EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies

Areeb Alowisheq, David E. Millard and Thanassis Tiropanis

Learning Societies Lab, School of Electronics and Computer Science
University of Southampton, Southampton, SO17 1BJ, UK
{aaa08r, dem, tt2}@ecs.soton.ac.uk

Abstract. Existing approaches to Semantic Web Services (SWS) require a domain ontology and a semantic description of the service. In the case of lightweight SWS approaches, such as SAWSDL, service description is achieved by semantically annotating existing web service interfaces. Other approaches such as OWL-S and WSMO describe services in a separate ontology. So, existing approaches separate service description from domain description, therefore increasing design efforts. We propose EXPRESS a lightweight approach to SWS that requires the domain ontology definition only. Its simplicity stems from the similarities between REST and the Semantic Web such as resource realization, self describing representations, and uniform interfaces. The semantics of a service is elicited from a resource's semantic description in the domain ontology and the semantics of the uniform interface, hence eliminating the need for ontologically describing services. We provide an example that illustrates EXPRESS and then discuss how it compares to SA-REST and WSMO.

Keywords: Semantic Web, Semantic Web Services, Ontologies, REST, SA-REST, WSMO.

1 Introduction

The emergence of Web Service technologies offers great business opportunities. Traditional Web Services, based on the SOAP/WSDL standards provide syntactic descriptions of services. Offering syntactic descriptions however, is insufficient for the automation or semi-automation of service discovery and composition, stating that a service accepts an integer and returns a string will not offer information on what the service does. In order to solve this problem research has been done to semantically, rather than syntactically, describe Web Services. The Semantic Web is a set of technologies enabling the semantic description of resources using standards such as RDF and OWL. Therefore it offers a solution to the lack of semantics in the Web Services world. The research community has introduced several approaches for Web Service semantic descriptions. These range from lightweight solutions like SAWSDL [1] to complex ones like OWL-S [2] and WSMO [3]. The complexity of these approaches stems from their heavy reliance on logical reasoning for the automation of discovery, matchmaking and composition. This complexity also means it will be very challenging for these features to be available at Web scale [4] [5] [6]. There is a trade-off between automation and scalability, and existing SWS approaches tend to focus on automation. Recently, there has been a rising interest in lightweight SWS for reasons of scalability and minimising complexity and design overhead [7] [8].

Another issue with these approaches, whether heavy or lightweight, is that in addition to semantically describing services they require a semantic description of the domain. This separation of domains and services descriptions stems from the SOA and RPC mindset these approaches are based on. This was the prevalent mindset in traditional Web Services when SWS research began. However another approach to Web Services came forward, known as RESTful Web Services. This approach is based on REST [9] where resources are key actors just as services are in SOA.

REST is an architectural style for network-based systems. It provides a set of constraints learnt from the Web's HTTP development and when applied can make systems scalable, reliable, reusable, resilient 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 was not introduced as an approach to designing web services, yet it has been adopted by the non-corporate Web Service community as alternative to SOAP/WSDL. Although not always adhering to the all of REST's constraints [10] [11] [12], RESTful Web Services are gaining popularity and are adopted by major service providers 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 to provide Web Services. In this paper we explain an approach we called EXPRESS [13] that offers Semantic RESTful Web Services by exploiting Semantic Web resources through a RESTful interface with the minimum of design and development overhead. EXPRESS uses the ontologies that describe classes, instances and relationships among them to create resources accessible via RESTful interfaces. Because the mapping between entities in an ontology and resources is direct, we created a tool that automatically creates a RESTful interface for the semantic resources, therefore simplifying the deployment process. The next section provides a brief overview of existing SWS approaches. In section 3 we discuss EXPRESS with an example. In section 4 we briefly compare EXPRESS with SA-REST and WSMO, then we conclude by highlighting the research questions and future directions.

2 Approaches to Semantic Web Services

We can classify SWS approaches into two main categories: in the first category are approaches that semantically enhance Web Services. The second are approaches that are based on manipulating semantic resources.

2.1 Semantically Enhancing Web Services Approaches

These approaches can be either based on SOA or based on REST.

