OWL-S Atomic services composition with SWRL rules

Domenico Redavid[1], Luigi Iannone[2], and Terry Payne[3]

[1] Dipartimento di Informatica, Università degli Studi di Bari
Campus Universitario, Via Orabona 4, 70125 Bari, Italy
{redavid}@di.uniba.it
[2] Computer Science Dept., University of Liverpool
Ashton Building, Ashton Street L69 3BX, Liverpool UK
{L.Iannone}@csc.liv.ac.uk
[3] School of Electronics and Computer Science, University of Southampton
Southampton, SO17 1BJ, United Kingdom
{trp}@ecs.soton.ac.uk

Abstract. This paper presents a method for encoding OWL-S atomic processes by means of SWRL rules and composing them using a backward search planning algorithm. A description of the preliminary prototype implementation is also presented.

1 Introduction

Semantic Web (SW) aims at proposing standards, tools and languages for knowledge representation on the Web. Amongst the other issues, it deals with the provision of semantics to Web Services in order to achieve a more abstract and flexible automation. The result of this effort is the notion of Semantic Web Services (SWS) [1]. This term refers to traditional Web services that have been annotated by means of SW languages and techniques so as to make possible their automatic discovery, composition and invocation. In order to achieve that, in literature there are different approaches which produced different frameworks, among which the most widespread are OWL-S [2], WSMO [3] and WSDL-S [4].

In this paper we will focus on OWL-S as underlying language for annotating Web Services. OWL-S provides an ontological framework based on which an abstract description of a service can be created. It is an upper ontology whose root class is the *Service* class that directly corresponds to the actual service that is described semantically (every service that is described maps onto an instance of this concept). The upper level *Service* class is associated with three other classes: *ServiceProfile* (specifies the functionality of a service), *ServiceModel* (specifies how to ask for the service and what happens when the service is carried out) and *ServiceGrounding* (specifies how the service has to be invoked). In particular, the service model tells a client how to use the service, by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes. For nontrivial services (those composed of several steps over time), this description may be used by a service-seeking agent in different ways.

The *ServiceModel* defines the concept *Process* that describes the composition of one or more services in terms of their constituent processes. A *Process* can be atomic (a non-decomposable service), composite (a set of processes within some control structure that defines a workflow) or simple (a service abstraction).

In this paper our aim is the composition of OWL-S atomic processes adopting SWRL [5] as language for the representation of their IOPR (Inputs, Outputs, Preconditions and Results) models. Such SWRL descriptions are used as input to generate candidate service compositions in order to achieve a given goal.

The rest of the paper is organized as follows: in section 2 we report the basic notions of the OWL-S process model with some considerations on the guidelines that should be followed in order to have useful metadata for the Web services to be described. Section 3 identifies some requirements needed for encoding an OWL-S atomic process by means of SWRL rules. An algorithm for SWRL rules composition is described in section 4. In section 5 an application example that shows the applicability of our method is presented, while sections 6 and 7 are devoted to related work and conclusions, respectively.

2 Preliminary Considerations

In this section we report the basic notions about the OWL-S process model with some considerations on the guidelines that should be followed in order to have useful metadata for the Web services to be described.

Each OWL-S process [2] is based on an IOPR 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. *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 *Result* 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 can occur. Therefore, an inCondition binds inputs to the corresponding outputs. 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. If needed, OWL-S provides some extra variables, called *ResultVars* and *Existentials*[4], that can be used in these formulas.

Formally, *Input* and *Output* are subclasses of the more general class *Parameter* declared in its turn as a subclass of *Variable* in SWRL ontology. Every parameter has a type, specified using a URI. Such type is needed to refer it to an entity within the domain knowledge of the service. The type can be either a *Class* or a *Datatype* (i.e.: a concrete domain object such as a string, a number, a date and so on) in the domain knowledge. Nevertheless, we argue that providing descriptions of Web services parameters using

[4] These entities appeared in OWL-S 1.2 Pre-Release, available at: http://www.ai.sri.com/daml/services/owl-s/1.2/

concrete datatypes gives very little in terms of added semantics. For example, consider the following declaration of the input in a process that retrieves books:

```
<process:Input rdf:ID="BookName">
    <process:parameterType
        rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</process:Input>
```

The fact that *process:parameterType* is declared as datatype means that the reference knowledge model of this input parameter is a concrete XML Schema datatype (string) instead of being an entity within a domain ontology. This mismatch becomes critical in automatic composition of services. Indeed, suppose that, during an hypothetical composition process, we need to find another service whose output will be fed into the service described above. Our composer, then, must necessarily consider those services that have as output a resource of the same type of our input parameter. In the example above, this type is string, hence every service that returns a string as an output can be composed with our service. Therefore, this would result in meaningless compositions of totally unrelated services due to the fact that parameters have been semantically poorly described. In the rest of this paper we consider only those services that have parameters (i.e. *Inputs* and *Outputs*) declared as entities in a domain ontology (i.e. not as datatype).

