Guidelines and Tool for Meaningful OWL-S Services Annotations

Domenico Redavid, Stefano Ferilli, Berardina De Carolis and Floriana Esposito
Computer Science Department, University of Bari "Aldo Moro", Via E. Orabona, 4, Bari, Italy

Keywords: Semantic Web Services, OWL-S Annotation, OWL, SWRL.

Abstract: The current tools to create OWL-S annotations have been designed starting from the knowledge engineer's point of view. Unfortunately, the formalisms underlying Semantic Web languages are often incomprehensible to the developers of Web services. To bridge this gap, it is desirable that developers are provided with suitable tools that do not necessarily require knowledge of these languages in order to create annotations on Web services. With reference to some characteristics of the involved technologies, this work addresses these issues, proposing guidelines that can improve the annotation activity of Web service developers. Following these guidelines, we also designed a tool that allows a Web service developer to annotate Web services without requiring him to have a deep knowledge of Semantic Web languages. A prototype of such a tool is presented and discussed in this paper.

1 INTRODUCTION

Ontologies are written by knowledge engineers and require specific skills for their use (Staab and Studer, 2004). Formalisms such as Description Logics (Baader et al., 2003), underlying the Web Ontology Language (OWL)[1] standards, are incomprehensible to the developers of Web services (Erl, 2005). As it happened for the Web, where increasingly powerful browsers have allowed ordinary users to use the Internet without knowing languages such as HTML or XML, in order to promote the technologies related to Semantic Web Services (SWS) (McIlraith et al., 2001), it is desirable that developers are provided with tools that do not necessarily require knowledge of languages such as Resource Description Framework (RDF)[2], OWL, OWL for services (OWL-S)[3] and Semantic Web Rule Language (SWRL)[4].

The objective of this work is to propose solutions that simplify the process of annotating SWSs in order to facilitate the operation of automatic service composition. The technical experts that are in charge of writing the Web services must be able to apply for compositions of services that meet given goals and requirements, and to get results expressed in languages and formalisms that are easy to understand according to their knowledge.

[1] http://www.w3.org/TR/owl2-new-features
[2] http://www.w3.org/RDF
[3] http://www.w3.org/Submission/OWL-S
[4] http://www.w3.org/Submission/SWRL

2 TECHNOLOGICAL BACKGROUND

A Web service, as defined by the World Wide Web Consortium (W3C)[5] is a software system designed to support interoperability between different machines on the same network. It has an interface that describes the service in a suitable language, through which other machines can interact with it. This interaction takes place through the exchange of SOAP messages, formatted according to the XML standard and typically exchanged using the HTTP protocol. A Web service is an abstract interface that needs to be implemented by a software agent. The agent is the concrete part (software or hardware) that sends and receives messages, while the Web service is a resource that contains the abstract set of functionalities provided by the agent. In a practical context, the agent maintaining the same functionalities can vary continuously. For example, the agent's implementation might be modified or rewritten in a different language, while the Web service stays unchanged over time. Thus, a Web service is independent from both implementation language and platform type. For it to be used, in general, a service must be described and advertised (Walsh, 2002). The Web Service Description Language (WSDL) specifications[6] have been designed for this purpose. WSDL provides all the de-

[5] http://www.w3.org
[6] http://www.w3.org/TR/wsdl

Redavid D., Ferilli S., De Carolis B. and Esposito F..
Guidelines and Tool for Meaningful OWL-S Services Annotations.
DOI: 10.5220/0005160601300137
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2014), pages 130-137
ISBN: 978-989-758-049-9

tails needed to invoke the service operations. Once the WSDL is created, it can be published in a register called universal Universal Description Discovery and Integration (UDDI) [7]. Using WSDL and UDDI any requester may seek for an appropriate service (i.e., having the desired characteristics) and understand how to invoke it.

The strengths of Web services can be summarized as follows:

- they allow interoperability between different software applications and on different hardware/software platforms;
- they are specified using a text-based data format, which is more understandable and easier to use for developers;
- as they are based on the HTTP protocol, no change is required to the security rules used as firewall filters;
- once published, they can be combined with each other (no matter who provides them and where they are made available) to form integrated complex services;
- they allow reuse of already developed applications;
- as long as the interface remains constant, the changes made to the service remain transparent.