2.1.1. SOA based SWS. SAWSDL [1] is a lightweight solution and the only W3C SWS recommendation. It annotates WSDL components such as inputs and outputs with references to ontologies. More ambitious W3C submissions for SWS, such as OWL-S and WSMO, are more complex. OWL-S [2] based on OWL, defines an ontology describing 3 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 how inputs relate to outputs and preconditions to effects. The grounding maps to a concrete service specification. The limitation of OWL-S is in using OWL as a language based on description logics. OWL-S is overcoming this by incorporating SWRL [14] for defining rules. WSMO [3], another approach, is based on 4 major elements for modelling: 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 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 modelling ontologies and rules. It is more expressive and complex than OWL. A criticism of WSMO is its drifting from W3C standards. Efforts have been made to bridge between them.

2.1.2. REST based SWS. RESTful WS are gaining more popularity, and interests in RESTful SWS are rising. SA-REST [8] 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. It uses GRDDL[2] or 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. hRESTs and microWSMO [7] are similar approaches to SA-REST. Another approach was introduced in [15] in their approach Semantic Bridge for Web Services (SBWS), they annotated WADL[4] documents linking them to ontologies.

2.2 Semantic Resources Based Approaches

Another part of the work in [15] involved providing a RESTful interface for Semantic data called Semantic REST. They mapped the HTTP methods into SPARQL commands that included proposed extensions for insertion, deletion and updating. In this way RDF datasets offering SPARQL endpoints can offer RESTful functionality integrating them with Web 2.0 clients.

---

[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, describes interfaces for RESTful WS, https://wadl.dev.java.net/

Another approach that is based on semantic constructs is Triple Space Computing (TSC) [16]. It is based on Tuple Space Computing. The communication is shifted to reading and writing RDF triples in a shared triple space.

3 EXPRESS

EXPRESS eliminates the need for describing services separately because it provides resources with a uniform interface. The uniform interface is the HTTP methods GET, PUT, DELETE, POST and OPTIONS which define consistent operational semantics on all resources. The resources that EXPRESS exploits are entities described semantically in an OWL ontology. So by combining the expressivity and semantics in ontologies and providing a uniform interface to them, RESTful SWS can be created.

A service provider using EXPRESS provides an OWL file describing the resources in a Web Service. This is run through an EXPRESS deployment engine to generate URIs for classes, instances and properties. The service provider then specifies which of the HTTP methods can be applied to these resources and this can differ for different kinds of users, providing a role based access control (RBAC) at the resource methods level. The method is simple and generic and can be applied to any ontology. It builds upon existing standards and does not introduce additional complexity.

In this section we will describe how the method is applied in a simple example. We chose Amazon's Simple Storage Service S3[5], because it is a real service, it is simple so we describe how EXPRESS works in a limited space, and it is familiar to readers interested in REST[6]. S3 enables storing and managing data programmatically on Amazon's servers. It also provides the owner of the data with the ability to charge for downloads. There are two main concepts to manage users' data, Objects (data files) and Buckets (containers of these data files). S3 provides URIs for these objects. For example a file with a name -or key as S3 calls it- *doc* in a bucket *b1* would have the following URI *http://s3.amazon.com/b1/doc*. S3 also enables owners to control access to their data. S3 provides both REST and SOAP API.

3.1 A RESTful Semantic S3 Service

If Amazon wanted to provide a RESTful Semantic Web Service for S3, it should provide an ontology describing resources in S3 and relationships between them. We assume that this OWL file is provided. The next listing describes the relevant parts:

```
:User      a   owl:Class.
:Name      a   owl:DatatypeProperty;
               rdfs:domain :User;      rdfs:range xsd:string.
:Bucket    a   owl:Class.
:Key       a   owl:DatatypeProperty;
               rdfs:domain :Object;    rdfs:range xsd:string.
:Owner     a  owl:ObjectProperty;
               rdfs:domain :Bucket;    rdfs:range :User.
```

[5] Amazon Simple Storage Service (Amazon S3), http://aws.amazon.com/s3/

[6] This example, as a non-semantic RESTful Web Service is explained in [11]

```
:RequestPay  a  owl:DatatypeProperty;
                rdfs:domain :Bucket;    rdfs:range xsd:boolean.
:CreationDate a  owl:DatatypeProperty;
                rdfs:domain :Bucket;    rdfs:range xsd:string.
:Objects     a  owl:ObjectProperty;
                rdfs:domain :Bucket;    rdfs:range :Object.
:Object            a owl:Class.
:ContainingBucket  a owl:ObjectProperty;
                rdfs:domain :Object    rdfs:range :Bucket.
```

The OWL 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://s3.amazon.com/User (a class URI)
http://s3.amazon.com/Bucket/MyBucket (a bucket instance URI)

Properties also have URIs, for example the bucket's creation date has this URI

http://s3.amazon.com/Bucket/MyBucket/CreationDate.

