Grounding OWL-S in SAWSDL

Massimo Paolucci[1], Matthias Wagner[1], and David Martin[2]

[1] DoCoMo Communications Laboratories Europe GmbH
{paolucci,wagner}@docomolab-euro.com
[2] Artificial Intelligence Center, SRI International
martin@ai.sri.com

Abstract. SAWSDL and OWL-S are Semantic Web services languages that both aim at enriching WSDL with semantic annotation. In this paper, we analyze the similarities and differences between the two languages, with the objective of showing how OWL-S annotations could take advantage of SAWSDL annotations. In the process, we discover and analyze representational trade-offs between the two languages.

1 Introduction

Semantic Web services have emerged in the last few years as an attempt to enrich Web services languages with ontological annotations from the Semantic Web. Overall, the goal of such efforts is to facilitate Web services interaction by lowering interoperability barriers and by enabling greater automation of service-related tasks such as discovery and composition. A number of proposals, such as OWL-S 0, WSMO 0 and WSDL-S 0, have been on the table for some time. They provide different perspectives on what Semantic Web services ought to be, and explore different trade-offs. Each of these efforts is concerned with supporting richer descriptions of Web services, but at the same time each has made an effort to tie in with WSDL, and through it to Web service technology. In the case of OWL-S, an ontology-based *WSDL Grounding* is provided, which relates elements of an OWL-S service description with elements of a WSDL service description.

Recently, Semantic Web services reached the standardization level with SAWSDL 0, which is closely derived from WSDL-S. A number of important design decisions were made with SAWSDL to increase its applicability. First, rather than defining a language that spans across the different levels of the WS stack, the authors of SAWSDL have limited their scope to augmenting WSDL, which considerably simplifies the task of providing a semantic representation of services (but also limits expressiveness). Second, there is a deliberate lack of commitment to the use of OWL 0 or to any other particular semantic representation technology. Instead, SAWSDL provides a very general annotation mechanism that can be used to refer to any form of semantic markup. The annotation referents could be expressed in OWL, in UML, or in any other suitable language. Third, an attempt has been made to maximize the use of available XML technology from XML schema, to XML scripts, to XPath, in an attempt to lower the entrance barrier to early adopters.

B. Krämer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 416–421, 2007.

Despite these design decisions that seem to suggest a sharp distinction from OWL-S, SAWSDL shares features with OWL-S' WSDL grounding: in particular, both approaches provide semantic annotation attributes for WSDL, which are meant to be used in similar ways. It is therefore natural to expect that SAWSDL may facilitate the specification of the Grounding of OWL-S Web services, but the specific form of such Grounding is still unknown, and more generally a deeper analysis of the relation between SAWSDL and OWL-S is missing. To address these issues, in this paper we define a SAWSDL Grounding for OWL-S. In this process we try to identify how different aspects of OWL-S map into SAWSDL. But we also highlight the differences between the two proposals, and we show that a mapping between the two languages needs to rely on fairly strong assumptions. Our analysis also shows that despite the apparent simplicity of the approach, SAWSDL requires a solution to the two main problems of the semantic representation of Web services: namely the generation and exploitation of ontologies, and the mapping between the ontology and the XML data that is transmitted through the wire.

The result of this paper is of importance for pushing forward the field of Semantic Web services by contributing to the harmonization of two proposals for the annotation of Web services. In the paper, we will assume some familiarity with OWL-S and SAWSDL, neither of which is presented. The rest of the paper is organized as follows. In section 2 we will analyze the similarities and differences between OWL-S and SAWSDL. In section 3, we will introduce an OWL-S grounding based on SAWSDL, with analysis of its strengths and weaknesses. In section 4 we will discuss the finding and conclude.

2 Relating SAWSDL to OWL-S

The first step toward the definition of a SAWSDL Grounding for OWL-S is the precise specification of the overlap between the two languages. Since the two languages have a very similar goal: provide semantic annotation to WSDL, they have some similarities. The first one is that both OWL-S and SAWSDL express the semantics of inputs and outputs of WSDL operations. SAWSDL does it via a direct annotation of the types and elements while the OWL-S Grounding maps the content of inputs and outputs to their semantic representation in the Process Model. The second similarity is that both languages support the use of transformations, typically based on XSLT, to map WSDL messages to OWL concepts. These transformations allow a level of independence between the message formats and the semantic interpretation of the messages, allowing developers to think of the implementation of their application independently of the semantic annotation that is produced. The third similarity is that both OWL-S and SAWSDL acknowledge the importance of expressing the category of a service within a given taxonomy. SAWSDL provides category information by annotating interface definitions. OWL-S provides this information in the Profile through its `type` specification or through the property `serviceCategory`.

