As mentioned in the introduction, a web service description specifies the properties of a web service by metadata. A semantic description does it by semantic metadata that is ontology-based and connects a web service property with the corresponding concept defined in an ontology.

4.2 Sources of Information

Various information sources of different quality describing different web service properties exist. In the following, we give a list of such sources and analyze the provided information:

- One of the most common information sources is the documentation of a web service. The documentation contains selected information about a web service. In most cases, the information given in the documentation is defined in natural language and thus not formalized and not in a machine-understandable form. Determining actual properties of a web service from such text becomes difficult or even impossible for a machine.
- Source code contains all information about operations including all inputs and outputs. The extractable information differs between programming languages. For instance, a method signature in Java reveals type information about input and output parameters. Programming languages with less restrictive signatures do not contain such type information. Furthermore, source code defines programming logic which contains information about the web service's functionality, the web service's preconditions that must be fulfilled before the execution of an operation, and effects of the execution of an operation of the service.
- API descriptions consisting of interface declarations in source code and explanations in natural language are another common information source. In contrast to the documentation, API descriptions focus on particular operations of a web service and not on the web service as a whole. In contrast to source code, the API description can be more detailed and often is focused on the relevant methods. In principle the explanations are as expressive as the documentation and share the same disadvantages. Furthermore, API descriptions often contain signatures copied from the source code.
- Comments included in the source code denote another information source that can also be used to determine information about operations as well as about the functionality of the whole web service. Comments, however, inherit the pros and cons of documentations and API-descriptions. If the source code is not accessible, the comments are also not accessible.
- Software specifications and UML-diagrams are information sources that provide the same kinds of information as API descriptions.
- Other information sources are contracts like end user license agreements (EULA). As they are written in natural language, interpreting them is as problematic as interpreting documentations or other sources written in natural language.
- Background knowledge is an information that is usually not provided with a web service. For instance, background knowledge is programming knowledge needed to understand technical aspects, knowledge about the application area (like payment),

and about relevant laws that clarify domain specific properties. All such background knowledge could be useful for specifying properties of web services.

4.3 Non-semantic Web Service Description

In the following subsections we examine the reuse of non-semantically described web services. We assume, the developer of the web shop in our use case scenario wants to integrate a new credit card payment service. The information sources the developer has about the new service are the service's documentation and API descriptions. In the following, we analyze which information is provided and how it can be used.

4.3.1 Description Example: Documentation and API-Description

Listing 1 shows a cutout of the documentation provided for the payment service as used in our example. The cutout describes the *Transaction*-operation and refers to the API-description, here called format description. The description of the operation is very short and does not contain very valuable information as it only inaccurately characterizes input parameters by stating that all transaction information including the payment service provider are submitted to the service. The specification of the output parameter, the transaction identification number, is more detailed because only one parameter exists and is mentioned in the description, but the exact format of it is also missing. While information about preconditions that must hold before the operation of a web service can be executed and effects of the operation's execution are completely missing in the sample documentation, some of these can be derived from respective laws and the contracts, like the fact that specified accounts must be valid.

```
Transaction requests submit all transaction information to the Payment Service and receive in return a transaction identification number. The Transaction is executed by a specified payment service provider.
```

```
A list of all needed and optional parameters of transaction requests are listed in the Transaction Request Format description.
```

Listing 1: Example Documentation of a Web Service Operation²

A cutout of an API description is depicted in Listing 2. It gives the names of the parameters and a detailed description of assigned values. For instance, the description states that the value of the *payerName*-parameter must be a combination of the payer's last name and first name separated by a comma. The *payerAccountID* is an optional parameter that is only needed for credit card payment and must then contain the payer's credit card number. In the case of the money order payment this parameter can be skipped or must be empty. Like the documentation, this API-description does not contain any statements about preconditions or effects relevant to the *Transaction*-operation. So if the web shop developer does not have that knowledge himself, he has to gain the needed information from other sources as listed before.

```
Transaction Request Format
```

```
The following list shows all valid name-value pairs that must or can be used in a transaction request:
```

<u>Name</u>	<u>Description</u>
payerName	The name of the payer. The format is: "LastName, FirstName"
payerAccountID	If credit card payment is chosen, this must be the payer's credit card number. If money order payment is

² This example follows the style of a Google API description. (<http://www.google.com/apis/download.html>)

chosen this parameter must be empty or skipped.

...

Listing 2: Cutout of an Example API-Description

Using only the documentation and the API-description, the web shop engineer in our example would have to collect the information about the interfaces of the web service's operations without tool support. Furthermore, the developer would have to make all decisions about which web service to use for which task and how to use it, in advance and by himself. If the preconditions and effects are not characterized in detail, the engineer needs domain knowledge to avoid mistakes and to correctly reuse the service.

4.3.2 Description Example: Operation Interface Description

To enable reuse and discovery of web services, information about a web service's interface is needed. The interface is the property of a web service which can be described very simply as it consists of all methods a web service provides as an endpoint for machine-to-machine communication, and the messages which those methods accept and which they return. A method interface can be described with languages like the Web Services Description Language (WSDL), which is a XML based general-purpose language designed for the description of web service interfaces (Christensen, 2001). WSDL is a widespread language adopted by many web service supporting application servers which often facilitate the dynamic generation of WSDL files out of the source code or java class descriptions. Because WSDL is designed for the syntactical specification of web service interfaces, a WSDL description does not include any semantic information about the service.

