Visual Modeling of OWL-S Services

James Scicluna, Charlie Abela and Matthew Montebello

Department of Computer Science and AI,
University of Malta

Abstract. The Semantic Web is slowly gathering interest and becoming a reality. More people are becoming aware of this and are trying to embed Semantic Web technologies into their applications. This involves the use of tools that can handle rapid ontology building and validation in an easy and transparent manner. In the area of Semantic Web Web Services (SWWS) an OWL-S specification defines a set of ontologies through which a semantic description of the service can be created. At times this is not an easy task and could result in an incorrect specification of the description or even lead the fainthearted user to resort to some other type of description language. This paper describes the OWL-S editor tool that provides two methodologies in which such a web services description can be developed without exposing the developer to the underlying OWL-S syntax. These methodologies are based on a mapping from WSDL to OWL-S and on modeling a composite service using standard UML Activity Diagrams.

1 Rationale

The Web Services paradigm is becoming an important area due to the distributed nature of this technology. In recent years we have seen the birth of a number of languages that somehow provide the developer with ways and means to describe and advertise services on the Web. Some of these languages have already been accepted as standards while others are still evolving and striving to become accepted by the general public.

WSDL [20] is one such language and has been accepted as a standard by the W3C. It is also at the base of other Web Service composition languages, such as BPEL4WS [11], WSCI [19] and OWL-S [15]. These provide the Web Service developer with constructs to create a composition of services, by possibly reusing and or extending existing ones. Creating Web Service descriptions with any of these languages requires a thorough understanding of the language thus making them unpopular with the inexperienced developer. As such developers strive to find tools that could help or even automate the process of creating service descriptions in an easy and transparent manner. A number of tools such as BPWS4J [11] and .NET [14] provide such functionality to both the experienced and inexperienced user. In the area of SWWS though, tools are an important issue since several related technologies are still evolving making things somewhat more difficult.

OWL-S is at present the only language that marries the area of the Semantic Web with that of Web Services. A typical service description in this language consists of four ontologies, namely, Service, Service Profile, Service Model and Service Grounding. The upper Service ontology links to the other ontologies that make up the semantic description. The Profile ontology gives an overview of the service by exposing the inputs, outputs, preconditions and effects as defined in the process model. Other types of information such as a human readable textual description and business contact information are also expressed in this ontology. The Service Model defines the processes that make up the service together with a definition of their respective parameters. Additionally, processes may be of a composite nature in which a choreography of other processes is described

using control constructs defined in the Service Model ontology. Finally, the Service Grounding ontology defines the mapping of the atomic processes and their parameters to the operations and message parts defined in the WSDL of the service.

To create an OWL-S description, the user must be very confident about the underlying markup language. One of the major problems in this case is that the OWL-S specification is still under development and many aspects of this language are still being developed. Such aspects include expressing conditions, effects and dataflow binding mechanisms. DRS [3] and SWRL [9] seem to be the markup languages through which conditions will be expressed. Such ontologies enable to express logical expressions in OWL and hence allow defining also preconditions and effects in the OWL-S Service Model ontology. These evolutions in the language are not simple to grasp and define, as they require the user to be knowledgeable in other logics related areas as well as in OWL-S.

The OWL-S Editor tool that we present in this paper provides the user with a simplified user interface so that the above-mentioned concepts can be developed graphically without exposing the user to manually edit the underlying OWL-S constructs. Massimo Paolucci et al. describe how a primitive DAML-S description can be obtained from an existent WSDL. Since OWL-S essentially builds over DAML-S the concepts expressed in that paper equally apply. The conversion method considered by Paolucci assumes a one to one mapping between a DAML-S Atomic Process (or an OWL-S Atomic Process) and a WSDL operation. This operation will result in a complete Service Grounding specification and a partial definition of the Service Model and Service Profile ontologies. A complete specification of all the ontologies that comprise OWL-S cannot be obtained from a WSDL because the former is richer and expresses a more detailed description of the service. We implemented this methodology in the OWL-S Editor tool through a wizard (OwlsWiz), which abstracts the details of the underlying conversion and lets the user to specify the missing information through a graphical user interface.