3 Encoding OWL-S atomic processes with SWRL rules

The aim of this section is to illustrate our approach for transforming process descriptions into sets of rules expressed in an ontology-aware rule language, namely Semantic Web Rule Language (SWRL). The motivation for doing this lies in the fact that, starting from this rule-based representation, an algorithm for detecting possible sequential composition of services (described in section 4) can be applied. SWRL [5] extends the set of OWL [6] axioms to include Horn-like rules [7]. The proposed rules are in the form of an implication between an antecedent (body) and consequent (head); both consist of zero or more conjunctive atoms. The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. An important characteristic of the rules is *safety*, i.e. only variables that occur in the antecedent of a rule may occur in the consequent. Furthermore, a rule with conjunctive consequent can be transformed into multiple rules each with an atomic consequent (Lloyd-Topor transformations [7]). A SWRL weakness is the non decidability of the whole language. A solution to this problem has been proposed in [8] where decidability is achieved by restricting application of SWRL rules to individuals explicitly introduced in the *ABox*. This kind of SWRL rules are called DL-safe.

In order to perform the transformation we impose some requirements on OWL-S process descriptions. It is necessary that all the entities within the process model are described in terms of a domain ontology. This means, basically, that Inputs and Outputs types, in order to satisfy this requirement, cannot be datatypes.

Within OWL-S, conditions (logical formulas) are either string literals or XML literals. The latter case is used for languages whose standard encoding is in XML, such

as SWRL. Body and head are logical formulas, whereby the OWL-S conditions can be identified with the body or with the head of a SWRL rule. Such conditions are expressed over *Input* and *Output*. Therefore, if the above requirement is met, conditions will be also expressed in terms of a domain ontology and will hence have the right level of abstraction.

After these considerations, we can describe the guidelines we follow for encoding an OWL-S process into SWRL.

- For every result of the process there exists an *inCondition* that expresses the binding between inputs variables and the particular result (output or effect) variables.
- 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*.
- If the *Result* contains an *Effect* composed of more atoms, the rule will be split into as many rules as the atoms are. Each resulting rule will have the same inCondition as antecedent and a single atom as consequent.
- The *preconditions* are conditions that must be true in order to execute the service. Since these conditions involve only the process *Inputs*, the corresponding SWRL rules should have the condition as antecedent and a boolean predicate that indicates whether the condition is true or not. In this work we consider always true all the *Preconditions*.

The first guideline is needed because there may be processes in which such binding is implicit in their OWL-S descriptions. Let us consider, for example, an atomic process having a single output. In this case there might be no *inCondition* binding inputs and output variables since, being the output the unique outcome, such binding is obvious. In this case, though, our encoding with SWRL rules would not be possible because the second guideline is not applicable. However, we can add a new *inCondition* that makes explicit such implicit binding.

For example, suppose we have a process declared only with the following *Parameters*, without *inCondition*:

```
<process:Input rdf:ID="BookName">
   <process:parameterType rdf:datatype="&xsd;#anyURI"> &kb;#BookTitle
   </process:parameterType>
</process:Input>

<process:Output rdf:ID="BookInfo">
   <process:parameterType rdf:datatype="&xsd;#anyURI"> &bibtex;#Book
   </process:parameterType>
</process:Output>
```

we should write the corresponding rule as follows:

```
kb:BookTitle(?process:BookName) → bibtex:Book(?process:BookInfo)
```

but the variable *process:BookInfo* does not appear in the antecedent of the rule (i.e. in the *inCondition*), consequently this is not a valid SWRL rule. Since every service

produces the output manipulating the inputs, we can suppose that there exists a predicate (*hasTransf* predicate) always true that binds every input to the output. In order to obtain valid rules, we add this predicate at antecedent of the rule:

```
kb:BookTitle(?process:BookName) ∧
kb:hasTransf(?process:BookName,?process:BookInfo) →
    bibtex:Book(?process:BookInfo)
```

including also the implicit *inCondition*.

4 A backward search algorithm for SWRL rules composition

In this section we present our SWRL composer prototype that implements a backward search algorithm for the composition task. It works as follows: it takes as input a knowledge base containing SWRL rules and a goal specified as a SWRL atom, and it returns every possible path built combining the available SWRL rules in order to achieve such goal. These rules comply with SWRL safety condition mentioned in the previous section.

