

SPARQL as an expression language for OWL-S

Marco Luca Sbodio¹ and Claude Moulin²

¹ Italy Innovation Center, Hewlett Packard Italiana, C.so Trapani 16,
10139 Torino, Italy

² University of Compiègne, CNRS, Heudiasyc, Centre de Recherches de Royallieu,
60205 Compiègne, France

Abstract. This paper presents a novel approach for using SPARQL as an expression language for OWL-S. It describes how SPARQL can be used to give a compact representation of the preconditions of a service, and of its results (with associated result's conditions and effects). The paper describes also how SPARQL can be used to give an expressive definition of the goal of a software agent. The approach presented in this paper has been adopted the EC funded project TERREGOV to describe eGovernment process.

1 Introduction

The *semantic web services* vision [1] is pursued by several emerging formalisms and frameworks, such as WSMO [7], SWSF [8] and OWL-S [9]. The objective of all these formalisms is the unambiguous denotations of web services capabilities, in order to allow automation of tasks such as discovery and composition. However this objective seems reached by the adding of sections enlarging the original formalisms.

We have adopted OWL-S in the EU funded TERREGOV project: Impact of e-Government on Territorial Government Service [10]. TERREGOV addresses the issue of interoperability of e-Government services for local and regional governments (see Section 7). Our work is based on the latest version of OWL-S [9]. We explored how SPARQL [6] can be used to express conditions (both preconditions and results conditions) and effects, and we show how an agent can exploit the SPARQL expressions during the service discovery process.

The paper is structured as follows. Section 2 presents a short overview of the approach and describes the general scenario. Section 3 gives a short summary of some major features of SPARQL. Section 4 and Section 5 describe respectively how SPARQL can be used in the service context and in the agent context. Finally Section 6 presents some conclusion and an outlook.

2 Overview

One of the key aspects of the semantic web services vision is the (partial) automation of the discovery of web services. The general scenario can be described

as following: some software agent (for brevity we also use the word *agent*, even if in this paper we do not refer to any specific agent platform or standard) is trying to perform some task. The agent has a knowledge base (*agent knowledge base*), which is an essentially RDF graph providing information on resources that the agent can use during the discovery process (for example as inputs for a service). The agent's task is defined as a *goal*, which must be fulfilled. The agent looks for a service having the appropriate capabilities to fulfill its goal. In the discovery process it is necessary to use the full semantic denotation of the web service operations (inputs/output types, preconditions, results, and effects) in order to assess if a service is appropriate to fulfill the agent's goal.

The service conditions and effects are usually modelled as logical expressions, and OWL-S has recently introduced the possibility of using SPARQL as an expression language (since version 1.2. of OWL-S). Our approach is based on the adoption of SPARQL as an expression language for modelling OWL-S preconditions, results conditions and effects. We show how this simplifies the denotation of a service using a compact SPARQL query, and we also extend the usage of SPARQL to the definition of the agent's goal.

We worked mainly on *information production web services*, which usually generates or returns some kind of information based on information given as input and (possibly) the world state. This kind of web services usually does not produce changes in the state of the world (effects), which is a peculiarity of the *world transition web services*. Information production web services are very common in the e-Government domain, which is the domain of the project TERREGOV. Information production web services are fully specified by their inputs/output types, preconditions and result condition. Nevertheless our approach is general, and it applies also to *world transition web services*, which requires also the specification of effects. We show how our approach applies to a well known example of a world transition web service, which requires preconditions, result conditions and effects.

3 SPARQL features

SPARQL features are extensively described in [6]. An in depth analysis of SPARQL semantics is available in [13] and in [14]. We adopt the notation used in [6].

