EXPRESS: EXPressing REstful Semantic Services

Areeb Alowisheq **David E. Millard**
Learning Societies Lab
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
{aaa08r, dem}@ecs.soton.ac.uk

Abstract

We introduce a new approach for RESTful Semantic Web Services called EXPRESS. It aims to exploit the similarities between REST and the Semantic Web, such as realization of resources, self describing representations, and uniform interfaces. EXPRESS is straightforward and systematic. It works by identifying resources in ontologies and provides them with a uniform interface. We also discuss other RESTful and non-RESTful approaches to Semantic Web Services and show how they relate.

1. Introduction

Web Services (WS) are a set of technologies to create a common interface for behaviour on the Web that will enable clients to invoke services remotely. Traditional WS based on WSDL/SOAP (Web Service Definition Language/Simple Object Access Protocol) do this by providing syntactic descriptions of the service being offered. However these are insufficient for service discovery, as they do not describe the semantics of the service, making it difficult for clients to find services that solve a given problem.

The Semantic Web is a set of technologies that allow for the semantic description of resources on the Web (using standards such as RDF and OWL). The Semantic Web therefore offers a solution to the lack of semantics in the Web Services world, and many people have explored the possibility of describing WS semantically to aid in their discovery. These range from light-weight solutions like WSDL-S and SAWSDL [1] to complex ones like OWL-S, WSMO and SWSF. The complexity of these approaches stems from their heavy reliance on reasoning for discovery and matchmaking. This complexity also means that they would not work at Web scale, which is a key requirement for Web Services.

REST [2] is an architectural style for network-based systems. It provides a set of constraints learnt from the Web's development and when applied can make systems scalable, reliable, reusable, and other desired features of the Web as a network-based system. Constraints of REST are: identification of resources, manipulation of resources through representations, self descriptive messages, and hypermedia as the engine of application state. REST has been adopted by the non-corporate WS community as alternative to WSDL/SOAP, and has provided a new paradigm for realizing WS. Although not always adhering to all of REST's constraints [3], RESTful Web Services are gaining more popularity and are adopted by major websites like Google, Amazon, and Yahoo.

The RESTful approach is a natural fit to the Semantic Web since the Semantic Web is based on resources and REST provides a uniform way of manipulating resources to provide Web Services.

In this paper we propose an approach called EXPRESS for making Semantic Web data available through a RESTful interface with the minimum of design and development overhead. EXPRESS uses ontologies describing classes, instances, and relationships between them to create resources and provides these resources with a RESTful interface. Because the mapping between entities -in an ontology- and resources is direct, it will be possible to provide tools that automatically create a RESTful interface for the semantic resources, greatly simplifying the deployment process. In the next section we will discuss the existing approaches for Semantic Web Services (SWS). Section 3 will explain what EXPRESS is and provide a simple example. In section 4 we conclude and discuss further work.

2. Approaches to Semantic Web Services

Table 1 illustrates how we categorize the different approaches to SWS. Shaded areas show different WS approaches. The approaches in the Semantically enhanced Web column are either SOA or REST based services with Semantic wrappers that weave them into the Semantic Web. The approaches in the Semantic Web column are not based on the mindset of traditional WS; they

acknowledge the fact that the Semantic Web is for machines and build upon it. An important aspect to note that in the Semantic Web the distinction between data and WS is blurred. That is because both WS and the Semantic Web were meant to be machine comprehensible. In this section we will discuss the different approaches in more detail.

Table 1. Approaches to SWS

	The Web	Semantically Enhanced Web	Semantic Web
Data	Web Pages	GRDDL, RDFa, Semantic Wikis	Linked Open Data
Web Services	SOA based REST based	SAWSDL, OWL-S, WSMO, SWSF SA-REST, SBWS	Semantic REST Presto Triple Space Computing

2.1. Semantically enhanced Web Services

These approaches can be either SOA based or REST based.