In details, the algorithm performs backward chaining starting from the goal in the same fashion Prolog-like reasoners work for query answering. The difference is that this algorithm does not rely just on Horn clause but on SWRL DL-safe rules. This means that, besides the rule base, it takes into account also the Description Logic ontology to which the rules refer.

The SWRL rule path found, and consequently the resulting OWL-S service composition, will be valid, in the sense that it will produce results for the selected goal, only if the SWRL rules in the path are DL safe. In other words the DL-safety means that rules are true for individuals that are **known**, i.e.: they appear in the knowledge base[5]. At present, the prototype performs DL-safety check. This guarantees that the application of rules is grounded in the ABox and consequently that the services that embody those rules can be executed.

5 Example

In this section we present an example that shows the applicability of our method. The dataset of OWL-S services can be found on Mindswap Web site [6]. In such dataset, there are some OWL-S atomic services and, based on these ones, some OWL-S composite services. The latter set will be used to validate our method. We will evaluate how many composite services in such set can be actually built automatically by our composer.

In table 1 we report the set of the atomic services with the information needed for the scope of this section. Among them, only two services have not inputs and outputs described as datatype in knowledge domain and only one service contains a *Precondition*. All services have no declared *inConditions*, hence we assume that for each of them there is only one *Result* corresponding to the service output and there is no *Effect*.

[5] It might not be the case in general, given the Open World Assumption holding in Description Logics, see [8] and chapter 2 in [9]

[6] http://www.mindswap.org/2004/owl-s/services.shtml

To obtain SWRL rules that satisfy the requirements described in the section 3, we have modified the atomic services as follows:

a)

```
<!--namespace of this ontology is &kb;-->
..
<owl:Class rdf:ID="BookTitle">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="BookEntity"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:DatatypeProperty rdf:about="hasBookName">
  <rdfs:domain rdf:resource="#BookTitle"/>
    <rdfs:range rdf:resource="&xsd;#string"/>
  </owl:DatatypeProperty>
..
```

b)

```
..
<process:Input rdf:ID="BookName">
  <process:parameterType
    rdf:datatype="&xsd;#anyURI">
      &kb;#BookTitle
  </process:parameterType>
</process:Input>
..
```

Fig. 1. The transformation of a datatype in a knowledge domain entity

- For every parameter having a datatype as type, we created a class in the domain ontology having a datatype property with the corresponding datatype as range (fig. 1a). The OWL-S descriptions have been modified assigning the newly created class to the corresponding *parameterType* (fig. 1b).
- For each service, we create two logical formulas. The first composed of unary atoms having the *parameterType* URI as their predicate and the input as their variable, for each input. The second composed of a unary atom having the *parameterType* URI as its predicate and the output its variable. We set these two logical formulas as, respectively, the antecedent and consequent of a new SWRL rule.
- Since every service produces the output manipulating the inputs, we can suppose that there exists a predicate (*hasTransf* predicate) always true that binds every input to the output. We did this in order to guarantee the SWRL safety condition, then we added *hasTransf* predicates to the antecedent of the rule built in the previous step. With this modification the antecedent can be identified with a new *inCondition*.

The obtained SWRL rule set is given as input to our composer and the resulting composition can be compared with the processes proposed in the Mindswap composite services examples. However, OWL-S composite processes can use control constructs (such as *iteration* and *selection*) that are more complex than the simple sequence, hence some considerations are needed w.r.t. composed services in 2:

- The French Dictionary service returns the meaning of a French word in French. To do this, it uses the processes of two atomic services: BabelFishTranslator and English Dictionary. It defines the sequences reported in the column 2 of the table 2 that are combined by means of other control constructs to return its result.
- Find Cheaper Book Price service returns the smallest price of a book along with the name of the bookstore that sells it. To do this, it uses the processes of three

ATOMIC SERVICE	DATATYPE INPUTS	DATATYPE OUTPUTS	SWRL CONDITION
Book Finder: Returns the information of a book whose title best matches the given string.	"BookName"	No	No
Zip Code Finder: Returns the zip code for the given city/state.	"City", "State"	No	No
Latitude Longitude Finder: Returns the latitude and longitude for a given zip code.	No	No	No
Barnes & Nobles Price Finder (BNPrice): Returns the price of a book as advertised in Barnes and Nobles web site given the ISBN Number.	No	No	No
Amazon Book Price Finder (AmazonPrice): Returns the price of a book as advertised in Amazon web site given the ISBN Number.	No	No	No
English Dictionary: Returns the meaning of a word from the dictionary.	"InputString"	"OutputString"	No
BabelFish Translator: Convert text from one language to another language using the online BabelFish translator services.	"InputString"	"OutputString"	One Precondition: "SupportedLanguagePair"
Currency Converter: Converts the given price to another currency.	No	No	No

Table 1. Some characteristics of the OWL-S atomic services datataset

atomic services: BookFinder, BNPrice and AmazonPrice. It defines the sequences reported in the column 2 of the table 2 that are combined by means of *selection* control construct to return its results.