Listing 3 illustrates the use of WSDL to describe the operation interfaces of a simplified version of a credit card payment service. The first lines (lines 2-6) of the description declare used namespaces, which define the used XML-elements and XML-types, and facilitate readability. After the namespace specifications, new complex data types are declared from line 7 to line 23, e.g. the validity information type that holds all needed information for a validity check. WSDL describes operations of a web service as network endpoints (Christensen, 2001) that communicate by retrieving and sending messages which contain the data delivered to or returned from the operation (lines 24-29). From line 31 to line 35 port types are defined and associated with corresponding operations, which are connected with the appropriate input and output messages. After the port types are defined the binding between the web service's endpoints, their port types, and the kind of their supported invocation is specified (in line 38 to line 55). The last part of the listing (from line 56) describes the web service and contains the locations (as URI) where the service's operations can be reached. Some further information can be gained from optional comments, but in general other information sources are needed to gain all knowledge needed to automate the reuse of a web service described with WSDL.

```
01 <?xml version="1.0"?>
02 <definitions name="SimplifiedCreditCardPayment"
03     targetNamespace="http://example.org/creditcardpayment.wsdl"
04     xmlns:tns="http://example.org/creditcardpayment.wsdl"
05     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
06     xmlns="http://schemas.xmlsoap.org/wsdl/">
07   <types>
08     <schema targetNamespace="http://example.org/creditcardpayment.xsd">
09       <element name="ValidityInformation">
10         <complexType>
11           <all>
12             <element name="OwnerName" type="xsd:string"/>
```

```

13         <element name="CreditCardID" type="xsd:string"/>
14         <element name="ValidityDate" type="tns:Date"/>
15         ...
16     </all>
17 </complexType>
18 <element name="Date">
19     ...
20 </element>
21 ...
22 </schema>
23 </types>
24 <message name="ValidityCheckInput">
25     <part name="body" type="tns:ValidityInformation"/>
26 </message>
27 <message name="ValidityCheckOutput">
28     <part name="body" type="string"/>
29 </message>
30 ...
31 <portType name="SimplifiedCreditCardPaymentPortType">
32     <operation name="ValidityCheck">
33         <input message="tns: ValidityCheckInput"/>
34         <output message="tns: ValidityCheckOutput"/>
35     </operation>
36     ...
37 </portType>
38 <binding name="SimplifiedCreditCardPaymentBinding"
39     type="tns:SimplifiedCreditCardPaymentPortType">
40     <soap:binding style="rpc"
41         transport="http://schemas.xmlsoap.org/soap/http"/>
42     <operation name="ValidityCheck">
43         <soap:operation
44             soapAction="http://example.org/creditcardpayment/ValidityCheck"/>
45         <input>
46             <soap:body use="encoded"/>
47         </input>
48         <output>
49             <soap:body use="encoded"
50                 namespace="http://soapinterop.org/xsd/"
51                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
52         </output>
53     </operation>
54     ...
55 </binding>
56 <service name="SimplifiedCreditCardPaymentService">
57     <documentation> This is a ... </documentation>
58     <port name="SimplifiedCreditCardPaymentPort"
59         binding="tns:SimplifiedCreditCardPaymentBinding">
60         <soap:address location="http://example.org/creditcardpayment"/>
61     </port>
62     ...
63 </service>
64 </definitions>

```

Listing 3: Description of a Simplified Payment Service in WSDL

Using WSDL, an engineer developing the web shop gets detailed descriptions of web services telling him how to invoke operations on web services and which format the returned answers have. This information is detailed enough to allow tools to support the engineer adjusting interfaces to access the web service, but the engineer still has to collect information about service functionality, execution and so on from other sources.

4.4 Semantics of Descriptions

We previously analyzed descriptions of web services. In the following, we inspect which kind of information serves which annotation purpose in connection with which kind of web service **semantics**. We distinguish between four categories of semantics:

- The first category *data semantics* includes the semantics of all inputs and outputs of a web service (Patil et al., 2004). For instance, the specification that the credit card payment service has a validity date parameter which has a specific format like “month/year”.
- The category *protocol semantics* includes the semantics related to the protocol that defines dependencies between web services and their operations, e.g. the order of execution. In the web shop example, the protocol defines that the payment information has to be validated before the transaction-operation can be called.
- The category *functional semantics* covers all semantics related to the functionality of a web service (Patil et al., 2004). For example, a credit card payment service should be annotated with metadata that refers to an agreed terminology of tasks, describing e. g. that a money transfer takes place.
- The last category, *non-functional semantics*, bundles all semantics of non-functional properties of a service. All properties of a web service that are not directly related to the web service’s functionality are non-functional properties. For example, the description of cost parameters, e. g. of the fee that must be paid to use the credit card payment service falls under this category. Further examples are special properties like security, quality of service, and so on.

The introduced categories of semantics are used to specify web services semantically. A service description specifying all those semantics for a service is called **semantic service specification**. The semantic service specification contains all information needed for the service advertising and discovery. The combination of data semantics with the location of the service (specified by an URI) and semantics of the used middleware protocols is a special subset of the semantic service specification called **semantic grounding specification**. In other words, the grounding specification describes how to access a web service.

4.5 Semantic Web Service Description

WSDL has been designed to specify the endpoints of web services, but lacks the ability to express any kind of semantics. **Knowledge representation languages** allow to give web service descriptions a formal semantics since they enable the definition of concepts, classes, types, and relations to specify conceptualizations or ontologies, respectively. While many such description languages exist with varying expressivity, we here introduce the language Flora-2 (Yang, 2005). The purposes of Flora-2 are the definition of ontologies and the description of all metadata relevant to web services discovery and invocation. Interoperability between web service descriptions is achieved if they commit to the same ontology, i.e. express a web service by referencing concepts and relations defined in a particular ontology. Consequently, knowledge representation languages enable web service descriptions including semantics.

There exist many other languages that can be used to annotate services. Some are specialized for the annotation of web services (see related work section), but for many of these languages fundamental tools, e.g. reasoners, are not or only partially implemented. In the following we use Flora-2 as it is supported by a fully implemented reasoner³ and is powerful in expressing

³ Download location for the XSB Flora-2 reasoner: <http://flora.sourceforge.net/>

schemata. A weakness of Flora-2 is its capability to depict processes. A – somewhat inelegant and not very powerful – workaround can be achieved by the means of reifications. The reification is implemented as a class that enables the manipulation and reasoning about its attribute's values. A term is reified by instantiating the reification class.