2.1.1. SOA based SWS. SAWSDL [1] developed from WSDL-S[4], is a light-weight solution and the only W3C SWS recommendation. It annotates WSDL components such as inputs and outputs with references to ontologies. Annotations also refer to mappings between these WSDL components and the ontologies. SAWSDL aims to be compatible with existing specifications and improve the automation of discovery and composition. Next we will discuss more ambitious W3C submissions for SWS, such as OWL-S WSMO and SWSF, and hence more complex. OWL-S [5] is based on OWL. OWL-S defines an ontology for describing WS. It describes three aspects of the service: profile, process and grounding. The profile is for advertising and discovery and contains non-functional and functional properties (inputs, outputs, preconditions and effects.) The service process describes the logic of the service how inputs relate to outputs and preconditions to effects. The grounding describes mapping from the ontological description to a concrete specification of a service, for example to WSDL. OWL-S limitation is in using OWL as a language based on description logics. OWL-S is overcoming this by incorporating SWRL [6] for defining rules. WSMO [7], another approach, is based on four major elements for modeling WS: ontologies, web services, goals and mediators. Ontologies provide the terminology to describe the domain and services. Web services describe service capabilities (preconditions, assumptions, postconditions and effects) and interfaces (choreography -defining exchanged messages- and orchestration.) Goals model service requester's requirements which are used for matchmaking with service capabilities. Mediators handle heterogeneity. WSMO uses WSML[1] as the language for modeling ontologies and rules. It is more expressive than OWL and more complex. One of the criticisms of WSMO is its drifting from W3C standards[8]. Efforts have been made to bridge between them. SWSF is one of the latest approaches to SWS. It builds upon the experiences of OWL-S and WSMO. It is related to BPEL4WS and like WSMO it has its own language for defining ontologies.

2.1.2. REST based SWS. With RESTful WS gaining more popularity on the Web, interests in RESTful Semantic WS are rising. SA-REST [9] is similar to SAWSDL, as it semantically annotates RESTful WS, but because there are no WSDL files for RESTful WS, it adds the annotations to web pages that describe the services. The idea is to use Microformats such as GRDDL[2] and RDFa[3] to embed the annotations in HTML files. By adding semantics SA-REST aims to provide an easier way to create and coordinate mashups. Another approach to RESTful SWS was introduced by Battle and Benson [10] in their approach Semantic Bridge for Web Services (SBWS), they annotated WADL[4] documents, similar to SAWSDL, this linked WADL components to ontologies.

2.2. [Semantic Web] Services

Another part of Battle's and Benson's [10] work involved providing a RESTful interface for Semantic data in a term they called Semantic REST. They mapped the HTTP (GET, PUT, POST and DELETE) into SPARQL commands including extensions proposed by HP's Jena team thus becoming (SELECT, INSERT, MODIFY and DELETE). In this way RDF datasets offering SPARQL endpoints can also offer new RESTful functionality enabling integrating them with Web 2.0 clients.

Presto [11] provides a RESTful interface for resolving OWL ontologies and endpoints for DL and SPARQL queries. This is effective when ontologies are large, e.g. in life sciences. Presto publishes the entities in OWL files and enables retrieval of axioms about these entities through a RESTful interface. We can view Presto as a RESTful WS for resolving to OWL ontologies. Although Presto does not aim to offer a general framework for WS, it shows the straight forward mapping from OWL entities to resources.

---

[1] Web Service Modeling Language, http://www.w3.org/Submission/WSML/

[2] Gleaning Resource Descriptions from Dialects of Languages, http://www.w3.org/TR/grddl/

[3] RDFa in XHTML, http://www.w3.org/TR/rdfa-syntax/

[4] Web Application Description Language, introduced to describe interfaces for RESTful WS, https://wadl.dev.java.net/

Another approach that is based on semantic constructs is Triple Space Computing (TSC) [12]. It is based on Tuple Space Computing. The communication is shifted from being message oriented as in WS, to reading and writing RDF triples in a shared triple space. TSC has been used in both web service coordination [13] and communication [14].

3. EXPRESS

In our method instead of providing semantic wrappers describing WS, we argue that by combining the expressivity and semantics in ontologies and providing a uniform interface to them, RESTful Semantic WS can be created with minimal overhead. In this section we will briefly describe the method then explain how the method can be applied in a simple example.