A SPARQL query is a tuple $\langle GP, DS, SM, R \rangle$, where:

- GP is a graph pattern
- DS is the RDF dataset being queried
- SM is a set of solution modifiers
- R is a result form

A graph pattern can be defined recursively as in [13]. There are several types of graph patterns (GP) in SPARQL: a basic graph pattern (GP) is a set of triples. Basic graph patterns can be combined in group graph patterns (GGP) or union graph patterns (UGP). The group graph pattern GGP is a set of graph patterns $\{GP_1 \dots GP_n\}$; a query solution for GGP is a solution for each

graph patterns $GP_{i=1\dots n}$. The union graph pattern UGP is also a set of graph patterns $\{GP_1\dots GP_n\}$, but a query solution for UGP is a solution only for some graph patterns $GP_{i=1\dots n}$. SPARQL allows also the definition of optional graph patterns. An optional graph pattern is a made of a pair of graph patterns: the second pattern modifies solutions of the first, but does not fail matching of the overall optional graph pattern.

The dataset DS specifies an RDF data model, consisting of triples, over which the SPARQL query is executed; DS may be a set of graphs: the default graph (which is always present, and which does not have a name) and a set of optional named graph (each of which is identified by an URI). The solution modifiers SM allows for modifications of the solution sequence (for example ordering the solutions, or avoiding repetitions, or limiting the number of solutions).

Finally, a SPARQL query can have different result forms (R), among which the *SELECT*, *ASK*, and *CONSTRUCT* result forms. The *SELECT* result form defines a set of variables over which the query solutions are projected. The *ASK* form simply checks if the query pattern has some solutions over the provided dataset, and returns yes/no answers. The *CONSTRUCT* form returns an RDF graph specified by a graph template: the result RDF graph is formed by taking each query solution, substituting for the variables into the graph template, and combining the triples into a single RDF graph by a set union.

4 Using SPARQL to define the service's preconditions and results

OWL-S has the notion of *service model*, which provides a description of what happens when the service is carried out; such description is useful for agent to decide if it can use the service in order to fulfill its goal. OWL-S has a class *Process*, which is a subclass of *ServiceModel*; the class *Process* represents the most general service model, and has further specializations. In the following we use the word *process* to identify an instance of *Process*.

An OWL-S process may have preconditions. Preconditions are logical expressions that must be satisfied in order to invoke the process. An OWL-S process may have several outcomes, each of which is modeled by an instance of *Result* (in the following we use the word *result* to identify such an instance). Each result is associated with some output, a result condition and some effects. Result conditions are logical expressions; if result's conditions are true, then the associated outputs and effects should occur; effects are also defined by logical expressions.

We suggest the usage of SPARQL *CONSTRUCT* query form to give a compact representation of a process result, with its preconditions, result's conditions and effects. Formally the preconditions are related to the process, and not to the specific result. However each result, being an outcome of the process, will happen if and only if the preconditions are satisfied, and therefore we can safely replicate them at result level. Symbolically we can say that a result RES_i can be defined by the following SPARQL query:

$$RES_i = \langle GP_i, DS, SM_i, CONSTRUCT T_i \rangle$$

where

- GP_i is a graph pattern
- DS is a dataset
- SM_i is an optional solution modifier
- T_i is a template pattern for the CONSTRUCT query

The result of the CONSTRUCT query is an RDF graph $GRAPH_i$, which can be interpreted as a description of what happens when the service is invoked and result RES_i is returned to the agent; $GRAPH_i$ is an *outcome* of the process, and it fully models the corresponding state of the world after the process invocation. The elements of the tuple RES_i are interpreted as following:

- DS is the agent knowledge base, which is used to bind the free variables in the graph pattern GP_i .
- GP_i represents the preconditions of the process. The CONSTRUCT query returns an empty (or partial) $GRAPH_i$ if GP_i has no solutions over DS ; this can be interpreted as the fact that the agent cannot fulfill the preconditions of the process. Conversely, if GP_i has solutions over DS , then a complete $GRAPH_i$ is returned.
- T_i is used to construct the outcome of the process according to the result conditions. T_i may contain also triples with variables, whose values will be taken from the solution of GP_i over DS . T_i gives the structure of $GRAPH_i$.

