Semantic RPC via Queries

Mohammad-Reza Tazari
Fraunhofer-IGD, Darmstadt, Germany
saied.tazari@igd.fraunhofer.de

ABSTRACT

A vision of the Semantic Web is to facilitate global software interoperability. Many approaches and specifications are available that work towards realization of this vision: Service-oriented architectures (SOA) provide a good level of abstraction for interoperability; Web Services provide programmatic interfaces for application to application communication in SOA; there are ontologies that can be used for machine-readable description of service semantics. What is still missing is a standard for constructing semantically formulated service requests that solely rely on shared domain ontologies without depending on programatic or even semantically described interfaces. *Semantic RPC* would then include the whole process from issuing such a request, matchmaking with semantic profiles of available and accessible services, deriving input parameters for the matched service(s), calling the service(s), getting the results, and mapping back the results onto an appropriate response to the original request. The standard must avoid realization-specific assumptions so that frameworks supporting semantic RPC can be built for bridging the gap between the semantically formulated service requests and matched programmatic interfaces. This poster introduces a candidate solution to this problem by outlining a query language for semantic service utilization based on an extension of the OWL-S ontology for service description.

1. INTRODUCTION

The visionary scenario for the Semantic Web in [1] reveals some of the features of semantic interoperability even in its first paragraph: "his phone turned the sound down by sending a message to all the other *local* devices that had a *volume control*." It is obvious that the phone is supposed to only broadcast a message without any dependency on the technical details of addressing and accessing the targeted devices; instead, it relies on shared concepts, here mainly (1) *turn the sound down* if you (2) *are in the vicinity of location l* and (3) *have a volume control*. We call this kind of messaging *semantic RPC*.

Common SOA solutions, such as Web Services and OSGi, concentrate on needed programmatic interfaces, where service registration and search are based on interface declarations and may use keyword- and /or taxonomy-based advertisement and inquiry mechanisms which restrict the process of service discovery to matches with no or little machine-interpretable semantics. Ontological approaches, such as OWL-S (www.daml.org/services/owl-s/), aim at providing more inference and reasoning potential for performing tasks, such as service matchmaking. As an OWL-based language for describing services, OWL-S allows to import domain ontologies and use concepts from them in the description of preconditions, inputs, outputs, and results of services (actually a kind of interface specification). Although the OWL-S profile ontology is designated for service advertisement, but there is still no specific solution for constructing service requests, which has lead to the redundant usage of the profile ontology also in forming requests as a sort of "query by example" (cf. [2] & [3]). In reply to the submission of such a request and as a result of the matchmaking task the requester receives a set of matched profiles; then it may decide to invoke an appropriate matching service using an associated grounding. Hence, current OWL-S–based solutions force thinking in terms of interfaces on the client side.

In this poster, an alternative solution is introduced based on an explicit query mechanism that bundles the process of semantic RPC in one step from the view point of the requester. A brokerage mechanism comprises the matchmaking and invocation tasks altogether. The broker is supposed to have access to service advertisements in form of OWL-S profiles; by receiving "service requests" in form of queries, the broker then tries to find and invoke the target services and return related "service responses". Figure 1 illustrates the process of resolving a query made by component$_1$ into calling an appropriate service realized by component$_2$ returning a related result, assuming a semantic interoperability platform that realizes both the broker and the client-side stub.

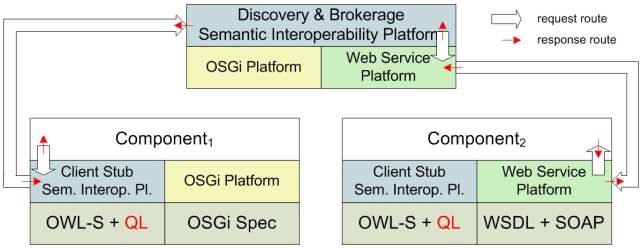

Figure 1: A semantic interoperability platform (the blue boxes) supporting "semantic RPC"

2. AN OUTLINE OF THE SOLUTION

The solution is based on a slightly new way of using OWL-S and the concept of "property paths[1]". OWL-S justifiably concentrates on domain-independent aspects of modeling services. The link to the domain ontology is made by introducing the concept of service category, on one side, and the usage of the domain concepts in the description of the process preconditions, inputs, outputs, and results., on the other side. However, there is no argument against bringing these two aspects together and using the OWL-S concept "service:Service" directly as the root of the domain-specific service ontology for defining specific service classes and relating such classes to the domain concepts. Figure 2 shows a simple ontology defined in this way that models lighting

[1]E.g., see www.w3.org/DesignIssues/Notation3

services based on an understanding of light sources and the fact that lighting services control light sources.