3.1. Method

A service provider describes entities and relationships between them as an ontology (described in an OWL file). The OWL file would be used by an EXPRESS deployment engine to create a RESTful interface for the service provider. It is used to generate URIs for resources like classes, instances and properties. The service provider then specifies which of the HTTP methods (GET, PUT, POST and DELETE) can be applied to these resources. The deployment engine generates stubs responding to these methods. The service provider can then map the stubs to existing business logic.

The clients using these services can identify which methods are allowed on which resources using the HTTP method OPTIONS [15]. The OWL file specifies the exchanged messages format thus offering decoupling.

The method is simple and generic and can be applied to any ontology. It also builds upon existing standards and does not introduce additional complexity.

3.2. Use Case

We have chosen a short and simple example to demonstrate the primary ideas in EXPRESS. In this example a pizza takeaway wants to enable ordering and delivering of pizzas via WS. We will refer to the pizza takeaway as the service provider. There are two types of clients: customers and a carrier service delivering pizzas.

The service provider needs to provide an ontology describing entities it wants clients to deal with. In this case they are: Pizzas, Customers, and Orders. The following listing describes the relevant parts of the ontology formatted in N3

```
:Pizza       a       owl:Class.
:Meat        a       :Pizza.
:Cheese      a       :Pizza.
:Veggie      a       :Pizza.
:Order       a       owl:Class.
:hasPizzas   a       owl:ObjectProperty;
  rdfs:domain :Order; rdfs:range :Pizza.
:OrderedBy   a       owl:ObjectProperty;
  rdfs:domain :Order; rdfs:range :Customer.
:hasStatus   a       owl:DatatypeProperty;
  rdfs:domain :Order;  rdfs:range xsd:string.
:hasTime     a       owl:DatatypeProperty ;
  rdfs:domain :Order; rdfs:range xsd:dateTime.
:Customer    a       owl:Class.
:hasAddress  a       owl:DatatypeProperty;
  rdfs:domain :Customer; rdfs:range xsd:string.
```

The OWL file is used to create a RESTful interface for the resources. The file is parsed; classes, properties and individuals are given URIs based on their names in the file. The following are examples of generated URIs.
http://www.server.com/Order (URI for a class)
http://www.server.com/Meat (URI for a pizza instance).
http://www.server.com/order11233 (URI for an order instance), the properties of this order also have URIs, for example this order's status has the following URI
http://www.server.com/order11233/hasStatus

The service provider then states, via mechanisms such as access control lists, which methods (GET, PUT, POST and DELETE) can be applied to each URI. If the server provider has several types of clients it can state that permitted methods on a URI differ depending on what type of client is accessing it.

After specifying the access control lists, stubs are automatically created. The service provider can map these stubs to existing services or code the logic inside them.

To illustrate how this works we will show how the client can use the service to order a pizza. We assume the client has already discovered the service. For the client to invoke the service it needs to have the OWL file. It can access it from the service in the same way it GETs any other resource. The purpose of the OWL file is to show the resource representation -and thus the exchanged messages format-, relationships, and special instances. The client also needs to know how to invoke HTTP methods on resources.

After the client has got the OWL file, to place an order it sends a POST request to *http://www.server.com/Order/*
The server will respond by creating a new order and sending back its URI to the client. For example *http://www.server.com/order11233*, the client then can invoke a PUT on the sent URI with the order information in the message. The format of the message is based on the OWL file discussed in the beginning of this section.
The format of the message in N3 is

```
:order11233     a       :Order ;
  :hasPizzas :Cheese ;
  :hasTime "2009-04-23T11:19:35"^^xsd:dateTime;
  :OrderedBy :c1245.
```

The `OrderedBy` property indicates which customer sent placed the order. There is a requirement that a customer must be created on the server before placing an order. The server conveys this in the OWL file using restrictions. The restriction that exists on the Order class is

$$Order \sqsubseteq \exists\, isOrdered.Customer$$

In this way the client can know before placing an order that a customer must exist and if it does not, then it needs to create it. This use of constraints to convey a shared understanding of workflow is a key difference between EXPRESS and other SWS approaches that are more formal about specifying choreography.

The process of creating customers is the same as placing orders because of the uniform interface; a POST request is sent to *http://www.server.com/Customer/* the server responds with creating a new customer and sending back the URI, the client PUTs the required customer information to the returned URI.