Despite their many strengths, Web services have limitations for which solutions must be found. The tasks of search, selection, composition and execution of Web services are delegated to the developers, since they know the semantics underlying each service. In this way, they can obtain suitable combinations which result in complex applications. In this practice lies the main limitation of the commonly used Web services. Indeed, since Web services were created to be used by machines, delegating to humans such tasks as discovery, selection and composition is too strict a constraint. To be able to automate these steps, machines should be able to understand the semantics of the services. This is achieved by enriching the existing Web services with a semantic layer that expresses the functionality of the service in a machine-understandable way.

The choice of enriching existing resources is clearly important in the context of the Web, where rewriting the information architecture is not feasible. The key to overcoming the syntax limitations is provided by the Semantic Web(Berners-Lee et al., 2001). Making the Web machine-understandable is the specific aim of the Semantic Web. Currently, the machines maintain information without understanding its meaning. When a search engine like Google stores Web pages in its cache, it is not able to distinguish whether the word 'espresso' refers to a train or a type of coffee, or if 'verdi' is a color or the name of a composer. With the Semantic Web this limitation is overcome, since the resources in the network are associated with information that specifies their semantics in a format suitable to be interpreted and processed automatically.

2.1 OWL-S and SWRL Characteristics

OWL-S enables semantic descriptions of Web services using the *Service Model* ontology, that defines the OWL-S process model. Each process is based on an IOPR (Inputs, Outputs, Preconditions, and Results) model. The *Inputs* represent the information that is required for the execution of the process. The *Outputs* represent the information that the process returns to the requester[8]. *Preconditions* are conditions that are imposed over the *Inputs* of the process and that must hold for the process to be successfully invoked. Since an OWL-S process may have several results with corresponding outputs, the *Results* entity of the IOPR model provides a means to specify this situation. Each result can be associated to a result condition, called *inCondition*, that specifies when that particular result may occur. It is assumed that such conditions are mutually exclusive, so that only one result can be obtained for each possible situation. When an *inCondition* is satisfied, there are properties associated to this event that specify the corresponding output (*withOutput* property) and, possibly, the *Effects* (*hasEffect* properties) produced by the execution of the process. *Effects* are changes in the state of the world. The OWL-S conditions (*Preconditions*, *inConditions* and *Effects*) are represented as logical formulas. Since OWL-DL offers limited support to formulate constructs such as property compositions without becoming undecidable, a more powerful language is required for the representation of OWL-S conditions. For this reason, in OWL-S these logical formulas are represented as simple string literals or XML literals. The former allow to use languages such as Planning Domain Definition Language (PDDL) (McDermott, 1998) and Knowledge Interchange Format (KIF)[9], on which efficient process-oriented reasoning systems can be applied. Since these systems work under the Closed World Assumption, a change of representation language from

[7] http://www.uddi.org/pubs/uddi_v3.htm

[8] Inputs, Outputs and Local variables (entities used within the process) are SWRL variables and their types are defined in the domain ontology.

[9] http://www-ksl.stanford.edu/knowledge-sharing/kif

DL is required. This requirement is the main disadvantage, since it generates a change of semantics where the presence of all necessary semantic constructs is not guaranteed (Borgida, 1996). The latter allows to use languages whose standard encoding is in XML, such as SPARQL[10] and SWRL (Horrocks et al., 2005). Since these are SW languages, their use overcomes the loss of semantics. SPARQL, however, was born as a query language for RDF, therefore the OWL-DL representation of effects is problematic. Moreover, SWRL is undecidable. A solution for this problem has been proposed in (Motik et al., 2005), where decidability is achieved by restricting the application of SWRL rules only to the individuals explicitly introduced in the ontology. This kind of SWRL rules, called DL-safe, makes this language the best candidate for representing OWL-S conditions (Redavid et al., 2013). Let us now briefly mention the features of SWRL that are relevant to our aims. WRL extends the set of OWL axioms to include *Horn-like* rules in the form of implications between an antecedent (body) and consequent (head), both consist of zero or more conjunctive atoms having one of the following forms:

- $C(x)$, with C an OWL class, $P(x,y)$, with P an OWL property,
- *sameAs(x,y)* or *differentFrom(x,y)*, equivalent to the respective OWL properties,
- $builtIn(r,z_1,\ldots,z_n)$, functions over primitive datatypes.

where x, y are variables, OWL individuals or OWL data values, and r is a built-in relation between $z_1,\ldots,z_n$ (e.g., $builtIn(greaterThan,z_1,z_2)$). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent hold also. A rule with conjunctive consequent can be transformed into multiple rules by means of Lloyd-Topor transformations. Each rule has an atomic consequent.