Composing an OWL-S service through control constructs defined in the Service Model ontology, is another difficult task. Although tools such as Protégé [16], through its OWL plug-in, and the Web Service Composer [4], enable visual editing of ontologies and services respectively, there exist no tools which enables users to visually create a composite service by using a standard diagrammatic technique without exposing the users to technical information related to the underlying markup. UML Activity Diagrams is a standard diagrammatic technique which is well suited for manipulating the constructs and concepts defined in an OWL-S Service Model ontology. As part of OwlsWiz our tool implements a visual composer which makes use of UML Activity Diagrams [17] to enable a visual composition of the service and the automatic generation of the process markup. This abstracts away the underlying structural details of the Service Model ontology enabling a fast and easy creation of a composite service. The user must still have to know the basic key features behind OWL-S, such as knowledge about preconditions and effects, but there is no need for an in-depth understanding of how these concepts are expressed in the markup.

The rest of the paper is structured as follows. We first give an overview of our tool and then we continue by giving a more in-depth view of both the OwlsWiz and the Visual Composer by making references to a Health Information System service. We discuss issues related to the adoption of a diagrammatic representation in the composer and also other solutions related to the definition of conditions and effects. We then evaluate the tool and make references to other related work. Finally we describe some planned future work and some concluding remarks.

2 Overview

Most of the effort was dedicated to the development of the wizard and the visual composer, nonetheless we are also proposing a general architecture, depicted in Figure 1, of a system in which OWL-S

descriptions can be created, validated and also visualized. The idea is to consider an OWL-S description as if it were an application project. Thus the user wont have to load the OWL-S ontologies one by one but rather open a single file and define the paths and file names of the ontologies that comprise the description. Furthermore, the experienced user will be given the facility to directly edit the markup if desired.

The user may either create a description from an existent WSDL or from an empty template as desired. Any required non-local ontologies are fetched from the Web and validated before further processing is done. Such validation is performed with the Jena Toolkit, version 2.1 [7]. During the validation stage, the classes and property identifiers are extracted and maintained in a concepts list. The markup editor uses this list to perform a simple markup completion procedure during editing. The resulting ontologies can be validated and viewed either as a directed graph or as a table of triples. This method is an adaptation of the RDF Validation service [18].

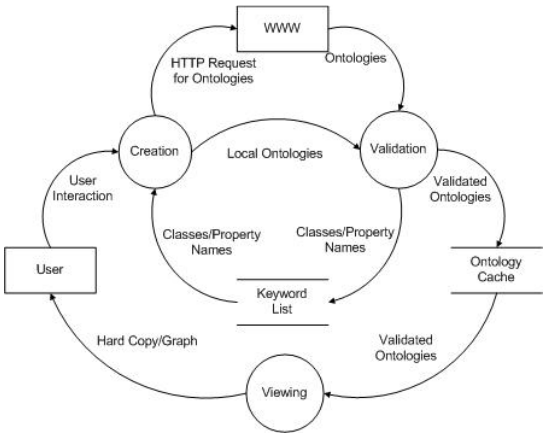

Fig. 1. General architecture of the OWL-S Editor

3 OWLSWIZ

The core feature in the OWL-S Editor is the wizard, which enables to create a full set of OWL-S ontologies from a WSDL description. This wizard presents to the user a step-by-step process in which the respective OWL-S ontologies are created. Initially, the user has to enter three basic types of information:

- A path to the input WSDL file
- The local target directory where the generated OWL-S description will be saved
- The base URL from where the description will be deployed.

The base URL directory is very important because it will be used to create the URI references of the various ontologies that comprise the OWL-S description. If this URL is incorrect, then the validation will not succeed but the description is created just the same. The second step involves the creation of the upper Service Ontology. Here the user can enter a general comment about the service and also add specific reference to imported URIs if the service is to make use of other ontologies. During the Profile generation step the user has to enter four basic types of information:

– A full human-readable Service Name
– A detailed human readable description of the service
– Business contact information of the entity providing the service
– A URL to the services rating

The business contact information is expressed as a modified VCard ontology. This ontology was simplified to enable to express simple types of information such as name, title, role, email and telephone number. For the service ratings, we are considering that there is some other service that provides rating information about services.