An Amazon developer specifies which methods (GET, PUT, POST and DELETE) can be applied to each URI. The stubs are generated then the Amazon developer maps these stubs to existing services. Before providing an example, we will explain the differences between the URI structure in the existing S3 and our proposed S3. In the existing S3 the URIs of buckets and objects have the following forms respectively

http://s3.amazon.com/{bucket name}
http://s3.amazon.com/{bucket name}/{object name}

In our proposed S3 service the forms of the URIs are

http://s3.amazon.com/Bucket/{bucket name}
http://s3.amazon.com/Object/{object name}

The difference in the URI forms stems from design decisions. In the existing S3 there are only two types of resources: buckets and objects. The routing of requests to the processes dealing with each type is based on the structure of the URI. If a request was to *http://s3.amazon.com/**myspace*** then this will be considered a bucket and will be routed to the function that processes buckets. However if the request was to *http://s3.amazon.com/myspace/**m1*** it will be considered an object and routed to the processing function. EXPRESS however, is designed for a general purpose and in most cases there will be more than two resources in the system and therefore the routing decisions could not be made based on URI structure only. The URIs are designed to include the type of the requested resources as shown above and this also acts in accordance with the W3C note on cool URIs[7].

Now we will provide a simple scenario of how the service works by showing how a user can create a bucket, add objects to it and delete it. The interaction starts by the client accessing the OWL file. It can access it 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. If the client wants to use the existing S3 services it will have to sign up with Amazon. The semantics of this action is creating a user. The OWL file contains the URIs of resources the client can manipulate. Restrictions in the OWL file such as $Bucket \sqsubseteq$

[7] Cool URIs for the Semantic Web http://www.w3.org/TR/cooluris/

$\exists\, Owner.User$ indicate that a user must be created before creating a bucket. As resources have a uniform interface - HTTP methods- the client knows how to create any resource, so the client will send a POST request to *http://s3.amazon.com/User* the message will contain required user information, specified by the OWL file as all the properties where user is the subject. In the excerpt of the S3 OWL file above the only property is required is the name, although other properties are required such as authentication information we did not discuss them due to space limitations. The service response will be creating a new user resource and returning its URI for example *http://s3.amazon.com/User/user1234*. The client can create a bucket in a similar approach. It sends a PUT request to *http://s3.amazon.com/Bucket/* and the message will be:

```
:MyBucket     a       :Bucket;
       :Key   "MyBucket"^^xsd:string;
       :Owner :user1234;
       :RequestPay  "false"^^xsd:boolean.
       :AccessControlPolicy "public-read"^^xsd:string.
```

Because the name of the bucket is specified by the client the PUT method is used instead of POST to create it. This complies with the HTTP standard. The server responds by creating the bucket at the requested URI which is *http://s3.amazon.com/Bucket/MyBucket*. In order to add an object to this bucket the client sends a similar PUT request to *http://s3.amazon.com/Object/* but with required Object properties and the file as a payload. This will make the files available at the S3 storage space and the client can then provide the URIs to its clients to download. Deleting a bucket is straightforward as a DELETE request is sent to a bucket's URI.

We can realize patterns in the way clients manipulate resources in EXPRESS, these patterns are further explained in [13]. To summarise, the Amazon developer needed to provide a domain ontology, specify control access on resources and then map the generated stubs to existing services. The resulting service will be semantic because: relationships between resources are described semantically, resources can be semantically associated to widely agreed on ontologies for example User could be defined as a subclass of foaf:person, and actions come down to adding, deleting and modifying assertions. The resulting service will also be RESTful, resources have URIs, resources have uniform interfaces, the exchanged messages are in OWL which is self-described, and the server guides the client by responding with URIs in which the client can follow where there are next states to go through.

4 Comparison to SA-REST and WSMO

In this section we will highlight the efforts in describing SWS in both SA-REST and WSMO. Creating the domain ontology is an effort that exists in all SWS approaches. We chose to compare to SA-REST, because like EXPRESS it recognises the increasing popularity of RESTful WS. SA-REST as explained in section 2 aims to integrate existing RESTful WS into the Semantic Web by semantically annotating their HTML documentations. Whereas EXPRESS is an approach of using OWL files and REST principles to describe and create RESTful SWS.