2.2 Related Works

To the best of our knowledge, there are no recent proposal of OWL-S tools for the OWL-S annotation. The Protégé (Knublauch et al., 2004) plugin called OWL-S Editor (Elenius et al., 2005) is most popular tool for the creation of OWL-S descriptions.

The OWL-S tab can be considered as the main point of user interaction, providing a more direct view of the OWL-S classes and instances than what Protégé provides by default. It contains the necessary panel representing all instances of the main OWL-S classes: Service, Profile, Process, and Grounding. Furthermore, it has the following

- WSDL Support to create a "skeleton" of OWL-S description based on a preexisting WSDL file.
- IOPR description and management.
- Graphical Overview of the "forest" of relationships.
- an integrated execution environment for the OWL-S so that developers could verify that their specifications reflect their intentions, and to try out different possibilities before deploying their services.
- Process Modeling to model composite processes. A composite process is constructed from subprocesses that can in turn be composite, atomic, or simple.

This plugin is available only for an old version of Protégé having a limited suport to OWL.

The OWL-S IDE project[11] is also concerned with the development of OWL-S services. The OWL-S IDE is a plugin for Eclipse[12] , which attempts to integrate the semantic markup with the programming environment. Developers can write their Java code in Eclipse, and run an ad hoc tool to generate an OWL-S "skeleton" directly from the Java sources. The idea of integrating SWSs more closely with the programming environ- ment used to develop the service implementations is a good one. However, Eclipse does not support ontology editing, and there is no KB from which to choose the domain concepts to which the OWL-S files should relate. Furthermore, it will often be more useful to generate the semantic markup before the Java (or other) code, as the semantic descriptions can be seen as a higher level of abstraction of the programming modules. The OWL-S IDE does not provide any graphical visualization of services or processes.

Another OWL-S Editor is presented in (Scicluna et al., 2004). It is a stand-alone program, providing WSDL import as well as a graphical editor and visualization for control flow and data flow definition. It is not integrated with an ontology editor and shares some of the drawbacks of the OWL-S IDE.

ODE SWS is a tool for editing SWSs "at the knowledge level" (Gómez-Pérez et al., 2004), describing services following a Problem-Solving Methods (PSMs) (Fensel and Motta, 2001) approach. The annotation task plays a subordinate role in this envi-

[10] http://www.w3.org/TR/rdf-sparql-query

[11] http://projects.semwebcentral.org/projects/owl-s-ide

[12] www.eclipse.org

ronment, whereas a simplified vision of the OWL-S annotation procedure is the main focus of our work.

The manage of SWRL rules the most popular tool is the SWRLTab a development environment integrated in Protégé-OWL[13]. It supports the editing and execution of SWRL rules and includes a set of libraries that can be used in rules, including libraries to interoperate with XML documents, and spreadsheets, and libraries with mathematical, string, RDFS, and temporal operators. A SWRL-based OWL query language called SQWRL (Semantic Query-Enhanced Web Rule Language)(O'Connor and Das, 2008) is also provided. This plugin was developed for Protégé-OWL version 3. Although keeps unchanged the SWRL semantics, it presents some issues: the OWL's open world assumption is not guaranteed leading, in some situations, to nonmonotonicity. Unlike OWL and SWRL, SQWRL adopts the unique name assumption when querying.

The main difference between these tool and our proposal lies in the fact that they are oriented to knowledge engineering rather than a WSDL developer.

3 GUIDELINES FOR A CORRECT ANNOTATION

With reference to the OWL-S and SWRL characteristics described in Sect. 2.1, we propose the following guidelines for encoding an OWL-S process into SWRL rules. These guidelines provide precise indications to develop a prototype.

