The OWL-S Java API

Evren Sirin and Bijan Parsia
MINDSWAP Research Group
University of Maryland, College Park, MD
evren@cs.umd.edu, bparsia@isr.umd.edu

Abstract

The OWL-Services (OWL-S) suite of ontologies is the most mature and probably the most widely deployed comprehensive Semantic Web Service technology. However, the intended semantics of OWL-S service descriptions is not expressed (or expressable, often) in OWL. Furthermore, working with OWL-S descriptions at the RDF or even the OWL level is quite difficult and tedious as they tend to be at the wrong level of abstraction. The OWL-S API is a Java library for working with OWL-S service descriptions. It provides parsing, serializing, validation, reasoning, and execution services for OWL-S versions 0.7 through 1.0.

1 Introduction

The OWL-S [OWL Services Coalition, 2004] suite of ontologies are designed to support the automation of a range of Web Service related tasks, such as discovery, composition, and execution. The ontologies are canonically expressed in the Web Ontology Language (OWL), and in particular, as of version 1.1, the DL subspecies of OWL. OWL's exchange syntax is RDF/XML and many processors work with an RDF based model, in part, to facilitate the smooth integration of OWL-S service descriptions and other Semantic Web knowledge bases.

However, working with the RDF triples directly can be quite cumbersome and confusing. Furthermore, the OWL DL axioms do not sufficiently constrain the OWL-S descriptions. There are huge chunks of the intended meaning of the various OWL-S constructs which are only specified in natural language. So, for programmatic generation of descriptions, for validation, for certain sorts of reasoning (e.g., planning) and for execution and monitoring, it is helpful to have service descriptions represented at a higher level of abstraction.

The OWL-S API is a Java library which provides this higher level of abstraction. These classes also support various useful services such as validation of OWL-S descriptions (beyond what is expressed in the ontologies), matchmaking, and execution.

2 The Design Objectives

The OWL-S API was designed to let programmers access and manipulate OWL-S service descriptions easily. Therefore, the basic purpose of the API is to provide a data model that covers the specifics of OWL-S. However, the design of the API was driven by many other factors and objectives:

Support for multiple OWL-S versions OWL-S ontologies are constantly being refined and extended by the OWL-S coalition. The radical changes in the ontologies between different versions make it harder to develop and maintain applications based on the structure of the OWL ontologies. For example, the OWL-S processes were modeled as OWL classes in OWL-S 0.9 whereas they are modeled as OWL individuals in OWL-S 1.0 and higher versions. The data model in the API should be general enough to support different versions of the ontologies.

Execution of services OWL-S Process Model defines how a service works. Processes are defined either as one-step directly-invocable *AtomicProcess*es or as *CompositeProcess*es that are composed of other processes combined with one (or more) of the control construct defined in the Process ontology. An execution engine should handle interpreting these control constructs. The invocation of *AtomicProcess*es are described by *Grounding* specifications that map the processes to WSDL operations. Some applications may extend the grounding specification to use other standards such as UPnP. The execution engine should support the invocation of WSDL services as a minimum requirement but it must also be flexible to handle other grounding specifications.

Extensibility of OWL-S descriptions One essential feature in describing Web Services with OWL-S is being able to extend the base OWL-S ontologies in order to describe specific features for a service. For this purpose, OWL-S profile ontology defines a construct named *ServiceParameter* so that concepts defined in other ontologies may easily be integrated into OWL-S profile descriptions. The API should let the users easily handle the concepts that are not part of the core OWL-S ontologies, thus not part of the core data model in the API.

3 Architecture of the OWL-S API

The OWL-S API was designed to achieve the objectives described in the previous section. The data model for services were created to reflect the structure of the OWL-S model. While the Java interfaces and methods were designed in conjunction with the classes and properties defined in the

