Towards a Proxy Architecture for Semantic Web Services

Eric Rozell, Tetherless World Constellation, Rensselaer Polytechnic Institute, Troy, NY

At the Tetherless World Constellation [i] (TWC), we have developed the S2S framework[ii] to serve as an integration point for various Web applications and services. This framework was originally designed in the context of oceanography for the purpose of creating customizable "data dashboards" that scientists could use as a "one-stop shop" for their diverse data needs. The framework has since evolved for the purposes of the Semantic eScience Framework[iii] project, and now provides a common interface to multiple Web service standards with semantic annotation capabilities, along with an extensible front-end architecture for combining Web applications and services. We are also working closely with sub-groups of the Federation of Earth Science Informatics Partners (ESIP) [iv] to integrate our semantically-enabled solution in their approaches to the federated search and data discovery infrastructure evolving amongst Earth scientists. In this position paper, we will discuss the importance of developing communication conventions or protocols for "proxies" that utilize the Semantic Web Service layer and back up our discussion with lessons learned from developing the S2S framework.

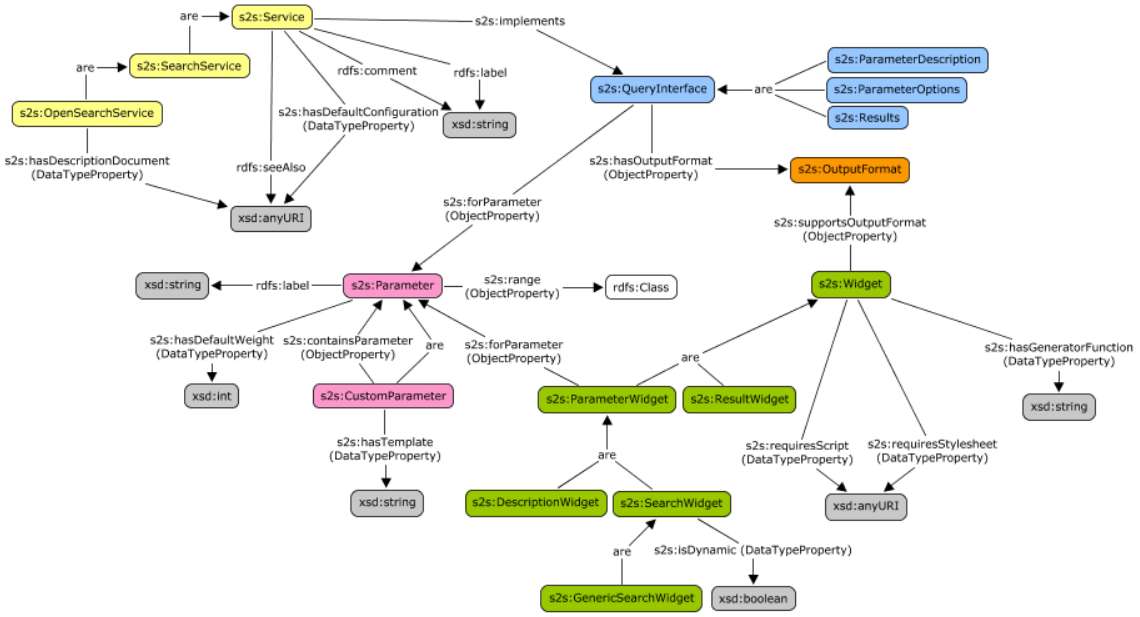

Figure 1: The S2S framework ontology.

Standardized Communication Protocols for Abstract Services

How do you communicate with a Semantic Web Service (SWS)? In the context of the S2S Framework, which has been implemented primarily for data and metadata search, the communication requirement is simple; typically, transactions are stateless, and only a communication protocol for Web service invocation is required. S2S defines query interfaces, parameters, and outputs, which are analogous to WSDL operations, inputs, and outputs, resp. (see Figure 1 for the S2S ontology

diagram). In order to invoke any Web service in S2S, we have developed a JSON service, where the service, query interface and set of parameter values are specified as input. The S2S back-end infrastructure then takes the submitted JSON, and sends it to an appropriate adapter, which has been developed to communicate with a specific Web service specification (e.g., SAWSDL, OpenSearch). Using the combination of the S2S vocabulary and a simple communication protocol based on JSON, we have developed an abstract interface for communicating with data search services through an intermediate "proxy" service.