When creating the Process Model, the atomic processes and their inputs and outputs are extracted from the WSDL, but this process requires further interaction with the user, as shown in Figure 2 below. At this stage the user can make use of the Visual Composer to create a composite process by plugging together the extracted atomic processes. It is also possible for the user to visualize a tree representation of the extracted atomic processes and their parameters.

During the creation of the Grounding ontology, no user interaction is required because this ontology is completely generated from the WSDL. However, as in the previous step, the user may view a tree representation of the mappings between the process model and WSDL information. The final step in the wizard simply informs the user that the OWL-S description has been created successfully and shows also the information related to filenames of the ontologies.

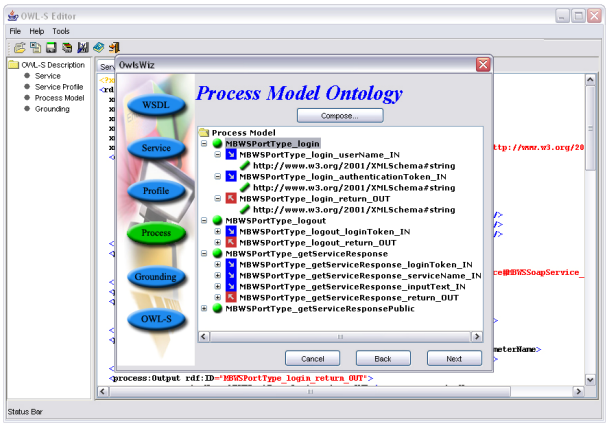

Fig. 2. Screenshot of the OwlsWiz at the Process Model generation step

4 Visual Composer

The Visual Composer is accessed from the wizard during the generation of the Service Model. The idea is to use the extracted atomic processes and their inputs/outputs to compose them graphically and automatically generate the markup. The graphical representation adopted in the composer involves the use of UML Activity Diagrams and the constructs implemented so far are Sequence, If-Then-Else and Split. These constructs were mapped to specific components in this diagrammatic representation. In order to differentiate between the different types of processes in OWL-S, some graphical changes had to be made to the Action component.

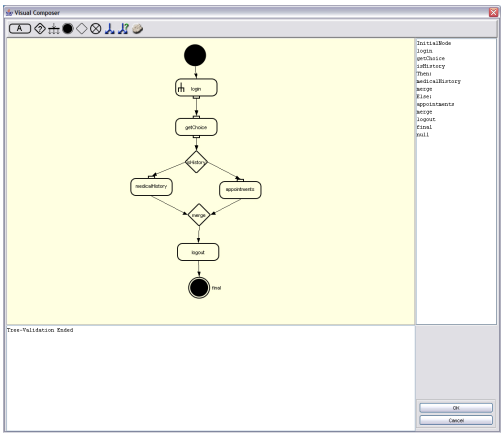

Fig. 3. A screenshot of the Visual Composer tool

To illustrate how the diagrams are mapped to the respective OWL-S control constructs and to processes, we will use a fictitious Health Information Service (HIS) [8] whose service composition is shown in Figure 3. A patient may log into the system and access two different services; a service which will return the patients medical history and another which returns the list of medical appointments.

Processes such as login, getChoice, medicalHistory, appointments and logout are represented as Action diagrams. Whenever a dataflow exists between processes, the notion of pins is used. The getChoice process for example has both an input and an output dataflow. OWL-S processes may also have preconditions and effects. We represent these with the rake symbol. This symbol is normally used to indicate that there are some sub-activities underlying the particular action. The If-Then-Else construct is represented using a Decision symbol. Such a construct must however by accompanied by a Merge in order to adhere to UML 2.0 specification which states that no action component can have more than one incoming edge. Although the example in Figure 3 does not use a split control construct, the latter is simply mapped to a fork symbol in the same representation.