As an example of the proposed approach we present a SPARQL representation of the `ExpressCongoBuyPositiveResult`, which is one of the results defined in the OWL-S example `ExpressCongoBuy`¹. The Express Congo Buy is a service for buying a book. It takes as input a Book ISBN number, and the customer's sign-in information. It has the effect of ordering the book if the book is in stock. In this case (`ExpressCongoBuyPositiveResult`), the output of the service is a message saying the book is ordered. We can represent the `ExpressCongoBuyPositiveResult` with the following SPARQL query:

```
PREFIX congo:
  <http://www.daml.org/services/owl-s/1.2/CongoProcess.owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX profileHierarchy:
  <http://www.daml.org/services/owl-s/1.2/ProfileHierarchy.owl#>

CONSTRUCT {
  _:expressCongoBuyBook a congo:InStockBook ;
  profileHierarchy:hasISBN ?expressCongoBuyBookISBN.
```

¹ The OWL-S definition is available at <http://www.ai.sri.com/daml/services/owl-s/1.2/CongoProcess.owl>

```

_:expressCongoBuyOutput a congo:orderShippedAcknowledgment .

_:expressCongoBuyShipment a congo:Shipment ;
  congo:shippedTo ?expressCongoBuyAcctID ;
  congo:shippedBook _:expressCongoBuyBook .
}
WHERE {

  ?expressCongoBuyBookISBN a profileHierarchy:ISBN .

  ?expressCongoBuySignInInfo a congo:SignInData ;
    congo:hasAcctID ?expressCongoBuyAcctID .

  ?expressCongoBuyAcctID a congo:AcctID .

  ?expressCongoBuyCreditCard a congo:CreditCard ;
    congo:cardNumber ?cardNumber ;
    congo:validity congo:Valid .
  FILTER ( datatype(?cardNumber) = xsd:decimal )
}

```

Notice that the graph pattern uses variables for instances of OWL-S `Input` and `Local`. The graph pattern contains also triples to constrain the RDF type of such variables.

The OWL-S description of `ExpressCongoBuy` has two preconditions. The first one says that the user must have an account ID with the Express Congo Buy, that is:

```
hasAcctID(ExpressCongoBuySignInInfo, ExpressCongoBuyAcctID)
```

Such precondition is represented in the CONSTRUCT graph pattern by the following triples:

```

?expressCongoBuySignInInfo a congo:SignInData ;
  congo:hasAcctID ?expressCongoBuyAcctID .
?expressCongoBuyAcctID a congo:AcctID .

```

The second precondition of `ExpressCongoBuy` says that the credit card number must be provided, and that the credit card must be valid, that is:

```

cardNumber(
  ExpressCongoBuyCreditCard,
  ExpressCongoBuyCreditCardNumber
)
& validity(ExpressCongoBuyCreditCard, Valid)

```

Such precondition is represented in the CONSTRUCT graph pattern by the following triples:

```

?expressCongoBuyCreditCard a congo:CreditCard ;
  congo:cardNumber ?cardNumber ;
  congo:validity congo:Valid .
FILTER ( datatype(?cardNumber) = xsd:decimal )

```

The template pattern of the SPARQL query defines the outcome of the process when the positive result occurs. Notice that the template pattern uses blank nodes (aka bNodes) to for instances of OWL-S `ResultVar` and `Output`; it also contains triples to constrain the RDF type of such bNodes.

If the agent knowledge base can provide solutions for the graph pattern of the CONSTRUCT query, then the agent has enough knowledge to satisfy the preconditions, and it may obtain the result `ExpressCongoBuyPositiveResult`. The corresponding CONSTRUCT query yields an RDF graph, which describes the state of the world after the process invocation.

5 Using SPARQL to define the agent's goal

A goal is a representation of an objective that the agent seeks to fulfill; the agent tries to discover a service whose execution allows the fulfillment of its goal. Some semantic web service formalisms, such as WSMO, have a formal definition of goal (see [2] [3] [4] [5]). Although OWL-S does not formally specify how to define the goal of a software agent, we propose a simple approach where a goal can be defined with a SPARQL query. Such an approach fits well with the usage of SPARQL to define process preconditions and results as described in Section 4.

