OWL-S Atomic services composition with SWRL rules

Domenico Redavid*, Luigi Iannone†, Terry Payne‡, and Giovanni Semeraro*

*Dipartimento di Informatica
Università di Bari, Bari 70126, Italy
†Computer Science Dept., University of Liverpool
Ashton Building, Ashton Street L69 3BX, Liverpool UK
‡School of Electronics and Computer Science, University of Southampton
Southampton, SO17 1BJ, United Kingdom
{redavid, semeraro}@di.uniba.it
L.Iannone@liverpool.ac.uk
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 and a grounding in BPEL are 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 [1], and WSMO [2]. In this paper[3] our aim is the composition of OWL-S atomic processes adopting SWRL [2] 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 process model denoted by services compositions can be grounded in BPEL to obtain an executable process.

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.

[1] OWL-S: Semantic markup for web services, http://www.w3.org/submission/owl-s/
[2] WSMO: Web service modeling ontology d2v1.3, http://www.wsmo.org/tr/d2/v1.3/
[3] This research was partially funded by the project DIPIS (Distributed Production as Innovative System), Apulia Region Strategic Project (2006-08)

Each OWL-S process 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. 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 concrete datatypes gives very little in terms of added semantics. For example, consider a service whose input has declared as *Datatype* within a knowledge domain, i.e. a string. This means that the reference knowledge model of this input parameter is a concrete XML Schema datatype 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 declared as entities in a domain ontology (i.e. not as datatype).

3 Encoding OWL-S atomic processes with SWRL rules and composition algorithm

In this section we explain our approach for transforming process descriptions into sets of rules expressed in an ontology-aware rule language, namely Semantic Web Rule Language (SWRL), and our composer implementation.

To our aim, it is important to underline two SWRL characteristics: every rule must respect the *safety* condition and every rule with conjunctive consequent can be transformed into multiple rules each with an atomic consequent [3]. Furthermore, we work exclusively with SWRL DL-safe rules [4] fragment. Within OWL-S, conditions (log-

ical formulas) can be declared using 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*, they will appear in the antecedent of each resulting rule togheter with *inConditions*. 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 service that returns book information whose process is declared having one input (*?process:BookName*), one output (*?process:BookInfo*), and none condition. We should write the corresponding rule as "*kb:BookTitle(?process:BookName)* $\rightarrow$ *bibtex:Book(?process:BookInfo)*", but the variable *process:BookInfo* does not appear in the antecedent of the rule, 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 obtaining the implicit *inCondition*.

The realized SWRL composer prototype 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. 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[4]. 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.

4 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 [5]. It is formed by four services, namely *BookFinder*, *BNPrice*, *AmazonPrice* and *CurrencyConverter*. Among them, only one service has not any inputs and outputs described as datatype in knowledge domain. 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 *Precondition* and *Effect*. To obtain SWRL rules that satisfy the requirements described in the section 3, we have modified the atomic services as follow:

- 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 The OWL-S descriptions have been modified assigning the newly created class to the corresponding *parameterType*.
- 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 some resulting composition are showed in Figure 1. In a) and b) the searched goal is the same, i.e. the price of a book, but with composition b) is possible to obtain the price in a given currency. The paths having the same service as starting point (*BookFinder* service) are been joined to form a *Split-like* construct. In the next section we deal with a possible grounding of the composition plan with BPEL.

5 Grounding service plan with BPEL

An interesting application of our composition method is the grounding of the resulting composition plan with Business Process Execution Language (BPEL)[6]. The deficiency

[4] It might not be the case in general, given the Open World Assumption holding in Description Logics, see [4] and chapter 2 in [5]

[5] Available at: http://www.mindswap.org/2004/owl-s/services.shtml

[6] Business process execution language for web services (BPEL) 1.1, http://www.ibm.com/developerworks/webservices/library/ws-bpel/ (2003)