4.1 Expressing Conditions and Effects

Currently, there is no specification in OWL-S that clearly defines how conditions and effects are to be expressed. This is somewhat changing in OWL-S 1.1 where an Expression construct defines the union between an AtomList (in SWRL) and a Formula (in DRS).

The Visual Composer adopts a subset of these languages in order to present the user with simple scenarios in which conditions, preconditions and effects can be expressed. Given the short time to develop these aspects, we couldnt implement full functionality of the mentioned languages but rather adopt adequate solutions for the most common cases in which the concepts expressed are used.

The condition of an If-Then-Else construct compares an output parameter of a process to a value. At present the wizard does not carry out the transformation from XSD complex datatypes to OWL concepts and this limits the comparison of parameters to atomic datatype values as expressed in the XML Schema Datatype specification. Hence, it is up to the user to make sure that the value to which the parameter is to be compared is of the same datatype. Optionally, the condition in an If-Then-Else may compare the outputs of two process parameters. In both cases, the conditions are

based upon a comparison of an equivalence mathematical operator (such as *equal to* or *greater than*) and the user must also make sure that the parameters are comparable with the selected operator. This type of condition is expressed as a DRS Atomic Formula construct where the subject is a reference to the process output parameter, the predicate is a reference to an equivalence operator and the object is either defined as a value or as a reference to another output parameter. The markup defined in the Process Model ontology at [8] is equivalent to the one generated for the condition of the isHistory If-Then-Else construct in Figure 3. The formula declares a variable of the same type as for the output parameter of the getChoice process and compares this to a value 0 as specified by the user. If the condition holds true, then the service lets the patient access the medicalHistory service.

For preconditions and effects, the composer adopts a rather different solution from the one above. We are assuming that there exist an ontology which defines the predicates and classes needed for the preconditions and effects defined by the particular service. Considering the login process in the HIS service, a typical precondition would involve a valid user name to be provided.

Depending on what the precondition or effect describes, a predicate may have two arguments rather than one. For example, the login process may have a loginUser effect which uses the user name and the respective authentication to let the user access the system. The idea is to use an IndividualPropertyAtom construct as defined in SWRL. This construct references to a propertyPredicate and to the arguments of the predicate, namely argument1 and argument2. In both cases, the respective variables must be declared.

4.2 Expressing Dataflow

The current version of OWL-S provides a simple way of handling dataflow. The idea is to use a sameValues property which defines ValueOf constructs which map the parameters of the processes. However, the problem with this technique is that if the same process instance is used in different parts of a choreography, the mapping of the parameters from the different calls to the process can be distinguished. The draft proposal for Version 1.1 is addressing this issue through the use of the Perform control construct which however was introduced late in the development stage of the Visual Composer at which point we had already opted for a different solution. The concept is basically the same as for the above mentioned construct but the idea of tags is used [2].

The use of this technique required an ontology which specifies the constructs to be used. The tag names are defined in the construct TagBind which is a sub-class of ControlConstruct. In the sequence control construct a set of Call constructs define a reference to the process to be executed and also to the associated tag name(s) that will be used to bind the processes parameters. Along with the set of Call constructs, another set of Dataflow construct is expressed. The number of such constructs will depend on how many dataflow bindings exist between the defined processes. For every single binding, a correspondent Dataflow construct must be expressed. This construct defines the properties source and destination. Both of these properties range over a ParameterInst class. The properties defined in this class include the tag name and a reference to the associated process and also whether the tag defines an input or an output flow. The markup of our fictitious HIS Process Model shows how such a technique can be used to define these types of dataflow.

5 Evaluation

Time constraints were very tight and there were several issues on which we had no control. Apart from that the API which transforms WSDL to OWL-S [13] was not available for download and

we had to adopt our own solution. We made the assumption that the input WSDL description doesnt define any types node. This limits the editor to handle atomic datatypes as defined in the XSD Schema Datatype Specification. However, the wizard was tested with extreme cases in which an input WSDL contained these types of elements. The result was that the information relative to atomic processes and their respective parameters was extracted correctly but without realizing the complex datatypes as OWL ontologies. This proved that the underlying engine performing this mapping is in fact robust.