We suggest the usage of SPARQL ASK query form to give a compact representation of a goal. Symbolically we can say that a goal *GOAL* can be defined by the following SPARQL query:

$$GOAL = \langle GPg, DS, SMg, ASK \rangle$$

where

- *GPg* is a graph pattern
- *DS* is a dataset
- *SMg* is an optional solution modifier

A goal is fulfilled if and only if the corresponding ASK query returns yes. The elements of the tuple *GOAL* are interpreted as following:

- *DS* is either the agent knowledge base or the RDF graph obtained by a CONSTRUCT query that defines a process outcome. In the former case the evaluation of the ASK query tells if the agent has already enough information to fulfill its goal; in the latter case the evaluation of the ASK query tells if the agent's goal is fulfilled in the state of the world obtained as outcome of the process.
- *GPg* represents a set of constraints, which define the objective of the goal.

The agent uses the ASK query *GOAL* over *DS*. If *GOAL* returns true, then the goal is fulfilled. If it returns false then the agent tries to discover a service whose execution allows the fulfillment of its goal. The discovery process takes the SPARQL query that defines a process result RES_i , and executes it over the agent knowledge base. Such execution yields an RDF graph $GRAPH_i$ as described in Section 4. If $GRAPH_i$ has solutions for the ASK query *GOAL*, then the agent can infer that the process having the result RES_i can be used to fulfill its goal.

As an example, consider an agent whose goal is to buy some book; the goal can be expressed by the following SPARQL query (we omit some namespaces):

```
PREFIX   akb: <http://www.example.com/agents/myagent#>

ASK
WHERE {
  ?anInterestingBook a congo:InStockBook ;
    profileHierarchy:hasISBN akb:anInterestingBookISBN .

  ?expressCongoBuyOutput a congo:orderShippedAcknowledgment .

  ?expressCongoBuyShipment a congo:Shipment ;
    congo:shippedTo akb:myCongoAcctID ;
    congo:shippedBook ?anInterestingBookBook .
}
```

Assuming that the agent has the following knowledge base (we omit prefix definitions):

```
akb:mySignInInfo a congo:SignInData ;
  congo:hasAcctID akb:myCongoAcctID .

akb:myCreditCard a congo:CreditCard ;
  congo:cardNumber "1234"^^xsd:decimal ;
  congo:validity congo:Valid .

akb:myCongoAcctID a congo:AcctID .

akb:anInterestingBook a profileHierarchy:Book .

akb:anInterestingBookISBN a profileHierarchy:ISBN .
```

The agent knows some information related to a credit card, and some information on an interesting book. The CONSTRUCT query that defines the *ExpressCongoBuyPositiveResult* has solutions over such knowledge base, and yields the following RDF graph (we omit namespaces, and use the N3 [11] notation):

```

[ ]    a congo:orderShippedAcknowledgment .

[ ]    a congo:Shipment ;
      congo:shippedBook
        [ a congo:InStockBook ;
          profileHierarchy:hasISBN akb:lotrISBN
        ] ;
      congo:shippedTo akb:myCongoAcctID .

```

Such RDF graph has solution for the goal presented above, and therefore the agent can infer that the process `ExpressCongoBuy` may have a result that can fulfill its goal, and can terminate the discovery process. Notice that the process `ExpressCongoBuy` is a potential match for the discovery process: the agent must check at invocation time that the `ExpressCongoBuy` invocation yields the result `ExpressCongoBuyPositiveResult`. The actual invocation may in fact yield a negative result because at invocation-time the requested book is not in stock.

6 Outlook and Conclusions

We presented an approach that shows how SPARQL can be effectively used as an expression language for OWL-S. The SPARQL CONSTRUCT form can be used as a compact definition of a process result, with associated result's conditions and effects. The CONSTRUCT definition comprehends also the process preconditions, and when evaluated over an appropriate data set it yields an RDF graph that fully describes the state of the world after the execution of the process. Such RDF graph can be exploited by a software agent trying to discover a process to fulfill its goal. As an extension to OWL-S, we suggested to express also the agent goal with a SPARQL ASK query, whose graph pattern defines the goal's constraints, and whose yes/no answer tells if the goal is fulfilled.