The efforts in SA-REST are creating the domain ontology then annotating the

HTML documentation. A developer must annotate documentation pages with descriptions such as sarest:input, sarest:output, sarest:operation, sarest:lifting or sarest:lowering linking them to the domain ontology. For example in the case of S3 there are approximately 30 pages, the developer must decide which ones to annotate, and then annotate them with inputs, outputs, and actions. This can increase maintenance costs especially if documentation pages are scattered. In terms of RESTfulness SA-REST is not concerned if the services it describes are actually RESTful. As mentioned in the introduction not all RESTful Web Services adhere to REST's constraints. On the other hand services designed in EXPRESS are RESTful.

In the case of WSMO much more effort is needed. The developer needs to understand WSML and its variants. And for each web service a capability and interface have to be defined. The capability consists of axioms describing: preconditions, assumptions, postconditions, and effects. Preconditions and postconditions describe the Web Service information space state before and after the Web Service execution, whereas assumptions and effects describe the world's state. Furthermore the developer needs to describe the interface consisting of the choreography and orchestration of the service. In the choreography transitions rules that guide the interaction with this service must be specified. In the orchestration the rules guiding how this service uses other services to achieve its overall functionality is stated. For a simple service like S3 at least 15 service descriptions need to be created. It must be noted however, that the efforts included to describe a Web Service in WSMO are in the aim to automate or semi-automate the discovery, composition and invocation of Web Services. Criticisms to this approach and similar ones question whether the overhead is practical, and whether the automation's limited scalability justifies such efforts [4].

5 Conclusions and Future Work.

In this paper we have explained EXPRESS, provided an example of how it works and briefly compared it to SA-REST and WSMO. The work done on EXPRESS is in its early stages. EXPRESS uses the OWL file as a description of a RESTful Semantic Service. The implemented deployment system parses OWL files and generates stubs to access the resources. It offers fine grained role based access control, controlling what can be accessed, how and who can access it. In order to understand EXPRESS better we would like to fully implement an existing system, develop the approach more, and analyse its applicability and constraints. EXPRESS's 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 motivate us to investigate it further. The research questions we would like to answer are:

1. How to perform automatic discovery and composition in EXPRESS?
2. How to facilitate transforming legacy systems to be Semantic and RESTful? Initial ideas are to start from ontologies derived from legacy DB schemas.
3. How to utilise the mapping between PI calculus and ROA Resource-Oriented Architecture [11] (an architecture influenced by REST) as described in [17] to answer the previous questions?

Our goal is provide pragmatic solutions that can contribute towards building infrastructure of the Semantic Web.

References

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

[2] 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 Member Submission, World Wide Web Consortium (W3C), 2004.

[3] J.D. 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 Member Submission, W3C, 2005.

[4] M. Klusch, “Semantic Web Service Description,” *CASCOM: Intelligent Service Coordination in the Semantic Web*, Birkhäuser Verlag, 2008.

[5] D. Fensel and F.V. Harmelen, “Unifying Reasoning and Search to Web Scale,” *Internet Computing, IEEE*, vol. 11, 2007, pp. 96–95.

[6] G. Hench, E. Simperl, A. Wahler, and D. Fensel, “A Conceptual Roadmap for Scalable Semantic Computing,” *IEEE International Conference on Semantic Computing*, 2008, pp. 562-568.

[7] J. Kopecky, K. Gomadam, and T. Vitvar, “hRESTS: An HTML Microformat for Describing RESTful Web Services,” *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2008, pp. 619-625.

[8] 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.

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

[10] R.T. Fielding, “A Little REST and Relaxation,” Jazoon’07 The International Conference for Java Technology, 2007.

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

[12] S. Vinoski, “RESTful Web Services Development Checklist,” *Internet Computing, IEEE*, vol. 12, 2008, pp. 96-95.

[13] A. Alowisheq and D.E. Millard, “EXPRESS: EXPressing REstful Semantic Services,” To appear in: *The 2nd Doctoral Workshop for 2009 IEEE/WIC/ACM International Joint Conference onWeb Intelligence and Intelligent Agent Technology* , Milan, Italy: 2009.

[14] 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 Member Submission, World Wide Web Consortium (W3C), 2004.

[15] 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.

[16] 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, pp. 300–306.

[17] H. Overdick, “The Resource-Oriented Architecture,” *2007 IEEE Congress on Services*, 2007, pp. 340-347.