The generated ontologies have been validated using the RDF Validation Service [18] and the OWL Ontology Validator [15]. When validating the base OWL-S ontologies, errors such as "Not a valid OWL DL subgraph" have been encountered. This error is generated because OWL-S falls under OWL-Full. As regards the process model ontologies we are still working on features regarding the way conditions, effects and dataflow are expressed. Most of the time we were basing our work on draft proposals that were described in papers (such as DRS and SWRL proposals) and mailing lists (Semantic Web Interest Group). Furthermore, one has to consider the new issues in OWL-S 1.1, specifically the expressions and dataflow binding mechanisms. We are still working on this part in order to generate ontologies that are compliant to OWL-S 1.1.

The use of standard APIs is also limited. During the initial stages of tool development, the Jena Toolkit [7] didnt support OWL Full. Thus we preferred to implement our own simple APIs in order to achieve our goals. Similarly, the OWL-S API [5] has been recently released at which point, most of the classes we needed were ready and was not feasible for us to switch to this API. However, we integrated the Jena Toolkit (version 2.1 with OWL Full support), which validates the ontologies created by the wizard.

The editor in general is limited in some editing features such as an efficient tokenizer for syntax highlighting purposes. Also, an efficient error handling and validation system is missing. The tool is limited to show errors in a separate window rather than pointing out in real time the errors or problems that the ontologies might have.

6 Related Work

At present, the number of tools available for creating OWL-S descriptions is limited though we expect that this number will increase in future months, as the OWL-S specification stabilizes and more research work is completed. We have found two tools that are somehow related to out editor: the Web Service Composer [5] and the Protégé OWL plugin [6].

The Web Service Composer consists of two modules, a visual composer and an inference engine. The former is used as an interface between the user and the engine and provides means to dynamically create a flow between a number of services. The inference engine is based on Prolog and handles OWL ontologies by converting the information into RDF triples and storing it in the composers knowledgebase. The engine uses a set of inbuilt axioms to provide service matching capability. It provides for two types of matches, namely an exact and a generic (subsumption) match. The composer makes use of a filtering mechanism to limit the number of services returned after the matchmaking process. The visual interface helps the user to create a flow between composable services and, generated automatically both the process model and the corresponding profile. The tool also provides for the execution of the composed service by invoking the individual services and passing data between the services according to the user-defined flow.

The Protg generic ontology editor enables to create individually the ontologies that make up an OWL-S service description. The tool presents graphically the properties, classes and individual which can be created. The user must be very confident with the constructs defined in the OWL-S

ontologies in order to create a whole description, and this is a bit of a drawback. When imports are added to the ontology, the asserted facts are also imported by making use of the in-built reasoner. These facts are listed in a tree view structure that makes it easy to understand the class hierarchies extracted from the imported ontologies. Furthermore, for service composition there is no standard graphical representation used.

7 Future Work

We plan to extend the tool to convert XSD complex datatypes into OWL concepts. This would require further work in other parts of the composer, such as in the parts where process parameters are compared to specific values.

A reasoner is another important feature that we are planning to add to the tool. This feature would enable classification and subsumption of the ontologies which are used by the service. It may also be used to automatically subsume the computed preconditions and effects of processes. This is however an area in OWL-S which is still under research and we hope that future version will fully tackle these issues.

The Visual Composer can be further extended to handle the other OWL-S control constructs. All the constructs defined in OWL-S can be represented using UML Activity Diagrams. Certain constructs are implicitly expressed in the visual composer (such as Sequence and dataflow mechanisms) while the others require the use of explicit constructs (such as Split, If-Then-Else, Split+Join etc). Also, the composer enables only to create a single composite process and hence it could be further extended to enable the creation of multiple composite processes.

Having the OWL-S API available, a slight re-design could be considered in order to make use of this standard official library. However, the OWL-S Editor presents some partial solutions with respect to conditions and dataflow that have not yet been implemented in this API. The idea is to extend the required classes of this library in order to maintain the features presented by the tool.