1. For every result of the process, there exists an *inCondition* that expresses the binding between input variables and that particular result's (output or effect) variables.
2. Every *inCondition* related to a particular result will appear in the antecedent of each resulting rule, whilst the *Result* will appear in the consequent. An *inCondition* is valid if it contains all the variables appearing in the *Result*.
3. If the *Result* contains an *Effect* made up of many atoms, the rule will be split into as many rules as the atoms. Each resulting rule will have the same inCondition as the antecedent and a single atom as the consequent.
4. The *Preconditions* are conditions that must be true in order to execute the service. Since these conditions involve only the process *Inputs*, they will appear in the antecedent of each resulting rule together with *inConditions*. In this work we consider all the *Preconditions* as being always true.

The first guideline is necessary because there may be processes where the binding is implicit. For example, consider an atomic process having a single output. Due to the single output, an explicit declaration of the inCondition is not necessary. However, if the inCondition is not specified, the second guideline is not applicable. To overcome this problem, we add a new inCondition that makes explicit the implied binding in an atomic processes with one output. This inCondition will be represented as an SWRL atom that is always true, and therefore as an OWL property, that will bind input and output in explicit way. In addition, in order to represent SWSs using SWRL rules, one needs to represent in them also preconditions, inConditions and effects explicitly declared in the service, that is, the entire IOPR model. Preconditions, inConditions and effects are represented as SWRL logical formulas (i.e., as an antecedent or a consequent). These logical formulas can be combined in order to obtain SWRL rules as reported in Table 1.

Table 1: OWL-S representation by means of a SWRL rule.

BODY	HEAD
Preconditions $\wedge$ inCondition	{Outputs} $\wedge$ Effect

Finally, a general problem concerns how the Web services are annotated. Web services must be annotated using ontological classes only, without ever using primitive types (string, int, etc.) as their semantics is too general. For example, consider an SWS for searching for books, with the following input:

```
<process:Input rdf:ID="_CAR">
<process:parameterType
rdf:resource="xsd:#string"/>
</process:Input>

*where xsd = http://www.w3.org/2001/XMLSchema
```

Due to the fact that `process:parameterType` is declared as a datatype, the class of this input is an XML Schema datatype (string) instead of being an entity belonging to the domain ontology. This problem becomes critical during the operation of SWS composition. In fact, suppose that we need to find a service whose output type is the same as the input type defined in the example. Since such an input is of type string, any service returning a string can be composed with that SWS. In this way, the final result might be incorrect due to the semantics of the primitive types. The problem is solved by annotating the parameters of the input and output using ontological classes without ever using primitive types. Referring

[13] http://protegewiki.stanford.edu/wiki/Protege-OWL

to the previous example, the input parameter should annotated with the class *Car* of an ontology describing vehicles.

```
<process:Input rdf:ID="_CAR">
<process:parameterType
rdf:resource="&kb;#Car"/>
</process:Input>

*where &kb; is the ontology URI of Car
```

The Fig. 1 show the correct placing and relations between Web service and SWS.

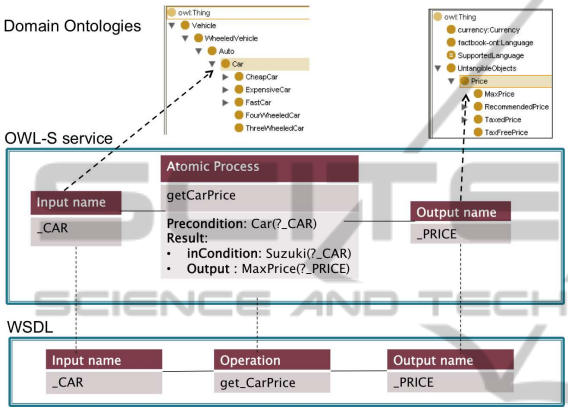

Figure 1: SWS abstraction layer.

4 OWL-S SEMANTIC ANNOTATOR

To ensure the proper annotation of Web services and a proper division of tasks among different users, we have developed a tool that allows *ad hoc* annotation using only the classes in the domain ontologies, thereby taking into proper consideration the dichotomy between the knowledge engineer and the developer of Web services. Thus, the developer of Web services will be able to annotate Web services without necessarily knowing the Semantic Web ontology and rule languages. Moreover, in a context where there are various professional roles, each in charge of specific tasks, a clear division of tasks must be ensured. For example, there may be users in charge of annotating Web services and others that must release them after checking. Once Web services are annotated, it is necessary to ensure their persistence in such a way that reusability is guaranteed. In this way, it will be possible to search for existing OWL-S descriptions through the Web before creating a new one. At the end of this section an abstract solution to manage OWL-S description will be proposed.