4.5.1 Data and Protocol Semantics

To specify data semantics of a web service operation, connections between ontology concepts and the corresponding input and output parameters, and the parameter types have to be build. The connections are made by instantiating appropriate ontology concepts for each parameter and web service operation.

Protocol semantics describe the protocol to follow for accomplishing a specific task by means of web services. The protocol defines the order of operations, including sequences, loops, forks, joins, and so on. In general the protocol semantics describe the workflow and the dataflow relevant for the web service's execution. A protocol may be specified in different notations that are more or less intuitive for the developer. In the following, the protocol semantics are specified with the help of pre-conditions and post-conditions, which are discussed in the following subsection.

4.5.2 Functional Semantics

The functional semantics describe the functionality of a web service operation. This can be expressed with the help of effects. Before the execution of an operation, certain pre-conditions have to be fulfilled. For instance, a money transfer from account A to account B can only take place if account A is solvent. Preconditions and effects can be used to express the protocol semantics and functional semantics of web services. Before we continue to examine how information about them can support the web service development process we want to analyze both in more detail.

We distinguish between internal preconditions which are checked (internally) by the application logic or ensured by type conditions of the programming language, and external, more general preconditions which must be fulfilled for a correct and successful execution of the web service and can not be checked within a web service implementation. Within the internal conditions, we further distinguish between implicit and explicit conditions. Implicit preconditions are conditions that are a consequence of standards, types and definitions. In the example of the credit card payment service, we assume that one of the parameters passed to the service is the validity date of the credit card number. An implicit internal precondition is that the date has to be in a specific format, e.g. has to contain a day, a month, and a year and has to be of a particular type used in the programming language to represent dates. Explicit preconditions are conditions that are specified for a specific operation. In our example, an explicit internal condition for the validity date parameter is that the value must denote a date in the future, which can only be checked programmatically by comparing the current date with the date represented by the parameter.

As an example of an external precondition, consider the validity check of a credit card. The validity of a credit card number may depend on several external facts that are not available within our payment web service, e.g. the current balance of the bank account of the card owner. Thus, evaluation of a credit card number is only possible for the credit card provider which has access to the relevant facts that determine the validity of a credit card number. In order to enable validity checks by third parties, credit card providers usually offer a web

service that allows to retrieve the validity of a credit card. As the determination of validity of a credit card is out of the scope of our payment web service since it is missing the necessary external background knowledge, the condition that a web service must be reachable that is able to check credit card validity, is an external precondition.

Effects describe the output and the impact of a web service's execution. While the term post-condition often focuses only on the description of the conditions regarding return values of the web service operations, we prefer the term effect to emphasize that the impacts of a web service execution are also considered. In our example above, an impact of the execution would be the real money transfer from the credit card account to the destination account while the post-condition describes that no money got lost during the transfer. While it is possible to describe the money transaction by post-conditions only, such a description would only focus on the fact that one account is decreased by a specific amount of money and another account is increased. The effect of a money transfer, however, can include more than changing account balances.

4.5.3 Semantic Description Example: Functional Semantics

Before we exemplify the use of knowledge representation languages for web service specifications, we briefly examine the definition of a web service ontology that defines the concepts and relations used by the later specification. Listing 4 shows a snippet of a simplified service ontology written in Flora-2 that defines amongst others the concepts *semanticSpecification*, *groundingSpecification* and *service*. The *semanticSpecification* specifies the semantics related to the preconditions and effects of an operation. For this purpose the *semanticSpecification*-class has two attributes, *precondition* and *effects*, which are inheritable attributes (*) of the type (\Rightarrow reification) defined by the reification class. The *paramPos* class specifies a type that is defined to connect parameter names with their position in a sequence of parameters. The *groundingSpecification* specifies the reference to the service implementation (*serviceImplRef*), the name of the described operation (*operationName*), and the sequences of input and output parameters (*inParamSequence* and *outParamSequence*). The *service*-class represents the concept that represents a service, including the grounding and the non-functional properties, of all operations of a service. For the following examination we skip non-functional properties since we examine them later.

```
semanticSpecification[precondition *=> reification,
                    effects      *=> reification].

paramPos[pos *=> ordinal,
         name *=> string].

groundingSpecification[serviceImplRef *=> string,
                      operationName  *=> string,
                      inParamSequence *=>> paramPos,
                      outParamSequence *=>> paramPos].

nonFunctionalProperties[...].
...

service[semanticSpec *=> semanticSpecification,
       groundingSpec *=> groundingSpecification,
       nfProp       *=> nonFunctionalProperties].
```

Listing 4: Definition of Service Related Concepts in an Example Ontology⁴

⁴ The Flora-2 examples follow the style of the ontology defined in the ASG-project, <http://asg-platform.org>
 ASG, Adaptive Services Grid, is an Integrated Project supported by the European Union

Based on a general service ontology as described in Listing 4, a domain ontology can be build that defines domain specific concepts, types and relations. Listing 5 is a cutout of a domain ontology defining the types *validityInformation* and *date* as mentioned in the documentation of the credit card payment service. Besides the two types, the domain ontology additionally specifies the *isValid* relation that has two arguments of the type *validityInformation* and *date*.

```

validityInformation[ownerName    *=> string,
                    credidCardId *=> string,
                    validityDate *=> date,
                    ...].

date[day    *=> ordinal,
     month  *=> ordinal,
     year   *=> ordinal].
...

isValid(validityInformation, date).