The user interface in general can be improved in various ways. A feature that will be added is a "help-in-context" system where the user may obtain the help needed at the particular step in the wizard. This would further ease the use of the tool and at the same time enable the user to get more familiar with OWL-S. Another feature is that of a markup completion tool where the editor suggests the lists of classes that can be used in some referenced ontology. Additionally, the reasoner may filter out these classes in order to provide only the classes and property names that are valid in the particular context.

8 Conclusion

The OWL-S Editor (OWL-S Editor site) tool is intended for users who need a fast way to create Semantic Web Service descriptions while using standard specifications such as WSDL and UML Activity Diagrams. The main objective behind this tool is to abstract away the underlying OWL-S construct details and to provide an easily accessible way of building complex descriptions. With this effort we wanted to consolidate on previous work and motivate others in adopting similar techniques and methodologies.

References

1. DAML Services, (November 2003), OWL-S, (Online), Available from: http://www.daml.org/services/owl-s/
2. Drew McDermott, (13 October 2003), Surface Syntax for OWL-S-PAI, (Online), Available from: http://www.daml.org/services/owl-s/1.0/surface.pdf
3. Drew McDermott, (12 January 2004), DRS: A Set of Conventions for Representing Logical Languages in RDF, (Online), Available from: http://www.daml.org/services/owl-s/1.0/DRSguide.pdf
4. Evren Sirin, (19 March 2004), OWL-S API, (Online), Available from: http://www.mindswap.org/2004/owl-s/api/
5. Evren Sirin, James Hendler, Bijan Parsia, (April 2003), *Semi-Automatic composition of Web Services using Semantic Descriptions*, Web Services: Modeling, Architecture and Infrastructure workshop (ICEIS 03). Angers, France
6. Holger Krubnauch, (25 May 2004), Protégé OWL Plugin, (Online), Available from: http://protege.stanford.edu/plugins.html
7. HP Labs Semantic Web Programme, (February 2004), Jena: A Semantic Web Framework for Java, (Online), Available from: http://jena.sourceforge.net/index.html
8. HIS, Health Information Service Ontologies, (Online), Available from: http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/his/
9. Ian Harrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, (19 November 2003), SWRL: A Semantic Web Rule Language combining OWL and RuleML, (Online), Available from: http://www.daml.org/2003/11/swrl/
10. IBM, (9 September 2002), BPWS4J, (Online), Available from: http://www.alphaworks.ibm.com/tech/bpws4j
11. IBM, (2002), Business Process Execution Language for Web Services, (Online), Available from: http://www-106.ibm.com/developerworks/library/ws-bpel1/
12. Martin Fowler, (2003), UML Distilled: A Brief guide to the Standard Object Modelling Language, 3rd ed, Addison-Wesley Pub Co
13. Massimo Paolucci, Naveen Srinivasan, Katia Sycara, Takuya Nishimura, (June 2003), Towards a Semantic Choreography of Web Services: From WSDL to DAML-S, Proceedings of First International Conference on Web Services (ICWS 03). Las Vegas, Nevada, USA, pp. 22-26
14. Microsoft. .NET Framework for Services, OWL-S Editor, (Online), Available from: http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/OwlSEdit.html
15. Sean Bachofer, Raphael Votz, (2003), OWL Ontology Validator, (Online), Available from: http://phoebus.cs.man.ac.uk:9999/OWL/Validator
16. Stanford Medical Informatics, (12 May 2004), Protege 2000, (Online), Available from: http://protege.stanford.edu/aboutus.html
17. Unified Modeling Language, (2003), UML 2.0, (Online), Available from: ¡http://www.uml.org/
18. World Wide Web Consortium, (19 August 2003), RDF Validation Service, (Online),Available from: http://www.w3.org/RDF/Validator/
19. World Wide Web Consortium, (8 August 2002), Web Services Choreography Interface, (Online), Available from: http://www.w3.org/TR/wsci/
20. World Wide Web Consortium, (26th March 2004), Web Service Description Language (Online), Available from: http://www.w3.org/TR/wsdl20/
21. World Wide Web Consortium, (2001), XML Schema, (Online), Available from: http://www.w3.org/XML/Schema