As mentioned above, our composer is not able to work with complex control sequences and composite services as inputs. In both cases, we have to check if our composer was able to retrieve the basic sequences reported above on the basis of which the composite services have been built. In order to verify this, in Tab. 2 we put as input of our composer the outputs of the sequences instead of the outputs of the services.

COMPOSITE SERVICE	SEQUENCES IN THE SERVICE	RETRIEVED? (Yes/No)
French Dictionary	1) BabelFishTranslatorProcess ⇒ EnglishDictionaryProcess 2) EnglishDictionaryProcess ⇒BabelFishTranslatorProcess	1) Yes 2) Yes
Book Price	1) BookFinderProcess ⇒ BNPriceProcess ⇒ CurrencyConverterProcess	1) Yes
Find Cheaper Book Price	1) BookFinderProcess ⇒ BNPriceProcess 2) BookFinderProcess ⇒ AmazonPriceProcess	1) Yes 2) Yes

Table 2. Some characteristics of the OWL-S composite services datataset

BookPrice service returns the price of a book in a desired currency. To do this, it uses the processes of three atomic services: Book Finder, BNPrice and Currency Converter. Since it uses only the sequence construct, the searched goal can be the output of the service and our system retrieves the correct composition.

6 Related work

To the best of our knowledge no approach in literature makes use of SWRL for the composition of Semantic Web Services. Researchers focussed either on semi-automated or fully automated methods for service composition, drawing inspiration especially from AI planning [10] and state machines [11].

One approach aims at integrating Semantic Web formalisms into classical planner methodologies. Berardi et al. [12] address the problem of automatic composition synthesis of e-Service. They developed a framework in which the exported behavior of an e-Service is described in terms of its possible executions (execution trees). Then they specialize the framework to the case in which such exported behavior (i.e., the execution tree of the e-Service) is represented by a finite state machine. In [13], the semantics underlying the DAML-S specification (the ancestor of OWL-S) has been translated into FOL, obtaining a set of axioms for describing the features of each service. By combining these axioms within a Petri Net, the authors have obtained process-based service models that enable reasoning about the interactions among the processes that form the structure of a service. Traverso and Pistore [14] propose a planning technique for the automated composition of Web services described in OWL-S process models, which can deal with nondeterminism, partial observables, and complex goals. Such technique facilitates the synthesis of plans that encode compositions of web services with the usual programming constructs, like conditionals and iterations. In [15] an approach for developing a Semantic Web service discovery and composition framework on top of the CLIPS rule-based system is presented. More specifically, it describes a methodology for using production rules over Web services semantic descriptions expressed in the OWL-S ontology.

Other approaches, in which our methodology can be framed, apply methodologies and tools developed in the field of AI planning directly on Semantic Web settings. Sirin and Parsia [16] demonstrate how an OWL reasoner can be integrated within an AI planner, called SHOP2 [17], for the composition of Semantic Web Services. The reasoner is used to store the world states, answer the planners queries regarding the evaluation of preconditions, and update the state when the planner simulates the effects of services. An approach for using SPARQL [18] as an expression language for OWL-S conditions is presented in [19]. It describes how SPARQL can be used to give a compact representation of the preconditions of a service, and of its results. To our knowledge there hasn't been any attempt to use SPARQL query engines for achieving service composition.

The first type of approach foresees a translation from the Semantic Web formalisms to a dedicated formalism so that tools developed in particular research areas can be applied maintaining the same performances. On the contrary, the second type of approach foresees a porting of the algorithms and methodologies from other research fields using the Semantic Web technologies. The advantage of this approach, in which we frame our methodology, is the direct use of the Semantic Web formalisms. In this manner, we are able to use methodologies coming from more consolidated research fields exploiting the advantages that Semantic Web guarantees, i.e. a distributed knowledge base and the semantic interoperability. Furthermore, the use of SWRL, in particular, allowed us to exploit its greater expressiveness with respect to OWL-DL itself. Indeed, OWL-DL, being a Description Logic, does not allow to formulate constructs like property

compositions without becoming undecidable. SWRL partially reliefs these constraints, especially in the fragment we adopted in our work (i.e.: DL-safe SWRL rules), providing a more powerful means that becomes very useful in most SWS description portions such as *inConditions* and *Effects*.