```

Listing 5: Definition of Classes and Relations in a Payment Domain Ontology

The concrete credit card payment service is described by instantiating (:) concepts and utilizing relations of the ontologies sketched before. In Listing 6, *creditCardPayment* is defined as an instance of the *service*-class defined in the service ontology of Listing 4. Furthermore, it shows in detail the semantic specification part of the description of the *validityCheck*-operation. The instance *validityCheckSpec* of the *semanticSpecification*-class is an assigned value (->) of the semantic spec attribute of the *creditCardPayment*-instance of the *service*-class. The specified preconditions, which are specified by reifications (\$ {...}), are that the *ValidityInfo* parameter of the operation is of the type *validityInformation*, and that *ActualDate* is another parameter of the operation which is of type *date*. The described effects include that the output parameter *Valid* is of the type *boolean* and that the relation *isValid* must be true for the given *ValidityInfo* and *ActualDate*. In addition, Listing 6 depicts the grounding specification of the *validityCheck*-operation (the *_#* represents an unnamed instance).

```

creditCardPayment:service[
  semanticSpec -> validityCheckSpec:semanticSpecification[
    precondition -> ${ValidityInfo:validityInformation, ValidityInfo:parameter,
                     ActualDate:date, ActualDate:parameter},
    effects      -> ${Valid:boolean, Valid:parameter,
                     isValid(ValidityInfo, ActualDate)}],
  groundingSpec -> validityCheckGround:groundingSpecification[
    serviceImplRef -> "http://example.org/creditcardpayment/":string,
    operationName  -> "ValidityCheck":string,
    inParamSequence ->> {_#:paramPos[pos -> 1, name -> "ValidityInfo":string],
                        _#:paramPos[pos -> 2, name -> "ActualDate":string]},
    outParamSequence ->> {_#:paramPos[pos -> 1, name -> "Valid":string]}],
  nfProp         -> validityCheckNFP:nonFunctionalProperties[...],
  ...].

```

Listing 6: Service Specification of a Simplified Credit Card Payment Service

As we have seen, if the description of a service is given in the above mentioned manner, the engineer reusing the web service gets detailed information about the preconditions and parameters of a service.

Given a proper semantic web service specification and its referenced ontologies, the annotated web service can be reused with only a small amount of extra knowledge. In addition, if the

language used to define the ontology is machine-interpretable, development tools can offer the developer semi-automatic support for the web service integration process. The first kind of support that can be given is discovery support by finding a service that matches a given request. A request can be the result of a planning process or the description of desired behaviors and properties specified by the engineer. The request can then be translated into a query that represents a generic service specification which specifies the desired properties stated in the request. The support tool queries an appropriate registry which returns a list of matching service specifications. In the case of a suitably specified web service, further assistance can be given in the form of semi-automatic web service invocation and integration into an existing workflow.

4.5.4 *Non-functional Semantics*

Many different non-functional properties exist. We provide a short list of common non-functional properties below:

- The *costs* of a web service are a non-functional property that is relevant for the customer who wants to use the service.
- *Security* is a non-functional property that is (like costs) relevant to most web services. Security enfolds aspects like communication and data encryption.
- The *quality of service* property is a bundle of different properties that all affects the quality of specific aspects of the service, its usage and its output. Examples for such properties are the response time of an operation call and the capacity of the service.
- The information about the *resource usage* of a web service can be of importance, if the web service needs special resources or if using the service is billed with respect to its resource usage.

In order to achieve a correct interpretation of non-functional properties logical descriptions have to be found. For this purpose different sources exist, for instance standards and laws. If a standard defines that an encryption algorithm has to be used with at least a specific key length, this information can be interpreted as a property of a certain security level. Another source for non-functional property specifications are laws which sometimes also specify a minimum level of encryption for specific data transfers.

4.5.5 *Semantic Description Example: Non-functional Semantics*

To enable the annotation with non-functional properties a concept defining non-functional properties is needed. In Listing 7 a refining of the *nonFunctionalProperties*-concept of the service ontology from Listing 4 is depicted that extends the concept by two simple non-functional properties, the service's name and the name of the provider.

```
nonFunctionalProperties[serviceName *=> string,
                        providerName *=> string].
```

Listing 7: Expansion to Listing 4: Definition of Service Related Concepts in an Example Ontology

The insertion of the concept of non-functional properties enables any domain ontology that is based on the service ontology to define domain specific non-functional properties and types needed by these. For the payment example in Listing 8, the cutout of Listing 5 is extended by the type *securityType*, which defines different levels of security. The different levels are defined as an inheritable instance of an array (->>) that contains a set of specified values ({..., ...}). In this example the service engineer can choose between *none* and *norm_X*. *none* stands

self-explanatory for no security support and *norm_X* for security as defined in the norm X. In addition, the *securityNFProps* is defined as a subclass (::) of the *nonFunctionalProperties*-concept that includes security.

```
securityType::enumeration[values *->> {"none", "norm_X"}].
securityNFProps::nonFunctionalProperties[security *=> securityType].
```

Listing 8: Expansion to Listing 5: Definition of Classes and Relations in a Payment Domain Ontology

In Listing 9, the service description of the concrete payment service of Listing 6 is refined. Since the security level should be specified, the *validityCheckNFP* is no longer an instance of the *nonFunctionalProperties*-concept but of the *securityNFProps*-concept. While the property values of *serviceName*, and *providerName* are set to string, the value of the *security* property must be chosen from the pre-defined ones. In the example, the *norm_X*-level is specified.

```
creditCardPayment:service[
...
  nfProp -> validityCheckNFP::securityNFProps[
    serviceName -> "SimplifiedCreditCardPaymentService":string,
    providerName -> "Credit Card Service Provider X":string,
    security      -> validityCheckSType::securityType[
      values ->> {"norm_X"}]].
...].
```

Listing 9: Expansion to Listing 6: Service Specification of a Simplified Credit Card Payment Service

The annotation with non-functional properties supports the service engineer, even if they are often not machine-interpretable. For instance, the name of the provider, especially if extended to complete address information, can be used to contact the provider in order to gain more information. The reference to a particular security specification allows the engineer to find out about the standards to which a service conforms, and which standards need to be fulfilled to communicate with the service.