We can realize some patterns in the way clients can manipulate resources, which is based on the type of the resources and operations allowed on them the following table illustrates the concept

Table 2. Patterns of manipulating resources

Resource	OPTIONS	Semantics
Class in an ontology	GET	Gets the structure of the class
	POST	A factory endpoint to create instances of this class
Instance or property	GET {only}	Read only instance or property
Instance or property	GET, PUT, DELETE	Modifiable instance or property

As an example of how access control lists can be used, on the URI *http://www.server.com/order11233/hasStatus* customer clients can only invoke GET. Carrier service delivering pizzas can invoke GET or PUT, but cannot modify other Order properties, however customer clients can.

4. Conclusions and Future Work.

In this paper we have proposed the EXPRESS approach as an alternative to more sophisticated WSMO or OWL-S methods, we believe its simplicity and nativity to both the Web and the Semantic Web, harvesting the strengths of both, and introducing the minimum level of complexity are features that are interesting to investigate. Since the adoption of OWL-S and WSMO is still yet to be seen, the feasibility of the complexity introduced by them and similar approaches is questionable. For future work we aim to develop a deployment engine for EXPRESS and investigate how we might use EXPRESS to model complex web services (such as transactions). We also aim to investigate methods that enable discovery of EXPRESS services with: minimal complexity, provide pragmatic solutions that that can contribute towards the deployment of linked data and help build the Semantic Web.

References

[1] J. Farrell and H. Lausen, *Semantic Annotations for WSDL and XML Schema*, World Wide Web Consortium (W3C), 2007.

[2] R.T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Doctoral dissertation, University of California, 2000.

[3] R.T. Fielding, "A Little REST and Relaxation," 2007. http://www.parleys.com/display/PARLEYS/A+little+REST+and+Relaxation

[4] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A.P. Sheth, and K. Verma, *Web Service Semantics WSDL-S*, W3C Submission, 2005.

[5]D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. Mcllraith, S. Narayanan, M. Paulocci, B. Parsia, T.R. Payne, E. Sirin, N. Srinivasan, and K. Sycara, *OWL-S: Semantic Markup for Web Services*, W3C Submission, 2004.

[6] I. Harrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, W3C Submission, 2004.

[7] J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, U. Keller, M. Kifer, B. König-Ries, J. Kopecky, R. Lara, H. Lausen, E. Oren, A. Polleres, D. Roman, J. Scicluna, and M. Stollberg, *Web Service Modeling Ontology (WSMO)*, W3C Submission, 2005.

[8] C. Bournez, *Team Comment on Web Service Modeling Ontology (WSMO) Submission*, W3C, 2005. http://www.w3.org/Submission/2005/06/Comment.html

[9] A.P. Sheth, K. Gomadam, and J. Lathem, "SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups" *IEEE Internet Computing*, vol. 11, 2007, pp. 91–94.

[10] R. Battle and E. Benson, "Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST)," *Web Semantics*, vol. 6, 2008, pp. 61–69.

[11] A. DeLeon and M. Dumontier, "Publishing OWL Ontologies with Presto," In OWLED, OWL: Experiences and Directions, 2008.

[12] J. Riemer, F. Martin-Recuerda, Y. Ding, M. Murth, B. Sapkota, R. Krummenacher, O. Shafiq, D. Fensel, and E. Kühn, "Triple Space Computing: Adding Semantics to Space-Based Computing," *The Semantic Web – ASWC 2006*, 2006, 300-306.

[13] D. Fensel, R. Krummenacher, O. Shafiq, E. Kühn, J. Riemer, Y. Ding, and B. Draxler, "TSC – Triple Space Computing," *e & i Elektrotechnik und Informationstechnik*, vol. 124, Feb. 2007, pp. 31-38.

[14] D. de Francisco, L. Nixon, and G. Toro del Valle, "Towards a Multimedia Content Marketplace Implementation Based on Triplespaces," *The Semantic Web ISWC*, 2008, pp. 875-888.

[15]L. Richardson and S. Ruby, *RESTful Web Services*, O'Reilly Media, 2007.