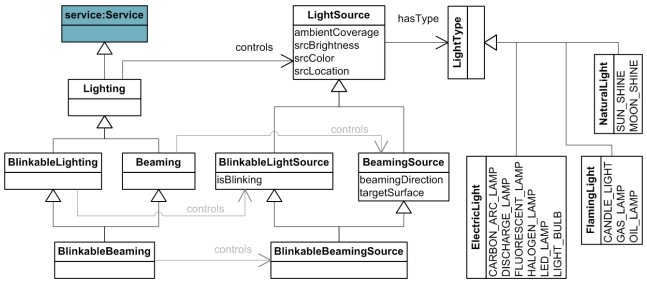

Figure 2: Example modeling of lighting services

A property path is a closed list of property URIs that relates a resource to a set of other resources or literal values as it could be reached by conventional "join" operations over an RDF database. Combining the notion of property paths with the explicit definition of service classes that are related to the domain ontology, we can now introduce the idea of *data view* of classes of services. Figure 2 is already imparting how the *Lighting* class of services views the "world" in terms of data. All data reachable from an instance of this class can be addressed with a unique property path so that at each point in time we can virtually construct a table containing those property paths as column names and rows containing the values associated with those columns at that given time. Then, the semantic of inputs, outputs, and effects of a large set of concrete services can be summarized the following way:

- effects either change certain columns of certain rows, add a new row, or remove an existing row
- inputs either select a certain set of rows by being incorporated in a filtering function, or are used in the expressions defining an effect
- outputs return the values in certain columns after filterng and just before or after the effects has taken place; outputs may undergo some conversion after having been selected, as well

The thesis is now that both components that realize services (and want to advertise their profiles) and clients that want to issue a service request can rely on this shared "data view" of the domain, virtually based on such an imaginary table as mentioned above. An example should help to better understand the idea: A software component that controls 4 light bulbs in a given apartment may define a subclass of the above *Lighting* class of services, e.g. stating that, as a subclass of Lighting services, it controls only four concrete light sources of type LIGHT_BULB whose "ambient coverage"s are not known and whose "brightness"es accept values equal to only 0 or 100. Then, using the concept of property paths, the OWL-S profile for a service 'my:turnOn' can both specify the role of its input parameter as a filter that selects the light source to be affected and state which property is affected with which new value. Similarly, a client component that wants to know the location of a given light source can formulate its request the following way: In the context of the service class `l:Lighting`, return the value in the "column" `l:controls!l:srcLocation` from a "row" whose "column" `l:controls` equals to `other:hallog1`. A request for turning off all light sources in a given location is equivalent to stating that an instance of the service class `l:Lighting` is sought that can set the "column" `l:controls!l:srcBrightness` equal to `0` in all "rows" whose "column" `l:controls!l:srcLocation` equals to `other:loc1`.

The two client-side examples in plain text show how a service request can be formulated as a query over the imaginary table. Hence, thinking in terms of SPARQL and ignoring the head part defining used namespaces, the envisioned query language can be outlined the following way:

```
CALL            ?s
WHERE           <SPARQL-style spec of the context of ?s>
[HAVING EFFECT  <var>=<value> {, <var>=<value>}*]
[WITH OUTPUT    <var> {, <var>}*]
[USING FILTER   best-match | random
                | <global-selection-criteria>]
```

where *1)* no explicit notion of property paths is needed because a chain of SPARQL-style conditions eventually specifies a certain property path, *2)* `HAVING EFFECT` is an abbreviation for `HAVING CHANGE EFFECT`; other variants are `HAVING ADD EFFECT` and `HAVING REMOVE EFFECT`; only one variant is allowed, however an `ADD` or `REMOVE` variant may be repeated, *3)* at least one of `HAVING EFFECT` or `WITH OUTPUT` is mandatory, *4)* each variable in the `HAVING EFFECT` or `WITH OUTPUT` clause must address a certain column of the imaginary table, in the latter case possibly associated with a unary aggregating operation, e.g. *min*, *max*, or *sum*, *5)* in contrast to the `WHERE` clause, the `USING FILTER` clause does not constrain property paths from the viewpoint of the referenced service class but *<global-selection-criteria>* are based on the OWL-S concept of 'profile:ServiceParameter', e.g. *AverageResponseTime*, in combination with unary operations *minOf* and *maxOf* that are supposed to be considered if the brokering mechanism finds more than one matching profile; in the absence of this clause, the request will be broadcasted to all components with matching service profiles.

The data structures that would result from parsing such a query have been realized within the EU-IST project PERSONA (www.aal-persona.org). So far we could cover all sorts of interoperability needs within the PERSONA system based on the above scheme. All the practical difficulties that have arisen so far could be mastered based on improvement of the domain ontology model, normally by adding missing concepts. Results of evaluating this solution will be available in the spring 2010, after the deployment and tests with real users has taken place in the pilot sites in the fall 2009 / winter 2010.

The improvements achieved by this solution are *1)* the omission of all syntactical dependencies, such as interfaces, which leads to more freedom in developing software components independently, and *2)* concrete SOA solutions can be bound once within the framework relieving individual components of dealing with heterogeneity of platforms.

The complete specification of the query language itself and the development of an appropriate parser are still open tasks. As the proof of concept is based on a special-purpose reasoner developed within PERSONA, adjustments in the concept, especially on the side of service advertisements, may be needed in order to guarantee its compliance with OWL DL.

3. REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[2] G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for DAML Web Services: Annotation and Matchmaking. In *Proceedings of the 2nd International Semantic Web Conference (ISWC'03)*, volume 2870 of *LNCS*, pages 335–350. Springer, 2003.

[3] C. Preist. A Conceptual Architecture for Semantic Web Services. In *Proceedings of the 3rd International Semantic Web Conference (ISWC'04)*, volume 3298 of *LNCS*, pages 395–409. Springer, 2004.