The approach presented in this paper is an evolution and simplification of the approach presented in [12]. We have applied our approach within the project TERREGOV, to describe eGovernment process. We have also built a Java implementation of a prototype discovery agent (*TERREGOV eProcedure Choice Module*), which takes advantage of the SPARQL approach described above to perform automatic discovery of services. We are currently working on extension of our agent, to allow automatic composition of services. The general idea is that whenever the preconditions of a process $Proc_i$ are not satisfied, the agent can generate at run-time a sub-goal, which is defined as an ASK query whose graph pattern is built using the graph pattern of the CONSTRUCT query that defines $Proc_i$. The agent can then look for another process $Proc_j$ whose outcome fulfills the sub-goal represented by the preconditions of $Proc_i$. The process is recursive and it yields to a sequence of services that can be used to satisfy the initial agent's goal.

The major benefit of using SPARQL as an expression language for OWL-S is the compactness of the definitions of the process results and agent's goal.

Furthermore the proposed ASK form for the definition of the agent's goal allows for an automatic check of goal fulfillment. This approach simplifies the operations that an agent must perform during the service discovery process, and it offers a natural extension for automatic composition as briefly outlined above.

7 Acknowledgments

This work has been developed within the TERREGOV [10] project, an integrated project cofunded by the European Commission² under the IST (Information Society Technologies) Program, e-Government unit, under the reference IST-2002-507749.

References

1. S. McIlraith, T. Son and H. Zeng Semantic Web Services. IEEE Intelligent Systems, Special Issue on the Semantic Web. 16(2):46-53, March/April, 2001.
2. Christoph Bussler, Dieter Fensel, Dumitru Roman, et al. Web service modeling ontology. Applied Ontology Journal, 1(1), 2005.
3. J. Domingue, D. Fensel, and D. Roman. Semantic Web Services with the Web Services Modeling Ontology (WSMO), AgentLink News 19, 2005.
4. J. de Bruijn, D. Fensel, U. Keller, and R. Lara. Using the Web Service Modeling Ontology To Enable Semantic e-Business, Communications of the ACM 48(12), 2005.
5. R. Lara, A. Polleres, H. Lausen, D. Roman, J. de Bruijn, and D. Fensel. A conceptual comparison between WSMO and OWL-S, 2005. Available on Internet at: www.wsmo.org/TR/d4/d4.1/v0.1/
6. Eric Prud'hommeaux, Andy Seaborne. SPARQL Query Language for RDF. W3C Working Draft 4 October 2006. Available on Internet at: <http://www.w3.org/TR/rdf-sparql-query/>
7. Web service modeling ontology. Available on Internet at: <http://www.wsmo.org/>
8. Semantic Web Services Framework (SWSF). Available on Internet at: <http://www.w3.org/Submission/SWSF/>
9. OWL-S 1.2 Pre-Release. Available on Internet at: <http://www.ai.sri.com/daml/services/owl-s/1.2/>
10. TERREGOV project: Impact of e-Government on Territorial Government Service. IST-2002-507749. See: <http://www.terregov.eupm.net>
11. T. Berners-Lee. A readable language for data on the Web. N3 formalism Available on Internet at: <http://www.w3.org/DesignIssues/Notation3.html>
12. Claude Moulin, Marco Luca Sbodio. Denotation of Semantic Web Services Operations through OWL-S. In proceedings of Workshop on Semantics for Web Services (SemWS'06), in conjunction with 4th European Conference on Web Services (ECOWS'06), 4-6 December 2006, Zurich, Switzerland.
13. Jorge Perez, Marcelo Arenas, Claudio Gutierrez. Semantics and Complexity of SPARQL. The Semantic Web - ISWC 2006, Springer
14. Axel Polleres. SPARQL Rules! GIA TECHNICAL REPORT 2006-11-28, November 2006

² The content of this paper is the sole responsibility of the authors and in no way represents the views of the European Commission or its services.