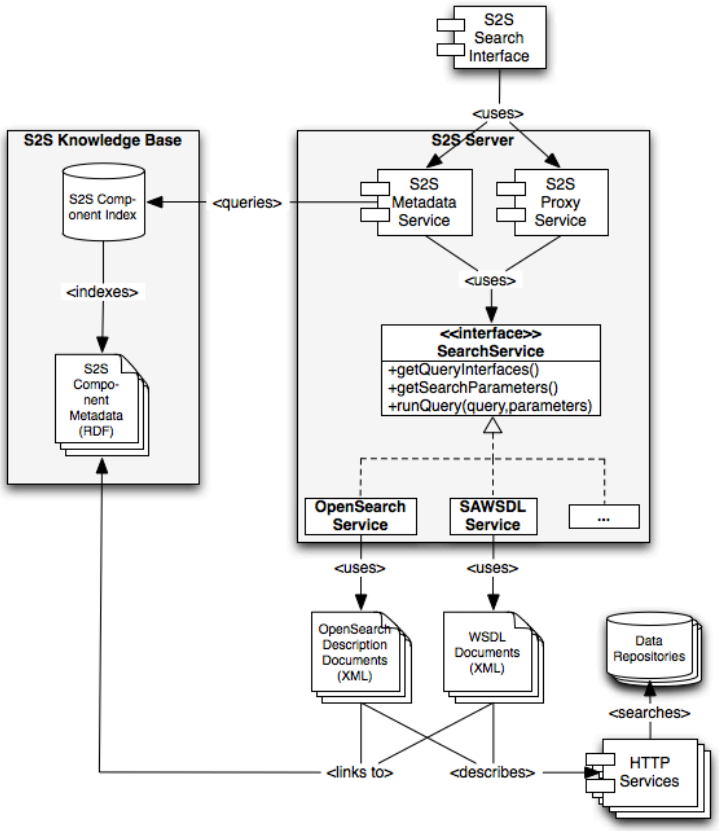

Figure 2: The S2S server-side architecture.

Such a model for SWSs in general could increase the usability and adoption of the abstract service layer. The success of Web service technologies, such as WSDL and SOAP, can be partly attributed to the development of clients and service development tools across many programming languages. Without a standard way of communicating with SWSs, it is difficult to develop the kinds of clients and tools that will expose SWSs to a wide audience. Consider OWL-S[v], there are constructs for defining groundings between the abstract profiles and processes, but the execution of the groundings is left to the client or some service mediator. The argument here is that the S2S framework has benefited from an intermediate "proxy" service that performed all the necessary "groundings". There is a need to develop such an intermediate service for OWL-S, or any other SWS ontology, and to standardize how clients communicate with that service. In addition to simplifying the development

of clients, the intermediate service provides a model for implementing of reusable groundings. In S2S, we used an adapter pattern[vi] and required that a core set of methods be implemented in each "grounding" (see Figure 2). Once an adapter for a specific Web service technology has been created, it can be reused on all other Web services that use the same technology. Using a similar architecture for SWSs, a grounding from a specific ontology to a syntactic Web service technology need only be implemented once, instead of for each client.

Discussion

The S2S framework has employed a generic service description (the S2S ontology) in the development of an intermediate "proxy" service to semantically annotated Web services. It should be noted that this solution has been designed in the context of stateless search and retrieval, but it will be interesting to see if it can be extended to enable more interactive, stateful transactions, such as those found in many of the use cases for SWSs. The S2S Framework provides a proof of concept for the utility of a common communication interface to Semantic Web Services. In order for such a communication interface to be developed, an abstract set of methods should be specified, which would be implemented for each grounding. These methods should be general enough to cover any Web service communication model, such as the four types of WSDL[vii] operations (*request-response*, *one-way*, *notification*, and *solicit-response*). Another requirement is the development of a standard for the communication protocol, or at the very least a convention. One candidate for the communication protocol is to use HTTP POST to submit RDF data using the OWL-S vocabulary. Section 6.1 of the OWL-S W3C submission discusses the correspondence between instances of OWL-S atomic processes and the four types of WSDL operations. This sort of work is on the horizon for S2S, and the development of standards for SWS "proxies", including an architecture design for groundings and a communication protocol, will facilitate the broader use of SWSs in general.

---

[i] TWC, http://tw.rpi.edu/
[ii] S2S, http://tw.rpi.edu/web/project/sesf/workinggroups/s2s
[iii] SeSF project, http://tw.rpi.edu/web/project/sesf
[iv] ESIP, http://esipfed.org
[v] OWL-S, http://www.w3.org/Submission/OWL-S/
[vi] Adapter pattern, http://en.wikipedia.org/wiki/Adapter_pattern
[vii] WSDL, http://www.w3.org/TR/wsdl