Despite their similarities, the two languages have also strong differences. The first one is in the use of WSDL. OWL-S uses WSDL exclusively at invocation time; therefore the WSDL description relates directly to atomic processes in the Process

Model; hence, in OWL-S, there is no direct relation between WSDL and the service Profile, which is used during the discovery phase. Instead SAWSDL uses WSDL both at both discovery and invocation time. Therefore, SAWSDL needs to relate to both the OWL-S Profile and the Process Model. The distinction is important since WSDL and the OWL-S Profile express two very different perspectives on the service: WSDL describes the operations performed by the service during the invocation; on the other hand, the OWL-S Profile takes a global view of the service independent of how this function is realized by the service. From the WSDL perspective, the Profile compresses the Web service to only one operation and it does not specify how this operation can be decomposed to more refined ones. The second difference is in SAWSDL agnostic approach toward semantics. In contrast to OWL-S, which is very committed to OWL and Semantic Web technology, SAWSDL does not make any commitment regarding the representational framework for expressing semantics. The authors of the SAWSDL specification explicitly state that semantics can be expressed in many different ways and languages. Such an agnostic approach extends the applicability of SAWSDL at cost of creating interoperability problems by mixing different annotation frameworks. The third difference is that SAWSDL, on the opposite of OWL-S, allows partial annotation of services. For example, it is possible to annotate the semantics of the attributes of a message, but not the semantics of the whole message. In turn the corresponding OWL-S Grounding will have to define the semantics of the elements that were not described.

Because of these differences, in order to be able to exploit the SAWSDL semantic annotations in the OWL-S Grounding we need to make three assumptions. The first one is that SAWSDL annotations are in OWL since OWL-S does not handle any other type of semantic annotation. The second assumption is that the semantic type of the complete message types is specified. This assumption is required since SAWSDL supports the specification of a schema mapping without a `modelReference`. In such a case, it may be known how to perform the mapping, but not the semantic type of the input or output. Finally, whole description needs to be semantically annotated. If these conditions are violated, then the semantic annotation of parts of the WSDL description will not be available, and therefore the grounding will have to be compiled manually.

3 Grounding OWL-S in SAWSDL

When the previous three assumptions are satisfied, we can take advantage of the SAWSDL semantic annotations in the definition of the mapping of the OWL-S Grounding. To define the OWL-S Grounding, we first need to specify which element of OWL-S maps to the corresponding element in SAWSDL. The class `WsdlAtomicProcessGrounding`, see Figure 2, specifies the correspondence between the Atomic Process and the WSDL operations through the two properties `owlsProcess` and `wsdlOperation`. The two properties `inputMap` and `outputMap` map the inputs and the outputs of OWL-S processes and WSDL operations.

```
<owl:Class rdf:ID="ModelRefMap">
  <owl:Restriction>
    <owl:onProperty rdf:resource="owlsParameter"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger"> 1
    </owl:cardinality>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="modelRef"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger"> 1
    </owl:cardinality>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="mapParam"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger"> 1
    </owl:cardinality>
  </owl:Restriction>
</owl:Class>

<owl:datatypeProperty rdf:ID="owlsParameter">
  <rdfs:domain rdf:resource="#ModelRefMap"/>
  <rdfs:range rdf:resource="&xsd;#anyURI"/>
</owl:datatypeProperty>

<owl:datatypeProperty rdf:ID="modelRef">
  <rdfs:domain rdf:resource="#ModelRefMap"/>
  <rdfs:range rdf:resource="&xsd;#anyURI"/>
</owl:datatypeProperty>

<owl:datatypeProperty rdf:ID="mapParam">
  <rdfs:domain rdf:resource="#ModelRefMap"/>
  <rdfs:range rdf:resource="&xsd;#literal"/>
</owl:datatypeProperty>
```