To ensure that every user has a defined role that allows him to perform only certain operations, the following subdivision of roles have been identified:

1. Annotators: can only create the OWL-S descriptions.
2. Publishers: can also publish the OWL-S descriptions.
3. Group Leaders: can assign permissions (Annotator, Publisher, Group Leader) to existing users, edit their data (username and password) or create new users.
4. Administrators: can perform all the operations of group leaders, and also manage the settings for the connection to the repository for the publication of OWL-S descriptions.

For it to be published, an OWL-S description requires a common Web server that can potentially be deployed anywhere on the Web. Once it has been created, a mechanism is needed that allows the localization of the resource on a physical machine which may be different from the one where the OWL-S annotator works. Currently, our prototype uses the File Transfer Protocol (FTP)[14] for this purpose.

4.1 A Prototype Implementation

In this section we present the details about the annotation procedure of the OWL-S annotator. Starting from a WSDL document, the user can create an OWL-S service by simply choosing an OWL domain ontology and annotating every single parameter of the service with an ontology class selected from the ontological class hierarchy (i.e., the taxonomic view of the ontology). Furthermore, once inputs and outputs have been annotated, the user can declare logical conditions corresponding to preconditions, inConditions, and Effects in an intuitive manner.

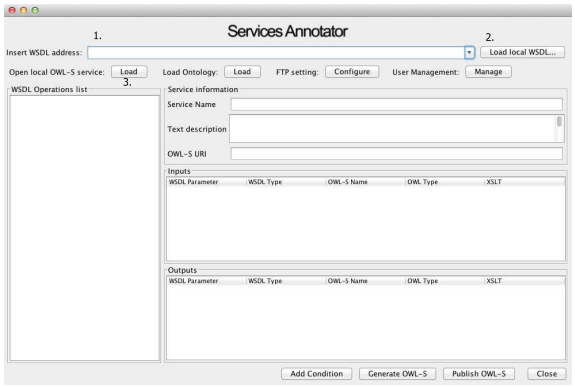

Figure 2: OWL-S annotator main page.

[14]http://www.w3.org/Protocols/rfc959/

The interface shown in Fig. 2 allows to:

1. enter the URL of the WSDL of the Web service to be annotated;
2. select a local WSDL document to be annotated;
3. load a previously created OWL-S description for editing purposes.

After loading a WSDL or an existing OWL-S description, the functionality of the various numbered items in the graphical interface shown in Fig. 3) is as follows:

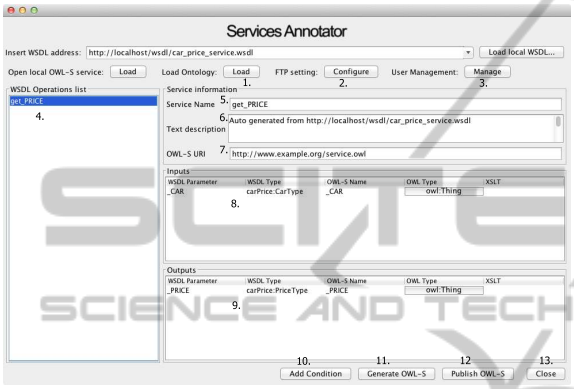

Figure 3: OWL-S annotator core characteristics.

1. allows to choose the domain ontologies to be used for annotating the parameters;
2. allows to configure the default settings for the publication of OWL-S (this option is available only to the Administrator);
3. is a button to access the panel that allows to manage users, register new ones, change their username and password and assign a set of permissions;
4. contains the list of all operations in the selected WSDL (for each of these operations an OWL-S atomic service can be created);
5. reports the name of the OWL-S service;
6. reports a text description of the OWL-S service;
7. reports the URI of the OWL-S service;
8. is the list of input parameters (for each of them, the name in OWL-S can be changed and an annotation can be made by clicking in the 'OWL class' column);
9. is the list of output parameters (for each of them, the name in OWL-S can be changed and an annotation can be made by clicking in the 'OWL class' column);
10. is a button to access the panel for the inserting SWRL logical constructs;
11. is a button for generating the OWL-S description;
12. is a button for publishing the OWL-S description on the Web (this option is not permitted to Annotator users);
13. is a button to close the application.