4.5.6 Semantic Description Example: Exploiting Semantic Descriptions

As mentioned above, the annotation of a web service with a knowledge representation language enables the discovery of web services. For instance the web shop developer of our application example wants to query a registry for a service by specifying a subset of desired preconditions. The preconditions may state that a parameter of the type *validityInformation* has to be assigned at the operation's call. Listing 10 shows such a query, which will return the matching service instance (in our example this will be at least the *creditCardPaymentService*) and the instance of the semantic specification of the operation (in our example this will be for the credit card payment service the *validityCheckSpec*). The query can be read in the following way: We want all X that are instances of the *service*-class and all Y that are instances of the *semanticSpecification*-class assigned to the *semanticSpec*-attribute of X. In addition, the *precondition*-attribute of Y must be a reification that contains at least two attribute-type pairs. One of these pairs specifies an attribute of the type *validityInformation* and the other one specifies an attribute of the *parameter*-type. The associated attributes (*_G* and *_H*) and additional pairs the reification may contain (*_*, *_*) are ignored.

```
X:service[
  semanticSpec->Y:semanticSpecification[
```

```
precondition-> ($[_G:validityInformation], $_H:parameter} , _)]].
```

Listing 10: Querying an operation by precondition

4.6 The Annotation Process

We explained the representation of web service descriptions and how these descriptions can be exploited, but have not yet outlined the **annotation process**. Thus, in this section we annotate the payment service of the application example step by step while introducing exemplary methods and tools that support the annotation process. Generally, we assume that descriptions are developed by the engineer who implements the web service. In this case the source code is well known and the background knowledge of the engineer should be sufficient. Figure 3 depicts the process of annotation.

The first step is the non-semantic annotation. The inputs of this step are gained from the different information sources introduced before and include the web services documentation (see Listing 1), and the interface descriptions of the web service's operations (see Listing 2). The outcomes of the non-semantic annotation are WSDL files (see Listing 3) and other kinds of non-semantically annotated API-descriptions and documents. The second step is the semantic annotation, which takes the same inputs as the first step in combination with relevant ontologies, such as web service ontologies (see Listing 4) and different domain ontologies (see Listing 5). The outcome of the semantic annotation is a semantic service specification (see Listing 6 and Listing 8).

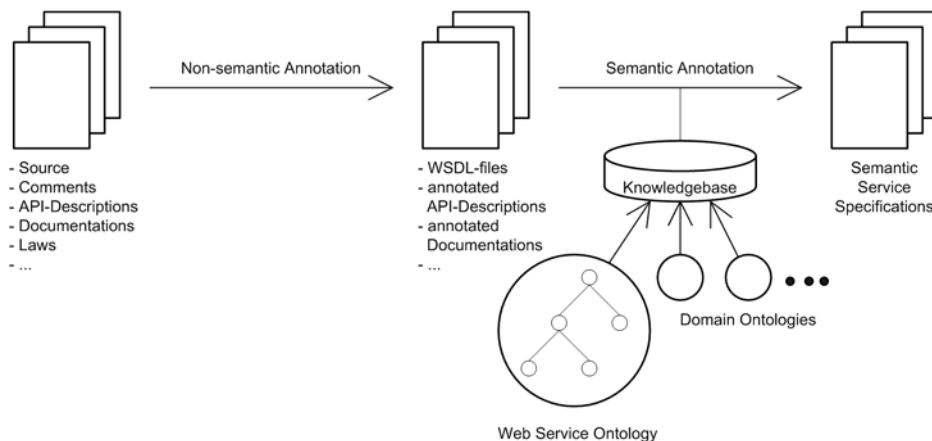


Fig. 3: The Annotation Process

Non-semantic annotation is semi-automatically supported by different tools. For instance, for many programming languages tools exist that support the generation of WSDL. One framework that supports this is Axis, a SOAP (the Simple Object Access Protocol is the communication protocol to access web services (Mitra, 2003)) engine that provides various tools to develop and use web services (Apache, 2005). Axis provides a tool named Java2WSDL, a translator that generates complete WSDL files out of a web service's Java code.

The process of semantic annotation consists of different steps. The first step is the classification of the web service. The result of the classification is the assignment of the web service to its domain. The service developer classifies services by decisions based on his

background knowledge or with the help of classification tools, e.g. ASSAM (Hess, 2004). ASSAM, a semi-automatic web service classification tool, applies text classification algorithms to the WSDL description of a web service to classify the service into a specific domain, e.g. banking. In advance, ASSAM can classify the functionality of operations and the types of input and output parameters. After the classification of the web service's domain the developer has to select an appropriate domain ontology, or, if no suitable one exists, define a new one. For the development of domain ontologies tool support exists as well: For example, the method described in (Sabou, 2005) supports the service engineer by extracting terms from textual sources (e.g. documentation) using text processing methods like parts of speech (POS) and dependency relations (DEP). In a second step, corresponding concepts and relations are derived and arranged in a hierarchy. The engineer only has to filter the proposed concepts and relations and refine the derived hierarchy.