Fig. 1. Definition of ModelRefMap

As first approximation, OWL-S inputs and outputs can be mapped directly to the results of the concepts representing the semantics of the message types. This way we can take advantage of the lifing elements of SAWSDL. The class `ModelRefMap`, shown in Figure 1 performs this mapping by defining the two properties `owlsParameter` and `modelRef`. The first property specifies the OWL-S parameter to be used, the second property points to the URI of the semantic markup of the message type. One complicating factor in the input and output mapping is that whereas a WSDL operation has only one input and one output, the corresponding Atomic Process in OWL-S may have multiple inputs and outputs. Therefore the straightforward mapping defined above needs a mechanism to select the portions of the input or output that derive from the semantic markup of the message. This can be achieved with rules that specify how the modelRef of a message type maps to and from an OWL-S Parameter. Such a rule could be expressed in a rule language such as SWRL 0. The property `mapParam` of `ModelRefMap` is defined to store such a rule. The cardinality restriction of at most 1 allows for the property not to be used in the grounding, in such case the mapping between the OWL-S parameter and the SAWSDL message is expected to be 1:1.

The last aspect of the grounding is to deal the SAWSDL annotation on the interface. Unlike the previous mappings, in this case there is no need to explicitly add information to the Grounding because first, the expression of service categories is

```
<owl:Class rdf:ID="WsdlAtomicProcessGrounding">
  <owl:Restriction>
    <owl:onProperty rdf:resource="owlsProcess"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:cardinality>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty rdf:resource="wsdlOperation"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:cardinality>
  </owl:Restriction>
</owl:Class>

<owl:objectProperty rdf:ID="owlsProcess">
  <rdfs:domain rdf:resource="#WsdlAtomicProcessGrounding"/>
  <rdfs:range rdf:resource="&owlsProcess;#AtomicProcess"/>
</owl:objectproperty>

<owl:datatypeProperty rdf:ID="wsdlOperation">
  <rdfs:domain rdf:resource="#WsdlAtomicProcessGrounding"/>
  <rdfs:range rdf:resource="&xsd;#anyURI"/>
</owl:datatypeProperty>

<owl:objectProperty rdf:ID="inputMap">
  <rdfs:domain rdf:resource="#WsdlAtomicProcessGrounding"/>
  <rdfs:range rdf:resource="#ModelRefMap"/>
</Owl:objectproperty>

<owl:objectProperty rdf:ID="outputMap">
  <rdfs:domain rdf:resource="#WsdlAtomicProcessGrounding"/>
  <rdfs:range rdf:resource="#ModelRefMap"/>
</Owl:objectproperty>
```

Fig. 2. SAWSDL to OWL-S Grounding

equivalent in OWL-S and SAWSDL; and second, the Profile of the service can be found through the Service specification of OWL-S. Therefore, it is possible to stipulate a fixed mapping between the two service descriptions. Such mapping first identifies the Profile corresponding to the Grounding under definition, and then proceeds with a one-to-one mapping between the interface annotation in SAWSDL and the ServiceCategory of OWL-S.

4 Conclusions

The analysis performed in this paper reveals the relation between OWL-S and SAWSDL with the objective of deriving automatically OWL-S Grounding from SAWSDL annotations. The results of our analysis is that whereas in principle such derivation is possible, a number of assumptions on the use of WSDL and the style of annotations are satisfied. When the assumptions are not satisfied, the Grounding can still be defined, but such a mapping has to be derived manually by programmer that understands the semantics of the WSDL specification.

The result of the derivation is a skeletal OWL-S specification that contains a Process Model in which only the atomic processes are specified, and a Profile in

which only the service category is specified. The atomic processes themselves will also be partially specified since SAWSDL does not provide any information on their preconditions and effects. An additional modeling problem is the handling of WSDL faults. In principle, they can be represented in OWL-S with conditional results, but the problem is that there is no knowledge in SAWSDL of what are the conditions of a fault since SAWSDL specifies only the annotation of the semantics of content of the message, instead of the conditions under which the fault occurs. These problems could be addressed by adding a specification of preconditions and effects to SAWSDL.

References

1. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.T., Sheth, A., Verma, K.: Web Service Semantics - WSDL-S. Technical report, W3C Member (submission November 7, 2005) (2005)
2. Farrell, J., Lausen, H.: Semantic Annotations for WSDL and XML Schema, W3C Candidate Recommendation (January 26, 2007), http://www.w3.org/TR/sawsdl/
3. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic Web rule language combining OWL and RuleML
4. Lausen, H., Polleres, A., Roman, D.: Web Service Modeling Ontology (WSMO). W3C Member (2005) (submission), http://www.w3.org/Submission/WSMO/
5. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services. W3C Member Submission (2004)
6. McGuinness, D.L., Harmelen, F. v.: OWL Web Ontology Language overview – W3C recommendation (February 10, 2004)