4.2 SWRL Annotation

Once all the service parameters have been annotated, one may create the service or continue with the addition of SWRL logical formulas by clicking on the 'SWRL constructs' panel. More specifically, it:

1. includes a Tab to enter preconditions;
2. includes a Tab to enter output, effects, and under what conditions they occur;
3. can be used to insert an OWL class in preconditions, selecting it from a tree view of the taxonomy (as in the case of record of the parameters of the service);
4. can be used to insert an OWL Property in preconditions.
5. after selecting a class or property, allows to enter the process parameter that will be the argument of the selected class or property;
6. after selecting a class or property, allows to enter the individual who will be the argument of the selected class or property;
7. displays in a comprehensible way all atoms entered for the SWRL construct;
8. allows to cancel the atoms entered for the SWRL construct;
9. allows to add the SWRL construct to the process;
10. contains a complete list of all the SWRL constructs entered in the process;
11. allows to save the changes.

By opening the 'Output/Effects' panel, the window shown in Fig. 4 is displayed. It consists of a panel that includes three tabs. These tabs allow to specify 'Output', 'Effects' and 'Conditions', respectively. In the Output panel the user can select the output to be returned, by clicking on the 'Output' button. Then, if the process returns multiple outputs, he will be asked to select which output must be returned. Let us now turn to examine the remaining tabs: 'Effects' and 'Conditions'. The elements contained in these tabs are the same as those appearing in the tab 'Preconditions'. The buttons carry out the same functions as those described for the Preconditions tab, as well. The difference lies in the fact that by selecting some items from the Preconditions tab they will be

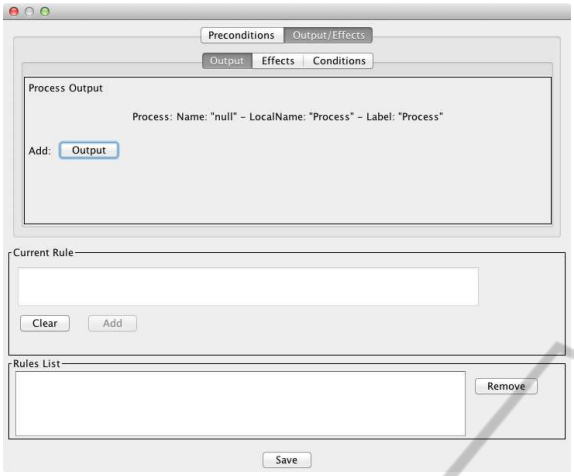

Figure 4: The “Output / Effects” panel.

added as preconditions of the process, while selecting them from the Effects tab they will be added as the effects of the process, and selecting them from the Conditions tab they will be added as necessary conditions to return specific effects and/or specific output.

Let us now show an example of insertion. Our SWRL construct will contain two conditions that are combined in order to form a logical condition. When these conditions are verified, the returned output will belong to a particular OWL class. Specifically, suppose the user wants to define that “the price of Suzuki cheap cars the service will return must be of type RecommendedPrice” (an OWL class). Then he must define the following two conditions on the input parameter *_CAR*:

- *Suzuki(?_CAR)*,
- *CheapCar(?_CAR)*.

To define these conditions he opens the tab ‘Output and Effects’, selects the tab ‘Conditions’ and clicks on the button to select the class ‘Suzuki’. Once the class is selected, he clicks on the button ‘Variable’ and selects the parameter. He repeats the same procedure for the second condition, this time selecting the class ‘*CheapCar*’. Now he must define the fact that the output returned by the service will be of type ‘*RecommendedPrice(?_CAR)*’. To do this, he selects the tab ‘Output’ and, after clicking on the ‘Output’ button, he selects the class ‘*CheapCar*’. The entered SWRL construct will be displayed as shown in Fig. 5.

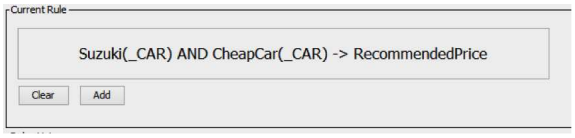

Figure 5: SWRL construct display panel.

Through this view he may have a summary of the SWRL atoms included in the construct. After entering all the constructs he deems as appropriate, he can go back to the main screen and generate the OWL-S service he has just created by means of the button ‘Generate OWL-S’. Later, he will also be able to publish it on the Web using the button ‘Publish’.