7 Conclusions and future work

In this paper we have presented a new method that exploits SWRL for OWL-S atomic services composition. We have proved that if the OWL-S services have a meaningful semantics and valid SWRL conditions it is possible to build composer exploiting only the Semantic Web technology to achieve the composition task. This work can be considered as a starting point for the solution of a broader issue like the orchestration of SWS. Indeed rules for service coordination can be added to the rules for SWS encoding completing the knowledge that is necessary for the orchestration.

Future work will mainly consist of augmenting the number of services that can be encoded into SWRL rules. In other words the system should be able in the future to handle composite services as input and to produce more complex control structures (such as selection and iteration). The latter seems to be the most challenging task since it will require more powerful algorithms for the composition task.

Furthermore, an interesting aspect to deal with is the management of knowledge bases when there are changes produced by the effects of a service execution. Semantic Web languages are based on Description Logics which implement monotonic reasoning. In other words, they do not provide any means for retracting or modifying the status of the knowledge base that is not adding some new facts. This is somewhat a too restrictive requirement to represent, for instance, service execution in such formalisms. Think for example of representing the status of an activity as a functional property [7]. Now, as soon as this activity changes its status (say, for instance, it passes from 'scheduled' to 'in progress'), an equivalent change should be carried out on its description in the knowledge base. At the moment there is no DL reasoner allowing for that, meaning that performing such change in a traditional knowledge base would lead to an inconsistency of the whole knowledge base.

References

[1] McIlraith, S.A., Son, T.C., Zeng, H.: Semantic web services. IEEE Intelligent Systems **16** (2001) 46–53

[2] OWL-S: Semantic markup for web services, http://www.w3.org/submission/owl-s/ (2004)

[3] WSMO: Web service modeling ontology d2v1.3, http://www.wsmo.org/tr/d2/v1.3/ (2006)

[4] WSDL-S: Web service semantics, http://www.w3.org/submission/wsdl-s/ (2005)

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

[6] OWL: Owl web ontology language overview, http://www.w3.org/tr/owl-features/ (2004)

[7] Lloyd, J.W.: Foundations of logic programming (second, extended edition). Springer series in symbolic computation. Springer-Verlag, New York (1987)

[7] a property allowing only one value for each instance it is applied to.

[8] Motik, B., Sattler, U., Studer, R.: Query answering for owl-dl with rules. Journal of Web Semantics: Science, Services and Agents on the World Wide Web **3** (2005) 41–60
[9] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook. Cambridge University Press (2003)
[10] Georgeff, M.P.: Planning. In Allen, J., Hendler, J., Tate, A., eds.: Readings in Planning. Kaufmann, San Mateo, CA (1990) 5–25
[11] Gurevich, Y.: Evolving algebras 1993: Lipari Guide. In Börger, E., ed.: Specification and Validation Methods. Oxford University Press (1994) 9–37
[12] Berardi, D., Calvanese, D., Giacomo, G.D., Hull, R., Mecella, M.: Automatic composition of transition-based semantic web services with messaging. In Böhm, K., Jensen, C.S., Haas, L.M., Kersten, M.L., Larson, P.Å., Ooi, B.C., eds.: VLDB, ACM (2005) 613–624
[13] Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: WWW '02: Proceedings of the 11th international conference on World Wide Web, New York, NY, USA, ACM Press (2002) 77–88
[14] Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In McIlraith, S.A., Plexousakis, D., van Harmelen, F., eds.: International Semantic Web Conference. Volume 3298 of Lecture Notes in Computer Science., Springer (2004) 380–394
[15] Meditskos, G., Bassiliades, N.: A semantic web service discovery and composition prototype framework using production rules. In OWL-S:Experiences and Directions Workshop at 4th European Semantic Web Conference (ESWC) (2007)
[16] Sirin, E., Parsia, B.: Planning for Semantic Web Services. In Semantic Web Services Workshop at 3rd International Semantic Web Conference (2004)
[17] Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: Shop2: An htn planning system. J. Artif. Intell. Res. (JAIR) **20** (2003) 379–404
[18] Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (Candidate Recommendation). Technical report, W3C (2007)
[19] Sbodio, M.L., Moulin, C.: SPARQL as an expression language for OWL-S. In OWL-S:Experiences and Directions Workshop at 4th European Semantic Web Conference (ESWC) (2007)