Once an ontology is selected the engineer has to connect the metadata describing the web service functionalities with ontology concepts. This process can be separated into different subprocesses. Each subprocess addresses one kind of semantics. Even if no special order exists for the execution of the single subprocesses, it makes sense to start annotating the metadata with data semantics. For this purpose the first step is to specify the operation's interfaces. If a non-semantic interface description exists, it can be translated into the knowledge description language used to notate the semantic specification. Dependent of the language used for the non-semantic description the translation can be automated to a certain degree. For instance, WSDL files can semi-automatically be translated, because of their machine-interpretable form. For the translation, the used types and the elements of the operation interfaces have to be mapped onto ontology concepts. For this task, tools like ASSAM or the deep annotation mechanism of the OntoMat-Service-Browser can be used (Agarwal, 2005). Using the OntoMat-Service-Browser, the engineer can build the connections between ontology concepts and the metadata using the drag and drop mechanism provided by the graphical user interface. The result of this step is the semantic grounding specification and the parts of preconditions and effects that specify input and output parameters (see highlighted parts of Listing 11).

```
creditCardPayment:service[
  semanticSpec -> validityCheckSpec:semanticSpecification[
    precondition -> ${ValidityInfo:validityInformation, ValidityInfo:parameter,
      ActualDate:date, ActualDate:parameter},
    effects      -> ${Valid:boolean, Valid:parameter,
      isValid(ValidityInfo, ActualDate)},
  groundingSpec -> validityCheckGround:groundingSpecification[
    serviceImplRef -> "http://example.org/creditcardpayment/":string,
    operationName   -> "ValidityCheck":string,
    inParamSequence ->> {_#:paramPos[pos -> 1, name -> "ValidityInfo":string],
      _#:paramPos[pos -> 2, name -> "ActualDate":string]},
    outParamSequence ->> {_#:paramPos[pos -> 1, name -> "Valid":string]},
    ...].
```

Listing 11: Data Semantics Derived from WSDL

The next step is the specification of the protocol and functional semantics. If the data semantics are specified, parts of the protocol and the functionality are already specified implicitly. The implicit specification of the protocol results from the specification of the input parameters. From this specification we can derive which information has to be gathered before an operation can be invoked. The implicit specification of the functionality is part of the parameter specifications. In other words: all parts of the protocol and the functionality that can be derived from the dataflow can be specified implicitly. The other parts of the semantics require an explicit specification by means of relations. As input of this process step information from all sources mentioned in the section about information sources is

imaginable. The explicit specifications of preconditions and effects are complex and have to be defined by the service developer manually. The highlighted parts of Listing 12 depict the parts of the service specification of the example service that specify the protocol and functional semantics of the web service. The dataflow of the validity check is described by the parameters *ValidityInfo*, *ActualDate* and *Valid*. To fully express the functionality of the *ValidityCheck*-operation the input and output parameters are not sufficient. A relation that describes the dependency between the validity information and the actual date is necessary. For this purpose the *isValid*-relation has to be specified.

```
creditCardPayment:service[
  semanticSpec -> validityCheckSpec:semanticSpecification[
    precondition -> ${ValidityInfo:validityInformation, ValidityInfo:parameter,
                    ActualDate:date, ActualDate:parameter},
    effects      -> ${Valid:boolean, Valid:parameter,
                    isValid(ValidityInfo, ActualDate)}},
  ...].
```

Listing 12: Protocol and Functional Semantics

The last step is the specification of the non-functional semantics of the web service. Some of the simple non-functional properties like the name of the service can be extracted automatically from the WSDL file. More complex non-functional properties like the security property are described by the engineer. For this purpose he also can use information from all information sources introduced in the section “Sources of Information”. In our example, the engineer can choose between two levels of security defined in the payment domain ontology (see Listing 8). The highlighted parts of Listing 13 show the parts of the specification that specify the non-functional semantics of the *ValidityCheck*-operation. As result of the semantic annotation process the engineer gets a semantic description of the web service.

```
creditCardPayment:service[
  ...
  nfProp -> validityCheckNFP::securityNFPProps[
    serviceName -> "SimplifiedCreditCardPaymentService":string,
    providerName -> "Credit Card Service Provider X":string,
    security     -> validityCheckSType::securityType[
      values ->> {"norm_X"}].
  ...].
```

Listing 13: Non-Functional Semantics

A common aspect of the non-semantic and semantic annotation processes is the effort of annotating. A description of a service containing all relevant properties leverages reuse and enables better tool support, but also requires much higher efforts from the engineer. A tradeoff between modeling efforts and management efforts has to be achieved (Oberle, 2006), where modeling efforts comprise the annotation and documentation process, while management efforts denote amongst others the reuse of the service. To decrease annotation efforts one has to increase the management efforts and vice versa. A reasonable tradeoff is usually achieved when the combination of both is minimal, however, for different scenarios there are other factors influencing the optimal combination. For instance, a service provider may increase his modeling efforts to decrease his customers' management efforts to gain an advantage over his competitors. Thus, everyone annotating a web service has to find his own tradeoff depending on his individual context.

5 RELATED WORK

In this chapter we have shown how web services can be annotated to ease web service discovery, reuse, composition, and further web service management tasks. The approach

followed in this chapter uses existing resources to fulfill the task of web service annotation. The techniques and methods presented here are independent of the used languages or tools. In addition, our approach uses one description language to specify all semantic descriptions. In this section we introduce some other approaches to describe web services.