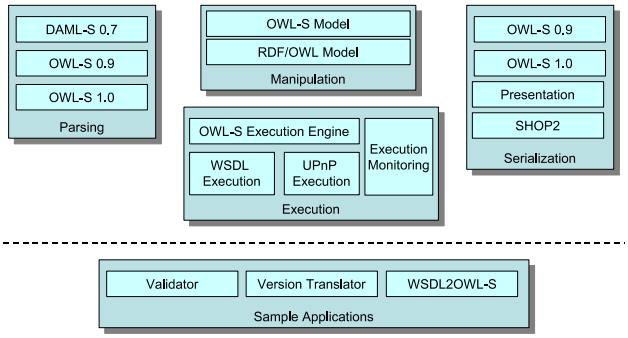

Figure 1: Basic components of the OWL-S API

OWL-S ontologies, there is no tight coupling between the two. A set of *Reader*s have been created to parse descriptions of different versions of OWL-S into the same data model. Therefore, there is one consistent view for all the services even if different versions of the ontologies have been used. Serialization of the descriptions are handled by a set of *Writer*s. *Writer*s are generally used to serialize the service for a specific OWL-S version but may also be used for other purposes, e.g. generating an HTML presentation for the service. The basic components of the API are shown in the Figure 1.

The API has been built using the Jena [HP, 2001] toolkit but the interfaces has been designed so that functionality of the OWL-S API is not bound to the specifics of the underlying RDF/OWL API. A basic *OWLResource* interface is provided for accessing the information in the RDF model and can easily be implemented for different RDF toolkits. Querying the RDF model makes it possible to get the extended parameters that are not part of the standard OWL-S ontologies. It is also possible to wrap the frequently used OWL concepts in a Java interface. This feature is similar to polymorphic views in the Jena toolkit and makes programming easier when applications are being developed for a fixed set of ontologies. Creation of Java interfaces can even be automated so a code template can be generated for a given OWL-S description, similar to how stubs are generated from WSDL descriptions.

The execution of processes are handled by a *ProcessExecutionEngine*. The default implementation provides the execution of control constructs such as *Sequence*, *Unordered*, *Split* and *Split+Join*. Conditional constructs, e.g. *If-Then-Else*, *Repeat-While*, *Repeat-Until*, is not yet supported because current release of OWL-S (1.0) does not specify how to express and evaluate the conditional expressions. The invocation of WSDL services are achieved through the Axis Web Services package. The API also provides for the execution of OWL-S services that have UPnP groundings. The specification of UPnP groundings were developed in collaboration with Fujitsu Labs of America, College Park and used to interact with devices in pervasive environments.

4 Applications

The OWL-S API is already being used in a number of in-house applications, and by various of our industrial partners. For example, Fujitsu Labs of America uses the API in their Task Computing framework [Masuoka *et al.*, 2003]. Indeed, we built the API, in part, to simplify our support of their work.

The API has also been used to translate OWL-S descriptions to SHOP2 [Nau *et al.*, 2003] planning domains [Wu *et al.*, 2003]. The translation algorithm described in [Wu *et al.*, 2003] was implemented as a *writer* in the API. The SHOP2 planner used the OWL-S execution engine to invoke Web Services for gathering information that is used in the planning process.

5 Conclusions and Future Work

This paper describes the basic features of the OWL-S API and the design issues behind the architecture of the API. To our knowledge, it is the only open-source API designed specifically for working with OWL-S ontologies. The source code, documentation and examples illustrating the usage of the API can be downloaded from the Web page http://www.mindswap.org/2004/owl-s/api.

We plan to keep the OWL-S API in sync with future releases of OWL-S. In particular, we plan to support the syntax and evaluation of the default condition language in OWL-S 1.1 (e.g., conjunction of SWRL atoms). When the expression of conditions are resolved we intend to extend the execution engine to handle the conditional control constructs.

6 Acknowledgements

OWL-S API was developed in collaboration with Fujitsu Labs of America, College Park and is currently being used in the Task Computing Environment. The authors would like to thank Ryusuke Masuoka and Zhexuan Song from FLA, CP for their feedback and help in the development of the OWL-S API.

References

[HP, 2001] Hewlett Packard, Jena semantic web toolkit, 2001. http://www.hpl.hp.com/semweb/jena.htm.

[Masuoka *et al.*, 2003] Ryusuke Masuoka, Bijan Parsia, and Yannis Labrou. Task computing - the semantic web meets pervasive computing. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, October 2003.

[Nau *et al.*, 2003] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20, December 2003.

[OWL Services Coalition, 2004] OWL Services Coalition. OWL-S: Semantic Markup for Web Services, 2004. OWL-S White Paper http://www.daml.org/services/owl-s/1.0/owl-s.pdf.

[Wu *et al.*, 2003] Dan Wu, Bijan Parsia, Evren Sirin, James Hendler, and Dana Nau. Automating daml-s web services composition using SHOP2. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, October 2003.