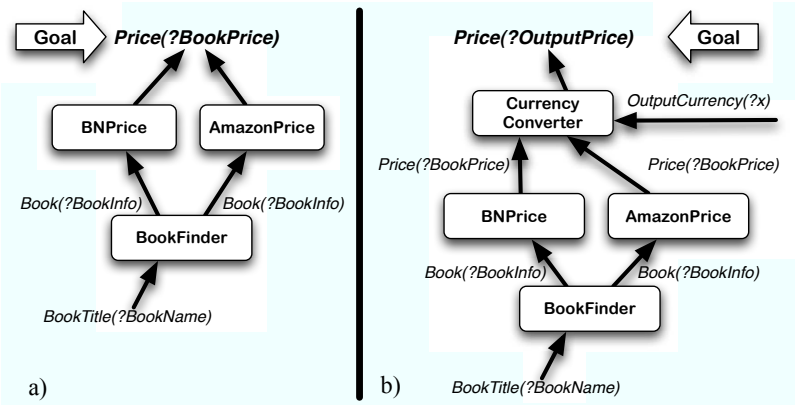

Fig. 1. Examples of composition (all concepts are defined with same namespace)

of BPEL is the impossibility to represent semantic information. On the other hand, OWL-S process model is designed to represent such kind of information, but, in general, such semantic information is superfluous during the execution of procesess. Since our method works at semantic abstraction level, it is possible to ground plans obtained with our SWRL composer using BPEL. BPEL makes use of the *activity* notion for process modeling. There exist two kinds of *activities* for this task i.e.: *Basic Activities* and *Structured Activities* [6]. *Basic Activities* (e.g.: "Invoke", "Receive" and "Reply") are used to model interaction between business partners. These activities can be nested in some *Structured Activities* to define BPEL workflow. Since OWL-S Atomic services can be grounded in WSDL, we have a set of services described with WSDL usable as base building blocks to create BPEL process models. The encoding between WSDL and BPEL is straightforward. The linking between building blocks is represented by the plan produced with the SWRL composer prototype. Analyzing a plan produced by SWRL composer, we note that:

- A rule path is a *Sequence* of rules (Atomic services).
- Two or more rule path can be split and then joined in one rule (Atomic service).
- No iterations are possible.

The correspondent BPEL *Activity* used to represent such plans are "Sequence" and "Flow". Moreover, it is necessary to use BPEL *Data Handling* to specify the data-flow into process model. It defines the notion of "Assignment" used between two basic activities to assign an output message or message parts (in case of complex message types) of first activity, as an input message or message part for next activity. These messages are originated from WSDL messages. Therefore it is possible to encode plans produced with SWRL composer with BPEL process model.

6 Related work

To the best of our knowledge no approach in literature makes use of SWRL for the SWS composition. Researchers focussed either on semi-automated or fully automated

methods for service composition, drawing inspiration especially from AI planning [7] and state machines [8]. Generally, two different approaches to perform the composition task have been adopted. One approach aims at integrating Semantic Web formalisms into classical planner methodologies. Berardi et al. [9] 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 [10], 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 [11] 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 [12] 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 [13] demonstrate how an OWL reasoner can be integrated within an AI planner, called SHOP2 [14], for the SWS composition. 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.

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.

7 Conclusion 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. Working at semantic abstraction level it is possible to map the resulting composition with XML-based languages for process management like BPEL. This work can be considered as a starting

point for the solution of a broader issue like the orchestration of SWS. Future work will mainly consist of augmenting the types 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.

References

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

[2] 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

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

[4] 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

[5] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook. Cambridge University Press (2003)

[6] Aslam, M.A., Auer, S., Shen, J., Herrmann, M.: Expressing Business Process Models as OWL-S Ontologies. In Eder, J., Dustdar, S., eds.: Business Process Management Workshops. Volume 4103 of Lecture Notes in Computer Science., Springer (2006) 400–415

[7] Georgeff, M.P.: Planning. In Allen, J., Hendler, J., Tate, A., eds.: Readings in Planning. Kaufmann, San Mateo, CA (1990) 5–25

[8] Gurevich, Y.: Evolving Algebras 1993: Lipari Guide. In Börger, E., ed.: Specification and Validation Methods. Oxford University Press (1994) 9–37

[9] 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

[10] 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

[11] 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

[12] 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)

[13] Sirin, E., Parsia, B.: Planning for Semantic Web Services. In Semantic Web Services Workshop at 3rd International Semantic Web Conference (2004)

[14] 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