While we have shown what specifications of web services are and how they can be achieved, different approaches exist for developing semantic descriptions of web services:

- In (Zaremba, 2006) the authors follow the same approach regarding the use of one description language to specify all semantics and the grounding. They introduce their own language, the Web Service Modeling Language (WSML), a description language they specifically developed for web service description. They also introduce the Web Service Modeling Ontology (WSMO), which defines all concepts relevant to web service descriptions. Their Web Service Execution Environment (WSMX) combines WSML and WSMO to support the description of web services and the exploitation of web service descriptions.
- In (Akkiraju, 2005) the authors describe WSDL-S, an approach that adds semantics to the WSDL specification. Like the approach followed in this chapter, WSDL-S reuses already existing techniques and languages. In addition, WSDL-S is upward compatible to WSDL, because the description is an extension to existing document elements of WSDL. The descriptions can refer (by means of URI) to ontology concepts defined in external ontologies. These ontologies can be defined using any description language.
- The authors of (Battle, 2005) also use one description language for all descriptions. For this purpose they introduce their own language, the Semantic Web Service Language (SWSL). SWSL consists of two components: a first-order logic language (SWSL-FOL) and a rule-based language (SWSL-Rules). By means of SWSL the Semantic Web Service Ontology (SWSO) is specified, which defines concepts required to describe semantics of web services.
- Beside approaches dealing with the web service description as a whole, many solutions for sub-problems exist. For instance, many proposals for ontologies defining concepts related to the description of web service properties exist. Such a proposal is the Web Ontology Language for Services⁵ (OWL-S) that builds upon the DAML-S (Sycara, 2003), and uses the Web Ontology Language (OWL) that is created as a language with the purpose of ontology definition (McGuinness, 2004). OWL-S contains three interrelated sub-ontologies which define concepts for the profile, the process model, and the grounding of a web service. With the help of the profile sub-ontology the functional semantics can be described, the process sub-ontology expresses the protocol semantics and the grounding sub-ontology the data semantics.

Different approaches exist in the field of semi-automatic annotation and reuse support:

- For instance, in (Patil, 2004), the METEOR-S Web Service Annotation Framework (MWSAF) is introduced that provides algorithms, which semi-automatically match items in WSDL files to ontology concepts by means of SchemaGraphs. SchemaGraphs are devised by the authors as graphs that are used to represent ontologies independent of the language the ontology is specified in.
- Another example is ODESWS (Gómez-Pérez, 2004) a tool suite that provides tools for semi-automatic annotation, design and composition of web services. For this purpose ODESWS provides miscellaneous ontologies to support semantic specifications.

⁵ <http://www.daml.org/services/owl-s/1.0/>

6 CONCLUSION

In this chapter we have shown how the process of annotation leads to semantic specifications that leverages web service use and eases web service development. We examined different information sources, their information content, and potential exploitations of the content. Different aspects of web service development have been addressed on the basis of an application example that illustrates the development of a web shop by composing other web services to a new one. Finally, the challenges of dealing with non-functional properties and the annotation process itself were analyzed.

The introduced techniques enable a semi-automatic reuse and management of web services. We illustrated, that under the condition that web services are semantically annotated using one language, e.g. Flora-2, web service management can be accomplished over one common interface. Therefore, web service development as well as management efforts are decreased, and web service employment is leveraged. Flora-2 presents a running implementation of a description language that is applicable for semantic web service description and its exploitations.

Besides partial automation of web service management, automating web service annotation is an underexplored research area. We conjecture that many annotation tasks can be automated so that semi-automatic annotation methods utilize the possible simplifications of automation and the power of manual annotation.

ACKNOWLEDGMENTS

This work is conducted with respect to the Adaptive Services Grid (ASG) project and the project Knowledge Sharing and Reuse across Media (X-Media), both funded by the Information Society Technologies (IST) 6th Framework Programme.

LITERATURE

Agarwal, S., Handschuh, S., & Staab, S. (2005), Annotation, Composition and Invocation of Semantic Web Services, in *Journal of Web Semantics*, 2(1).

Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A., & Verma, K. (2005), *Web Service Semantics - WSDL-S*, W3C Member Submission 7 November 2005, Retrieved April 4, 2006, from <http://www.w3.org/Submission/2005/SUBM-WSDL-S-20051107/>

Alonso, G., Casati, F., Kuno, H. & Machiraju, V (2004), *Web Services – Concepts, Architecture and Applications*, Berlin, Heidelberg, New York: Springer.

Andrews, T., Curbera, F., Dholakia, H., Golland, Y., Leymann, F., Klein, J., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., & Weerawarana, S. (2005), *Business Process Execution Language for Web Services Version 1.1*, Retrieved February 20, 2006, from <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>

Apache, Software Foundation (2005), *Axis User's Guide*, Retrieved February 20, 2006, from <http://ws.apache.org/axis/java/user-guide.html>

Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Langworthy, D., Malhotra, A., Nadalin, A., Nagaratnam, N., Nottingham, M., Prafullchandra, H., von Riegen, C., Schlimmer, J., Sharp, C., & Shewchuk, J. (2004), *Web Services Policy Framework (WS-Policy)*, Retrieved February 20, 2006, from <ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf>

Battle, S., Bernstein, A., Boley, H., Grosz, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., & Tabet, S. (2005), *Semantic Web Services Language (SWSL)*, W3C Member Submission 9 September 2005, Retrieved April 4, from <http://www.w3.org/Submission/2005/SUBM-SWSF-SWSL-20050909/>

Cabrera, L. F., Copeland, G., Feingold, M., Freund, R. W., Freund, T., Johnson, J., Kaler, C., Klein, J., Langworthy, D., Langworthy, D., Little, M., Nadalin, A., Orchard, D., Robinson, I., Shewchuk, J., & Storey, T. (2005), *Web Services Coordination (WS-Coordination)*, Retrieved February 20, 2006, from <ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf>.

Christensen, E., Curbera, F., Meredith, G. & Weerawarana, S. (2001), *Web Services Description Language (WSDL) 1.1*, 15 March 2001, Retrieved February 20, 2006, from <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

Feier, C., Domingue, J. (2005), *WSMO Primer*, in Feier, C. (Ed.), *WSMO Final Draft 01 April 2005*, Retrieved February 20, 2006, from <http://www.wsmo.org/TR/d3/d3.1/v0.1/20050401/>

Fensel, D., and Bussler, C., (2002), *The Web Service Modeling Framework WSMF*, *Electronic Commerce Research and Applications*, 1(2).

Gómez-Pérez, A., González-Cabero, R., & Lama, M. (2004), *A Framework for Design and Composition of Semantic Web Services*. *AAAI Spring Symposium on Semantic Web Services*, 113-120. Stanford, USA.