5 CONCLUSIONS AND FUTURE WORKS

This work aimed at providing a tool for manual annotation of Web services, that would take into account the strengths and weaknesses of Semantic Web technologies. We started from the description of Web services, by specifying their meaning within the Web community and describing the WSDL, the de facto XML-based standard that allows to abstract away from the concrete implementation of the service. Then we analyzed the syntactic limitations of Web Services, and specifically their constraining human experts to search, select, compose and execute the Web Service. The key to overcoming the syntax limitations is provided by the Semantic Web. In particular, OWL allows to specify formal ontologies that can be used to attach a meaning to WSDL inputs and outputs. OWL-S is an OWL ontology that models Web services at the abstract level by simplifying the semantic association to the input parameters and output. In order to specify the process model of a service, rule based languages are necessary. As we have seen, OWL-S allows to specify logical constructs in SWRL, an extension of OWL, which enables the description of preconditions and effects.

This paper proposed a set of guidelines for a correct annotation, and implemented them in a tool that can be used by a Web service developer that is not an expert in Semantic Web technologies in order to annotate his services. As future works we plan to extend the ontology management tab in order to further facilitate the developer during the annotation procedure, and to introduce the management of OWL-S composite processes, a validator that will allow to check the correctness of the descriptions before generating the OWL-S service, and a graphical tab to visualize the complex services. Furthermore, and most important, we plan to introduce additional modules that exploit NLP approaches able to suggest the most appropriate OWL entities to be used for annotation.

ACKNOWLEDGEMENTS

This work was partially funded by the Italian PON 2007-2013 project PON02_00563_3489339 'Puglia@Service'.

REFERENCES

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., editors (2003). *The Description Logic Handbook*. Cambridge University Press.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific American*.

Borgida, A. (1996). On the relative expressiveness of description logics and predicate logics. *Artif. Intell.*, 82(1-2):353–367.

Elenius, D., Denker, G., Martin, D., Gilham, F., Khouri, J., Saadati, S., and Senanayake, R. (2005). The OWL-S Editor - A Development Tool for Semantic Web Services. In Gómez-Pérez, A. and Euzenat, J., editors, *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 78–92. Springer.

Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Fensel, D. and Motta, E. (2001). Structured Development of Problem Solving Methods. *IEEE Trans. Knowl. Data Eng.*, 13(6):913–932.

Gómez-Pérez, A., González-Cabero, R., and Lama, M. (2004). Development of Semantic Web Services at the Knowledge Level. In Zhang, L.-J., editor, *ECOWS*, volume 3250 of *Lecture Notes in Computer Science*, pages 72–86. Springer.

Horrocks, I., Patel-Schneider, P. F., Bechhofer, S., and Tsarkov, D. (2005). OWL rules: A proposal and prototype implementation. *J. of Web Semantics*, 3(1):23–40.

Knublauch, H., Fergerson, R. W., Noy, N. F., and Musen, M. A. (2004). The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In McIlraith, S. ., Plexousakis, D., and van, Harmelen, r. a. n. k., editors, *The Semantic Web ISWC 2004*, volume 3298 of *Lecture Notes in Computer Science*, chapter 17, pages 229–243. Springer Berlin / Heidelberg, Berlin, Heidelberg.

McDermott, D. (1998). PDDL — the planning domain definition language.

McIlraith, S. A., Son, T. C., and Zeng, H. (2001). Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53.

Motik, B., Sattler, U., and Studer, R. (2005). Query Answering for OWL-DL with rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60.

O'Connor, M. J. and Das, A. K. (2008). SQWRL: A Query Language for OWL. In Hoekstra, R. and Patel-Schneider, P. F., editors, *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Redavid, D., Ferilli, S., and Esposito, F. (2013). Towards Dynamic Orchestration of Semantic Web Services. *T. Computational Collective Intelligence*, 10:16–30.

Scicluna, J., Abela, C., and Montebello, M. (2004). Visual Modelling of OWL-S Services. In *IADIS International Conference WWW/Internet*.

Staab, S. and Studer, R., editors (2004). *Handbook on Ontologies*. International Handbooks on Information Systems. Springer.

Walsh, A. E., editor (2002). *Uddi, Soap, and Wsdl: The Web Services Specification Reference Book*. Prentice Hall Professional Technical Reference.