Hess, A. & Kushmerick, N. (2003), *Learning to Attach Semantic Metadata to Web Services*, *International Semantic Web Conference 2003*.

Hess, A., Johnston, E. & Kushmerick, N. (2004), *ASSAM: A Tool for Semi-automatically Annotating Semantic Web Services*, *International Semantic Web Conference 2004*, 320-334.

IBM, BEA Systems, Microsoft, Arjuna, Hitachi, IONA (2005). *Web Services Transaction (WS-Transaction)*, Retrieved February 20, 2006, from <http://www-128.ibm.com/developerworks/library/specification/ws-tx/>.

McGuinness, D. L., van Harmelen, F. (Ed.)(2004), *OWL Web Ontology Language - Overview*, W3C Recommendation 10 February 2004, Retrieved February 20, 2006, from <http://www.w3.org/TR/2004/REC-owl-features-20040210/>

Mitra, N. (Ed.) (2003), *SOAP Version 1.2 Part 0: Primer*, W3C Recommendation 24 June 2003, Retrieved February 20, 2006, from <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>

Oberle, D. (2006), *Semantic Management of Middleware*, New York, USA: Springer.

Patil, A. A., Oundhakar, S. A., Sheth, A. P., and Verma, Kunal (2004), METEOR-S Web Service Annotation Framework, *WWW 2004*, ACM Press, 553-562

Pistore, M., Spalazzi, L., & Traverso, P. (2006), A Minimalist Approach to Semantic Annotations for Web Processes Compositions, *European Semantic Web Conference 2006*.

Sabou, M., Wroe, C., Goble, C., & Stuckenschmidt, H. (2005), Learning Domain Ontologies for Semantic Web Service Descriptions, in *Journal of Web Semantics*, 3(4).

Sycara, K. P., Paolucci, M., Ankolekar, A., & Srinivasan, N., (2003): Automated discovery, interaction and composition of Semantic Web services, In *Journal of Web Semantics*, 1(1): 27-46.

Wolff, F., Oberle, D., Lamparter, S., Staab, S. (2005). Economic Reflections on Managing Web Services Using Semantics. In *EMISA-2005 - Enterprise Modelling and Information Systems Architectures*. Klagenfurt, Austria, October 24-25.

Yang, G., Kifer, M., Zhao, C., Chowdhary, V. (2005), *Flora-2: User's Manual*, Retrieved February 20, 2006, from <http://flora.sourceforge.net/docs/floraManual.pdf>

Zaremba, M., Kerrigan, M., Mocan, A. & Moran, M. (2006), Web Services Modeling Ontology, in Cardoso, J., & Sheth, A., (Ed.), *Semantic Web Services, Processes and Applications*, Kluwer Academic Publishers.

AUTHOR BIBLIOGRAPHY

Christoph Ringelstein is a doctoral student in the Department of Computer Science at the University of Koblenz-Landau. His research interests include web service annotation, knowledge representation, ontology engineering, and the semantic web. He received his diploma in computer science from the University of Koblenz-Landau. Contact him at the Dept. of Computer Science, University of Koblenz-Landau, Universitaetsstrasse 1, 56070 Koblenz, Germany; cringel@uni-koblenz.de

Thomas Franz is a doctoral student in the Department of Computer Science at the University of Koblenz-Landau, Germany. His research interests include knowledge representation, personal information management, and the semantic web. He received his M.Sc. in computer science from the University of Freiburg. Contact him at the Dept. of Computer Science, University of Koblenz-Landau, Universitaetsstrasse 1, 56070 Koblenz, Germany; franz@uni-koblenz.de

Prof. Dr. Steffen Staab is professor for databases and information systems in the Department of Computer Science at the University of Koblenz-Landau and heads the research group on information systems and the semantic web (ISWeb). His research interests range from ontology management and ontology learning to the application of Semantic Web technologies in areas such as multimedia,

personal information management, peer-to-peer - and web services. In particular, he is interested in combining sound and diligent ontology engineering with real world concerns and applications. His research activities have led to over 100 refereed publications and 7 books as well as to the foundation of Ontoprise GmbH, a company focusing on the exploitation of semantic web technology.

INDEX

Term 1 - Annotation

- Also known as:
- Similar to: annotation process
- Associated in the manuscript with:
- Notable appearances of this term can be found on:
Page 2 –term definition
Page 5 –the goal of annotation
Page 14 – the process of annotation

Term 1 – Web Service Specification

- Also known as: **Description**
- Similar to:
- Associated in the manuscript with:
- Notable appearances of this term can be found on:
Page 2 –term definition
Page 5 –
Page 14 –

Term 2- Web Service

- Also known as:
- Similar to:
- Associated in the manuscript with:
- Notable appearances of this term can be found on:
Page 1 –term definition
Page -
Page -

Term 3- Knowledge representation Language

- Also known as:
- Similar to:
- Associated in the manuscript with:
- Notable appearances of this term can be found on:
Page 10 – aim of the use of knowledge representation languages
Page -
Page -

Term 4- Semantics

- Also known as:
- Similar to:
- Associated in the manuscript with:

- Notable appearances of this term can be found on:
Page 9-different kinds of semantics
Page -
Page -

Term 5- Semantic Grounding Specification

- Also known as:
- Similar to:
- Associated in the manuscript with:
- Notable appearances of this term can be found on:
Page 10- term definition
Page -
Page -

Term 6- Semantic Service Specification

- Also known as:
- Similar to:
- Associated in the manuscript with:
- Notable appearances of this term can be found on:
Page 10- term definition
Page -
Page -

Term 7- Web Service Ontology

- Also known as:
- Similar to:
- Associated in the manuscript with:
- Notable appearances of this term can be found on:
Page 2- term definition
Page -
Page -